



THE UNIVERSITY OF QUEENSLAND  
A U S T R A L I A

# **Energy Efficiency Models for Scientific Applications on Supercomputers**

Mark Endrei

B.E. (Hons)

*A thesis submitted for the degree of Doctor of Philosophy at  
The University of Queensland in 2019*

School of Information Technology and Electrical Engineering

# Abstract

Supercomputers are a powerful class of High Performance Computing (HPC) systems used for the most challenging science problems, including drug discovery, earthquake prediction, and climate change impacts. Scientific applications running on supercomputers use highly parallel constructs to iteratively process enormous, multi-dimensional data sets, such as billions of base pairs in a genome or grid units in the atmosphere. The largest HPC systems can consume as much electricity as a small town, so rising power costs and constraints are driving a growing focus on energy efficiency. Techniques that reduce energy consumption help to reduce constraints and costs on important research.

Scientists running applications on HPC systems encounter a number of barriers using existing energy optimisation methods. Existing methods are typically aimed at parallel application developers and HPC system administrators, rather than application users. Users often do not have the required level of system administration and programming skills. Access to tuning parameter controls may require system privileges that are not available to typical users. The tuning process is often complex and time consuming, which can be a further deterrent when scientists naturally want to focus on their research.

The complexity of optimising energy efficiency is driven by a range of factors. Optimisation methods must manage large search spaces covering the unique characteristics of a particular system and workload and their effect on performance and energy efficiency. Settings for optimum performance and energy efficiency can diverge, so trade-off options need to be identified that guide a suitable balance between energy use and performance. There is also inherent observational and prediction uncertainty in optimisation processes that needs to be considered.

This thesis presents a number of significant advances in the field of energy efficiency optimisation of parallel applications:

- The energy usage and performance impacts of bottlenecks in the system architecture and of user controllable settings are analysed.
- Statistical and machine learning system models are developed that can be trained at low cost to accurately predict trade-off options using parameters that users can control.
- A novel technique for assessing the impact of experimental error in Pareto-optimal trade-off analysis is presented.
- The design and implementation of a new tool known as HPCProbe that prototypes the proposed optimisation approach are described in detail.
- HPCProbe is used to provide a comprehensive experimental evaluation of the method for a collection of parallel kernels and scientific applications.

These advances can allow HPC application users to make accurate performance and energy trade-off decisions, at low cost, and without specialised programming or system operations skills.

## **Declaration by author**

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly-authored works that I have included in my thesis.

I have clearly stated the contribution of others to my thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, financial support and any other original research work used or reported in my thesis. The content of my thesis is the result of work I have carried out since the commencement of my higher degree by research candidature and does not include a substantial part of work that has been submitted to qualify for the award of any other degree or diploma in any university or other tertiary institution. I have clearly stated which parts of my thesis, if any, have been submitted to qualify for another award.

I acknowledge that an electronic copy of my thesis must be lodged with the University Library and, subject to the policy and procedures of The University of Queensland, the thesis be made available for research and study in accordance with the Copyright Act 1968 unless a period of embargo has been approved by the Dean of the Graduate School.

I acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material. Where appropriate I have obtained copyright permission from the copyright holder to reproduce material in this thesis and have sought permission from co-authors for any jointly authored works included in the thesis.

## **Publications included in this thesis**

- [1] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, “A bottleneck-centric tuning policy for optimizing energy in parallel programs,” *Advances in Parallel Computing*, vol. 32, pp. 265–276, 2018.
- [2] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, “Statistical and machine learning models for optimizing energy in parallel applications,” *The International Journal of High Performance Computing Applications*, 2019.

## **Submitted manuscripts included in this thesis**

No manuscripts submitted for publication.

## **Other publications during candidature**

- [3] C. Jin, B. R. de Supinski, D. Abramson, H. Poxon, L. DeRose, M. N. Dinh, M. Endrei, and E. R. Jessup, “A survey on software methods to improve the energy efficiency of parallel computing,” *The International Journal of High Performance Computing Applications*, vol. 31, no. 6, pp. 517–549, 2017.
- [4] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, “Energy efficiency modeling of parallel applications,” in *Proceedings of SC18, the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE, 2018, pp. 17:1–17:13.

## **Contributions by others to the thesis**

No contributions by others.

## **Research Involving Human or Animal Subjects**

No animal or human subjects were involved in this research.

## Acknowledgements

This thesis is the result of a challenging and stimulating adventure made possible by the generosity and support of my family, learned advisors and colleagues.

This experience would not have happened without the support and patience of my wife, Györgyi. I also thank my son and daughter, Dominic and Judy, for their enthusiasm and proof reading along the way, and my parents for their influence.

I thank my teacher and principal advisor, Prof David Abramson, for encouragement, guidance and latitude in the just right doses, and for the opportunity to be part of outstanding research and eResearch services teams. Also my associate advisors, Dr Chao Jin, as a guiding mentor and confidant, and Prof Zahir Tari (RMIT University), for valued perspective and insight.

This work is part of a larger collaboration. I thank Prof Bronis R de Supinski (Lawrence Livermore National Laboratory), an exemplar for an aspiring researcher. I also thank Dr Luiz DeRose (Cray Inc.) and Ms Heidi Poxon (Cray Inc.) for enabling, supporting and guiding this work. I gratefully acknowledge the supporting grant and infrastructure provided by Cray Inc.

Lastly, I am grateful for the advice and many discussions shared with my colleagues Dr Minh Ngoc Dinh and Dr Hoang Anh Nguyen, and for the help of Tracey Miller in navigating the administrative events of my candidature.

## **Financial support**

This research was supported by an Australian Government Research Training Program Scholarship, by the Australian Research Council under the Linkage grant scheme (project number LP150100837), and by Cray Inc.

## **Keywords**

High performance computing, energy efficiency, performance, regression modelling, machine learning.

## **Australian and New Zealand Standard Research Classifications (ANZSRC)**

ANZSRC code: 080110, Simulation and Modelling, 30%

ANZSRC code: 080309, Software Engineering, 30%

ANZSRC code: 080501, Distributed and Grid Systems, 40%

## **Fields of Research (FoR) Classification**

FoR code: 0801, Artificial Intelligence and Image Processing, 30%

FoR code: 0803, Computer Software, 30%

FoR code: 0805, Distributed Computing, 40%

# Contents

|  |             |
|--|-------------|
| <b>Abstract</b>                                  | <b>ii</b>   |
| <b>Contents</b>                                  | <b>vii</b>  |
| <b>List of Figures</b>                           | <b>xi</b>   |
| <b>List of Tables</b>                            | <b>xiii</b> |
| <b>List of Algorithms</b>                        | <b>xv</b>   |
| <b>List of Listings</b>                          | <b>xvii</b> |
| <b>List of Abbreviations and Symbols</b>         | <b>xix</b>  |
| <b>Chapter 1 Introduction</b>                    | <b>1</b>    |
| 1.1 Exascale Computing . . . . .                 | 1           |
| 1.2 Computing Energy Efficiency . . . . .        | 2           |
| 1.3 Challenges . . . . .                         | 3           |
| 1.4 Contributions . . . . .                      | 5           |
| 1.5 Thesis Structure . . . . .                   | 7           |
| <b>Chapter 2 Literature Survey</b>               | <b>9</b>    |
| 2.1 Multi-Objective Optimisation . . . . .       | 9           |
| 2.2 Imbalance Detection Methods . . . . .        | 12          |
| 2.3 Auto-Tuning Methods . . . . .                | 14          |
| 2.4 Model Based Methods . . . . .                | 16          |
| 2.4.1 Deterministic Models . . . . .             | 16          |
| 2.4.2 Statistical Models . . . . .               | 18          |
| 2.5 Power Measurement Tools . . . . .            | 19          |
| 2.6 Summary and Research Opportunities . . . . . | 20          |
| <b>Chapter 3 Energy Tuning Controls</b>          | <b>25</b>   |
| 3.1 Definitions . . . . .                        | 25          |
| 3.1.1 Electrical Power and Energy . . . . .      | 26          |

|                  |   |           |
|------------------|---|-----------|
| 3.1.2            | Energy Efficiency and Performance . . . . .           | 26        |
| 3.2              | HPC Architectural Considerations . . . . .            | 27        |
| 3.2.1            | SISD . . . . .  | 27        |
| 3.2.2            | SIMD . . . . .  | 28        |
| 3.2.3            | MIMD . . . . .  | 29        |
| 3.2.4            | DVFS . . . . .  | 32        |
| 3.3              | Tuning Parameters Analysis . . . . .                  | 33        |
| 3.3.1            | Experiments Setup . . . . .                           | 33        |
| 3.3.2            | Thread Scaling . . . . .                              | 34        |
| 3.3.3            | CPU Frequency Scaling . . . . .                       | 36        |
| 3.3.4            | Problem Size Scaling . . . . .                        | 38        |
| 3.3.5            | Node Scaling . . . . .                                | 42        |
| 3.4              | Conclusion . . . . .                                  | 43        |
| <b>Chapter 4</b> | <b>The Architecture of an Energy Tuning Framework</b> | <b>47</b> |
| 4.1              | Application Instrumentation . . . . .                 | 48        |
| 4.2              | Response Measurement . . . . .                        | 49        |
| 4.2.1            | Power Measurement . . . . .                           | 49        |
| 4.2.2            | Measurement Evaluation . . . . .                      | 50        |
| 4.3              | Response Prediction . . . . .                         | 52        |
| 4.3.1            | Model Predictors . . . . .                            | 53        |
| 4.3.2            | Sampling . . . . .                                    | 54        |
| 4.3.3            | B-Spline Model Formulation . . . . .                  | 55        |
| 4.3.4            | Neural Network Model Formulation . . . . .            | 57        |
| 4.3.5            | Model Evaluation . . . . .                            | 59        |
| 4.4              | Trade-Off Analysis . . . . .                          | 60        |
| 4.4.1            | Pareto Front Evaluation . . . . .                     | 61        |
| 4.4.2            | Measurement and Modelling Error . . . . .             | 61        |
| 4.5              | Results Reporting . . . . .                           | 63        |
| 4.6              | Orchestration . . . . .                               | 64        |
| 4.7              | Data Model . . . . .                                  | 64        |
| 4.8              | Architecture Summary . . . . .                        | 65        |
| <b>Chapter 5</b> | <b>Implementation Details</b>                         | <b>67</b> |
| 5.1              | Deployment Model . . . . .                            | 67        |
| 5.1.1            | Application Instrumentation . . . . .                 | 68        |
| 5.1.2            | Response Measurement . . . . .                        | 69        |
| 5.1.3            | Response Prediction . . . . .                         | 69        |
| 5.1.4            | Trade-Off Analysis . . . . .                          | 72        |
| 5.1.5            | Results Reporting . . . . .                           | 78        |



|                     |  |            |
|---------------------|--|------------|
| 5.1.6               | Orchestration . . . . .                    | 80         |
| 5.2                 | Implementation Summary . . . . .           | 83         |
| <b>Chapter 6</b>    | <b>Evaluation</b>                          | <b>85</b>  |
| 6.1                 | Study Overview . . . . .                   | 85         |
| 6.1.1               | Platform . . . . .                         | 85         |
| 6.1.2               | Study Parameters . . . . .                 | 85         |
| 6.1.3               | Experiments . . . . .                      | 86         |
| 6.2                 | Kernels Study . . . . .                    | 86         |
| 6.2.1               | Stencil Kernel . . . . .                   | 87         |
| 6.2.2               | Transpose Kernel . . . . .                 | 90         |
| 6.2.3               | Nstream Kernel . . . . .                   | 93         |
| 6.3                 | Applications Study . . . . .               | 94         |
| 6.3.1               | AMG Application . . . . .                  | 96         |
| 6.3.2               | LAMMPS Application . . . . .               | 100        |
| 6.3.3               | LULESH Application . . . . .               | 102        |
| 6.3.4               | WRF Application . . . . .                  | 103        |
| 6.4                 | Study Observations . . . . .               | 107        |
| 6.5                 | 3-Fold Cross Validation . . . . .          | 109        |
| 6.6                 | Conclusion . . . . .                       | 110        |
| <b>Chapter 7</b>    | <b>Conclusion and Future Directions</b>    | <b>113</b> |
| 7.1                 | Thesis Summary . . . . .                   | 113        |
| 7.2                 | Key Contributions . . . . .                | 114        |
| 7.3                 | Future Research Directions . . . . .       | 115        |
| 7.3.1               | Iterative Application Training . . . . .   | 115        |
| 7.3.2               | Heterogeneous Systems . . . . .            | 116        |
| 7.3.3               | Search Optimisation . . . . .              | 116        |
| 7.3.4               | Policy Development . . . . .               | 116        |
| 7.3.5               | Uncertainties Propagation . . . . .        | 117        |
| <b>Bibliography</b> |  | <b>119</b> |
| <b>Appendix A</b>   | <b>HPCProbe User Manual</b>                | <b>129</b> |
| A.1                 | HPCProbe Module . . . . .                  | 129        |
| A.1.1               | Command Line Options . . . . .             | 129        |
| A.1.2               | Initialisation File Options . . . . .      | 130        |
| A.1.3               | Counter Regular Expressions File . . . . . | 136        |
| A.1.4               | Sample Initialisation File . . . . .       | 137        |
| A.1.5               | Experiment Results Summary File . . . . .  | 137        |
| A.2                 | HPCModel Module . . . . .                  | 138        |

|       |                                     |     |
|-------|-------------------------------------|-----|
| A.2.1 | B-Spline Regression Model . . . . . | 138 |
| A.2.2 | Neural Network Model . . . . .      | 141 |
| A.2.3 | Pareto Trade-Off Analysis . . . . . | 141 |
| A.3   | HPCPlot Module . . . . .            | 145 |
| A.3.1 | Pareto Plots . . . . .              | 145 |
| A.3.2 | Surface Plots . . . . .             | 145 |

## **Appendix B HPCProbe API 147**

|       |                           |     |
|-------|---------------------------|-----|
| B.1   | HPCProbe Module . . . . . | 147 |
| B.1.1 | Name . . . . .            | 147 |
| B.1.2 | Description . . . . .     | 147 |
| B.1.3 | Classes . . . . .         | 148 |
| B.1.4 | Functions . . . . .       | 150 |
| B.1.5 | Data . . . . .            | 150 |
| B.1.6 | File . . . . .            | 150 |
| B.2   | HPCModel Module . . . . . | 150 |
| B.2.1 | Name . . . . .            | 150 |
| B.2.2 | Description . . . . .     | 150 |
| B.2.3 | Classes . . . . .         | 151 |
| B.2.4 | Data . . . . .            | 161 |
| B.2.5 | File . . . . .            | 161 |
| B.3   | HPCPlot Module . . . . .  | 161 |
| B.3.1 | Name . . . . .            | 161 |
| B.3.2 | Description . . . . .     | 161 |
| B.3.3 | Classes . . . . .         | 162 |
| B.3.4 | Functions . . . . .       | 177 |
| B.3.5 | Data . . . . .            | 177 |
| B.3.6 | File . . . . .            | 178 |

## **Appendix C Artefacts Archive 179**

|       |  |     |
|-------|--|-----|
| C.1   | Description . . . . .                    | 179 |
| C.1.1 | How Software can be Obtained . . . . .   | 179 |
| C.1.2 | Hardware Dependencies . . . . .          | 179 |
| C.1.3 | Software Dependencies . . . . .          | 179 |
| C.1.4 | Archive Contents . . . . .               | 180 |
| C.1.5 | Datasets . . . . .                       | 180 |
| C.2   | Installation . . . . .                   | 181 |
| C.3   | Experiment Workflow . . . . .            | 183 |
| C.4   | Evaluation and Expected Result . . . . . | 183 |
| C.5   | Experiment Customisation . . . . .       | 184 |

# List of Figures

|      |  |    |
|------|--|----|
| 2.1  | Pareto Optimality . . . . .  | 10 |
| 2.2  | Clusters for Typical Power-performance Trade-Offs . . . . .        | 10 |
| 2.3  | Pareto Front Iterations . . . . .                                  | 11 |
| 2.4  | Relaxed Pareto Front . . . . .                                     | 12 |
| 2.5  | Operations per Miss Trace . . . . .                                | 13 |
| 2.6  | Roofline Plot – In-Memory/In-Cache and Scalar/Vector FMA . . . . . | 17 |
| 3.1  | CPU Pipeline Topology . . . . .                                    | 28 |
| 3.2  | CPU Topology . . . . .   | 30 |
| 3.3  | Node Topology . . . . .  | 31 |
| 3.4  | Cluster Topology . . . . .   | 32 |
| 3.5  | Stencil Thread Scaling – Compact Thread Placement . . . . .        | 35 |
| 3.6  | Stencil Thread Scaling – Scatter Thread Placement . . . . .        | 36 |
| 3.7  | Stencil Hyper-Thread Scaling – Compact Thread Placement . . . . .  | 36 |
| 3.8  | Stencil Thread Scaling – Ring Thread Placement . . . . .           | 37 |
| 3.9  | Stencil Per Core Current Scaling Frequency . . . . .               | 37 |
| 3.10 | Stencil Frequency Scaling – Compact Thread Placement . . . . .     | 39 |
| 3.11 | Stencil Frequency Scaling – Scatter Thread Placement . . . . .     | 39 |
| 3.12 | Stencil Frequency Scaling – Ring Thread Placement . . . . .        | 40 |
| 3.13 | Stencil Problem Size Scaling . . . . .                             | 40 |
| 3.14 | Sparse Problem Size Scaling . . . . .                              | 41 |
| 3.15 | Stencil Cache Analysis . . . . .                                   | 41 |
| 3.16 | Transpose Cache Analysis . . . . .                                 | 42 |
| 3.17 | Stencil Node Scaling . . . . .                                     | 43 |
| 4.1  | HPCProbe Component Model . . . . .                                 | 48 |
| 4.2  | Probability Density Function . . . . .                             | 51 |
| 4.3  | Sample population distribution . . . . .                           | 52 |
| 4.4  | HPCModel Class Diagram . . . . .                                   | 53 |
| 4.5  | Uniform, Random, and Latin Hypercube Sampling . . . . .            | 54 |
| 4.6  | Deep Neural Network with Multiple Hidden Layers . . . . .          | 57 |
| 4.7  | Basic Structure of Hidden Unit $h$ . . . . .                       | 58 |

|      |   |     |
|------|---|-----|
| 4.8  | Pareto Front – Performance and Energy Efficiency . . . . .                  | 60  |
| 4.9  | Pareto Trade-Off Zone Construction Steps . . . . .                          | 62  |
| 4.10 | HPCPlot Class Diagram . . . . .   | 63  |
| 4.11 | HPCProbe Orchestration Components . . . . .                                 | 64  |
| 4.12 | HPCProbe Data Model . . . . .   | 65  |
| 5.1  | HPCProbe Deployment Model . . . . .   | 68  |
| 5.2  | Model Fit by Polynomial Degree . . . . .                                    | 70  |
| 5.3  | DNN Model Convergence by Data Scaling Method . . . . .                      | 72  |
| 6.1  | Stencil Measured Energy Efficiency . . . . .                                | 88  |
| 6.2  | Stencil Pareto Front – Observed and Predicted . . . . .                     | 89  |
| 6.3  | Stencil Energy Efficiency – Observed and Predicted . . . . .                | 91  |
| 6.4  | Stencil Performance – Observed and Predicted . . . . .                      | 92  |
| 6.5  | Transpose Pareto Front – Observed and Predicted . . . . .                   | 93  |
| 6.6  | Nstream Pareto Front – Observed and Predicted . . . . .                     | 95  |
| 6.7  | AMG Pareto Front – Observed and Predicted . . . . .                         | 98  |
| 6.8  | AMG Pareto Front for 40, 48 and 60 Nodes – Observed and Predicted . . . . . | 99  |
| 6.9  | LAMMPS Pareto Front – Observed and Predicted . . . . .                      | 101 |
| 6.10 | LULESH Pareto Front – Observed and Predicted . . . . .                      | 104 |
| 6.11 | WRF Total Precipitation . . . . .   | 105 |
| 6.12 | WRF Pareto Front – Observed and Predicted . . . . .                         | 106 |
| C.1  | Archive Contents . . . . .  | 180 |

# List of Tables

|      |  |     |
|------|--|-----|
| 3.1  | Kernels Summary . . . . .  | 34  |
| 3.2  | RAPL Power Capping Controls . . . . .                                | 38  |
| 4.1  | Parameters Used in the Models . . . . .                              | 54  |
| 6.1  | System Specification . . . . .                                       | 86  |
| 6.2  | Experimental Parameters . . . . .                                    | 86  |
| 6.3  | Stencil Results Summary . . . . .                                    | 90  |
| 6.4  | Transpose Results Summary . . . . .                                  | 94  |
| 6.5  | Nstream Results Summary . . . . .                                    | 96  |
| 6.6  | AMG Results Summary . . . . .  | 97  |
| 6.7  | AMG Results Summary for 48 Nodes . . . . .                           | 100 |
| 6.8  | LAMMPS Results Summary . . . . .                                     | 102 |
| 6.9  | LULESH Results Summary . . . . .                                     | 103 |
| 6.10 | WRF Results Summary . . . . .  | 107 |
| 6.11 | Models RMSE Comparison – 8 Test Cases . . . . .                      | 108 |
| 6.12 | Models Pareto Points Comparison – 8 Test Cases . . . . .             | 108 |
| 6.13 | 3-Fold Cross Validation Summary – Energy Efficiency Models . . . . . | 110 |
| 6.14 | 3-Fold Cross Validation Summary – Performance Models . . . . .       | 111 |



# List of Algorithms

|     |  |    |
|-----|--|----|
| 5.1 | Get Pareto Front with Extended Limits – Part 1 . . . . .                         | 73 |
| 5.2 | Get Pareto Front with Extended Limits – Part 2 . . . . .                         | 74 |
| 5.3 | Scale and Transform Front to Set Required Inner/Outer Limit . . . . .            | 75 |
| 5.4 | Get Vertices of Polygon/Path Enclosing All Pareto Points Within Limits . . . . . | 76 |





# List of Listings

|     |   |     |
|-----|---|-----|
| 5.1 | Pareto Set DataFrame . . . . .                | 73  |
| 5.2 | Extended Pareto Set DataFrame . . . . .       | 75  |
| 5.3 | Trade-Off Zone Pareto Set DataFrame . . . . . | 77  |
| A.1 | Sample HPCProbe Initialisation File . . . . . | 137 |



# List of Abbreviations and Symbols

| Abbreviations |  |
|---------------|--|
| ALPS          | Cray Application Level Placement Scheduler                     |
| AMG           | Algebraic Multi-Grid solver                                    |
| AVX           | Intel Advanced Vector Extensions                               |
| BIOS          | Basic Input/Output System                                      |
| BS            | Basis Spline   |
| CMOS          | Complementary Metal Oxide Semiconductor                        |
| CPU           | Central Processing Unit  |
| CoD           | Intel Cluster on Die   |
| CrayPAT       | Cray Performance Analysis Tool                                 |
| DCT           | Dynamic Concurrency Throttling                                 |
| DGEMM         | Double-precision General Matrix Multiplication                 |
| DIMM          | Dual In-line Memory Module                                     |
| DRAM          | Dynamic Random Access Memory                                   |
| DVFS          | Dynamic Voltage and Frequency Scaling                          |
| DVS           | Dynamic Voltage Scaling  |
| EC2           | Amazon Elastic Compute Cloud                                   |
| ECM           | Execution-Cache-Memory   |
| ED2P          | Energy Delay Squared Product                                   |
| EDP           | Energy Delay Product   |
| EFOM          | Energy Figure Of Merit   |
| FMA           | Fused Multiply-Add   |
| FOM           | Figure Of Merit  |
| FPGA          | Field-Programmable Gate Array                                  |
| Flops         | Floating Point Operations                                      |
| GPU           | Graphics Processing Unit                                       |
| HPC           | High Performance Computing                                     |
| HTT           | Intel Hyper-Threading Technology                               |
| LAMMPS        | Large-scale Atomic/Molecular Massively Parallel Simulator      |
| LAN           | Local Area Network   |
| LLC           | Last Level Cache   |
| LULESH        | Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics |
| MIMD          | Multiple Instruction stream–Multiple Data stream               |

---

| Abbreviations – continued |  |
|---------------------------|--|
| MISD                      | Multiple Instruction stream–Single Data stream |
| MPI                       | Message Passing Interface                      |
| MTBF                      | Mean Time Between Failures                     |
| NN                        | Neural Network                                 |
| NUMA                      | Non Uniform Memory Access                      |
| PAPI                      | Performance Application Programming Interface  |
| PBS                       | Portable Batch System                          |
| PCIe                      | Peripheral Component Interconnect Express      |
| PMPI                      | MPI Profiling Interface                        |
| PRK                       | Parallel Research Kernels                      |
| PUE                       | Power Usage Efficiency                         |
| Q-Q                       | Quantile-Quantile                              |
| QPI                       | Intel QuickPath Interconnect                   |
| RAPL                      | Intel Running Average Power Limit              |
| RMS                       | Root Mean Squared                              |
| ROB                       | Re Order Buffer                                |
| RS                        | Reservation Station                            |
| SB                        | Store Buffer                                   |
| SGEMM                     | Single-precision General Matrix Multiplication |
| SIMD                      | Single Instruction stream–Multiple Data stream |
| SISD                      | Single Instruction stream–Single Data stream   |
| SMT                       | Simultaneous Multi-Threading                   |
| SSE                       | Intel Streaming SIMD Extensions                |
| SSH                       | Secure Shell                                   |
| WAN                       | Wide Area Network                              |
| WPS                       | WRF Preprocessing System                       |
| WRF                       | Weather Research and Forecasting model         |
| YAML                      | YAML Ain’t Markup Language                     |

---



---

| Symbols    |   |
|------------|---|
| $e_{\eta}$ | Computer energy efficiency              |
| $p_{\mu}$  | Computer performance                    |
| $\rho$     | Spearman’s rank correlation coefficient |
| $\sigma$   | Standard deviation                      |

---

# Chapter 1

## Introduction

*“System power is the primary constraint for the exascale system: simply scaling up from today’s requirements for a petaFlop computer, the exaFlop computer in 2020 would require 200 MW, which is untenable. The target is 20-40 MW in 2020 for 1 exaFlop.”*

— Office of Science, U.S. Department of Energy, *The Challenges of Exascale*

### 1.1 Exascale Computing

Supercomputers are powerful High Performance Computing (HPC) systems capable of running the most demanding applications. HPC systems typically use highly parallel and interconnected hardware and software to achieve this goal. They are commonly benchmarked on their floating-point operations per second (Flops/s) performance. Exascale computers are the next generation of supercomputers capable of performing at least one exaFlops/s, or  $10^{18}$  Flops/s. The TOP500 project [5] ranks supercomputers internationally using the Linpack Benchmark [6] with performance reported in 64-bit floating-point operations per second.

Scientific researchers are a key user group for HPC systems. They use computational models and simulations for physical systems that operate at all scales, from quantum to cosmological, cells to biospheres, and storms to the global weather. Scientific applications are often performance constrained because model accuracy, resolution, and speed to solution are largely limited by the available computational power. This means, for example, that current global weather models are limited to around 10 km resolution [7]. Small features, such as storm cells, can be a significant source of error in these models. Exascale computing enables the next generation of scientific models and simulations, with accurate modelling of small scale features that provides qualitative improvements in computational model fidelity.

## 1.2 Computing Energy Efficiency

A key driver for improving the energy efficiency of computers has been to extend the time needed between battery charges for our mobile devices. Technologies such as dynamic voltage and frequency scaling (DVFS), concurrent computing, and heterogeneous architectures allow our latest laptop computers and mobile devices to provide great performance all day long without recharging.

Devices can use DVFS to select lower power states when the processor is not being fully utilised. Low power states allow the device to run longer without charging. Concurrent computing improves processor efficiency by avoiding stalls when a resource is busy, as processing can continue on other tasks instead of blocking until the resource is available. Heterogeneous architectures allow the CPU to offload some processing to specialised accelerators that are optimised for very specific workloads such as video or sound processing.

These technologies are also important for managing power costs in data centre operations. Data centres in the U.S. consumed an estimated 70 billion kWh in 2014, or about 1.8% of total U.S. electricity consumption [8]. The growth in data centre consumption levelled off at around 4% from 2010 to 2014, due in part to the adoption of improved energy efficiency practices in the data centre industry. Further improvements in energy efficiency practices were expected to restrict growth to a similar amount from 2014 to 2020 [8].

Performance and power usage improvements through chip design improvements that increase clock frequencies and miniaturisation are reaching their limits as we approach the end of Moore's law and Dennard scaling [9]. Moore's law predicted a doubling of chip transistor counts every two years, while Dennard scaling predicted that increasing transistor density enabled faster clock speeds with the same power consumption.

Performance gains in the current generation of supercomputers are typically achieved with architectures that are:

- Massively parallel, allowing concurrent computing and data analysis at the largest scales.
- Heterogeneous, integrating flexible CPUs and specialised accelerators or coprocessors.
- Highly interconnected, using deep memory hierarchies and fast connections for data movement within nodes and across the cluster.

The Summit supercomputer at Oak Ridge National Laboratory, U.S., is currently number one on the TOP500 list [5]. It has around 200,000 CPU cores, 2.2 million accelerator cores, 2.8 petabyte

memory, and a 250 gigabit/s interconnect network. Summit achieves 140 petaFlop/s performance using 10 MW power, so its energy efficiency is around 14 gigaFlops/J [5].

Comparing this level of energy use with residential energy use helps provide some perspective. The average monthly U.S. residential energy consumption in 2017 was 867 kWh [10], which is 1.2 kW power on average ( $867 \text{ kWh} / (30 \text{ days} \times 24 \text{ h/day})$ ). This means the 10 MW required by Summit is enough to power a town of around 8,300 homes. The average U.S. residential price in 2017 was about 13 cents per kWh [10], making the yearly power bill for 8,300 homes around US\$11 million! At these scales there are clearly significant opportunities for energy and cost savings if supercomputer energy efficiency improvements can be made.

Power consumption metrics for HPC systems also do not include the power used by supporting infrastructure, such as UPS, power distribution, lighting, and air conditioning [11]. A study of 289 data centres reports a 2016 average power usage efficiency (PUE) of 1.64 [12], with computer room air conditioning and other infrastructure using 64% of the IT load. A 10 kWh IT load will actually require 16.4 kWh when supporting infrastructure is added. This means that a 1.0 kWh IT load saving can provide a power saving of 1.64 kWh, or 164% of the IT load saving.

As well as the concern with energy efficiency within the HPC community, there is increasing public concern about the carbon footprint of the IT industry, that is evident in news media coverage. Business and economy section articles, such as “Iceland will soon use more energy mining bitcoins than powering its homes” [13] and “Data centre power use greater than Woolworths, Coles combined” [14], highlight the growing energy consumption of data centres compared to other sectors of the economy.

In summary, there are a range of concerns driving the high level of research activity in HPC energy efficiency. As we reach the limits of computer chip miniaturisation, innovative new techniques are needed to continue increasing performance while meeting energy constraints. The large amount of energy consumed by HPC systems and their supporting infrastructure, along with the increasing cost of that energy, provide strong environmental and financial incentives. Society as a whole is also showing increasing concern about how energy resources are allocated across the economy.

## 1.3 Challenges

HPC performance tuning is a mature field with well established practices for optimising the performance of a parallel application on a particular system. Extending these practices to allow trade-offs between performance and energy efficiency requires simultaneous optimisation of objective functions for performance and energy. The objective functions also have a range of input parameters that have divergent affects on performance and energy efficiency. This results in a multi-objective optimisation problem where there is often not a single solution that optimises both objectives. Instead, optimal

solutions exist along a Pareto frontier that establishes the trade-off ranges for each objective.

While there is a high level of research activity in HPC energy efficiency, as discussed in Chapter 2, the effectiveness of current optimisation methods is generally constrained by a combination of the following challenges:

1. Only a subset of the parameters that affect performance and energy efficiency may actually be controllable.

Many input parameters affect the trade-off between performance and energy efficiency. These parameters include application characteristics, like computational intensity, and memory and communication access patterns, and system factors, like cache design, and memory and network bandwidth.

The parameters that different HPC stakeholder groups can control will vary. For example, system developers have more control over parameters like cache and memory capacity, but less control of application characteristics. Application developers have more control over application characteristics and less control of system characteristics. Application users will have even more limited control. These users typically have the most control over job submission parameters, and limited control elsewhere.

This means that application users are often excluded because an optimisation method uses parameters that they cannot access. Even if parameters are accessible, the process of acquiring some measurements can increase the difficulty of automating the optimisation method as a practical tool.

2. The cost of searching for trade-off options must be offset by the achieved benefits.

As stated in item 1, there are many input parameters that affect performance and energy efficiency. This means optimisation tools must search a large and complicated parameter space.

A large parameter space can generate a full factorial design with many hundreds of thousands of combinations. It will typically not be feasible to conduct an exhaustive search for all Pareto efficient solutions, because the energy used in very large search runs can never be amortised over the energy saved in real world runs.

3. Reducing the search cost has an impact on the accuracy of trade-off predictions.

Search optimisation methods or objective function models are used to reduce the search cost by decreasing the amount of search space that needs to be covered. Each approach has limitations that need to be carefully considered to achieve trade-off options that are within the required error margins. For example, search optimisation results may be impacted by local minima or maxima due to measurement noise, and model results may be impacted by simplifications and approximations used in the model.



In both cases, a mechanism is required to indicate when the accuracy of the results is impacted by such limitations.

4. Valid trade-off options are excluded when measurement and prediction error is not considered.

Instrumentation measurement error and/or model prediction error mean that trade-off options occur *near* the Pareto frontier, within error limits, not just directly *on* the frontier. When error limits are not considered, the solution set is incomplete as only solutions that lie directly on the Pareto frontier are provided.

Performance and energy efficiency response curves also typically have regions that plateau or level off, which can result in large gaps in the trade-off solution set when error limits are not considered. Methods that do not consider error limits in the optimisation process are not able to provide the complete set of the Pareto efficient objective function solutions, or trade-off options.

This thesis presents research that addresses each of these challenges.

## 1.4 Contributions

This work provides a framework for accurately predicting, at low cost, the Pareto optimal performance and energy efficient configurations for parallel programs. Specifically, this thesis presents the following contributions:

1. An analysis of how changes in energy usage patterns arise from hardware and software interactions.

This work investigates how current HPC system architecture bottlenecks impact energy efficiency and performance for representative memory bound scientific computing kernels. Power and performance transitions are identified for each kernel, where tuning parameter sensitivity varies across the transition. The analysis shows significant divergence between performance and energy efficiency, confirming the requirement for a multi-objective optimisation strategy.

2. Energy and performance trade-offs using parameters that parallel application users can directly control.

This work lets parallel application users select trade-off options by setting basic HPC job scheduling parameters for running their application. Users achieve their required performance and energy trade-off option by selecting from the predicted optimal node counts, core counts, and CPU frequencies.

The proposed framework treats parameters that cannot be controlled as constraints, and optimises within the constraints. For example, a job will have a memory footprint that will typically depend on the problem size. The problem size is normally a constraint of the specific problem

the user wishes to solve. Similarly, the memory architecture of the HPC system being used is a constraint on the job memory footprint. Application users also need to operate within constraints imposed by facility managers and power suppliers, such as available processor and power controls.

This work shows that significant tuning opportunities can be accurately predicted within these constraints, for a broad set of kernel and application case studies.

3. A practical method to predict energy efficiency and performance trade-off options from few input measurements.

The proposed prediction method uses three strategies to minimise the amount of model training data required. First, sensible defaults are defined where practical, for parameters such as thread and process placement. Next, parameters that cannot be accessed or are difficult to control are accepted as constraints, such as problem size or memory hierarchy. Finally, the presented modelling techniques only require a small number of training samples to accurately predict trade-off options within the defined parameter defaults and constraints.

These strategies provide four orders of magnitude reduction in the parameter search space, allowing an accurate model to be constructed from as few as 12 measurements. To be practical, an energy tuning method must minimise the required training measurements so the energy used for training can be quickly recovered by the energy saved in live runs.

4. Multi-objective energy efficiency and performance models that accurately predict parallel application responses.

This work presents two independent modelling techniques that are capable of accurately predicting energy and performance objective functions for parallel applications. The first technique uses a B-spline piecewise polynomial model and ordinary least squares regression. The second technique uses a machine learning model implemented with a feed-forward artificial neural network. The results show that either modelling technique can be used to make accurate predictions from the selected input parameters. Further, building the models only needs a small set of sample measurements, which minimises the effort and resources required for training.

An important benefit of the modelling techniques demonstrated is model fit against training data can be quantified. Irregular objective function responses that exceed the resolution of the model will impact the statistical significance of the results. The case studies all show model results within the expected limits. For questionable results where this is not the case, users can be alerted that further analysis is needed.

5. A trade-off zone approach that improves Pareto optimisation, given measurement and/or modelling error.

This work introduces the novel concept of a trade-off zone for Pareto analysis of noisy data sets. This innovation ensures that the complete set of energy and performance trade-off options

are available to users. All solutions within error limits of the Pareto frontier are available, not just solutions directly on the front.

The proposed trade-off zone approach is generalisable as it is beneficial in any Pareto analysis scenario where the data set is noisy.

6. The development of a prototype implementation of the proposed energy tuning framework, HPCProbe.

HPCProbe is the first practical prototype of a tool allowing scientists to tune the energy use and performance of their parallel applications using accessible tuning parameters. HPCProbe is available in The University of Queensland's institutional repository, UQ eSpace, as described in Appendix C.

## 1.5 Thesis Structure

This thesis details the key contributions of the author. It sets the research context within the related work, and presents the novel features of the proposed energy tuning framework, along with the implementation and evaluation of a working prototype. A brief overview and reading guide for the rest of this document follows:

Chapter 2 introduces historical and state of the art energy efficiency tuning techniques, and shows where this research fits in the prior work. This review identifies important research gaps in the prior work and highlights contribution opportunities for this study.

Chapter 3 reviews computer energy efficiency controls, highlighting their benefits and limitations for use in a tuning framework. This analysis guides key design decisions for the solution framework presented in Chapter 4.

Chapter 4 highlights architectural features of an energy tuning framework and describes an innovative design for realising the key components. It investigates components for energy efficiency data instrumentation and measurement, predictive modelling, measurement and prediction error analysis, and results analysis and reporting. Orchestration requirements and data modelling considerations are also presented.

Chapter 5 provides details of the novel techniques used to implement a prototype demonstrating the architectural and design features of the proposed energy tuning framework. It also discusses challenges that were encountered in the implementation process.

Chapter 6 presents an evaluation the proposed energy tuning framework using case studies that increase in complexity. The case studies start with parallel software kernels commonly used within

scientific applications. More complex mini-applications used for system benchmarking are considered next. The final case study uses a full-scale weather modelling application.

Chapter 7 concludes this study with a summary of the research outcomes. It also considers future directions for the energy tuning framework and for research into supercomputer energy efficiency.

# Chapter 2

## Literature Survey

A wide-range of prior work [3] explores energy-efficient computing. This chapter introduces tools and state-of-the-art techniques for improving the energy efficiency of HPC systems. The aim is to set the context for this work within existing methods.

The chapter starts with an overview of multi-objective optimisation. Techniques using Pareto optimisation for analysing performance and energy usage trade-offs are reviewed. Approaches that combine energy and performance into a single objective are also considered.

Methods for determining optimal configurations for performance and energy usage are then surveyed in three groupings: Imbalance detection or slack recovery methods that use lower power states during times of processor underutilisation to save power with minimal performance impact; Auto-tuning methods that automatically search large configuration spaces for the optimal power and performance settings; Model-based methods that use analytic models to predict optimal settings using a range of input parameters.

Techniques for measuring system power usage are reviewed next, including on-board sensors, discrete/external power meters, and model-based approaches.

The chapter concludes with a summary of the literature review and a discussion of the identified research opportunities.

### 2.1 Multi-Objective Optimisation

Finding the right balance between performance and energy usage for parallel applications is a bi-objective optimization problem that does not have a single solution. The solution is a set of Pareto points defining available trade-off options [15, 16].

One part of a Pareto optimal solution cannot be improved without making another part worse. Geometric examples can be used to show how this works [17]. In Figure 2.1, the area of circles A and B need to be maximised while remaining within the triangle, and without overlapping each other. Figure 2.1 (c) is not Pareto optimal because the area of one circle can be increased without reducing the area of the other. Lack of Pareto optimality is an alert that a better solution may have been missed [17].

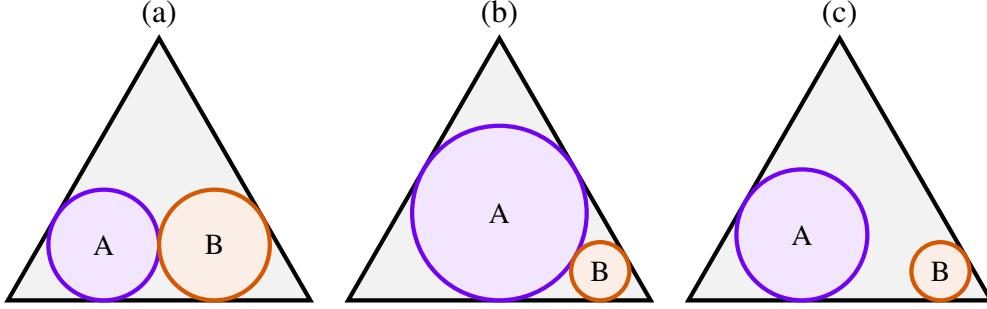


Figure 2.1: Pareto Optimality – (a, b) Pareto Optimal and (c) Not Pareto Optimal

Michanan et al. [18] study power and performance trade-offs for embedded system cache configuration parameters. Pareto solutions are categorised into power, balanced, and performance optimised clusters. Figure 2.2 shows a data set plotted by the energy efficiency and performance of each point. Both objectives need to be maximised so Pareto solutions (circled) occur along a front at the top-right of the data set. A optimisation policy favouring performance would use points in the performance optimised cluster. Balanced applies when performance and energy efficiency are equally important. Power applies when maximising energy efficiency is more important.

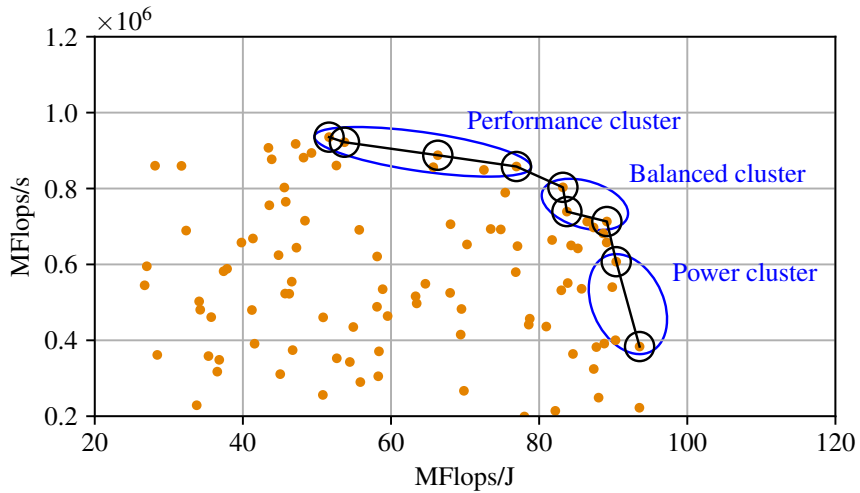


Figure 2.2: Clusters for Typical Power-performance Trade-Offs

Mezmaz et al. [16] use a hybrid genetic algorithm to identify trade-off options for completion time and energy consumption for parallel applications running on heterogeneous cloud computing infrastructure. Tuning controls include application task placement on cloud instances and processor DVS settings. A 47% energy saving is reported for a Fast Fourier Transformation task graph. Future work aims to reduce solving costs associated with the evaluation framework.

Bailey et al. [19] use a linear programming formulation to find Pareto-efficient solutions for selecting the CPU frequency and number of cores for each of the nodes executing an application. The proposed approach aims to provide theoretical bounds for evaluating systems rather than runtime optimisation. Power-constrained performance improvements of over 40% are reported.

Peachey et al. [20] present a *cosine seeking complex* algorithm for Pareto optimisation of two objectives. Search-based optimisation techniques can be efficient, but may have difficulty locating a globally optimal Pareto set for complicated search spaces. Noise on the objective function, such as measurement error, can result in a locally rather than globally optimal solution.

Sen and Wood [21] propose CPU speed governors for energy optimal computing that seek to constrain system operations to the Pareto frontier. Three new governors aim to maximise energy efficiency, maximise performance for a set power limit, and maximise power saved for a set performance level.

Care is also needed to ensure that Pareto analysis does not exclude meaningful trade-off options. This can occur when there are suboptimal points very close to the Pareto front [18] or measurement error is a factor [22].

Michanan et al. [18] use multiple iterations of Pareto analysis to bring in excluded solutions near the front. Figure 2.3 shows four iterations where Pareto points are removed from the data set and a another Pareto analysis iteration is performed. This approach is affected by the distribution of data points near the front, so there is a risk that more distant points than intended are included.

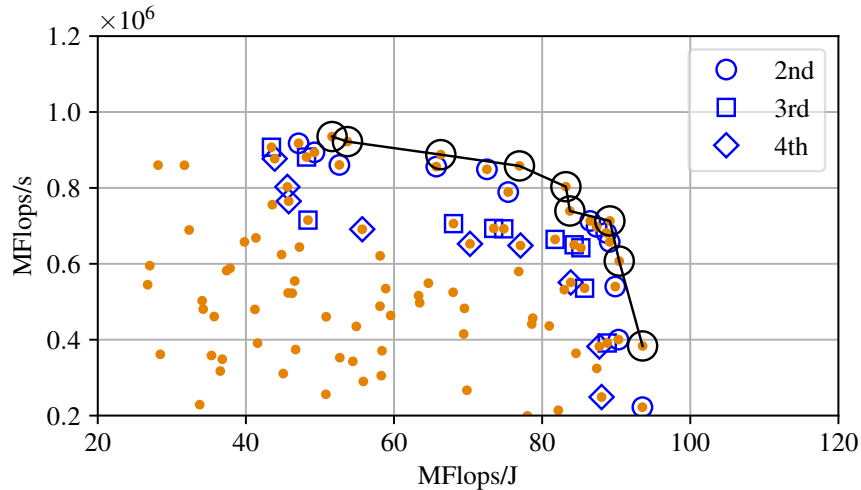


Figure 2.3: Pareto Front Iterations

Balaprakash et al. [22] use a *relaxed* Pareto front where measurement error margins of each objective are added to the Pareto points. Points that are within measurement error limits of Pareto points are added to the front. This approach is also affected by the distribution of points near the front. Figure 2.4 shows that points within limits of the frontier line can be still be excluded, in this case, along the front and to its upper left and lower right.

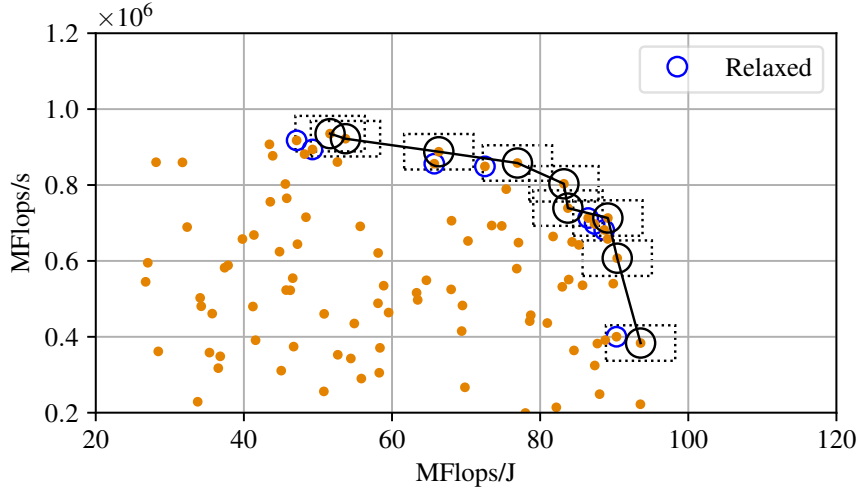


Figure 2.4: Relaxed Pareto Front

Multi-objective optimisation can be avoided if the objectives can be combined as a single objective. EDP (energy delay product –  $E \times D$ ) and ED2P (energy delay squared product –  $E \times D^2$ ) are widely used [23, 24, 25, 26, 27, 28], with ED2P placing more importance on performance than EDP. Multiple objectives can also be combined into a single result as a weighted sum [29, 30]. Relative superiority [31] is another option. These approaches all provide a single solution that can be weighted to favour performance over energy use or vice versa. A combined objective solution will lie somewhere along the Pareto front if it is Pareto optimal.

## 2.2 Imbalance Detection Methods

Imbalance detection methods aim to lower energy consumption with minimal impact on performance [32, 33, 34, 35]. Typically, these methods identify the opportunity of load imbalance and processor underutilisation at runtime to lower power consumption without significantly hurting performance. Often, profiling or tracing identifies finer regions with different degrees of idleness in a parallel program. According to each region’s pattern, such as phases with memory stalls and intensive computation, some techniques use Dynamic Voltage and Frequency Scaling (DVFS) to optimise CPU frequency in each phase.

Freeh and Lowenthal [32] show that changing CPU frequency during MPI program execution can achieve energy savings without unduly increasing execution time. CPU frequency settings provide  $g$  energy-performance points called *gears*. The program is divided into  $n$  *phases* using profiling, as figure 2.5 shows, and a gear is prescribed for each phase. Phase boundaries occur at abrupt changes in *memory pressure* measured in operations per miss. A heuristic is used to find the best gear for each phase across the  $n \times g$  search space. Program updates are needed to add sampling code for application profiling and gear changes at identified phase boundaries.

A similar approach has been applied to MPI processes distributed across multiple nodes [25]. A



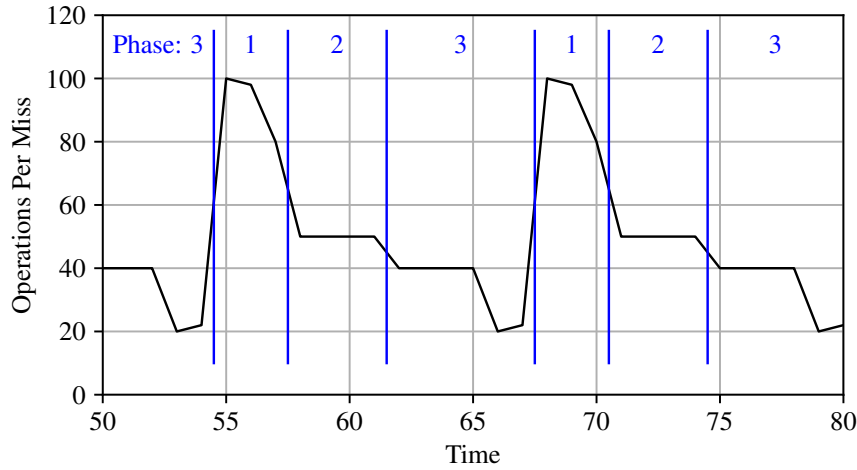


Figure 2.5: Operations per Miss Trace

benefit approaching 13% less energy used is reported for unbalanced workloads without modifying the application. Less benefit is achieved for load balanced applications where an optimum CPU frequency applies uniformly across the cluster.

Rountree et al. [35] are able to profile applications without programmer involvement using a custom MPI Profiling Interface (PMPI) library. The custom library intercepts selected MPI calls to build up a task graph for an application. The application is executed at each available CPU frequency to determine the execution time of each task at each frequency. A linear programming solver is then used to generate a task schedule that optimises energy savings while meeting an allowed time delay. Analysis costs are not considered. The focus is on providing optimal energy savings targets for evaluating lower-cost heuristic algorithms, or comparing supercomputer designs with and without DVS for example.

The Adagio system [36] is able to identify blocking MPI tasks at runtime. CPU frequency is lowered to fit the critical path using metrics from previous iterations of the task. This approach works best for task with repeating temporal patterns, and multi-core processors that can scale core frequencies independently.

Imbalance regions also occur during network communication between processes running on the HPC cluster. A number of techniques detect periods of MPI communications and tune the CPU frequency to reduce the energy used with minimal impact on performance [37, 26, 38]. This approach works well for non-iterative applications, since there is no assumption of predictable execution patterns for tasks within the application.

Other techniques use Dynamic Concurrency Throttling (DCT) to adjust the number of active cores to reduce the power wasted by core contention. Curtis-Maury et al. [24] profile OpenMP applications using hardware event counters to predict instructions per cycle (IPC) when the number of processors

and threads per processor are varied. The model requires a set of performance metrics captured in two observations of each parallel region of the application, including the IPCs achieved. Linear regression is also used to fit coefficients to a system-specific *transfer function*. The dynamically throttled results compare favourably with the best static processor and thread count configurations. This indicates that strategies tuning a fixed concurrency setting for the whole application run can provide similar benefits.

Load imbalance techniques can avoid the complexities of multi-objective optimisation by maintaining performance for the critical path of execution, while identifying activities off the critical path that can use lower performing low power states without affecting overall performance.

## 2.3 Auto-Tuning Methods

Auto-tuning tools are used to automatically optimise objectives such as power and performance, typically by searching large configuration spaces for the right input parameter settings. The state-of-the-art auto-tuning tools apply various search algorithms, such as machine learning algorithms and heuristic algorithms, to prune search spaces that consist of environment settings, compiler flags, code transforms, and application-specific parameters. Efficient search algorithms help to minimise the cost of tuning by constraining search effort to practical levels. Techniques can be exclusively off-line or online, or a combination of both. Online tuning occurs while the application runs, while off-line methods take place prior to live runs.

Ansel et al. [39] introduce the OpenTuner application tuning framework for Python programmers. OpenTuner provides a standard set of search techniques that can be extended with user-defined, domain-specific search techniques and combined into search ensembles. Search ensembles aim to minimise the search convergence time needed to locate optimal configurations. Results reported for stencil auto-tuning show optimal performance is reached after covering less than 2% of the full search space.

Li and Martinez [40] present heuristics for online tuning of the processor count and DVFS setting of a parallel application running on a multi-core processor. A combined binary search and hill-climbing optimisation is used in the processor count dimension. Measuring execution time and instructions per cycle at the highest DVFS setting then allows a target chip-wide DVFS setting to be computed. Simulation results show optimum power savings for a  $16 \times 16$  search space can be achieved in  $4 \times 3$  iterations. The authors envision an implementation involving new hardware micro-controllers and OpenMP extensions.

Gschwandtner et al. [15] use the Insieme optimising compiler to analyse time, energy and resource usage trade-offs for parallel applications. A Pareto set of the best configurations is derived and used

to compile multi-versioned code that can be dynamically selected at runtime. The Insieme differential evolution search is reported to use 5-12% of the evaluations needed for brute-force searches. A close correlation between time and energy is observed when resource usage is low.

Miceli et al. [41] describe tuning plug-ins available in the AutoTune Periscope Tuning Framework, including an energy tuning plug-in. MPI and OpenMP programs are instrumented and statically analysed in a pre-processing stage. The energy plug-in uses RAPL counters to monitor runtime power usage. A ternary search is used to reduce the search effort for the  $8 \times 3$  frequency by governor search space. Energy saving achievements are not reported.

Rahman et al. [30] use the POET code transformation engine to investigate performance and energy tuning of computationally intensive kernels. Compiler optimisations such as loop parallelisation, blocking, and unrolling are evaluated. A model for estimating power usage from machine-specific and runtime counter statistics is also assessed. A power consumption reduction of 20-30% with minimal performance degradation is reported when the allocated tuning priorities are 30% to performance and 70% to power.

Sarood et al. [42] present PARM resource manager for optimising resource allocation within a power budget. RAPL is used to dynamically tune the power draw of allocated nodes. The Charm++ runtime enables the number of allocated nodes to dynamically shrink or expand. Regression analysis of application specific profiles is used to model power usage. The reported job throughput improvements are largely achieved by reducing the delay between job arrival and start, as a result of dynamic power capping to enable over-provisioning of node allocations.

Sourouri et al. [27] present a method of fine-grained auto-tuning of OpenMP threads, and core and un-core CPU frequency. C++ preprocessor directives are added to the application code to enable energy monitoring and dynamic DVFS tuning using an exhaustive search. The approach is evaluated using a seismic wave simulator. The reported results show energy savings of up to 20% for a performance loss of 3.5% at most. The assessment shows static tuning consumes under 2% more energy than the proposed dynamic approach, given the overheads of changing the DVFS setting dynamically.

Tiwari et al. [28] analyse the interaction between energy consumption and compile optimisations using the Active Harmony auto-tuner and CHiLL code generation framework. The tuning parameters used are CPU frequency and code-transformations, such as tiling and loop unrolling factors, for computation intensive kernels. Energy savings of 5% for a performance loss of 3.9% are reported.

Abramson et al. [43] use the Nimrod toolkit for parametric studies. Nimrod is an alternative with more generalised capabilities such as search optimisation and resource management for distributed systems. A plain text schedule file specifies how to evaluate the objective function and the optimisation methods to use. It has a relatively wide user base with a range of scientific uses, such as molecular

modelling [44], functional MRI analysis [45], and computational fluid dynamics [46].

In summary, auto-tuners can provide significant tuning benefits at low cost when optimising performance and energy usage for parallel applications. These tools are aimed at application developers and HPC system administrators, rather than application users. Tools using single objective approaches are ideal when fixed optimisation priorities apply, but tools providing full visibility of the available trade-off ranges are more flexible. None of the reviewed tools have built-in mechanisms for including valid trade-off options excluded by very small margins within experimental error or noise.

## 2.4 Model Based Methods

Model-based methods aim to predict system responses, such as power use or performance, from a set of input parameters. Most power models extend existing performance models, such as Amdahl's Law, the roofline model, iso-efficiency, or the execution-cache-memory model. These models are usually deterministic or functional models because they predict an exact outcome, such as  $power = voltage \times current$ .

Stochastic or probabilistic models that predict a probable outcome with an associated error term are also used when dealing with partial or noisy data sets. These models are expressed in the form  $y = f(x) + \varepsilon$  with the error term,  $\varepsilon$ .

### 2.4.1 Deterministic Models

Woo and Lee [47] extend Amdahl's Law for multi-core, energy efficient computing. Amdahl's Law is used to calculate theoretical performance of a program that has sequential and parallel regions when using multiple processors. Performance per watt is then derived using processor idle power and power use per core in sequential and parallel regions. The technique provides theoretical limits aimed at processor designers rather than application users.

Cho and Melhem [23] extend Amdahl's Law further to model DVFS and processors that can be independently turned off. The model is used to predict frequencies for serial and parallel regions in an application to minimise energy or EDP. The approach provides additional processor design insights on parallelisation, performance and energy use trade-offs with variable speed processors.

Choi et al. [48] extend the roofline performance model to include energy limitations on performance. The roofline performance model sets two platform-specific ceilings based on peak processing performance and memory bandwidth. The roofline model of energy uses the same *operations/unit storage* workload characterisation metric, but it extends platform-specific computation and I/O time costs with energy costs. The approach aims to help algorithm designers and performance turners understand the relationships between, and bounds on, performance and energy use.

Ofenbeck et al. [49] analyse roofline plots with measured data for common numerical functions. Figure 2.6 shows an application with a set configuration (App 1) has a specific operational intensity and performance resulting in a point in the plot. Another application and configuration (App 2) appears at another point. Both points are below the roofline since it sets upper bounds on peak performance.

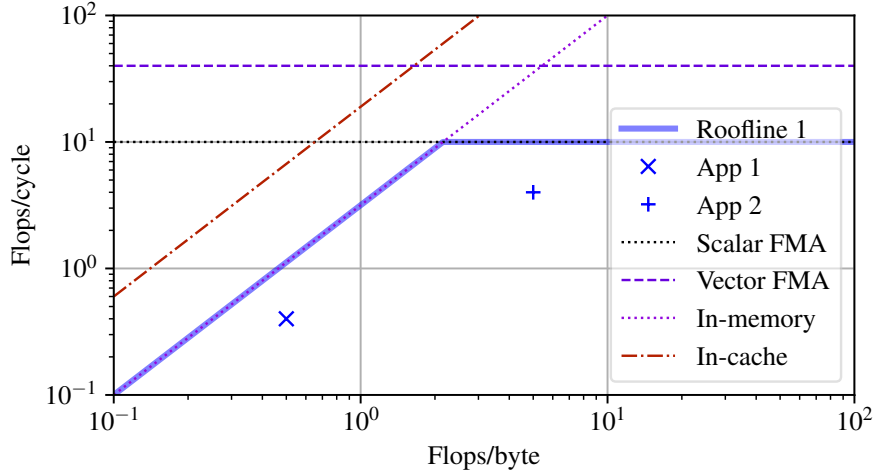


Figure 2.6: Roofline Plot – In-Memory/In-Cache and Scalar/Vector FMA

Roofline parameters can have multiple values depending on the application and platform. For example, the measured computational intensity varies depending on how *operations* and *unit storage* are defined. The computational intensity of a single-precision Flop will be double that of a double-precision Flop if *unit storage* is in bytes. Figure 2.6 shows using scalar (SISD) or vector (SIMD) operations will have a similar effect. The computational intensity of applications working in system cache or memory will also vary.

The value of the roofline approach is in the insight provided to algorithm coders on performance and energy bounds, rather than predicting system responses for complex applications.

Hofmann and Fey [50] use the execution-cache-memory (ECM) performance model and system tuning to minimise parallel application energy to solution. ECM is used to guide vector and streaming (SIMD), and cache blocking performance optimisations for a single-core. Power draw per core is then tuned using CPU frequency and Cluster on Die (CoD) settings. Last, the number of cores is tuned to avoid memory saturation. The method is applied to a 2D Jacobi solver on recent Intel processors. CPU socket-level energy consumption reductions of over  $2\times$  are reported.

Song et al. [51] apply the concept of iso-efficiency in a model that predicts application energy consumption as parameters such as concurrency and clock frequency change. Platform-specific model inputs include computational speed and throughput. Application-specific inputs include the level of parallelism, and average time and energy costs of computations and I/O. The reported model predic-

tion error for system energy use is within 5% on average. Platform-specific parameters are determined automatically, but some application parameters require expert observation and investigation.

Ramapantulu et al. [52] use a measurement-driven analytical model to obtain optimal configurations for energy and time trade-offs on heterogeneous clusters. Workload-specific measurements including the number of instructions, and work and stall cycles are obtained by micro-benchmarking each operation within a parallel application. Platform-specific measurements include CPU idle power, active power across cores and frequencies, and stalls power; memory idle and active power, and I/O power. The model is evaluated on a homogeneous AMD cluster and a heterogeneous AMD/ARM cluster. The energy reduction reported when changing from the homogeneous to heterogeneous cluster is up to 58% while maintaining the same completion time. The stated model prediction error is less than 15%.

Mitra et al. [53] present a model for exploring energy-optimal work partitioning for an application running on heterogeneous systems. CPU and GPU inputs for the model include active and idle power, computational rates, and work assignment split. Application inputs include total computational cost and time to solution. Evaluation reports using SGEMM and DGEMM show performance and energy trade-offs exist when partitioning work between CPU and GPU that are consistent with the model.

Varghese et al. [54] extend prior work [53] to understand energy use under CPU frequency and core scaling. The relationship between power and frequency is established by curve fitting measured data. This data includes platform-specific power and performance metrics for CPU-bound and memory-bound benchmarks while varying core count and CPU frequency. The eventual aim is to model a general workload using application-specific instruction mix data and the platform-specific benchmark data. The incremental model development approach used is clearly required to manage the level of complexity needed to create better, more generalisable deterministic models.

## **2.4.2 Statistical Models**

Barnes et al. [55] use multivariate regression to predict performance at large processor scales from training data for smaller processor counts. Three polynomial models are formulated with input parameters that include computation and communication metrics collected using PMPI profiling and extrapolated to larger configurations, application input variables, and processor count. Prediction errors of between 6.7% and 17.3% are reported from an evaluation of seven applications using between 16 and 64 training runs. The best predictions are generally provided by the model with the lowest RMS error across the input data.

Lee et al. [56] construct piecewise polynomial and artificial neural network models for predicting application performance. The application parameter spaces used are very large. Training data for the presented evaluation consists of 600 samples selected using a combination of regional, uniform

and random sampling approaches. Prediction errors ranging 2.2% to 10.5% are reported for two applications on three HPC platforms. Assessment of error sensitivity to sample size indicates that *a priori* selection of optimal sample size remains a challenge.

Lively et al. [57] use linear regression to predict performance and power use for hybrid MPI/OpenMP applications. Model training requires five data points for each application consisting of 40 PAPI counters captured in three single-node and two two-node runs. A principal component analysis method is used to select counter events with the strongest correlation to the application response. Training data counter rates are extrapolated to larger configurations using curve fitting. The models can then be used to predict time and power from a subset of counter events on a larger configuration. The reported model accuracy for four programs running on up to 16 MPI processes with 8 OpenMP threads per process is 97%. CPU frequency tuning was not assessed.

Johnston et al. [58] construct a random forest regression model using 37 OpenCL kernels with four problem sizes running on 15 different GPU/accelerator platforms. Model training data for each kernel on each platform is collected using a workload characterisation tool. The source code for each kernel is instrumented by a developer to enable profiling. A simulated-annealing algorithm is used to detect a model configuration providing an approximate global minimum prediction error across the configuration space. An unseen kernel is instrumented then profiled to capture platform-independent characteristics. The reported average prediction error is 1.2% for the execution time of an unseen kernel over the range of platforms.

## 2.5 Power Measurement Tools

HPC energy efficiency researchers use a variety of measurement tools for power and energy usage evaluations. Ideally, the system provides inbuilt sensors that can be used. Alternatives include power meters that connect to AC mains sockets used by the system, or hook-on to individual components of a system, such as CPU sockets or memory modules. External power meters are generally not practical for large scale systems. Model-based approaches that estimate power consumption using performance metrics are also used.

IBM Blue Gene systems [59] and Cray XC systems [60] are examples that provide various on-board sensors for measuring temperature, voltage and current. Performance monitoring tools access sensor counter data to derive power and energy metrics.

Examples of meters commonly used by researchers for monitoring mains socket power include the WattsUp Pro [22, 61, 34, 30, 21, 28] and Yokogawa WT210 [33, 52]. The WattsUp Pro and Yokogawa WT210 have been discontinued, but precision ammeters, such as the  $\mu$ Current Gold [53], and precision multi-meters [32, 35] are also used. These meters typically need to be paired with a data ac-

quisition device to capture measurements at the required sample rate. AC power measurement sample rates are often low as DC power supplies act as filters between mains and components. Mains power meters are a great way to capture total system power usage, but will generally not be suitable when high temporal resolution is required.

PowerInsight [62] and PowerMon [63] are both hook-on power meters. For example, PowerMon can be used for component-level CPU and DRAM power measurements [22, 48]. While generally very accurate with high sample rates, hook-on meters can be invasive. Harnesses with Hall effect or resistive sensors typically need to be connected in series with the power connection of components to be metered. DRAM metering may require a DIMM riser with sensor resistor.

The Running Average Power Level (RAPL) [64] interface available on current Intel processors uses an on-board energy model to estimate power use. Estimated CPU package and DRAM power measurements can be accessed using model specific registers (MSR) [19, 50, 42, 38]. Researchers including Desrochers et al. [61] report that RAPL DRAM estimates generally match within 20% of measured values.

## 2.6 Summary and Research Opportunities

This survey confirms that there is a considerable amount of active research occurring in the field of parallel computing energy efficiency. The research shows that there are a range of techniques that provide significant opportunities for improving energy efficiency. It is also apparent that there are a number of promising areas for further research.

Some authors make important observations on system architecture bottlenecks that impact performance and energy efficiency. For example, tuning the active core count in multi-core processors to match memory bandwidth [50], or reducing frequency when CPU stalls are high [32]. Further investigation of these bottlenecks and the associated changes in tuning sensitivity would help position and support a proposed research direction.

Auto-tuning tools successfully manage very large configuration parameter spaces using powerful search optimisation techniques. These tools are generally aimed at application developers with in-depth knowledge of the application coding and build cycle. Improving the accessibility of the optimisation techniques used would open up auto-tuning capabilities for a wider group of users.

Deterministic models are widely used to model various aspects of HPC systems. Simple models generally provide the best insights, but that simplicity limits the scope and accuracy that can be achieved. These models provide intuitive explanations of system response limits as theoretical bounds for application tuning, but accurately predicting responses for a specific application and platform can



be more challenging.

HPC platforms running scientific applications are complex systems that are difficult to model using a deterministic approach. System responses can be impacted by low-level factors in unpredictable ways [18]. CPU pipeline performance and power use depend on factors such as instruction type, available execution units, and branch prediction and instruction reordering optimisations. Memory access models need to model deep memory hierarchies, including multiple cache levels and non-uniform memory access with different capacity, bandwidth, latency and power requirements at each level. Similar hierarchies occur in interconnects with bandwidth, latency and power usage varying for core, socket, node, rack, and cabinet connectivity.

Application parameters then interact with platform parameters to further increase system complexity. Application instruction, memory, and network usage patterns vary, along with work mappings to processing units. Code and compiler optimisations also change performance and power responses. As a result, deterministic models depend on white-box application analysis that can be difficult to automate [55] and exclude HPC user groups that are not compiler and code profiling experts.

System complexity can be managed by reducing the scope of a model. It may be possible to reduce scope by focusing on application classes that have similar platform interaction patterns. For example, the vast majority of scientific applications are iterative with relatively stable processing patterns [25]. Load balanced workloads generally require a uniform frequency-voltage setting for the run and across processing units [25], rather than fine-tuning through the run. Dynamic Concurrency Throttling as a program executes can achieve similar results to static manual tuning [24], indicating that tuning for a fixed concurrency configuration for the full run cycle can be as effective as dynamic, phase-based approaches. Such application and architecture knowledge can potentially be applied to improve problem tractability.

Statistical regression and machine learning models have been successfully used to avoid some of the issues associated with white-box analysis [55, 56, 57, 58]. The models use a wide range of input parameters, or predictors, from application configuration parameters, parallelisation, CPU frequency, to profiling or CPU counter data. Models that use measured predictors, such as profiling or counter data, require strategies for determining the measured input when predicting the response for unseen data. Measured inputs can be practical if their values can be extrapolated for unseen data [55, 57], however models-within-models may suffer from compounding prediction error. New statistical models that treat the system as a black-box have the potential to make accurate performance and energy trade-off predictions using accessible predictors, and minimising training costs.

Pareto analysis provides a set of performance and energy usage trade-off options. Pareto analysis may still exclude valid trade-off options that are off the Pareto front but virtually indistinguishable from points on the front [22, 18]. These points are equivalent as far as performance or energy use

are concerned. Other policy guidance and inputs are needed, such as conservative versus aggressive resource allocation policy.

Experimental error analysis can identify statistically equivalent points and provide a more complete set of trade-off options. Experimental error has a number of other impacts on Pareto analysis. Search optimisation methods may terminate at local optima induced by noise or error and provide locally optimal rather than globally optimal solutions. Evaluation of Pareto front prediction methods can also be impacted by experimental error. Relatively small differences in response values can significantly impact the distribution of Pareto points across the search space. New methods that consider measurement and prediction error are needed to fully evaluate the accuracy of predicted trade-off options against measured data.

There are a number of strategies for minimising the cost of an optimisation process. The search space dimensions and resolution can be reduced to the minimums that provide an optimal solution in the required scenarios. Analytical models that predict responses across the search space can avoid the cost of capturing measurement data for the full search space. Low cost sampling techniques allow accurate predictions from a small amount of measured data. Sampling scope can potentially be reduced by focusing prediction accuracy where required, close to the Pareto frontier for example. Avoiding requirements for expert direction, or extra hardware or software tools, can also reduce cost. Methods that are easy to automate provide further opportunity for reducing effort and improving usability.

In summary, a number of research opportunities have been identified. Further analysis of tuning controls and response sensitivities will validate that easy-to-access controls can provide significant trade-off opportunities. New statistical models that allow a black-box view of the application and platform have the potential to achieve accurate trade-off opportunities at low cost. New error analysis methods will provide a more complete set of trade-off options, and better evaluation of actual versus predicted trade-off options. New policies need to be developed to guide users in their trade-off decisions.



Parts of the following publication have been incorporated in Chapter 3.

- [1] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, “A bottleneck-centric tuning policy for optimizing energy in parallel programs,” *Advances in Parallel Computing*, vol. 32, pp. 265–276, 2018.

Chapter 3 Publication Contributor Statement

| Contributor       | Statement of contribution | %   |
|-------------------|---------------------------|-----|
| M. Endrei         | Writing of text           | 100 |
|                   | Proof-reading             | 25  |
|                   | Theoretical derivations   | 100 |
|                   | Numerical calculations    | 100 |
|                   | Preparation of figures    | 100 |
|                   | Initial concept           | 60  |
| C. Jin            | Proof-reading             | 10  |
|                   | Supervision, guidance     | 20  |
| M. N. Dinh        | Proof-reading             | 10  |
|                   | Supervision, guidance     | 20  |
| D. Abramson       | Proof-reading             | 10  |
|                   | Supervision, guidance     | 20  |
|                   | Initial concept           | 10  |
| H. Poxon          | Proof-reading             | 10  |
|                   | Supervision, guidance     | 10  |
|                   | Initial concept           | 10  |
| L. DeRose         | Proof-reading             | 10  |
|                   | Supervision, guidance     | 10  |
|                   | Initial concept           | 10  |
| B. R. de Supinski | Proof-reading             | 25  |
|                   | Supervision, guidance     | 20  |
|                   | Initial concept           | 10  |

# Chapter 3

## Energy Tuning Controls

This chapter examines definitions, architectural features, and tuning parameters that are important for understanding and controlling the performance and energy efficiency of parallel applications. The aim is to investigate if:

- There are HPC job submission parameters accessible to application end-users that can be used to tune performance and energy efficiency.
- The tuning parameters identified can significantly improve performance and energy efficiency, providing a benefit that exceeds measurement error limits.
- The tuning search space size can be managed to practical levels by reducing search ranges of parameters to the most influential values, requiring an exhaustive search of less than 1,000 measurements.
- Maximum energy efficiency and performance diverge, requiring use of a multi-objective optimisation strategy rather than a simple race to halt strategy.

As well as providing an initial feasibility assessment of the proposed approach, the findings of this chapter inform the selection of modelling techniques, such as deterministic or probabilistic methods, model input parameters or predictors, and sampling strategies.

### 3.1 Definitions

This section provides the formal definitions of electrical energy, electrical power, and HPC energy efficiency and performance.

### 3.1.1 Electrical Power and Energy

Power is the amount of energy  $E$  transferred per unit of time  $t$ , measured in joule per second, or watt. Electrical energy is the electric charge  $Q$  in coulomb passing through an electric potential  $V$  in volt. Electric current  $I$  is electric charge per unit time. Ohm's law equates current  $I$  to electric potential  $V$  divided by load  $R$ . Equations 3.1 and 3.2 follow from these definitions.

$$\begin{aligned} P &= E/t \\ &= V \cdot Q/t \\ &= V \cdot I \\ &= V^2/R \text{ (watt)} \end{aligned} \tag{3.1}$$

$$E = P \cdot t \text{ (joule)} \tag{3.2}$$

### 3.1.2 Energy Efficiency and Performance

HPC energy efficiency  $e_\eta$  is assessed using operations executed per unit of electrical energy used, as shown in equation 3.3.

$$e_\eta = \text{operations}/E \text{ (ops/joule)} \tag{3.3}$$

Energy efficiency units for typical HPC operations include Flops/J (Flops processed per joule used), Bytes/J (bytes transferred per joule), or Updates/J (updates made per joule). Using equation 3.2, this can be equivalently expressed as operations per second per watt (for example, Flops/s/W, which is performance achieved per watt used).

HPC performance  $p_\mu$  is assessed using operations executed per second, as shown in equation 3.4.

$$p_\mu = \text{operations}/t \text{ (ops/second)} \tag{3.4}$$

Performance units for typical HPC operations include Flops/s, Bytes/s, or Updates/s.

Flops/s and Flops/s/W (or Flops/J) are in wide use, including in the international supercomputer rankings published by the TOP500 project [5].

## 3.2 HPC Architectural Considerations

Flynn's taxonomy [65] classifies parallel computing architectures by the concurrency in instruction and data stream processing. This results in the following four categories of concurrent processing:

- *SISD* (single instruction stream–single data stream) processing is conventional single-core processing, including pipelined and superscalar processors.
- *SIMD* (single instruction stream–multiple data stream) processing includes array and vector processors.
- *MISD* (multiple instruction stream–single data stream) processing is the least common. It is typically only used for fault critical systems where processors operate on the same data stream and must agree on the result (for example, space flight control systems).
- *MIMD* (multiple instruction stream–multiple data stream) processing includes multi-core and multi-threaded processors, and HPC clusters.

The specification of the Cray XC series system used for the evaluations conducted during this research project is listed in Table 6.1. Flynn's *SISD*, *SIMD* and *MIMD* categories provide a useful structure for analysing concurrent processing capabilities of the system and potential tuning controls for optimising performance and energy efficiency.

### 3.2.1 SISD

*SISD* designs can provide instruction level parallelism using pipelined and superscalar processing. Our laboratory system uses Intel Broadwell CPUs. Figure 3.1 shows the pipeline topology of each core in a Broadwell processor. Each stage of the pipeline can operate in parallel. This means each core can process multiple instructions in parallel, each at a different pipeline stage.

The Broadwell CPU also supports superscalar processing where multiple instructions are retired in one clock cycle. The execution engine of each core has multiple execution units that operate in parallel, allowing processing of up to eight instructions per cycle.

When the pipeline runs out of instructions that are ready to process the CPU will stall. Pipeline bottlenecks can cause CPU stalls when the:

- Re Order Buffer is full of instructions that are not ready to be retired,
- Reservation Station has no entries that are ready to execute, or

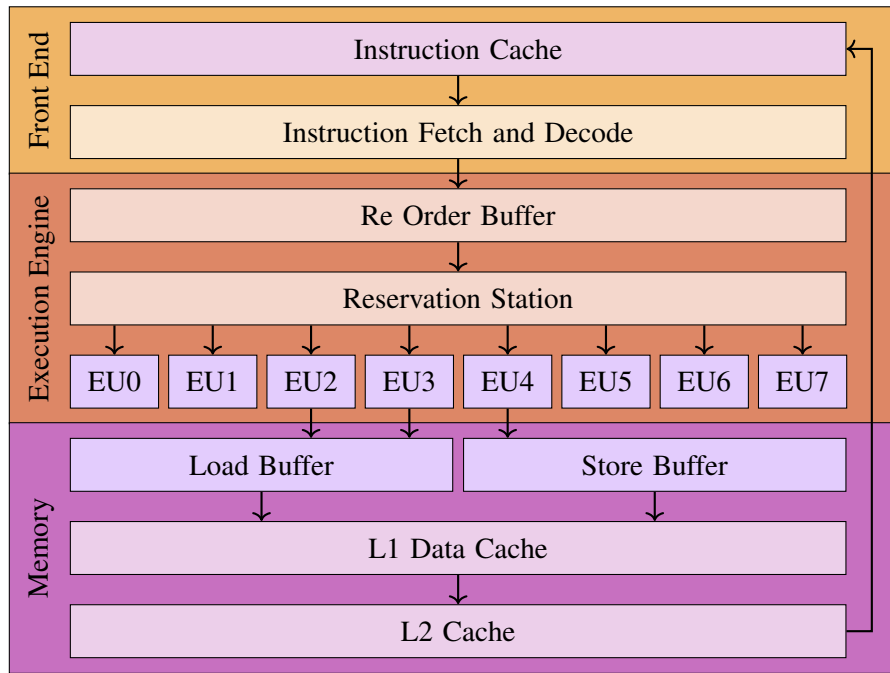


Figure 3.1: CPU Pipeline Topology

- Store Buffer overflows due to backlogs accessing memory

CPU stalls reduce both performance and energy efficiency because the program requires more time and therefore energy to execute. Tuning a program for CPU core parameters (such as instruction selection, loop/branching, and memory optimisation) is typically a program design or compile time challenge, and not controllable by parallel program users.

However, users can change the CPU frequency used to run a program using job submission parameters. The CPU frequency can be lowered when stalls dominate. Tuning can reduce the inefficiency of a high clock speed when the CPU is stalling, while having minimal impact on performance.

### 3.2.2 SIMD

Intel Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX) provide instruction level data parallelism capabilities at the CPU core level. AVX2 includes fused multiply-add (FMA) operations. Each FMA operand consists of eight single precision or four double precision floating point numbers. The dual 256-bit FMA execution units (EU0 and EU1 in Figure 3.1) enable the Broadwell theoretical peak performance of 16 double-precision Flops/cycle/core  $((1 \text{ multiply} + 1 \text{ add}) \times 4 \text{ double-precision floating point values} \times 2 \text{ units})$ .

Enabling SSE or AVX instructions can significantly reduce the number of CPU cycles required to complete some operations, which in turn provides large gains in performance and energy efficiency. Once again, this is a program design or compile time activity, not typically controllable by end users.



### 3.2.3 MIMD

The MIMD model uses multiple processing elements that have access to a shared or distributed program and data memory. Processing elements can be cores in a multi-core CPU, CPU sockets in a multi-CPU node, or compute nodes in a multi-node HPC cluster.

Figure 3.2 shows how cores and LLC in the Broadwell CPU microarchitecture are grouped in two equal-sized clusters on a pair of interconnect rings [66]. This configuration is known as Cluster on Die (CoD). QPI (QuickPath Interconnect) queues connect the cluster rings. Each cluster has a memory controller, but QPI links (used to connect CPU sockets) and PCIe links (used to connect peripherals such as GPUs) are only available on the first cluster. Each core has 2.5 MB LLC that can be accessed by all cores, providing the total LLC size of 55 MB.

Broadwell cores also support thread level parallelism (two threads per core) using Hyper-Threading Technology (HTT), a simultaneous multi-threading (SMT) technique. SMT maintains processor state for each thread, but threads share processor functional units. The aim is to increase instruction level parallelism by using functional units left idle by a single thread.

Shared resources, such as LLC and memory controllers, can become bottlenecks that limit core and thread scalability. Performance and energy efficiency are impacted as contention for shared resources increases. Tuning the number of active cores and their placement across sockets in a node can reduce this contention. Application users can specify core counts and placement per node when submitting a job with the system resource manager.

The interconnect links provide bidirectional bandwidth of 38.4 GB/s, but latency can vary significantly depending on how a connection traverses hierarchical memory and interconnect structures.

Hofmann et al. [67] measured access latencies of all memory hierarchy levels for a range of Intel server processors. For Broadwell processors they report L1 access latency of 4 clock cycles, L2 latency of 12 cycles, LLC latency of up to 47 cycles, and DRAM latency of up to 280 cycles. Cross-ring/CoD access through the ring interconnect queues adds six cycles. The latency is around 360 cycles if DRAM from another CPU socket (or remote NUMA domain) is used.

CPU and DRAM consume most of the power (around 95%) used by each node in an HPC cluster. Intel specifies a CPU thermal design power rating of 145 W for our Broadwell CPU. Desrochers et al. [61] measured the power consumption of a 16 GB DDR4-2133 DIMM for a number of benchmarks running on a 16 core Haswell CPU. Their measured values for sleep and OpenMP stream benchmarks were 0.5 W and 5 W respectively. Our laboratory system has eight 16 GB DDR4-2400 DIMMs per node, so we can expect DRAM power consumption of over 40 W per node for memory intensive workloads.

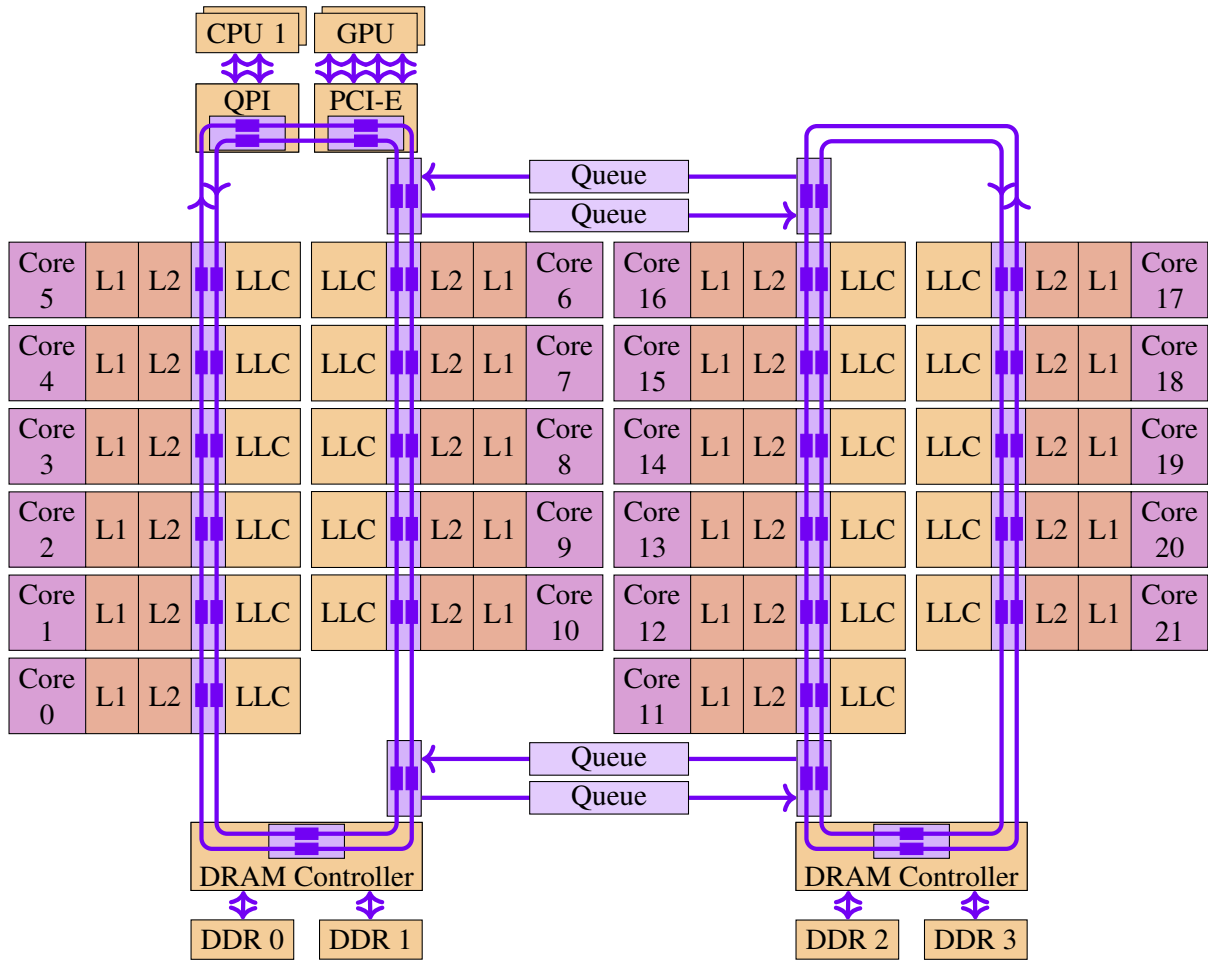


Figure 3.2: CPU Topology

Operating systems and applications that maximise the use of memory in the local NUMA domain are able to reduce the latency penalty, which improves performance and energy efficiency. Figure 3.3 shows NUMA domain information for a single compute node reported by the Linux operating system using the `lstopo` command. We can see that CoD mode is disabled in node BIOS settings as the operating system reports only one NUMA domain per socket.

System administrators and application designers control NUMA memory allocation using operating system policy controls. Despite policy controls that preference local memory allocation, imbalance may cause allocations to migrate away from the local node. On Linux systems, administrators and NUMA-optimised codes can manage this situation using page migration functions provided in the `numactl` package.

Figure 3.3 also shows operating system visibility of Hyper-Threads, with two Hyper-Threads per core detected. The Hyper-Thread ids detected for NUMA Node 0 are 0-21 and 44-65. Hyper-Thread ids for NUMA Node 1 are 22-43 and 66-87 (not shown). HPC resource management systems allow users to specify the number of Hyper-Threads per core that a job should use.

Figure 3.4 shows the cluster interconnect hierarchy for the laboratory environment. The Cray XC

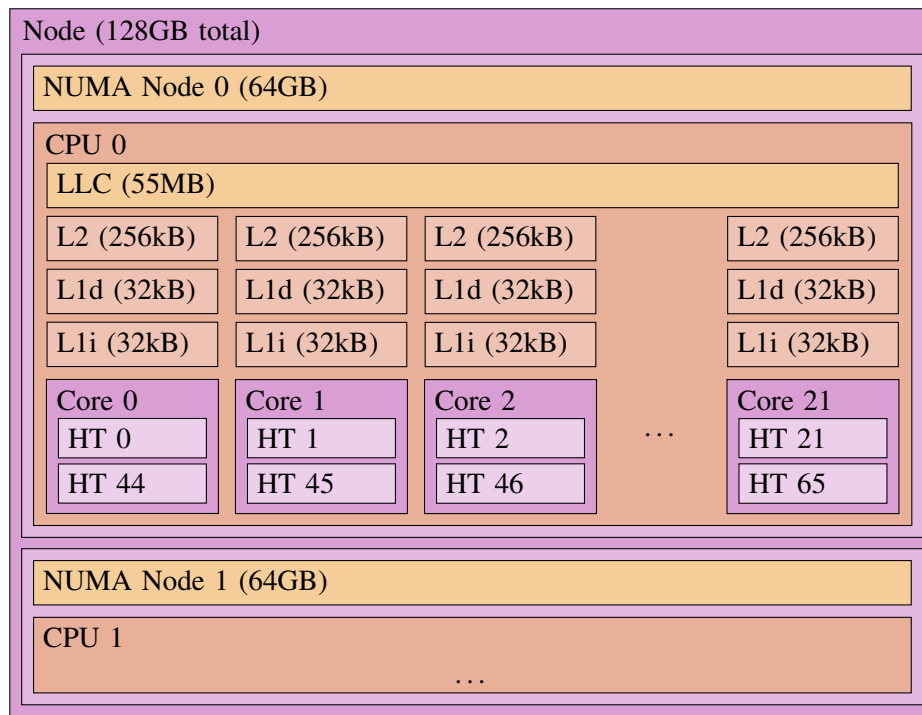


Figure 3.3: Node Topology

series architecture has two processor sockets per compute node and four compute nodes per blade. Compute nodes on a blade communicate using the blade interconnect unit. The interconnect unit also provides communications between blades over a three level interconnect network:

- Rank 1 fully connects 16 blades in a chassis using a backplane.
- Rank 2 connects six chassis backplanes in a cabinet group, over 2 or 4 hops.
- Rank 3 fully connects cabinet groups using optical cables.

HPC application users may not be aware of the cluster interconnect topology, but system administrators will typically configure the system resource manager with interconnect topology information. The scheduling system can use this information to favour node placements within a chassis or cabinet group. This approach reduces the likelihood that jobs submitted by different users will compete for chassis or cabinet group interconnect bandwidth.

Application users can control the number of nodes allocated to a job, the number and placement of cores on each node, and the number of threads per shared memory process running on a node. Users are in a position to submit a job with optimised settings for these parameters that minimise resource contention at the cluster, node, socket, and core level.

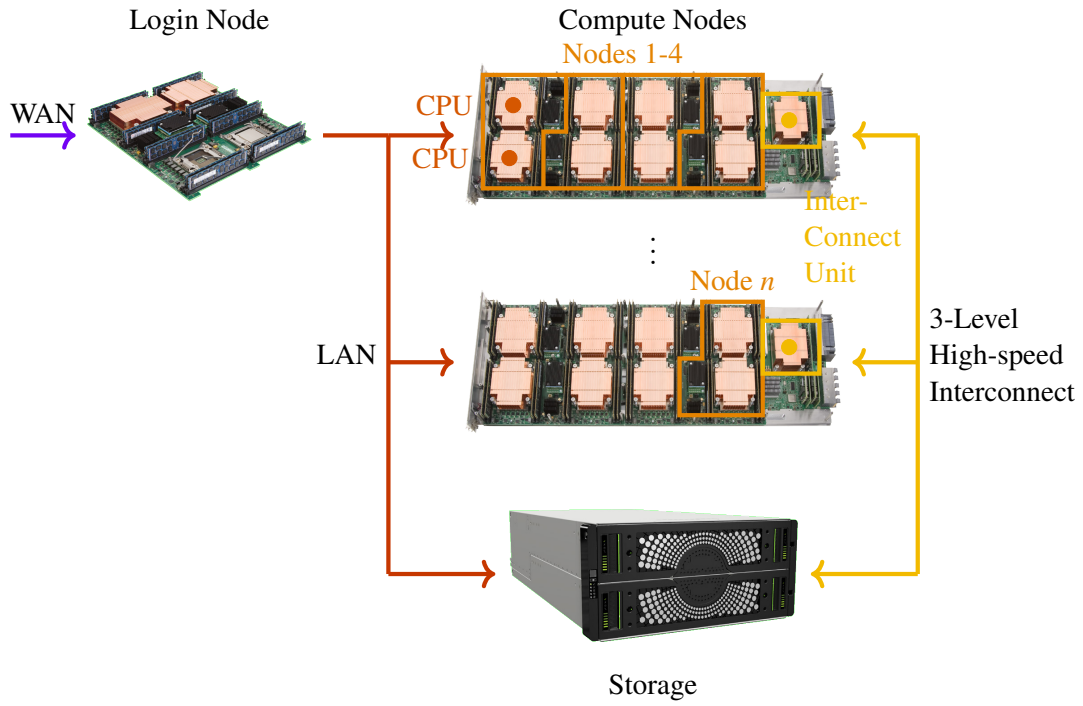


Figure 3.4: Cluster Topology

### 3.2.4 DVFS

In addition to the concurrency controls considered, DVFS has been widely used in parallel computing to lower energy consumption with minimal performance degradation [3].

For microprocessors that use complementary metal oxide semiconductor (CMOS) circuits, like our Broadwell CPUs, energy consumption is proportional to the clock cycle time and the square of the operating voltage [68]. This relationship can be stated as shown in equation 3.5, which derives from equations 3.1 and 3.2.

$$E \propto V^2 \cdot t \quad (3.5)$$

However, voltage and frequency controls cannot be used independently as the settling time for a CMOS gate varies with its operating voltage. The longer settling time means the clock frequency must be decreased as voltage decreases. Since energy use per cycle reduces quadratically for a linear reduction in performance, an overall saving in energy use can be achieved [69].

On Linux systems, users can use the `cpupower` utility to control CPU frequency. HPC resource managements tools may also provide controls as job submission parameters. For example, the Cray Application Level Placement Scheduler (ALPS) `aprun` command provides the `--p-state` argument for setting a CPU frequency cap, or power state, across all cores allocated for the program run.

Broadwell processors dynamically control the DFVS setting of each core, using the CPU frequency setting as an upper limit. The frequency of each core is automatically lowered for lighter workloads, and as required to meet power and thermal constraints. Processor and operating system controls mean the user CPU frequency selection is a cap rather than a fixed setting.

The Linux kernel CPUFreq subsystem uses the `intel_pstate` driver for frequency scaling on Intel systems. The `sysfs` virtual file system provides access to driver settings and status information, such as actual core frequency over a sample interval. Actual core frequency may also be calculated using the `IA32_APERF` and `IA32_MPERF` Intel model specific registers [64].

### 3.3 Tuning Parameters Analysis

Our architecture analysis in the previous section identified candidate parameters for optimising energy efficiency and performance of parallel applications running on HPC systems. This section examines the effectiveness of the following user-accessible tuning parameters:

- Thread placement and count
- CPU frequency
- Problem size
- Node counts

This analysis has several goals. First, to show that tuning a parameter provides significant performance and/or energy efficiency benefits. Second, to identify suboptimal parameter settings or setting ranges that can be excluded to reduce the tuning search space. Lastly, to confirm use cases where performance and energy efficiency diverge, indicating that a multi-objective optimisation strategy is required, rather than a simple race to halt strategy.

#### 3.3.1 Experiments Setup

All experiments run on a Cray XC system equipped as Table 6.1 shows. Each run uses a single, exclusively allocated 44-core node. Energy and power consumption for the entire node are monitored with Cray power management counters (`pm_counters`). Intel RAPL counters are used to monitor CPU package and DRAM power and energy. CrayPAT derived and PAPI counter data is used for CPU stall, cache, and memory metrics. Performance and energy counter events are collected in separate runs to avoid counter multiplexing, which reduces application perturbation and improves measurement accuracy.

The Parallel Research Kernels (PRK) [70] listed in Table 3.1 provide configurable OpenMP workloads for this evaluation on memory-bound application workloads. Kernel iterations are set to ensure run time is at least 10 times the measurement sample rate. Each kernel uses double precision floating

point values. The 95% confidence interval for the mean of five measurement samples is 4.5% or less for the presented experiments, calculated using the  $t$ -distribution as described in section 4.2.2.

Table 3.1: Kernels Summary

| Name      | Description   | Configuration  | Size Limits<br>(Cache / Mem) |
|-----------|---|--|------------------------------|
| Stencil   | Explicit stencil operation on a 2D square discretisation grid | Stencil radius: 2,<br>Iterations: Cache 10k / Mem 50 | 2.6k / 89k                   |
| Sparse    | Canonically indexed, sparse-matrix by dense-vector product    | Difference stencil radius: 2,<br>Iterations: 1k      | 0.8k / 28k                   |
| Transpose | Dense matrix transposition<br>( $C = A^T$ )                   | Blocking/tiling: disabled,<br>Iterations: 20k        | 2.6k / 89k                   |

The size limits listed in table 3.1 are calculated as follows. The memory limit depends on the available physical memory, or resource manager settings. The stencil code uses `malloc` to allocate memory for two double-precision floating point matrices. The maximum matrix order for this kernel can be calculated using equation 3.6.

$$Matrix\ order = \sqrt{\frac{sockets/node \times memory\ capacity/socket}{2\ matrices \times bytes/DP\ float}} \quad (3.6)$$

The laboratory HPC system has two CPU sockets per node, RAM capacity of 64 GB per socket, and requires eight bytes per double-precision floating point, so equation 3.6 limits the in-memory matrix order to 89,400. The L3 cache capacity of our system is 55 MB per socket, so the in-cache limit is 2,600.

### 3.3.2 Thread Scaling

HPC resource management tools such as the Portable Batch System (PBS) allow users to allocate the required nodes, sockets, cores, and threads to a job. Jobs can also be allocated the required MPI processes per node and OpenMP threads per process. The MPI runtime uses a distributed memory model where each MPI rank, or parallel process running on a node, has private memory and communicates with other processes in the cluster using a messaging interface. OpenMP programs use a shared memory model where parallel threads running within a process communicate using variables in shared memory.

Thread placement or affinity policy controls the assignment of processes and threads to sockets and cores within a node. *Compact* allocates all cores in one socket before moving to another, while *scatter* utilises all sockets uniformly.

Figures 3.5 and 3.6 show the effect of compact and scatter affinity at problem sizes of 500, 10k and 70k, with one OpenMP thread per CPU core. At small problem sizes the performance and

energy efficiency responses are similar, with compact experiencing a minor decline at 24 threads. The compact decline becomes significantly more pronounced as the problem size increases.

Small problem sizes operating within cache exhibit compute intensive behaviour where scaling improves as threads are added. As the problem size increases, there is a transition to memory-bound processing. Compact performance and energy efficiency start to decline at around 12 threads due to memory access contention as more cores are used in the first socket. When cores are added in the second socket, both responses begin improving again at the 24 thread point. The scatter policy clearly provides improved scaling with 24 threads across two sockets. The MFlops/s rate is almost double that of the rate with 12 threads in one socket.

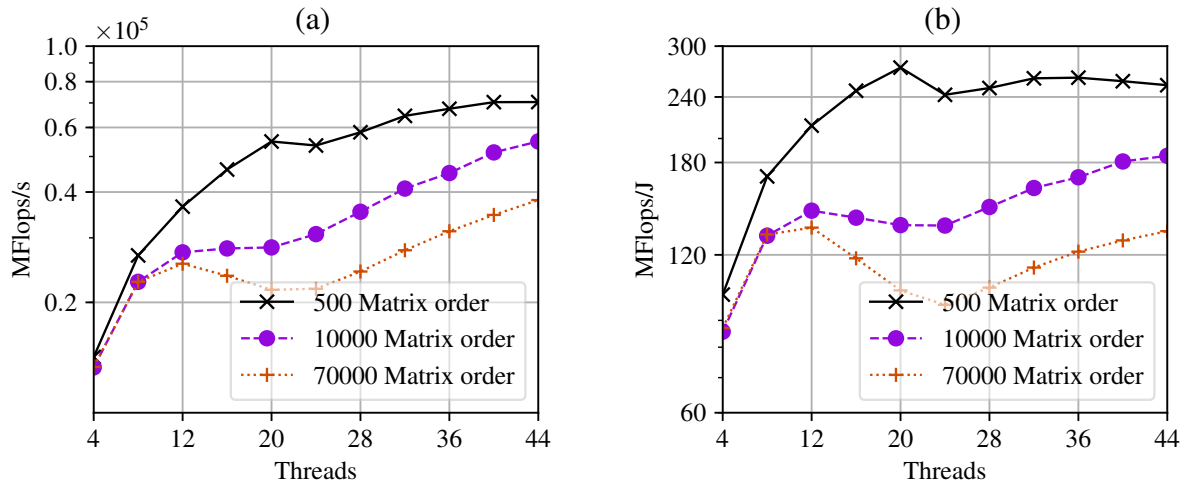


Figure 3.5: Stencil Thread Scaling – Compact Thread Placement (a) Performance and (b) Energy Efficiency

For matrix order of 70k, figure 3.6 shows that maximum performance of 50 GFlops/s and maximum energy efficiency of 220 MFlops/J both occur at 20 threads. These values drop to 38 GFlops/s and 130 MFlops/J at 44 threads, so thread count tuning provides a 24% performance improvement and 40% energy efficiency improvement compared to simply maximising threads.

Figures 3.5 and 3.6 also show divergence between thread counts for maximum performance and maximum energy efficiency, which confirms a multi-objective optimisation strategy is required. For example, Figure 3.6 shows performance peaking at around 40 threads for matrix order of 10k, but energy efficiency peaks at around 28 threads.

Figure 3.7 shows compact affinity with HTT enabled. This time we set one OpenMP thread per Hyper-Thread with two Hyper-Threads per core. The responses show that 12 Hyper-Threads in a socket can saturate memory bandwidth as we observed for 12 cores. Hyper-Threading performance is generally below that achieved without Hyper-Threading, in all but the most compute-intensive cases. Even in these cases the benefit and measurement errors are similar magnitudes. For problem

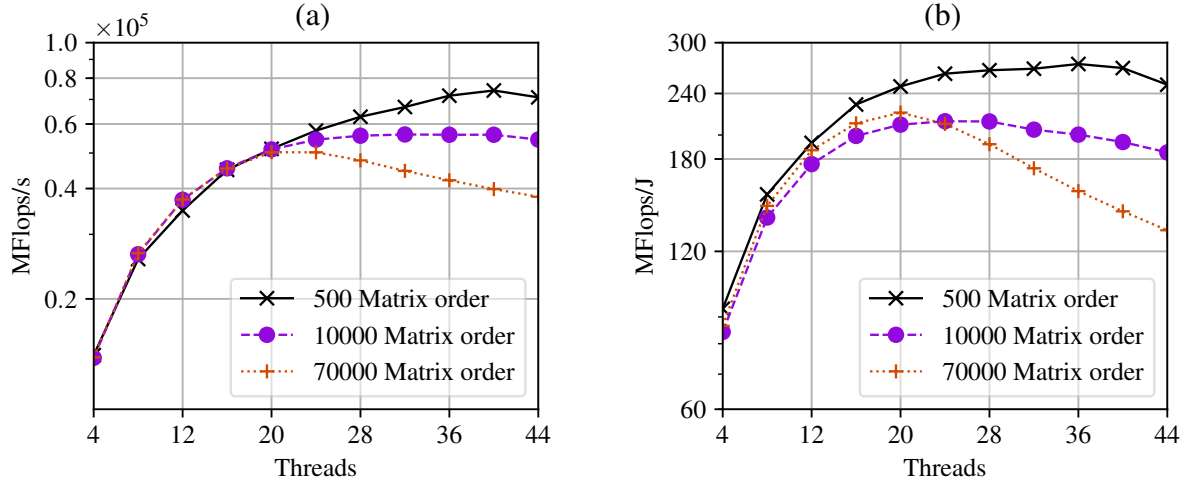


Figure 3.6: Stencil Thread Scaling – Scatter Thread Placement (a) Performance and (b) Energy Efficiency

size of 500 at 12 threads and 24 Hyper-Threads performance is 35.5 GFlops/s vs 37.1 GFlops/s, or 1.5% improvement, but the measurement confidence interval is also around  $\pm 1.5\%$ .

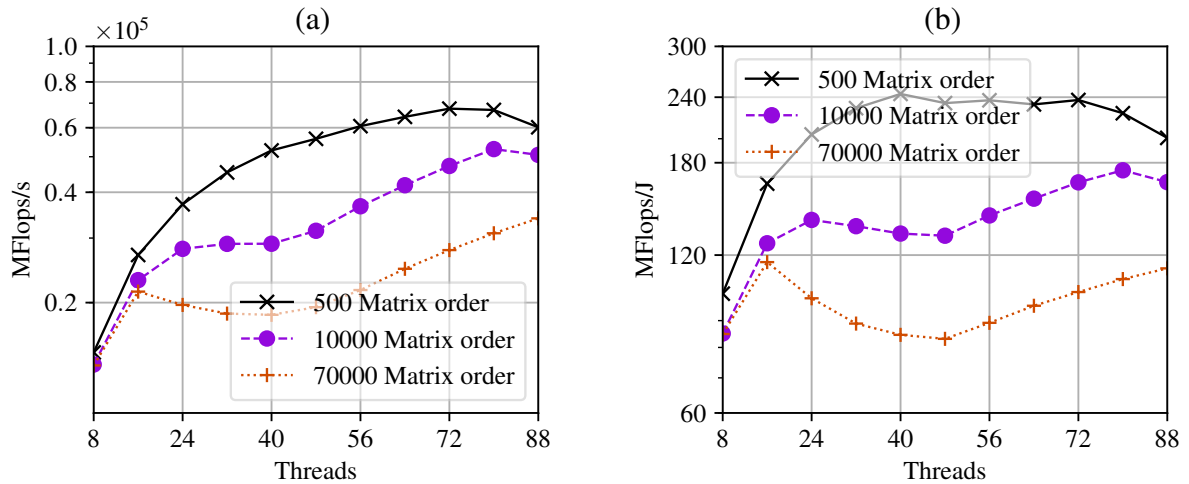


Figure 3.7: Stencil Hyper-Thread Scaling – Compact Thread Placement (a) Performance and (b) Energy Efficiency

Figure 3.8 shows scatter affinity with threads distributed in four even groups across the two CPU rings per socket (CoD), and two sockets per node. The results are identical to the Figure 3.6 results. With CoD disabled in the node BIOS, LLC and memory controllers are shared across the rings, so placement on rings within a CPU has no discernible impact on the responses.

### 3.3.3 CPU Frequency Scaling

Users can use frequency scaling to set a CPU frequency cap (or upper limit) for a job run, as discussed in section 3.2.4. Figure 3.9 shows two stencil runs using CPU frequency caps of 1.4 and 2.0 GHz, 12 OpenMP threads across 6 cores in each socket (12 cores in total), and matrix order of 40k.



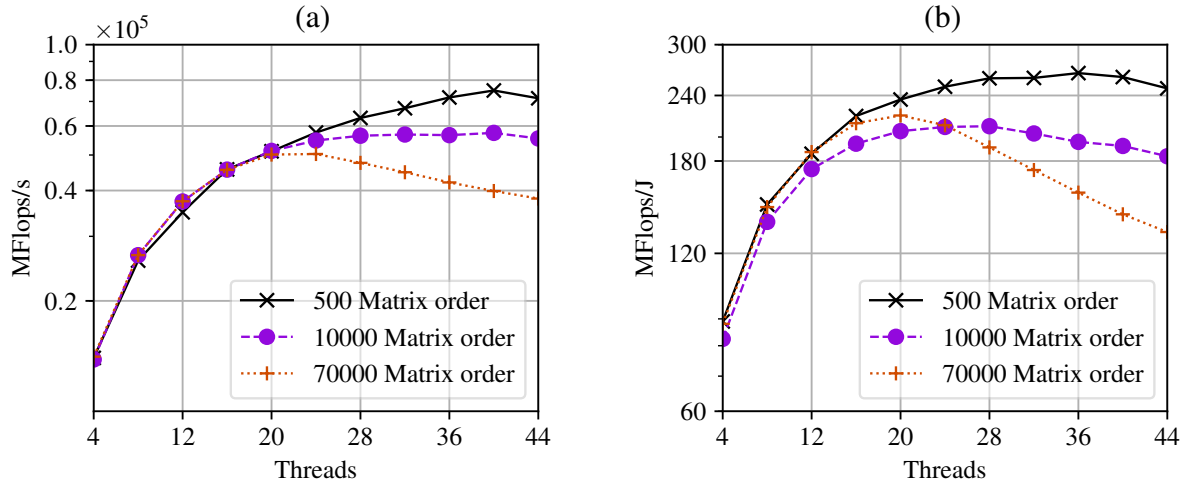


Figure 3.8: Stencil Thread Scaling – Ring Thread Placement (a) Performance and (b) Energy Efficiency

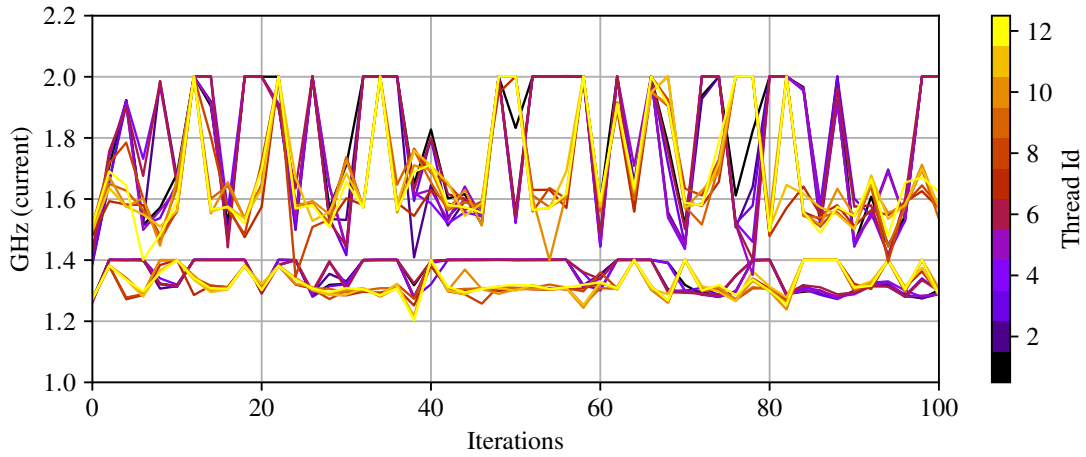


Figure 3.9: Stencil Per Core Current Scaling Frequency – Matrix Order 40k, 12 Threads, 1.4 and 2.0 GHz Cap

Stencil iterations proceed from 0 to 100 along the  $x$ -axis, with the current/actual frequency of each core/OpenMP thread plotted against the  $y$ -axis. The current CPU scaling frequencies for allocated cores on the compute node are read once every two stencil iterations from Linux CPUFreq subsystem sysfs files:

```
/sys/devices/system/cpu/cpu<0-n>/cpufreq/scaling_cur_freq
```

Table 3.2 shows the RAPL power capping controls for laboratory compute nodes. Short-term and long-term averaging window constraints are shown for both CPU sockets. The given power limits apply for the specified time windows. The DRAM long-term averaging window constraint is disabled. RAPL controls are accessed using files in the following sysfs folder:

```
/sys/class/powercap/intel-rapl
```

RAPL controls may also be accessed using Intel model specific registers [64]:

- MSR\_PKG\_POWER\_LIMIT
- MSR\_DRAM\_POWER\_LIMIT

Table 3.2: RAPL Power Capping Controls

| Domain    | Enabled | Constraint | Max Power | Power Limit | Time Window |
|-----------|---------|------------|-----------|-------------|-------------|
| Package-0 | True    | Short-term | 290W      | 174W        | 7.8ms       |
|           | True    | Long-term  | 145W      | 145W        | 1.0s        |
| DRAM-0    | False   | Long-term  | -         | -           | -           |
| Package-1 | True    | Short-term | 290W      | 174W        | 7.8ms       |
|           | True    | Long-term  | 145W      | 145W        | 1.0s        |
| DRAM-1    | False   | Long-term  | -         | -           | -           |

Imperfect load balance across threads has the potential to impact the accuracy of a simplified model using a static thread count as a predictor. In a similar way, hidden complexity in DVFS control systems may impact the effectiveness of CPU frequency cap as a model predictor.

The OpenMP threads transition point at 12 threads seen in Section 3.3.2 can now be used to identify DVFS frequency cap tuning responses. Figure 3.10 shows that MFlops/s performance for grid size of 70k using the compact thread placement policy flattens from around 1.8 GHz at 12 threads. At 16 threads this drops back to around 1.4 GHz which confirms there are DVFS tuning opportunities for memory-bound workloads.

The 16 thread curves also show significant divergence between optimal performance and energy efficiency. Peak performance occurs at 2.1 GHz when energy efficiency is 126 GFlops/J, down 11% from its peak of 142 GFlops/J. Peak energy efficiency occurs at 1.2 GHz for a 6% performance drop from 24 to 22 GFlops/s, providing further confirmation that a multi-objective optimisation strategy is required.

Fig 3.11 shows linear frequency scaling for the same thread ranges using the scatter thread placement policy, indicating that this policy also benefits from frequency scaling. Linear scaling also shows that underlying dynamic frequency variation below the set cap occurs without significant impact on performance.

Figure 3.12 confirms that thread placement on rings also has no discernible impact on the CPU frequency scaling responses.

### 3.3.4 Problem Size Scaling

Problem size is typically a constraint of the problem the user wishes to solve, so it is not evaluated as a tuning parameter. Instead, problem size scaling experiments provide insight on how interactions

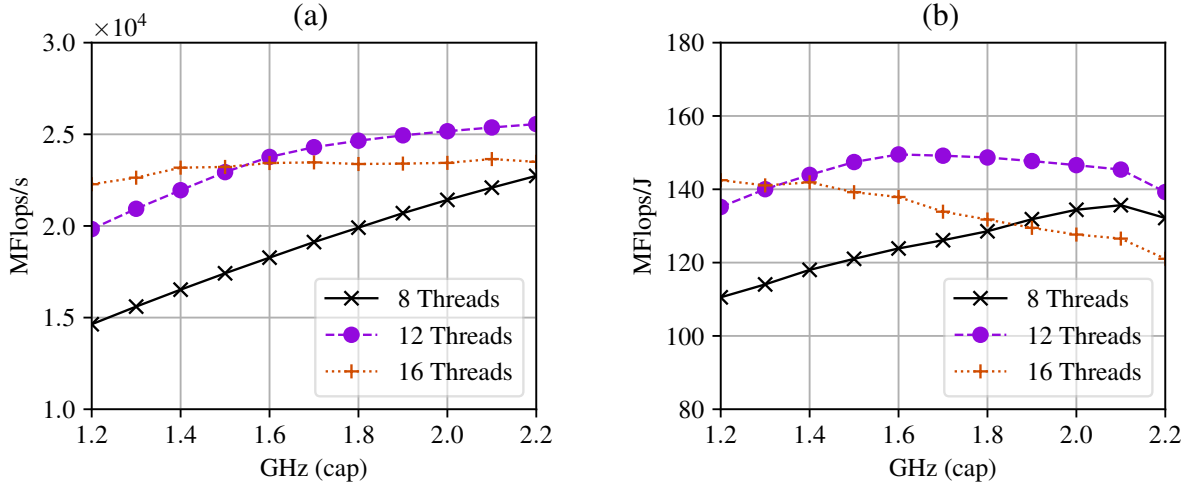


Figure 3.10: Stencil Frequency Scaling – Compact Thread Placement (a) Performance and (b) Energy Efficiency

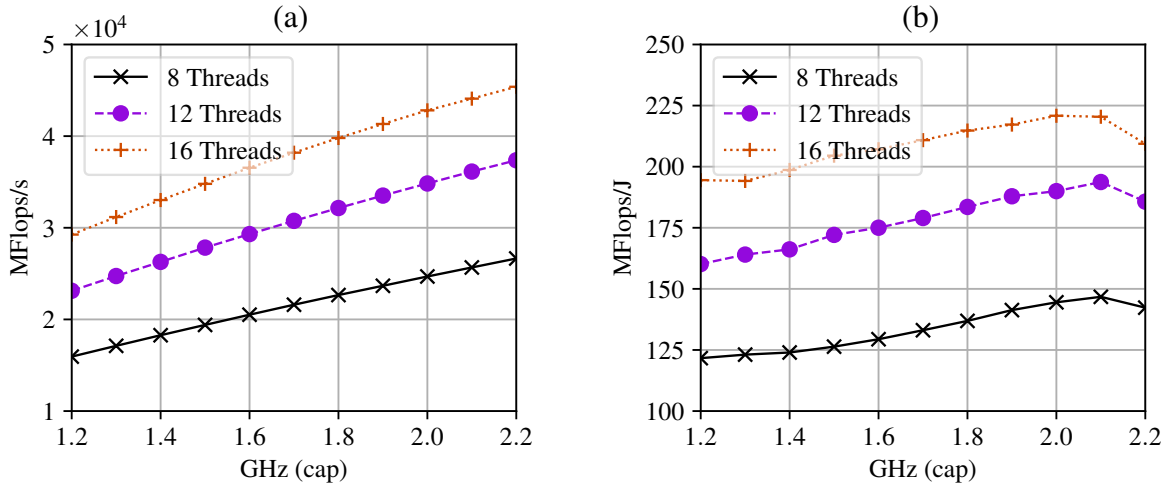


Figure 3.11: Stencil Frequency Scaling – Scatter Thread Placement (a) Performance and (b) Energy Efficiency

between applications and the memory system affect performance and energy efficiency.

Problem size sweeps for each kernel at around 75% of the maximum thread and CPU frequency cap settings, or 34 threads and 1.8 GHz, are used to evaluate scaling across the memory hierarchy. Measurement temporal resolution limits the sweep range at the lower end, while memory size limits it at the upper end. Figure 3.13 shows the Stencil kernel with numbered performance transitions points.

Stencil has transitions (1, 2) at matrix orders of around 5k and 30k in Figure 3.13 (a). Performance exceeds 70 GFlops/s before the first transition but drops sharply as matrix order reaches a problem size that exceeds LLC size. The processing rate remains approximately 55 GFlops/s until the second transition at matrix order 30k. From this point, the performance curve declines steadily.

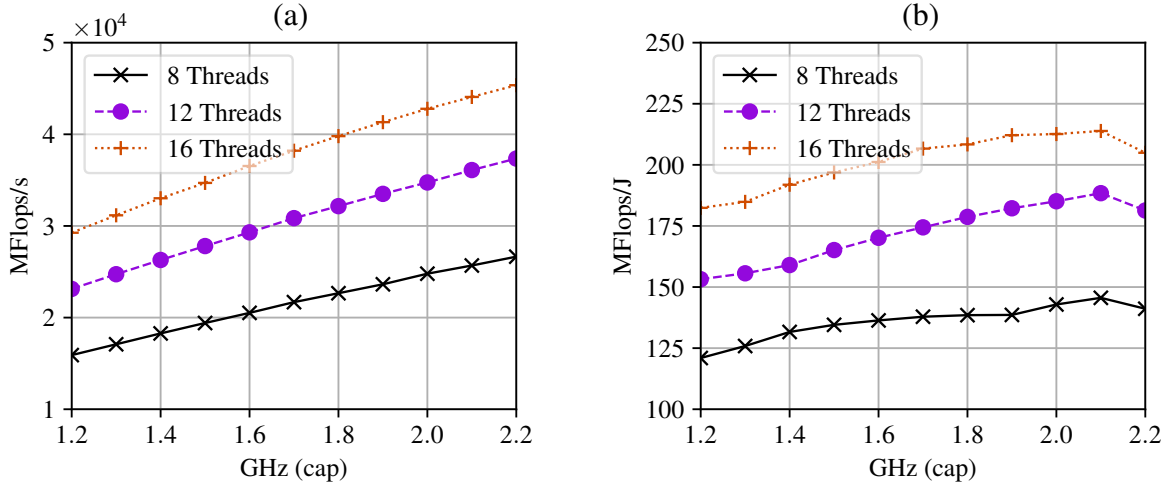


Figure 3.12: Stencil Frequency Scaling – Ring Thread Placement (a) Performance and (b) Energy Efficiency

Figure 3.13 (b) provides a breakdown of stall types for Stencil. The second transition at 30k may not be predicted by model-based methods, such as the ECM or Roofline models, as there does not appear to be a related memory system transition. It corresponds to an increase in CPU stall cycles. Re Order Buffer (ROB) and Store Buffer (SB) stalls are both flat for Stencil, but Reservation Station (RS) stalls are increasing as performance tapers off. The RS (filtered) curve applies a Savitzky–Golay smoothing filter to address noise. RS stalls occur when entries are not available in the instruction pipeline.

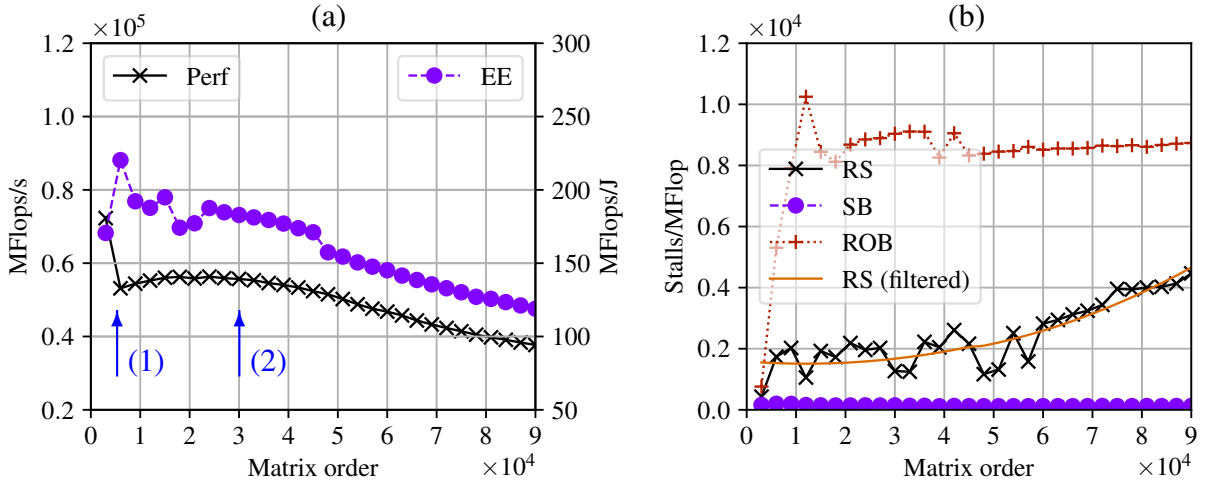


Figure 3.13: Stencil Problem Size Scaling – (a) Performance and Energy Efficiency, and (b) Stalls

RS stalls also dominate for Transpose (not shown), however, Figure 3.14 shows that SB stalls dominate for Sparse. SB stalls occur when the CPU front-end allocates store buffers faster than the execution engine commits data to cache or memory.

Figure 3.15 (a) and (b) provide a close-up view of the LLC saturation transition for Stencil. As the processing rate drops, the combined effects of increasing LLC miss rate and power consumption

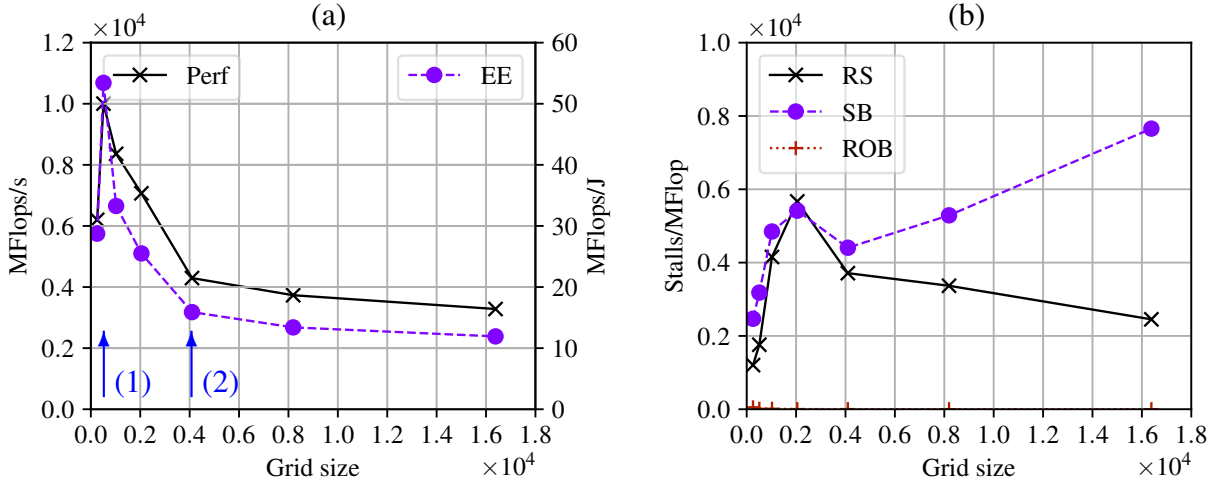


Figure 3.14: Sparse Problem Size Scaling – (a) Performance and Energy Efficiency, and (b) Stalls

are observed. Figure 3.15 (a) shows that Stencil socket and DRAM power consumption increases substantially from 60 W and 40 W to around 90 W each. The jump in socket and DRAM power use combined with the drop in performance confirm that moving data across the memory hierarchy consumes a significant energy efficiency penalty.

Sparse (not shown) has a similar pattern. However, Figure 3.16 shows that the ramp up to maximum DRAM power occurs at lower problem sizes for transpose. Peak DRAM power coincides with LLC miss rates above 20-40%.

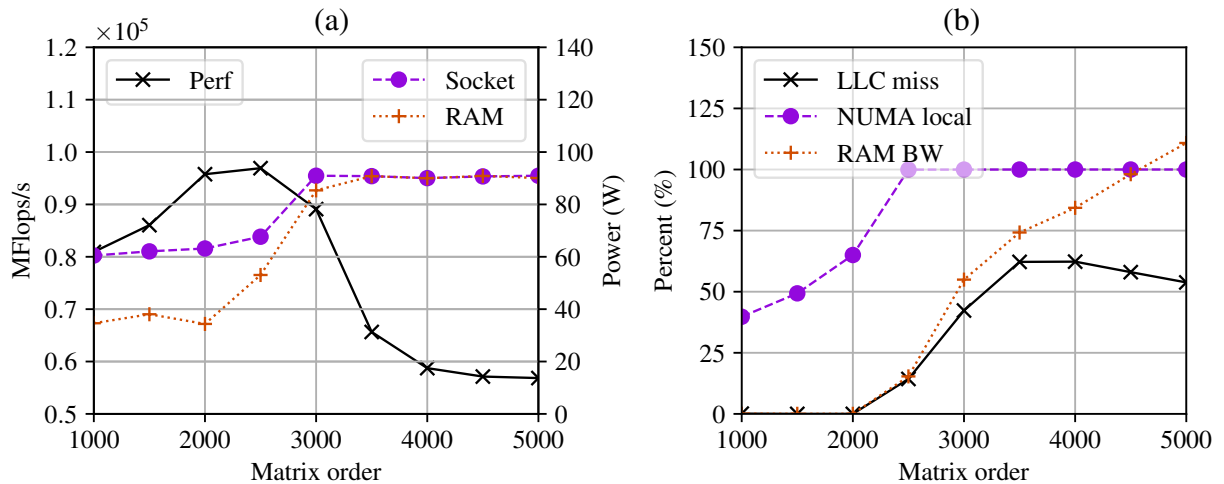


Figure 3.15: Stencil Cache Analysis – (a) DRAM Power, (b) DRAM Bandwidth, NUMA Local Memory Access, and LLC Miss Rate

Figure 3.15 (b) shows percent DRAM bandwidth utilisation for Stencil ramping up towards the node limit (76.8 GB/s × 2 CPUs), indicating that memory bandwidth becomes a constraint at this transition. The NUMA local curve shows that CPU sockets are directing 100% of in-memory processing to their local DRAM bank rather than the DRAM bank of the other CPU socket, so NUMA misses are not

a significant factor. Sparse exhibits similar behaviour. However, NUMA misses are a factor for Transpose, with NUMA memory access to the local socket levelling off at around 85%. Memory bandwidth utilisation for Transpose is accordingly lower.

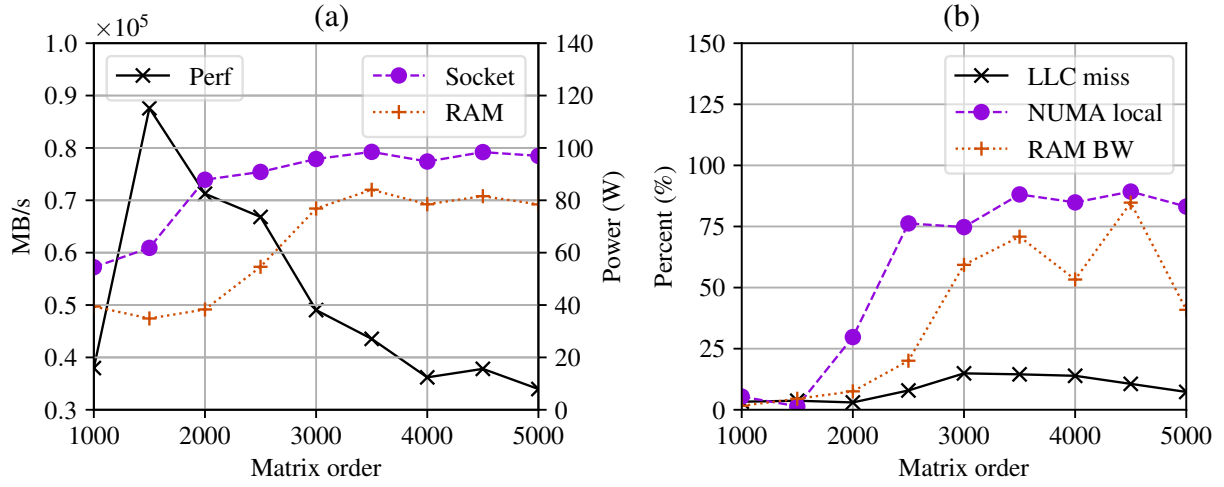


Figure 3.16: Transpose Cache Analysis – (a) DRAM Power, (b) DRAM Bandwidth, NUMA Local Memory Access, and LLC Miss Rate

Performance counters do not provide a direct measure of memory bandwidth. Equation 3.7 shows the formula used to derive memory bandwidth utilisation in bytes per second, using event counters for LLC local and remote misses, elapsed time, and cache line size of 64 bytes.

$$Bandwidth = \frac{64 \times (\text{OFFCORE\_RESPONSE\_0:L3\_MISS\_LOCAL} + \text{OFFCORE\_RESPONSE\_0:L3\_MISS\_REMOTE})}{time} \quad (3.7)$$

The results show that problem scale impacts where a program operates in the memory hierarchy. This in turn impacts the optimal settings for performance and energy efficiency tuning parameters, such as CPU frequency and thread count. Metrics such as cache miss and stall rates, and local and remote memory bandwidth can be used to identify where a code is operating in the memory hierarchy.

### 3.3.5 Node Scaling

The single node method described here can also be applied to multi-node systems. Multi-node results show the in-cache performance and energy efficiency benefits seen for a single node scale as nodes are added. Figure 3.17 (a) shows that super-linear scaling is achieved in some cases, such as performance more than doubling with the jump from 32 to 64 nodes at matrix order 24k (1). Super-linear scaling occurs because this problem size operates in LLC for 64 nodes but not for 32 nodes. Figure 3.17 (b) shows in-cache energy efficiency benefits also extend to larger problem sizes as the number of nodes increase.

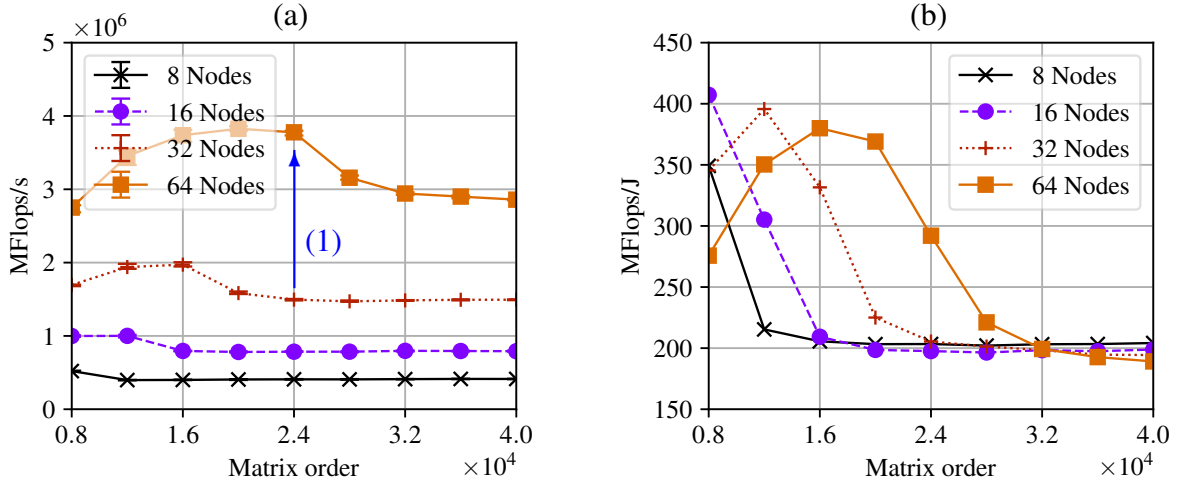


Figure 3.17: Stencil Node Scaling – (a) Performance and (b) Energy Efficiency

### 3.4 Conclusion

This chapter investigated performance and energy efficiency tuning parameters, and associated search space dimensions, that are controllable by parallel application users. Figures 3.5 to 3.12 show that an  $11 \text{ thread} \times 11 \text{ CPU frequency}$  search space of 144 measurements provides acceptable resolution for the single node performance and energy efficiency response curve plots.

The effect of OpenMP thread placement was evaluated for a range of test cases, including core level HTT, and scatter versus compact at CoD level and CPU socket level. Socket level scatter placement of one OpenMP thread per core is equivalent or superior to other thread placement test cases.

The lowest thread counts are also generally suboptimal for both performance and energy efficiency. This indicates that there is an opportunity to reduce search space resolution at low thread counts. For CPU frequency, performance and energy efficiency responses generally taper off at higher frequencies indicating that higher search space sensitivity is needed at higher frequencies.

Experimental analysis shows that thread and CPU frequency scaling provide significant tuning opportunities, with improvements of 24% in performance and 40% in energy efficiency identified. Results also show that a multi-objective optimisation strategy is required, where CPU frequency tuning to achieve maximum performance can require an 11% trade-off in energy efficiency.

Problem size responses were analysed to assess scaling across the memory hierarchy. Metrics including cache miss and stall rates, and NUMA local versus remote memory bandwidth, were used to identify code operation within the memory hierarchy. While these metrics provide valuable insight, they are system responses themselves, rather than predictors for system responses.

The memory footprint of a program will also typically depend on the parameters of the problem to be

solved. Such parameters are not tunable, since changing these parameters changes the problem itself.





Parts of the following publication have been incorporated in Chapter 4.

- [2] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, “Statistical and machine learning models for optimizing energy in parallel applications,” *The International Journal of High Performance Computing Applications*, 2019.

Chapter 4 Publication Contributor Statement

| Contributor       | Statement of contribution | %   |
|-------------------|---------------------------|-----|
| M. Endrei         | Writing of text           | 90  |
|                   | Proof-reading             | 25  |
|                   | Theoretical derivations   | 100 |
|                   | Numerical calculations    | 100 |
|                   | Preparation of figures    | 100 |
|                   | Initial concept           | 60  |
| C. Jin            | Writing of text           | 5   |
|                   | Proof-reading             | 10  |
|                   | Supervision, guidance     | 20  |
| M. N. Dinh        | Writing of text           | 5   |
|                   | Proof-reading             | 10  |
|                   | Supervision, guidance     | 20  |
| D. Abramson       | Proof-reading             | 10  |
|                   | Supervision, guidance     | 20  |
|                   | Initial concept           | 10  |
| H. Poxon          | Proof-reading             | 10  |
|                   | Supervision, guidance     | 10  |
|                   | Initial concept           | 10  |
| L. DeRose         | Proof-reading             | 10  |
|                   | Supervision, guidance     | 10  |
|                   | Initial concept           | 10  |
| B. R. de Supinski | Proof-reading             | 25  |
|                   | Supervision, guidance     | 20  |
|                   | Initial concept           | 10  |

# Chapter 4

## The Architecture of an Energy Tuning Framework

HPC performance tuning is a mature field that is already supported by a number of sophisticated and widely used tuning frameworks. An energy tuning framework that is controllable by application users needs to extend a conventional performance tuning framework with components that enable users to:

- Instrument the program under evaluation.
- Measure the energy consumed by the running program.
- Predict performance and energy efficiency responses.
- Trade-off performance and energy efficiency using the predicted responses.
- Consider measurement and prediction error limits in trade-off options.

Figure 4.1 shows the component model for the proposed energy tuning framework, named HPCProbe. This framework is designed to address the challenges identified in Chapter 1 by:

- Only using tuning parameters that parallel application users can control.
- Minimising the cost of finding optimal solutions by minimising the required input measurements.
- Accurately predicting application performance and energy efficiency trade-off options.
- Considering measurement and modelling error to ensure valid trade-off options are not eliminated.

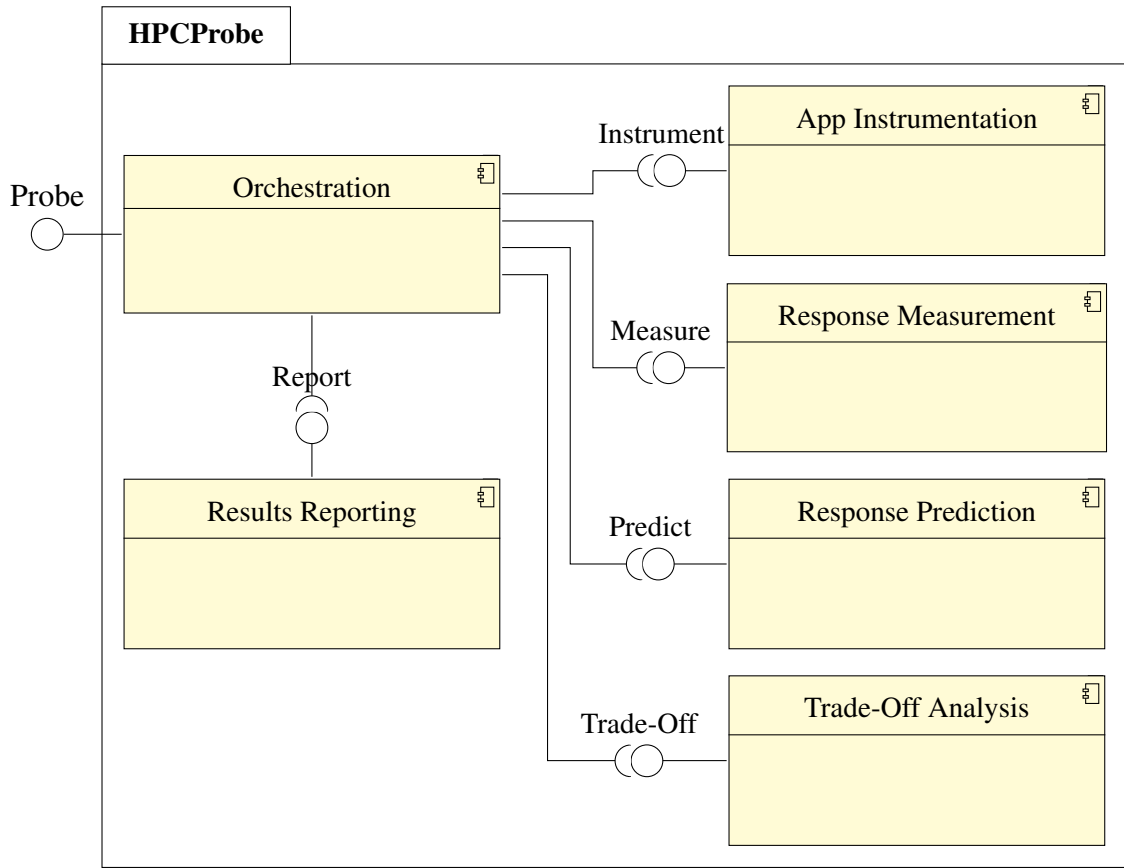


Figure 4.1: HPCProbe Component Model

In addition, HPCProbe uses a modular and extensible design with pluggable interfaces that support portability across instrumentation, measurement and modelling frameworks. The proposed design includes all components in an integrated package for trading off performance and energy in parallel programs.

The rest of this chapter describes in detail how the design of each component addresses the identified challenges.

## 4.1 Application Instrumentation

This component enables and reads the required counters for measuring performance and energy usage at appropriate times in the application run cycle.

When evaluating overall application performance and energy efficiency, these counters only need to be enabled at the start of each application run, and read when the run completes. If there is a risk of counter overflow, with long runs for example, intermediate reads may be required to detect and manage overflows.

The performance metric may simply be wall-clock time (or elapsed real time) for an application run. It could also be a figure of merit reported by the application itself, such as operations per second.

Where performance is a cumulative metric such as time in seconds, a cumulative energy metric such as total energy in joules is used. Where performance is a rate metric such as operations per second, the energy efficiency  $e_\eta$  for an application run can be derived from performance  $p_\mu$ , wall time  $t$ , and total energy  $E_T$ , as equation 4.1 shows.

$$e_\eta = \frac{p_\mu \cdot t}{E_T} \quad (4.1)$$

Current tuning frameworks provide utilities for instrumenting programs using performance counters on the system. Examples include commercial frameworks like Intel VTune Amplifier [71] and CrayPAT [60, 72], and open source frameworks such as Linux perf tools [73]. Programmers can also manually instrument code using an API such as Performance Application Programming Interface (PAPI) [74], or direct access to the hardware registers.

HPCProbe uses CrayPAT to instrument applications using PAPI preset, user defined, and native events and hardware information.

## 4.2 Response Measurement

This component measures the system performance and power usage responses of the program under evaluation. CPU hardware counters can be used to monitor a wide range of events, including cache miss rates, stall cycles, and operations per second.

### 4.2.1 Power Measurement

Power consumption metrics are an essential input for an energy tuning framework. As discussed in 2.5, a range of approaches are available, from mains socket and hook-on power meters, model-based methods using processor counters, to motherboards and peripherals equipped with inbuilt sensors. HPCProbe uses Cray power management counters (pm\_counters) and Intel Running Average Power Level (RAPL) counters to monitor energy and power consumption for all nodes.

Cray XC systems have node-level inbuilt sensors to measure temperature, current, and voltage. Cray pm\_counters provide real-time power and energy measurements, which are updated at a frequency of 10 Hz. CrayPAT is used to evaluate program behaviour. It instruments the program, collects the specified counters at runtime, and aggregates and reports the collected counters. Cray pm\_counters can also be read directly from the Linux sysfs folder, `/sys/cray/pm_counters`, as a program launches and terminates.

The RAPL interface is used for component level measurements, including CPU and DRAM energy use. CrayPAT is used to collect the required RAPL power counter data.

## 4.2.2 Measurement Evaluation

In addition to capturing required performance and energy usage metrics, energy tuning frameworks must also assess measurement confidence intervals. For example, measurement data may show performance improvements of 5% are possible at 20% energy cost. If the measurement confidence interval is  $\pm 10\%$  it may be difficult to justify the high energy cost for a performance improvement that is not statistically significant.

This section examines statistical methods for analysing the confidence level of performance and energy measurements. HPCProbe uses normal and  $t$ -distributions for confidence intervals analysis. Q-Q (quantile-quantile) plots are used to visually compare the measured data distribution with the standard normal distribution.

### Normal Probability Distribution

Physical measurements that have a range of random factors influencing measurement errors are often normally distributed. As such, computer performance measurements are also typically assumed to be normally distributed when analysing confidence levels. Normality tests can be used to check this assumption.

A confidence interval for the mean of a set of repeated measurement samples can be calculated using the standard deviation and normal probability distribution table.

The confidence interval of  $n$  normally distributed samples with mean of  $\bar{x}$  and standard deviation of  $\sigma$  is given by equation 4.2.

$$c = \bar{x} \pm z^* \cdot \frac{\sigma}{\sqrt{n}} \quad (4.2)$$

The required  $z^*$  for the confidence interval comes from the standard normal distribution table. For example, equation 4.3 and Figure 4.3 show the 95% confidence interval for normally distributed data.

$$P(X < z^*) = \frac{1 + 0.95}{2}, \text{ when } z^* = 1.96 \quad (4.3)$$

### Student's $t$ -distribution

Computer performance analysts are usually restricted to a small number of measurements due to test case complexity, and time and cost limitations.

The Student's  $t$ -distribution is used for small samples from normally distributed populations, and the confidence interval for the sample mean is now given by equation 4.4.

$$c = \bar{x} \pm t^* \cdot \frac{\sigma}{\sqrt{n}} \quad (4.4)$$

The required  $t^*$  for the confidence interval now comes from the  $t$ -distribution table. In this case, equation 4.5 and Figure 4.2 show the 95% confidence interval for the  $t$ -distribution table at five samples (four degrees of freedom).

$$P(X < t^*) = \frac{1 + 0.95}{2}, \text{ when } t^* = 2.78 \quad (4.5)$$

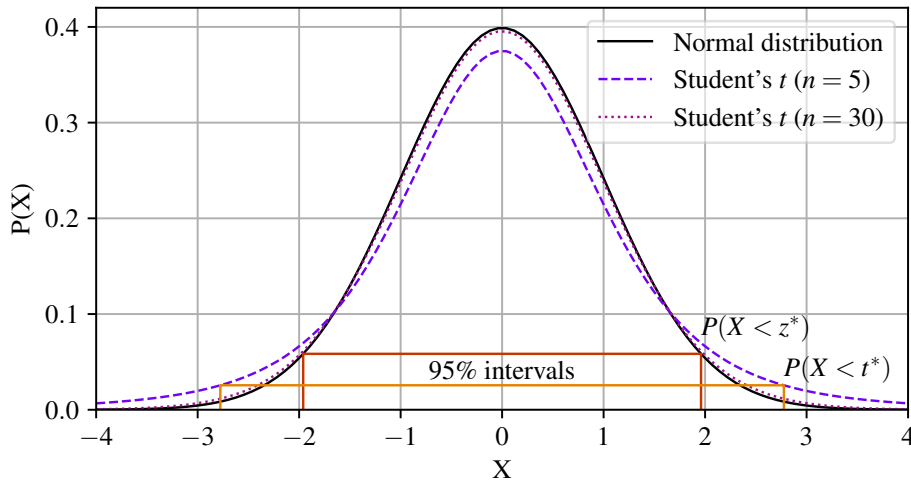


Figure 4.2: Probability Density Function

Figure 4.2 also shows that the  $t$ -distribution approaches the normal distribution as the sample size increases. As such, the  $t$ -distribution is typically recommended for sample sizes under 30.

Performance analysts can use this approach to assess the statistical significance of performance improvement measurements [75].

### Normality Test

The Q-Q plot can be used to visually compare a measured data distribution with the standard normal distribution [76]. Figure 4.3 (a) shows 100 energy measurement samples from our stencil case study plotted against the normal distribution line. The sample points show good alignment with the normal line, indicating that the data set is normally distributed. The results were similar for our FLOPs per second and wall clock time measurements.

Figure 4.3 (b) shows the Probability Density Function plot for our sample energy data set, which provides an alternative view of the sample distribution. The PDF plot overlays the data set histogram, estimated data set distribution function, and standard normal distribution function.

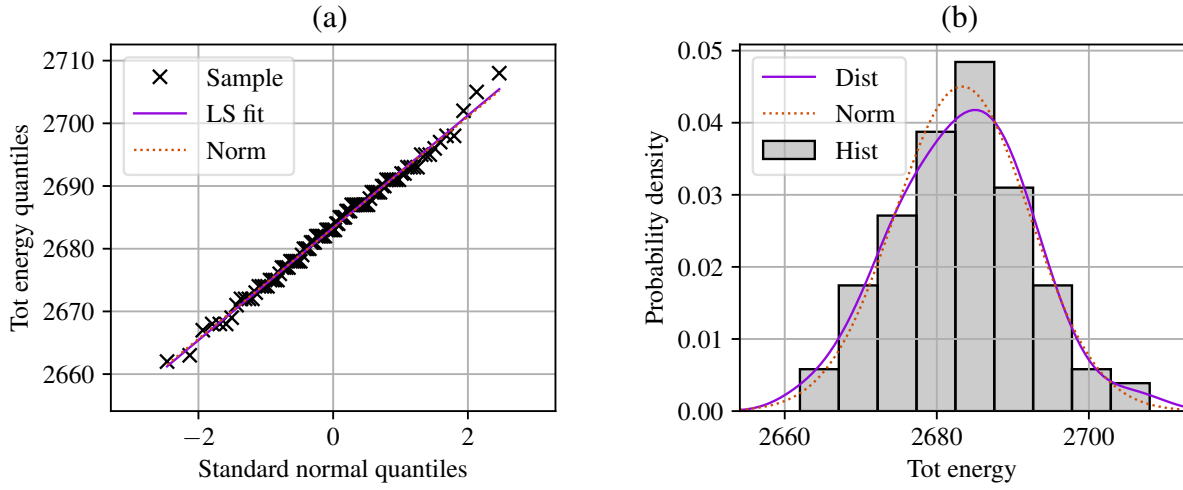


Figure 4.3: Sample population distribution

## 4.3 Response Prediction

The cost of tuning an application must be amortised over future runs of the application. To achieve an effective return on tuning effort, energy tuning frameworks must minimise the cost of accurately predicting optimal configurations. As stated at the start of this chapter, the aim for HPCProbe is to predict performance and energy trade-off options with accuracy of  $\pm 10\%$ , using 10% or less of the input parameter search space.

Chapter 2 examines related work based on deterministic, or functional models such as the Roofline or ECM models. This work evaluates stochastic, or statistical models that predict a probable outcome with an associated error term  $\epsilon$  as equation 4.6 shows.

$$y = f(x) + \epsilon \quad (4.6)$$

Stochastic models can deal with data uncertainties and limitations arising from partial or noisy data sets. When  $\epsilon$  values are normally distributed around a mean of zero, statistical techniques can be used to assess how accurately the model fits the data.

The class diagram for the HPCModel package of HPCProbe is shown in Figure 4.4. The package includes three stochastic model types for evaluation, derived from the *Model* base class. The *PolyReg* class implements a basic polynomial model using ordinary least squares regression. *SplineReg* im-



plements a B-spline (or basis spline) piecewise polynomial model also using ordinary least squares regression. The *DnnReg* class implements a deep neural network regressor model.

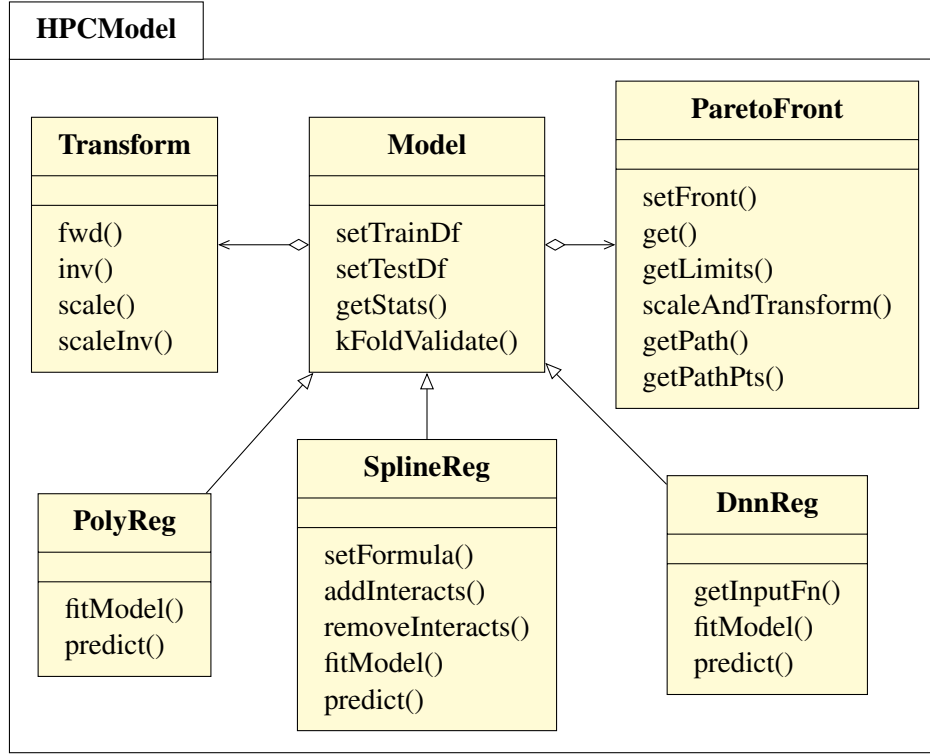


Figure 4.4: HPCModel Class Diagram

Each model type provides methods to fit the model to training data and to predict responses for unseen input data. The *Model* base class provides methods to set training and test data, get model evaluation statistics, and validate the fitted model.

The *Transform* class provides data transforms to mitigate normal distribution deviations and scaling functions to improve model convergence. The *ParetoFront* class is part of the analysis component described in section 4.4.

### 4.3.1 Model Predictors

Model input parameters, or *predictors*, are required that are user-controllable and can accurately model system *responses*: energy efficiency and performance. Chapter 3 shows that a complex relationship exists between system characteristics, workload features, concurrency, CPU frequency scaling and the observed system response. The definitions in Table 4.1 are used to express these relationships as shown in equations 4.7 and 4.8.

$$E(w_n, s_m) = F_e(n_i, c_j, f_k) \quad (4.7)$$

$$P(w_n, s_m) = F_p(n_i, c_j, f_k) \quad (4.8)$$

Table 4.1: Parameters Used in the Models

| Parameter             | Description  |
|-----------------------|--|
| $w_n$                 | Workload or application                                |
| $s_m$                 | System or architecture                                 |
| $E(w_n, s_m), e_\eta$ | Energy efficiency of $w_n$ on $s_m$                    |
| $P(w_n, s_m), p_\mu$  | Performance of $w_n$ on $s_m$                          |
| $n_i$                 | Homogeneous nodes count of $i$                         |
| $c_j$                 | Cores or threads count of $j$                          |
| $f_k$                 | CPU frequency or DVFS setting of $k$                   |
| $bs(x)$               | Transform to generate B-spline bases for parameter $x$ |
| $F_e(n_i, c_j, f_k)$  | Energy efficiency function                             |
| $F_p(n_i, c_j, f_k)$  | Performance function                                   |
| $e_T$                 | Total energy (J) required to run workload              |
| $t_T$                 | Total time (s) required to run workload                |

### 4.3.2 Sampling

Energy tuning frameworks that use predictive modelling must have efficient methods for selecting the data samples required to train the model. Using more training samples can improve model accuracy, but too many samples may make the training costs prohibitive.

The `setTrainDf` method of the `Model` class provides three sampling options for evaluation, each using around 10% of the search space. Model training samples can be generated using uniform (or non-random), random, and Latin hypercube sampling [77]. Figure 4.5 shows an 11 core count  $\times$  11 frequency search space with each method.

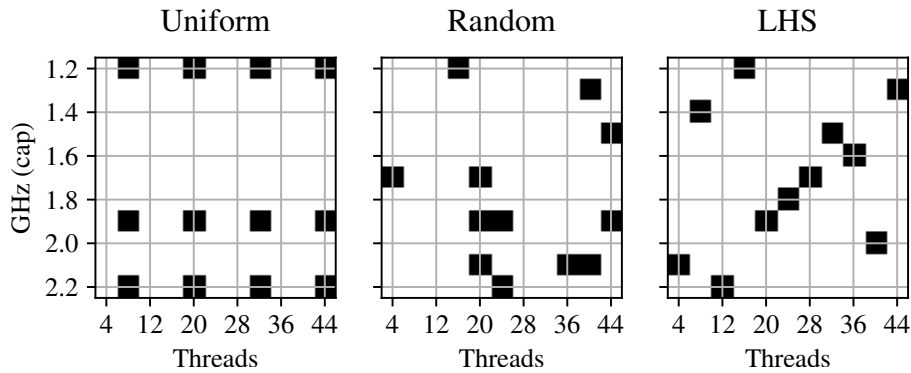


Figure 4.5: Uniform, Random, and Latin Hypercube Sampling

Random sampling has 12 randomly selected samples making up 10% of the search space. Latin hypercube sampling has a randomly selected sample from each row and column, resulting in 11 samples.

The trade-off problem between minimising sample size while maximising model accuracy can be made more tractable with the use of application and architecture findings from Chapter 3. These findings inform sample selection for uniform sampling.

With uniform sampling, the threads dimension has higher resolution with samples at four thread points versus three points in the frequency dimension. Suboptimal points at the lowest thread counts are excluded. Frequency sample points provide higher resolution where needed at higher frequencies.

### 4.3.3 B-Spline Model Formulation

The initial formulation of the regression models is guided by responses of representative software kernels to variations in node count, core count, and CPU frequency setting from section 3.3. The models require polynomial terms to capture the observed curvilinear performance and energy efficiency responses.

The inflexibility of polynomials [78] may result in undesirable oscillations in the predicted response or an overall response shape that is dominated by a few samples. To minimise the impact of this inflexibility, B-spline piecewise polynomials are used.

Spline functions fit a smooth curve along a series of points, or knots. The curve that joins each pair of points, or *spline*, is constructed piece-wise from polynomial functions. A B-spline, or basis spline function, improves the continuity at the knots, which may otherwise affect model continuity.

These models have several configuration options to tune their accuracy and efficiency. Increasing the polynomial degree and degrees of freedom allows the spline to fit more complex curves. Interactions between terms, linear and non-linear terms, transforms to reduce effects of data distribution skew, and sampling method also need to be considered.

The level of variability that each predictor drives in the response can be observed and knowledge of expected interactions between predictors can be used. For example, increasing core counts drive resource contention that produces non-linear responses. Increasing CPU frequencies also typically drive non-linear responses once core contention within the node starts to dominate. Such observations lead to the following model settings and simplifications:

- B-spline degrees of freedom is three to balance underfit and overfit in the overall predicted response;
- Node count, core count, and frequency are quadratic polynomial terms to balance underfit and overfit of the spline segments between knots;
- Node and core count have an interaction term as the term coefficient is significant;
- Core count and frequency also have an interaction term with a significant coefficient;
- Frequency and node count have no interaction term as the coefficient is insignificant;

- Natural logarithm transform of response to mitigate data skew.

Model settings are evaluated with an iterative approach that assesses the data distribution and correlation, the model fit, and the coefficient magnitudes. The optimal polynomial term order and B-spline degrees of freedom at the transition point between model underfit and overfit are also determined experimentally. For example, a near zero coefficient for the node count and frequency interaction term confirms that it can be removed from the model with little impact.

The resulting statistical models for energy efficiency and performance of a parallel application that runs on a homogeneous HPC cluster use equations 4.9 and 4.10.

$$\begin{aligned} \log_e(e_\eta) \sim & bs(n_i) + bs(c_j) + bs(f_k) \\ & + bs(n_i) : bs(c_j) + bs(c_j) : bs(f_k) \end{aligned} \quad (4.9)$$

$$\begin{aligned} \log_e(p_\mu) \sim & bs(n_i) + bs(c_j) + bs(f_k) \\ & + bs(n_i) : bs(c_j) + bs(c_j) : bs(f_k) \end{aligned} \quad (4.10)$$

The  $\log_e$  transform applied to model responses mitigates normal distribution deviations.

These equations use Wilkinson notation [79], which is accepted by regression analysis tools, such as MATLAB, and by R and Python. The operator ‘ $\sim$ ’ means *is modeled by* while ‘+’ adds a term to the model, and ‘:’ includes interactions between terms.

In addition to energy efficiency and performance rates, the models can support cumulative responses such as total energy and total time based on equations 4.11 and 4.12.

$$\begin{aligned} \log_e(e_T) \sim & bs(n_i) + bs(c_j) + bs(f_k) \\ & + bs(n_i) : bs(c_j) + bs(c_j) : bs(f_k) \end{aligned} \quad (4.11)$$

$$\begin{aligned} \log_e(t_T) \sim & bs(n_i) + bs(c_j) + bs(f_k) \\ & + bs(n_i) : bs(c_j) + bs(c_j) : bs(f_k) \end{aligned} \quad (4.12)$$

The natural exponential is applied to model predictions to reverse the  $\log_e$  transform.

Equation 4.13 shows the right-hand side expansion for equations 4.9 to 4.12. Each model has 28 predictor terms (1 intercept,  $3 \times 3$  spline B-spline terms,  $2 \times 3$  spline B-spline  $\times$  3 spline B-spline

interaction term). Predictor and response training data is collected then ordinary least squares regression is used to determine the term coefficients,  $\beta_0$  to  $\beta_{27}$ .

$$\begin{aligned}
& \beta_0 + \beta_1 \cdot bs_1(n_i) + \beta_2 \cdot bs_2(n_i) + \beta_3 \cdot bs_3(n_i) \\
& + \beta_4 \cdot bs_1(c_j) + \beta_5 \cdot bs_2(c_j) + \beta_6 \cdot bs_3(c_j) \\
& + \beta_7 \cdot bs_1(f_k) + \beta_8 \cdot bs_2(f_k) + \beta_9 \cdot bs_3(f_k) \\
& + \beta_{10} \cdot bs_1(n_i) \cdot bs_1(c_j) + \beta_{11} \cdot bs_2(n_i) \cdot bs_1(c_j) \\
& + \beta_{12} \cdot bs_3(n_i) \cdot bs_1(c_j) + \beta_{13} \cdot bs_1(n_i) \cdot bs_2(c_j) \\
& + \beta_{14} \cdot bs_2(n_i) \cdot bs_2(c_j) + \beta_{15} \cdot bs_3(n_i) \cdot bs_2(c_j) \\
& + \beta_{16} \cdot bs_1(n_i) \cdot bs_3(c_j) + \beta_{17} \cdot bs_2(n_i) \cdot bs_3(c_j) \\
& + \beta_{18} \cdot bs_3(n_i) \cdot bs_3(c_j) \\
& + \beta_{19} \cdot bs_1(c_j) \cdot bs_1(f_k) + \beta_{20} \cdot bs_2(c_j) \cdot bs_1(f_k) \\
& + \beta_{21} \cdot bs_3(c_j) \cdot bs_1(f_k) + \beta_{22} \cdot bs_1(c_j) \cdot bs_2(f_k) \\
& + \beta_{23} \cdot bs_2(c_j) \cdot bs_2(f_k) + \beta_{24} \cdot bs_3(c_j) \cdot bs_2(f_k) \\
& + \beta_{25} \cdot bs_1(c_j) \cdot bs_3(f_k) + \beta_{26} \cdot bs_2(c_j) \cdot bs_3(f_k) \\
& + \beta_{27} \cdot bs_3(c_j) \cdot bs_3(f_k)
\end{aligned} \tag{4.13}$$

#### 4.3.4 Neural Network Model Formulation

Artificial Neural Networks are networks of simple processing units or neurons that generate predicted responses for a set of input values or *features*. Figure 4.6 shows a *feed-forward* neural network with three hidden layers. As with the regression models, the neural network inputs are node count, core count, and CPU frequency setting. The input layer has a neuron to pass the value of each feature to the hidden layers.

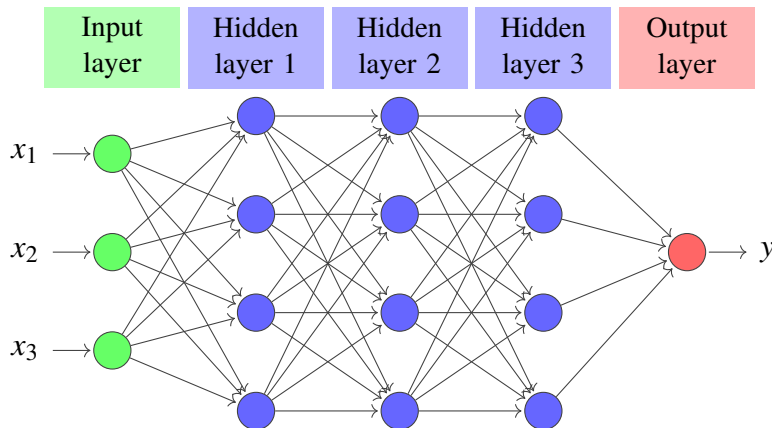


Figure 4.6: Deep Neural Network with Multiple Hidden Layers

Figure 4.7 shows the basic structure of a *hidden unit*, which is a neuron in the hidden layer. The output  $b_h$  of hidden unit  $h$  is the weighted sum of inputs  $a_1, \dots, a_n$ , as equation 4.14 shows. Weights on the hidden unit inputs  $w_{h,1}, \dots, w_{h,n}$  determine the effect that each has on the hidden unit output.

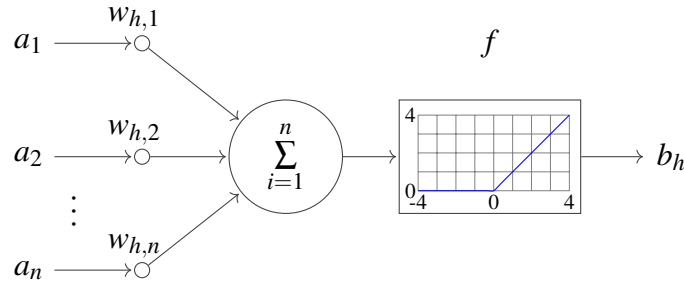


Figure 4.7: Basic Structure of Hidden Unit  $h$

$$b_h = f\left(\sum_{i=1}^n w_{h,i} \cdot a_i\right) \quad (4.14)$$

The hidden unit *activation function*  $f$  implements the rectifier function  $f(x) = \max(0, x)$  to model observed behavior. Rectified linear units (ReLUs) outperform sigmoid or hyperbolic tangent functions [80] and so are now widely used for deep learning. Deep learning architectures generally consist of two or more hidden layers.

Network training uses *back-propagation* to tune the input weights to minimise output error. A *loss function* at the output determines the error between network predictions and observed training samples. An *optimiser function* iteratively analyses the loss gradient to tune the weights of hidden units.

The output layer differs for classification and regression problems. Classifiers work with discrete or categorical *labels*, and require an output unit for each label. Regressors work with continuous, real numbers and use a single output unit that can generate a continuous response.

Several neural network configuration options can be used to tune the accuracy and efficiency of machine learning models. Increasing the number of hidden layers and the number of hidden units per layer can allow the model to fit more complex functions. The models also need sufficient training steps to converge on a fit of the required precision. However, too many steps may overfit the observed data.

Input data scaling is typically required to avoid saturating the network. Scaling outputs can also improve the learning rate and convergence. Scaling methods such as min-max normalisation and standardisation are typically used. For standardisation scaling, the scaled feature  $x'$  for feature mean

$\bar{x}$  and standard deviation  $\sigma_x$  is given by equation 4.15.

$$x' = \frac{(x - \bar{x})}{\sigma_x} \quad (4.15)$$

The following neural network configuration was experimentally determined to meet the requirements for model fit and error limits:

- DNN regressor model to provide numerical performance and energy predictions;
- Three hidden layers of 10, 20, and 10 hidden units to provide fan-out between input and hidden layers, and fan-in between hidden and output layers;
- 2,000 training steps to ensure model convergence without overfit;
- Natural logarithm transform of response to mitigate data skew;
- Standardisation scaling of response to aid learning rate and convergence;
- Standardisation scaling of features/predictors to avoid network saturation.

### 4.3.5 Model Evaluation

Correlation analysis,  $k$ -fold cross validation, and RMS error and  $R^2$  statistics are used to evaluate the model. These quantities are derived from many measurement samples in order to provide a statistically significant validation of model accuracy.

Spearman's rank correlation coefficient,  $\rho$ , measures the correlation between predictors and responses. Values near zero indicate weak correlation. This statistic is useful when validating predictor selection hypotheses.

Values of the  $R^2$  statistic near 100% indicate a model fits training data well. With  $n$  observations of  $y$  and model fitted values of  $f$ ,  $R^2$  is calculated by dividing the residual sum of squares by the total sum of squares, as equation 4.16 shows.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - f_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (4.16)$$

RMS error, the standard deviation of the prediction error, measures how closely observed data fits data forecasts that the model generates. RMS error values near zero indicate a close fit between

model forecasts and observed data. RMSE is calculated by taking the square root of the mean of the residual sum of squares, as equation 4.17 shows.

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (f_i - y_i)^2}{n}} \quad (4.17)$$

The  $k$ -fold cross validation method partitions the data set into  $k$  equal-size subsets.  $k - 1$  subsets serve as model training data and the remaining subset is the model test data. The process is repeated  $k$  times to test model accuracy across the full data set.

The *getStats* method of the Model class provides access to a range of evaluation data for models, including RMS error and  $R^2$  statistics. The *kFoldValidate* method uses 3-fold cross validation to confirm that the model does not overfit a subset of the data.

## 4.4 Trade-Off Analysis

HPCProbe performs trade-off analysis by computing a trade-off zone for a set of performance and energy efficiency observations (or predictions). Pareto optimisation is widely used to identify optimal trade-off points between competing objectives. Figure 4.8 shows energy efficiency plotted on the  $x$ -axis and performance plotted on the  $y$ -axis. Both responses need to be maximised, so Pareto-optimal points lie on the top-right of the data set (circled). The line joining these points forms the Pareto front. Points off the front are not Pareto-optimal as points on the front always provide an improvement in one parameter with less impact on the other.

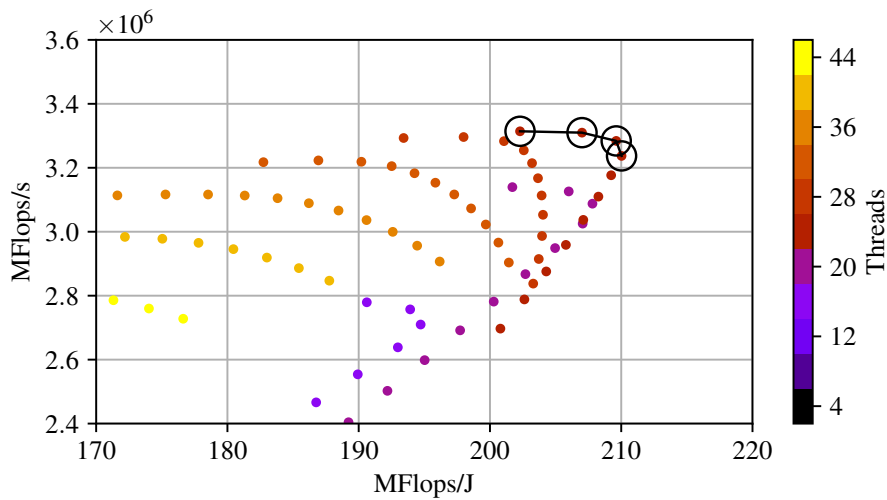


Figure 4.8: Pareto Front – Performance and Energy Efficiency

HPCProbe provides the *ParetoFront* class in Figure 4.4 for identifying performance and energy efficiency trade-off points. The *setFront* method computes Pareto points in the given data set when a *ParetoFront* instance is initialised. The identified points are accessible using the *get* method.



As stated at the start of this chapter, an important design goal for this component is ensuring that valid trade-off options are not incorrectly eliminated due to measurement or modelling error. Figure 4.8 shows performance points to the left of the Pareto front that may need to be included if within error limits of the front, as may energy efficiency points below the front.

#### 4.4.1 Pareto Front Evaluation

HPCProbe evaluates RMS error between predicted Pareto points and the observed values, but also uses further, specific techniques to assess the accuracy of the predicted fronts. Pareto front accuracy is assessed using the following metrics:

- Coincident point count, that is, the count of search space points that occur in both observed and predicted fronts.
- Non-coincident observed front point count, grouped by distance to nearest neighbour on predicted front.
- Non-coincident predicted front point count, grouped by distance to nearest neighbour on observed front.
- Predicted and observed minimums and maximums for each objective (energy efficiency or performance).
- Predicted and observed trade-off ranges for each objective compared to their values when threads and CPU frequency are at the maximum settings.

To normalise across dimensions, the distance to the nearest neighbour is measured as steps in the search space. For example, the thread count dimension may have 11 increments from 4 to 44 in steps of 4. The distance from 36 to 40 would be one search space step or  $1/11 = 9\%$ , and 36 to 44 would be two search space steps or  $2/11 = 18\%$ .

Pareto front evaluation data is accessible using the *getStats* method of the Model class.

#### 4.4.2 Measurement and Modelling Error

Response observations exhibit measurement error while model forecasts exhibit prediction error. As such, the HPCProbe design must allow for both measurement and prediction errors when identifying trade-off options between performance and energy efficiency. Valid trade-off options within error limits lie in a zone close to the Pareto front, rather than only lying directly on it. This work introduces the concept of a *trade-off zone* that includes all values near the Pareto front that are not statistically distinguishable from those on the front.

Figure 4.9 shows the steps to construct the trade-off zone:

1. Plot the data set against the trade-off parameters (energy efficiency and performance in our case).
2. Plot the Pareto front along the Pareto-optimal points.
3. Extend the Pareto front outer limits horizontally and vertically to encompass all points that are off the front but are within the error limits for the respective axes.
4. Scale and translate the Pareto front by the axes error limits to set the inner limits of the trade-off zone.
5. Close the two curves, which creates a polygon that represents the trade-off zone.

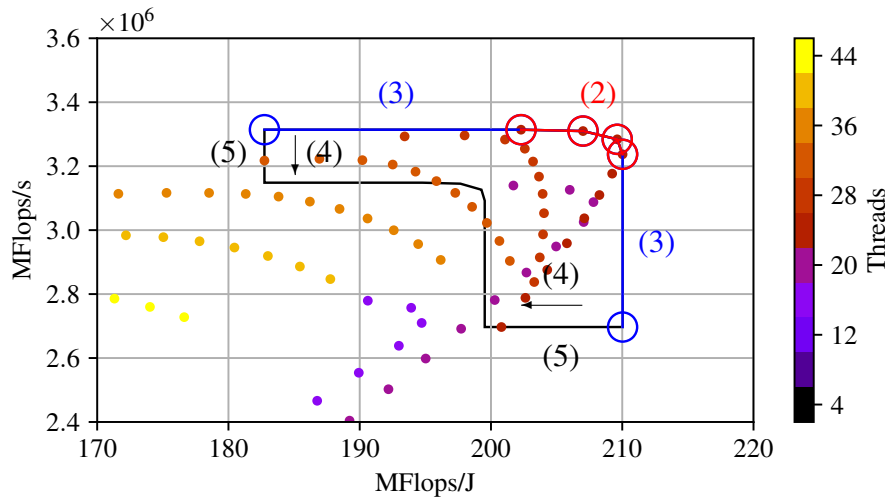


Figure 4.9: Pareto Trade-Off Zone Construction Steps

The trade-off zone polygon encloses the set of points that may be Pareto optimal when we compensate for the error limits of each axis. These points provide the trade-off options for energy efficiency and performance. Since measured and predicted error limits differ, the measured and predicted Pareto fronts have distinct trade-off zones.

The *getPath* method of the *ParetoFront* class provides access to the trade-off zone polygon. The *getPathPts* method provides access to the list of Pareto optimal data points enclosed by the trade-off zone.

The models only use measured data for the response variable, which provides two benefits. First, model coefficients are not biased by random error in the response measurements [81]. Consequently, the regression estimates tend to average training data values, plus or minus measurement error. Second, models can predict for unseen predictor data, allowing large parameter spaces to be practically explored with a small number of training measurements.

In our laboratory environment, measurement error levels can be controlled to around 5% by ensuring that overall execution time is large compared to program initialisation and shutdown time and to the temporal resolution of system power counters. In less controlled environments with larger measurement error, the size of the trade-off zone will increase as more data points fall within the error limits. This effect occurs even if the full search space is measured. The estimated measurement error and model error are used to assess the level of alignment or overlap between the observed and predicted trade-off zones.

## 4.5 Results Reporting

HPCProbe provides the HPCPlot package for reporting and visualising optimisation results, as Figure 4.10 shows. The *Plot* class is the base class for generating 2D and 3D graphical representations of the results. The *Plot* class has associated experiment details, including a list of experiment *ResultSets*. Experiment *ResultSet* data is read, selected, and merged into a plot data set by the *PlotDf* class. Data manipulation operations such as filtering, slicing, and vector arithmetic can then be used on plot data.

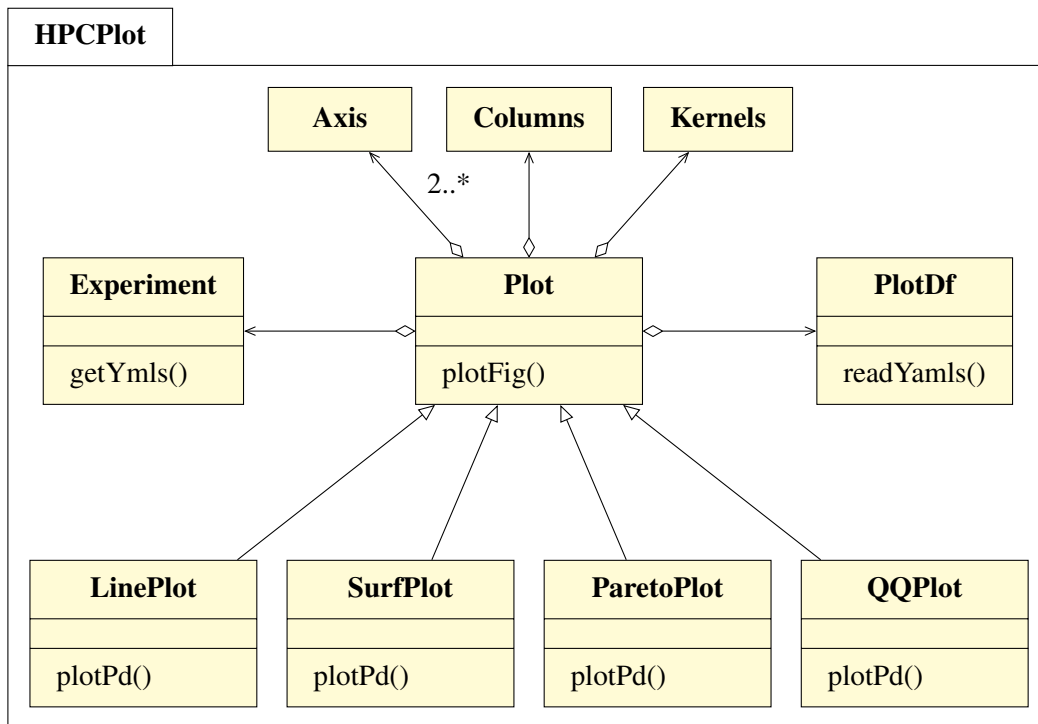


Figure 4.10: HPCPlot Class Diagram

Plots have two or more *Axis* instances, and reference *Columns* and *Kernels* meta data such as column units and scale, and kernel or application specifics such columns for optimisation.

The *LinePlot* class provides 2D *x/y*-axis plots. The *SurfPlot* class adds a *z*-axis for 3D surface plots. The *ParetoPlot* class provides visualisation of the trade-off zone plotted against energy efficiency and performance data points.

The Results Reporting component also accesses model and Pareto front evaluation metrics using the *getStats* and *kFoldValidate* methods of the *Model* class.

## 4.6 Orchestration

HPCProbe orchestration components are shown in Figure 4.11. The *Experiment* class in the HPCProbe sub-component provides methods for configuring and running energy tuning experiments, for reading and parsing job results, and for statistical analysis of results.

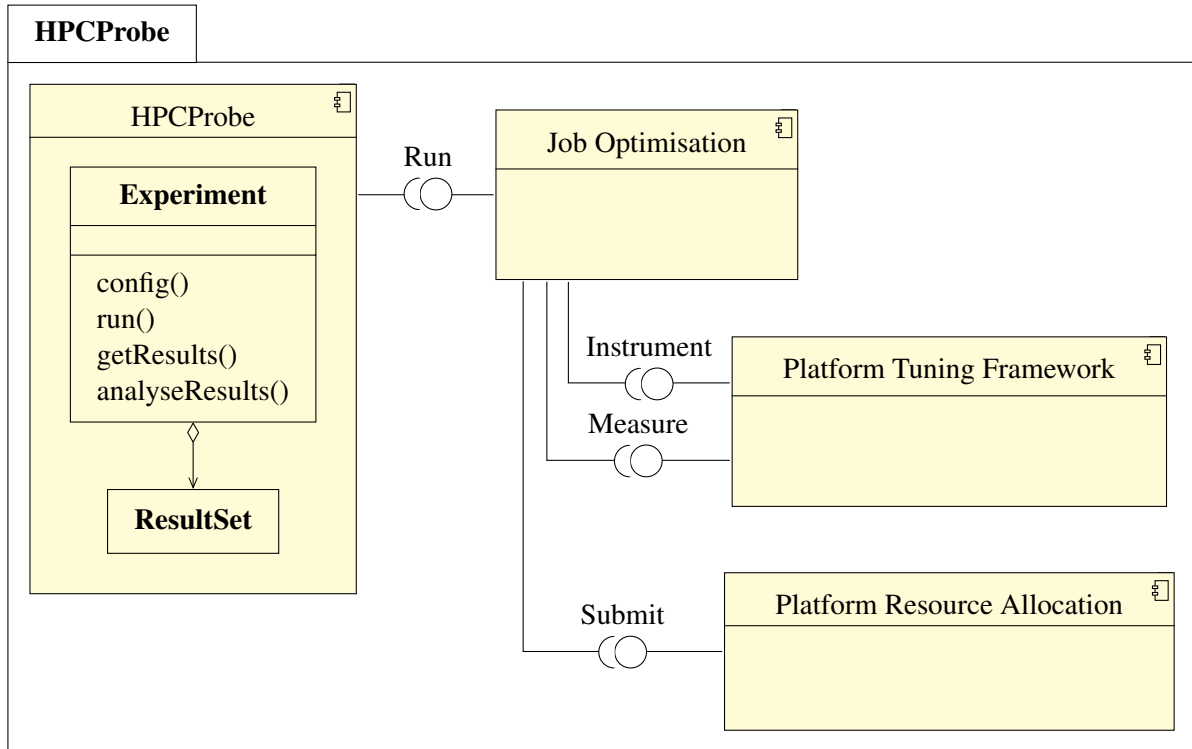


Figure 4.11: HPCProbe Orchestration Components

HPCProbe can integrate with different platform tuning and resource allocation frameworks, which improves cross-platform portability. HPCProbe can also integrate with job optimisation frameworks such as Nimrod/O [29] for search heuristics, results caching, and resource allocation using Nimrod/G [82]. Nimrod/G supports a wide range of cluster, grid, and cloud platforms, including PBS, Globus, Amazon EC2, and Microsoft Azure.

## 4.7 Data Model

Figure 4.12 provides an overview of key entity and relationship structures for HPCProbe. The *Experiment* class defines parameter ranges for the energy tuning experiment. Each run of an experiment generates a *ResultSet*. The *ResultSet* class uses the *Parameters* class to define the input parameters for each program run within an experiment. Parameters include CPU frequency, thread count, node count, and thread placement settings.

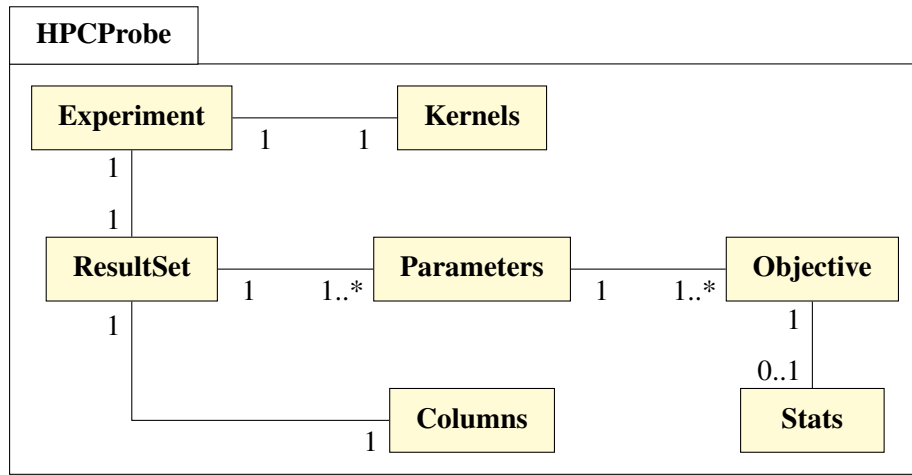


Figure 4.12: HPCProbe Data Model

The system responses or objectives for each run are captured in one or more *Objectives*. *Objectives* include measured values such as total time, total energy, and operations per second. Experiments with repeated runs also capture statistics for each objective using the *Stats* class. Statistics captured include sample means, median, standard deviation and *t*-distribution confidence interval.

The *Kernels* class specifies attributes of programs being tuned, such as columns to be optimised and if optimisation requires minimum or maximum values. The *Columns* class specifies *Parameter* and *Objective* attributes such as measurement units and scale.

Class, method, and attribute descriptions are available in Appendix B.

## 4.8 Architecture Summary

This chapter examined novel architectural features for a new tuning framework that enables parallel application users to make performance and energy trade-off decisions. Key architectural innovations that differentiate HPCProbe from traditional tuning frameworks include:

- Development of stochastic models that can make accurate predictions while tolerating small and noisy training data sets;
- A scheme for generating predicted trade-off ranges that can be validated using statistical techniques;
- A system for dealing with measurement and modelling error in multi-objective optimisation analysis;
- A component-based architecture where components are substitutable to suit the application tuning environment.

Additionally, this chapter presented advanced elements of the HPCProbe energy tuning framework, including its modular design and pluggable interfaces that can flexibly integrate with existing instrumentation and measurement tools. The outcome is an energy tuning architecture that delivers a new and innovative solution for parallel application users.

# Chapter 5

## Implementation Details

Chapter 4 presented the functional components and relationships that provide a logical design structure for HPCProbe. This chapter presents details for a HPCProbe prototype implementation that realises the proposed components design.

HPCProbe includes features of traditional HPC performance tuning frameworks, such as application instrumentation, response measurement, and results reporting. This chapter focuses on novel features of these components, along with implementation details for components that provide new predictive models and trade-off techniques for responses that are subject to error. Implementation of the following functionality is considered in detail:

- Application instrumentation and response measurement using CrayPAT.
- Response prediction using Python B-spline and machine learning models.
- Algorithms for constructing the Pareto trade-off zone used in trade-off analysis.
- APIs for numerical and visual results reporting.
- Integrated orchestration using Nimrod/O, PBS, and CrayPAT.

This chapter is intended to provide a robust demonstration of how the research contributions of this thesis may be implemented in a practical tool.

### 5.1 Deployment Model

The deployment model for the HPCProbe implementation is shown in Figure 5.1. The deployment model places deployment artefacts on the HPCProbe user workstation and the HPC cluster login node. The deployment artefacts implement functions assigned to components in Chapter 4. HPCProbe does

not interact directly with the HPC cluster. This interaction is via the PBS resource management tools, qsub and aprun.

The component numbers overlaying each artefact and the associated legend in Figure 5.1 provide traceability between the design and implementation. This traceability provides a demonstration of the implementation completeness.

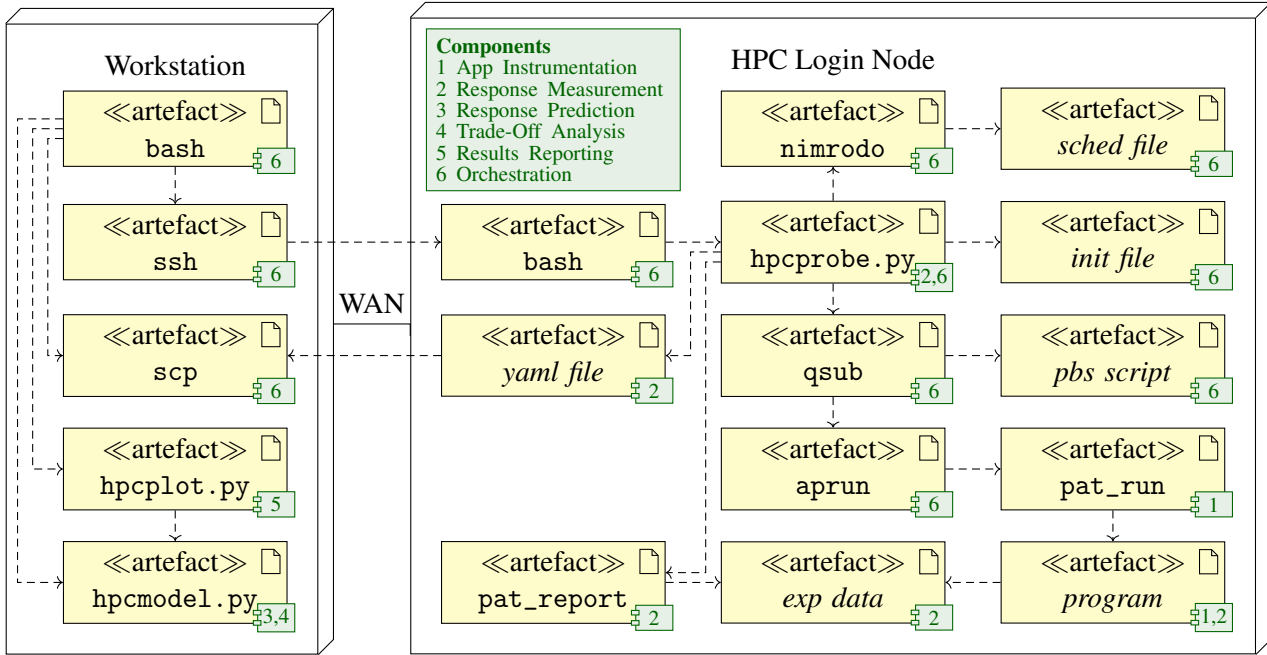


Figure 5.1: HPCProbe Deployment Model

The following sections present key implementation features of the deployment artefacts that realise each component.

### 5.1.1 Application Instrumentation

On the Cray Linux Environment (CLE) used for this work, the `pat_build` or `pat_run` utilities can be used to instrument programs without the need to modify source code.

The `pat_build` utility requires that the original program be compiled and linked with the `perftools-base` module loaded. A program written in C is instrumented as follows:

```
$ module load perftools-base
$ cc -o <program> <program>.c
$ pat_build -w <program>
```

The `pat_build -w` option enables a tracing experiment where code is instrumented synchronously by inserting hooks within the program. CrayPAT also supports sampling experiments using `ex-`



ternal or asynchronous instrumentation. Tracing experiments work well when tuning overall program performance and energy use, as in our case. The instrumented program is written to the file `<program>+pat`.

The `pat_run` utility provides an alternative to `pat_build` with the advantage that it can be used without recompiling. Program instrumentation and measurement can be invoked in one step when the program is launched, as section 5.1.6 shows.

Instrumenting programs with `pat_build` generally provides greater data collection capability and flexibility than using `pat_run`, but `pat_run` can provide the metrics required for performance and energy efficiency tuning without the need to recompile. Avoiding the need to compile programs simplifies the tuning process for both the program user and the tuning orchestration framework. HPCProbe works with both `pat_build` and `pat_run`.

### 5.1.2 Response Measurement

HPCProbe uses CrayPAT to access PAPI, RAPL and Cray PM counter data for an instrumented program. The `hpcprobe.py` module captures experiment data reported by the program itself at runtime and by the CrayPAT `pat_report` utility post program runtime for analysing experiment data written by the program instrumentation to the `<exp data>` folder.

The Python regular expressions library is used in `hpcprobe.py` to parse performance data output from the program and `pat_report` at the counter group and individual counter levels. This allows individual counters to be uniquely identified across groups, so node level and thread level counters can be distinguished for example.

The regular expression counter parser configuration is loaded from a YAML configuration file. New counters or performance reporting tools can be integrated in the HPCProbe framework by updating this configuration information.

The `hpcprobe.py` module writes the experiment `ResultSet` in YAML format to a summary file. The `ResultSet` captures the experiment input parameters and the measured responses or objectives. The `hpcplot.py` module reads the required YAML files into a Python pandas `DataFrame` for analysis, prediction, and reporting.

### 5.1.3 Response Prediction

HPCProbe provides basic polynomial, B-spline, and deep learning predictive model implementations that extend a common *Model* base class. The results in Sections 6.2, 6.3 and 6.5 demonstrate that a small sample of measured data is sufficient to fit system- and workload-specific coefficients for the B-spline and deep learning models.

## PolyReg Model

The *PolyReg* basic polynomial regression model is implemented using the Ordinary Least Squares `sklearn.linear_model.LinearRegression` model provided in the Python scikit-learn library [83]. The polynomial degree is a key configuration parameter for this model. Increasing the polynomial degree allows the model to fit more complex responses. Using a polynomial degree of two will generate a quadratic fit, and a degree of three will generate a cubic fit, and so on.

Figure 5.2 shows that selecting the optimum polynomial degree is a balance between underfitting with lower order polynomials and overfitting with higher order polynomials.

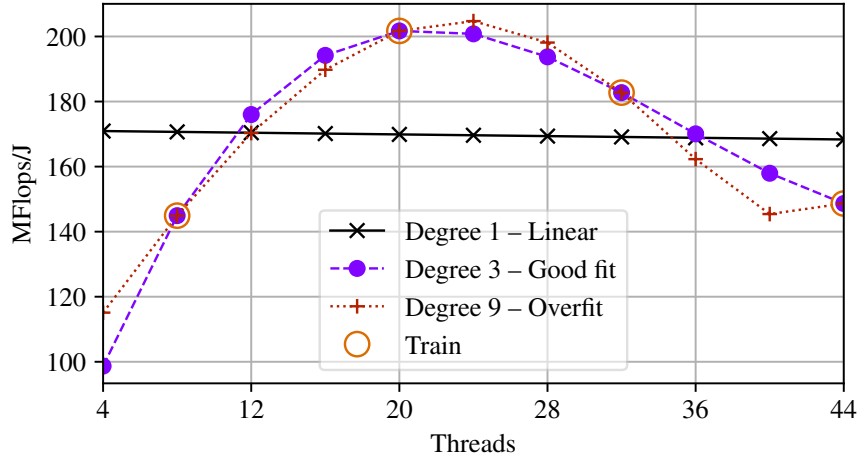


Figure 5.2: Model Fit by Polynomial Degree

The HPCModel package also provides a number of vectorised data transforms and scalers that can improve model performance. The implementation provides common transforms for mitigating normal data distribution deviations, including  $x' = \log_e x$ ,  $\log_2 x$ ,  $\log_{10} x$ ,  $e^x$ , and  $\sqrt{x}$ . The implemented data scalers include min-max normalisation, as equation 5.1 shows, and standardisation using the sample mean  $\bar{x}$  and standard deviation  $\sigma$ , as equation 5.2 shows. Feature scaling is a common requirement in machine learning algorithms.

The *PolyReg* model provides a baseline for demonstrating that more advanced modelling techniques are required to meet our training and accuracy requirements.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (5.1)$$

$$x' = \frac{x - \bar{x}}{\sigma} \quad (5.2)$$

## SplineReg Model

The `statsmodels.formula.api` module in the Python StatsModels library [84] allows regression models to be specified with plain-text Wilkinson notation [79] similar to the formula style used in the R programming language. This notation simplifies the process of specifying model parameters such as the B-spline degrees of freedom and polynomial degree for each model term, and interactions between terms. Model fit can be tuned using these parameters.

The HPCProbe implementation provides methods to auto generate formula, such as generating first and higher order term interactions, removing interactions that are not significant, and setting lower and upper bounds for terms. Basis spline terms are initialised with settings for the required polynomial degree and spline degrees of freedom. For a cubic B-spline the degrees of freedom and number of knots are equivalent.

The *SplineReg* B-spline model fits an Ordinary Least Squares model using the specified model formula and training data. This model consistently outperform basic polynomial models in our experiments.

## DnnReg Model

Many toolkits and frameworks are available to implement deep learning models. HPCModel uses Python scikit-learn [83] and TensorFlow [85]. TensorFlow classification and regression deep neural networks are implemented using the `DNNClassifier` and `DNNRegressor` classes. Classifier networks are used to predict classes, such as cat or dog image labelling by features of the image. Regressor networks are used to predict continuous values, such as house price by locality features. Our requirement is to predict continuous performance and energy usage values by system and workload features, so *DnnReg* uses the `DNNRegressor` class.

Parameters for instantiating a `DNNRegressor` set the topology of the neural network, including input units for each feature, hidden layers, and hidden units per hidden layer. Sample input data (features or predictors), output data (responses), and number of training steps (iterations for convergence) are then used to train the `DNNRegressor`.

Training neural networks typically involves some randomisation, such as training weights or sample ordering. This randomisation may result in prediction variations from one training run to the next. To ensure consistent, repeatable results from run to run, a `tf.estimator.RunConfig` is initialised using a constant random seed and assigned using the `DNNRegressor config` parameter.

Our model input and output parameters have very different scales and ranges, such as  $10^9$  Hz versus  $10^1$  threads or  $10^6$  MFlops/s versus  $10^2$  MFlops/J, so it is good practice to normalise data. Figure 5.3 shows model convergence for no scaling, min-max scaling, and standardisation scaling. It

shows model convergence improves significantly with standardisation scaling, where model inputs and outputs are scaled to have zero mean and unit variance.

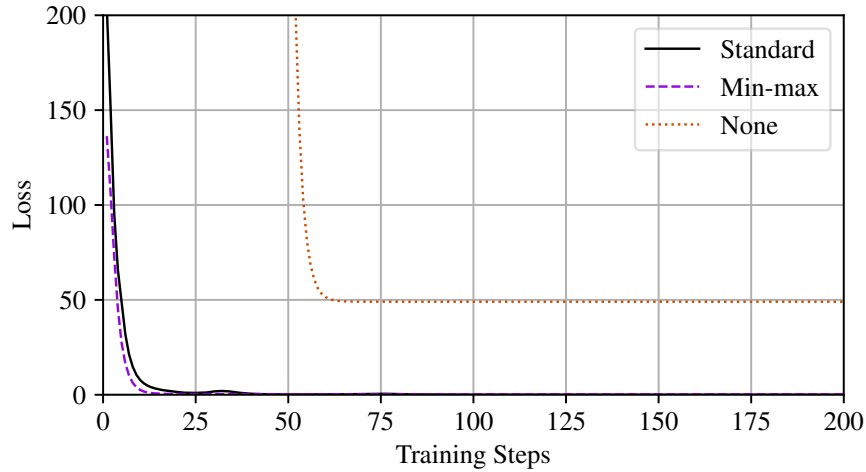


Figure 5.3: DNN Model Convergence by Data Scaling Method

#### 5.1.4 Trade-Off Analysis

As introduced in Chapter 4, a key innovation of this work is the introduction of a trade-off zone approach allowing for measurement and modelling error in the trade-off data. HPCProbe implements the calculations to determine the trade-off zone as follows:

1. Calculate the Pareto-optimal points forming the Pareto front.
2. Extend the front horizontally and vertically to set outer error limits.
3. Scale and translate the Pareto front to set inner error limits.
4. Extend the front to include points bounded by the trade-off zone.

The ParetoFront class in the HPCModel component provides methods that implement the required calculations.

##### Calculate Pareto Front

Exhaustive methods for identifying Pareto-optimal trade-off points are practical for HPCProbe because the search space size is limited [86] by minimising the number of tuning parameters and the tuning resolution needed. Exhaustive methods also avoid the risk of missing valid points that is associated with approximation methods.

The exhaustive method used in the *setFront* function starts with a multi-key sort by the data values for trade-off, worst-to-best for performance then best-to-worst for energy. The first point in the sorted data set is the best energy point, which becomes the first Pareto point. The algorithm then iterates

Listing 5.1: Pareto Set DataFrame

|    | ee         | ar           | fq      | tn | pe | nd |
|----|------------|--------------|---------|----|----|----|
| 62 | 210.024999 | 3.237124e+06 | 1900000 | 24 | 2  | 64 |
| 63 | 209.616948 | 3.284022e+06 | 2000000 | 24 | 2  | 64 |
| 64 | 207.011428 | 3.309843e+06 | 2100000 | 24 | 2  | 64 |
| 65 | 202.290166 | 3.314084e+06 | 2200000 | 24 | 2  | 64 |

through the rest of the data set adding any points with equal or better performance compared with the previously added Pareto point.

HPCProbe uses the DataFrame structure from the Python pandas library [87] for manipulating experiment data. DataFrame indexing allows data sorting and slicing without losing context, such as the tuning parameter settings for each Pareto point. Listing 5.1 shows the DataFrame representation of the stencil Pareto front seen in Figure 4.8. The data set index for each point is listed in the first column. The ee and ar columns have the energy efficiency and performances responses for each point. The fq, tn, pe, and nd columns are the frequency, thread count, process count, and node count settings for each point.

### Extend Pareto Front Horizontally and Vertically

HPCProbe then extends the horizontal and vertical limits of the Pareto front calculated by *setFront* using the *getLimits* algorithm, which is shown in two parts in Algorithms 5.1 and 5.2. Algorithm 5.1 shows the required variable initialisations, depending on whether optimal responses are maximised or minimised. For example,  $x$  and  $y$  need to be maximised if they are energy efficiency in MFlops/J and performance in MFlops/s. The outer  $x\_lim$  in this case is 1 – the  $x$  error limit (line 5), so any points with  $x$  values below the front that are within 95% of  $x$ -maximum are within error limits.

#### Algorithm 5.1 Get Pareto Front with Extended Limits – Part 1

```

1 function GETLIMITS
2    $px\_lim \leftarrow (pts_{ppts_n,2}, pts_{ppts_n,3})$  ▷ Get last/first front (x,y) points
3    $py\_lim \leftarrow (pts_{ppts_1,2}, pts_{ppts_1,3})$  ▷ as current end/start points
4   if  $max\_x = true$  then ▷ Get x limit when x is being maximised/minimised
5      $x\_lim \leftarrow 1 - MAX(lims_1)$ 
6   else
7      $x\_lim \leftarrow 1 + MAX(lims_1)$ 
8   end if
9   if  $max\_y = true$  then ▷ Get y limit when y is being maximised/minimised
10     $y\_lim \leftarrow 1 - MAX(lims_2)$ 
11  else
12     $y\_lim \leftarrow 1 + MAX(lims_2)$ 
13  end if
14   $pf1 \leftarrow null$  ▷ Index of new starting
15   $pf2 \leftarrow null$  ▷ and ending front points

```

Algorithm 5.2 shows that *getFront* then iterates through the data set searching for points within the

calculated limits,  $x\_lim$  and  $y\_lim$ , that extend the front horizontally and vertically (lines 16 to 36). For the MFlops/J and MFlops/s example, we want to extend the front vertically below  $x$ -maximum to create a new  $y$ -minimum that allows for  $x$  values within 95% of  $x$ -maximum (line 20). The front is extended horizontally to the left of  $y$ -maximum to create a new  $x$ -minimum in a similar manner (line 29).

---

**Algorithm 5.2** Get Pareto Front with Extended Limits – Part 2

---

```

16  for  $i \leftarrow 1$  to  $LEN(pts)$  do                                ▷ Find the index of points beyond either end of the front
17       $(x,y) \leftarrow (pts_{i,2}, pts_{i,3})$                             ▷ but still within limits
18      if  $max\_x = true$  then
19          if  $x < px\_lim_1$  and  $y > py\_lim_2 \times y\_lim$  and  $(pf1 = null \text{ or } pts_{pf1,1} > x)$  then
20               $pf1 \leftarrow i$                                 ▷ Found  $x$  that is less than current  $pts_{pf1,1}$  and within  $y$  limit
21          end if
22      else
23          if  $x > px\_lim_1$  and  $y < py\_lim_2 \times y\_lim$  and  $(pf1 = null \text{ or } pts_{pf1,1} < x)$  then
24               $pf1 \leftarrow i$                                 ▷ Found  $x$  that is greater than current  $pts_{pf1,1}$  and within  $y$  limit
25          end if
26      end if
27      if  $max\_y = true$  then
28          if  $x > px\_lim_1 \times x\_lim$  and  $y < py\_lim_2$  and  $(pf2 = null \text{ or } pts_{pf2,2} > x)$  then
29               $pf2 \leftarrow i$                                 ▷ Found  $y$  that is less than current  $pts_{pf2,2}$  and within  $x$  limit
30          end if
31      else
32          if  $x < px\_lim_1 \times x\_lim$  and  $y > py\_lim_2$  and  $(pf2 = null \text{ or } pts_{pf2,2} > x)$  then
33               $pf2 \leftarrow i$                                 ▷ Found  $y$  that is greater than current  $pts_{pf2,2}$  and within  $x$  limit
34          end if
35      end if
36  end for
37  for  $i \leftarrow 1$  to  $LEN(ppts\_lim)$  do                            ▷ Convert list of point indices to list of points
38       $front\_lim_i \leftarrow (pts_{ppts\_lim_i,2}, pts_{ppts\_lim_i,3})$ 
39  end for
40  if  $pf1 \neq null$  then                                            ▷ Found new  $x$  ending point for front
41       $front\_lim \leftarrow front\_lim \cup (pts_{pf1,2}, px\_lim_2)$         ▷ so add it to end of front
42  end if                                                            ▷ using previous  $y$  value
43  if  $pf2 \neq null$  then                                            ▷ Found new  $y$  starting point for front
44       $front\_lim \leftarrow (py\_lim_1, pts_{pf2,3}) \cup front\_lim$         ▷ so add it to start of front
45  end if                                                            ▷ using next  $x$  value
46  return  $front\_lim$ 
47 end function

```

---

A new list of Pareto points is initialised by copying the current list (lines 37 to 39). If a new vertical limit was found a new point is prepended to the list using the  $y$  value of the new point and the  $x$  value of the Pareto point that is currently first in the list (line 41). Similarly, if new horizontal limit was found it is append to the list (line 44).

Listing 5.2 shows the stencil Pareto front DataFrame with added starting and ending points that extend the front limits as plotted in Figure 4.9, step 3.

Listing 5.2: Extended Pareto Set DataFrame

|    | ee         | ar           | fq      | tn | pe | nd |
|----|------------|--------------|---------|----|----|----|
| 55 | 210.024999 | 2.696985e+06 | 1200000 | 24 | 2  | 64 |
| 62 | 210.024999 | 3.237124e+06 | 1900000 | 24 | 2  | 64 |
| 63 | 209.616948 | 3.284022e+06 | 2000000 | 24 | 2  | 64 |
| 64 | 207.011428 | 3.309843e+06 | 2100000 | 24 | 2  | 64 |
| 65 | 202.290166 | 3.314084e+06 | 2200000 | 24 | 2  | 64 |
| 87 | 182.741915 | 3.314084e+06 | 2200000 | 32 | 2  | 64 |

## Scale and Transform Pareto Front

The *scaleAndTransform* method sets the Pareto front inner and outer limits as Algorithm 5.3 shows. Method parameters include the required  $x$  and  $y$  offsets for the transformed front and the  $x$  and  $y$  scaling factors needed to fit within the offsets. Setting the inner limit starts by scaling each Pareto point by the required  $x\_scale$  and  $y\_scale$  (lines 3 and 5). For fronts that maximise both responses, the inner limit is scaled down to fit between a point  $y\_off$  below  $x$ -minimum of the front, and a point  $x\_off$  left of  $y$ -minimum.

---

### Algorithm 5.3 Scale and Transform Front to Set Required Inner/Outer Limit

---

```

1 function SCALEANDTRANSFORM( $x, y, x\_scale, y\_scale, x\_off, y\_off, inner$ )
2    $n \leftarrow \text{LEN}(x)$ 
3   for  $i \leftarrow 1$  to  $n$  do                                     ▷ Scale front  $x, y$  values by required amounts
4      $plim_i \leftarrow (x\_scale \times x_i, y\_scale \times y_i)$ 
5   end for
6   if  $inner = true$  then                                       ▷ Set indices needed to calculate front origin
7      $a, b, c, d \leftarrow 1, n, n, 1$ 
8      $plim \leftarrow \text{REVERSE}(plim)$                              ▷ Reverse inner limit list to close polygon
9   else
10     $a, b, c, d \leftarrow n, n, 1, 1$ 
11  end if
12  if  $max\_x = true$  then                                       ▷ Calculate  $x, y$  origin for required offset
13     $x\_origin \leftarrow x_a - plim_{b,1} - x\_off$ 
14  else
15     $x\_origin \leftarrow x_c - plim_{d,1} + x\_off$ 
16  end if
17  if  $max\_y = true$  then
18     $y\_origin \leftarrow y_c - plim_{d,2} - y\_off$ 
19  else
20     $y\_origin \leftarrow y_a - plim_{b,2} + y\_off$ 
21  end if
22  for  $i \leftarrow 1$  to  $n$  do                                     ▷ Transform front by required offset
23     $plim_{i,1} \leftarrow plim_{i,1} + x\_origin$ 
24     $plim_{i,2} \leftarrow plim_{i,2} + y\_origin$ 
25  end for
26  return  $plim$ 
27 end function

```

---

When setting the inner limit, points in the scaled front are reversed to provide a simple, closed polygon

that is not self-intersecting (line 8). Next, the required origin point for the scaled front is calculated. When  $x$  is maximised,  $x\_origin$  for the inner limit is the difference between the first and last  $x$  values of the unscaled and scaled fronts, less the required  $x\_off$  (line 13). The algorithm finishes by transforming the front by the required offset (lines 22 to 25).

### Extend Pareto Front using the Trade-Off Zone

The HPCProbe *getPath* and *getPathPts* methods integrate the all the described trade-off analysis steps to complete the implementation. The *getPath* method provides the list of polygon vertices that defines the trade-off zone, as Algorithm 5.4 shows.

First, the *getLimits* method shown in Algorithm 5.1 and 5.2 is called to extend the front horizontally and vertically (line 2). The required inner and outer scaling of  $x$  and  $y$  values that allows for the  $x$  and  $y$  margins of error is calculated (lines 11 and 18) using the calculated ranges and offsets (lines 7 to 10). Next, the *scaleAndTransform* method shown in Algorithm 5.3 is called to set the Pareto front outer limits (line 25) and inner limits (line 26). The *getPath* method returns a list of trade-off zone vertices that is the union of the inner and outer points lists.

---

#### Algorithm 5.4 Get Vertices of Polygon/Path Enclosing All Pareto Points Within Limits

---

```

1  function GETPATH
2       $(x,y) \leftarrow (\text{GETLIMITS})^T$                                 ▷ Transpose points within limits into  $x$  and  $y$  values
3      if  $x_0 < x_n$  then                                           ▷ check points are in ascending  $x$  order
4           $x \leftarrow \text{REVERSE}(x)$ 
5           $y \leftarrow \text{REVERSE}(y)$ 
6      end if
7       $\text{rng\_pfx} \leftarrow \text{MAX}(x) - \text{MIN}(x)$                             ▷ Calculate  $x$  and  $y$  ranges
8       $\text{rng\_pfy} \leftarrow \text{MAX}(y) - \text{MIN}(y)$ 
9       $x\_off\_in, x\_off\_out \leftarrow \text{lims}_{1,1} \times \text{MAX}(x), \text{lims}_{1,2} \times \text{MAX}(x)$     ▷ and inner and outer offsets
10      $y\_off\_in, y\_off\_out \leftarrow \text{lims}_{2,1} \times \text{MAX}(y), \text{lims}_{2,2} \times \text{MAX}(y)$     ▷ from front
11     if  $\text{rng\_pfx} > 0$  then                                         ▷ Calculate scaling for inner and outer fronts
12          $x\_scale\_in \leftarrow 1 - x\_off\_in / \text{rng\_pfx}$ 
13          $x\_scale\_out \leftarrow 1 + x\_off\_out / \text{rng\_pfx}$ 
14     else                                                         ▷ No  $x$  scaling if  $\text{rng\_pfx}$  is zero
15          $x\_scale\_in \leftarrow 1$ 
16          $x\_scale\_out \leftarrow 1$ 
17     end if
18     if  $\text{rng\_pfy} > 0$  then
19          $y\_scale\_in \leftarrow 1 - y\_off\_in / \text{rng\_pfy}$ 
20          $y\_scale\_out \leftarrow 1 + y\_off\_out / \text{rng\_pfy}$ 
21     else                                                         ▷ No  $y$  scaling if  $\text{rng\_pfy}$  is zero
22          $y\_scale\_in \leftarrow 1$ 
23          $y\_scale\_out \leftarrow 1$ 
24     end if
25      $\text{plim\_out} \leftarrow \text{SCALEANDTRANSFORM}(x,y,x\_scale\_out,y\_scale\_out,\text{plim\_out},\text{false})$ 
26      $\text{plim\_in} \leftarrow \text{SCALEANDTRANSFORM}(x,y,x\_scale\_in,y\_scale\_in,\text{plim\_in},\text{true})$ 
27     return  $\text{plim\_out} \cup \text{plim\_in}$ 
28 end function

```

---



Listing 5.3: Trade-Off Zone Pareto Set DataFrame

|    | ee         | ar           | fq      | tn | pe | nd |
|----|------------|--------------|---------|----|----|----|
| 48 | 200.287563 | 2.781450e+06 | 1600000 | 20 | 2  | 64 |
| 49 | 202.701038 | 2.867379e+06 | 1700000 | 20 | 2  | 64 |
| 50 | 204.961197 | 2.948861e+06 | 1800000 | 20 | 2  | 64 |
| 51 | 207.062290 | 3.025374e+06 | 1900000 | 20 | 2  | 64 |
| 52 | 207.807341 | 3.087693e+06 | 2000000 | 20 | 2  | 64 |
| 53 | 206.000412 | 3.126039e+06 | 2100000 | 20 | 2  | 64 |
| 54 | 201.707767 | 3.139495e+06 | 2200000 | 20 | 2  | 64 |
| 56 | 202.623305 | 2.788294e+06 | 1300000 | 24 | 2  | 64 |
| 57 | 204.287983 | 2.875768e+06 | 1400000 | 24 | 2  | 64 |
| 58 | 205.784968 | 2.958860e+06 | 1500000 | 24 | 2  | 64 |
| 59 | 207.110386 | 3.037038e+06 | 1600000 | 24 | 2  | 64 |
| 60 | 208.260791 | 3.109792e+06 | 1700000 | 24 | 2  | 64 |
| 61 | 209.233177 | 3.176638e+06 | 1800000 | 24 | 2  | 64 |
| 62 | 210.024999 | 3.237124e+06 | 1900000 | 24 | 2  | 64 |
| 63 | 209.616948 | 3.284022e+06 | 2000000 | 24 | 2  | 64 |
| 64 | 207.011428 | 3.309843e+06 | 2100000 | 24 | 2  | 64 |
| 65 | 202.290166 | 3.314084e+06 | 2200000 | 24 | 2  | 64 |
| 66 | 203.294764 | 2.837753e+06 | 1200000 | 28 | 2  | 64 |
| 67 | 203.718035 | 2.914736e+06 | 1300000 | 28 | 2  | 64 |
| 68 | 203.966412 | 2.986643e+06 | 1400000 | 28 | 2  | 64 |
| 69 | 204.039257 | 3.053001e+06 | 1500000 | 28 | 2  | 64 |
| 70 | 203.936380 | 3.113366e+06 | 1600000 | 28 | 2  | 64 |
| 71 | 203.658047 | 3.167327e+06 | 1700000 | 28 | 2  | 64 |
| 72 | 203.204976 | 3.214512e+06 | 1800000 | 28 | 2  | 64 |
| 73 | 202.578336 | 3.254595e+06 | 1900000 | 28 | 2  | 64 |
| 74 | 201.067652 | 3.283264e+06 | 2000000 | 28 | 2  | 64 |
| 75 | 197.991527 | 3.296168e+06 | 2100000 | 28 | 2  | 64 |
| 76 | 193.422145 | 3.293120e+06 | 2200000 | 28 | 2  | 64 |
| 77 | 201.437336 | 2.903674e+06 | 1200000 | 32 | 2  | 64 |
| 78 | 200.642754 | 2.966066e+06 | 1300000 | 32 | 2  | 64 |
| 79 | 199.683442 | 3.022587e+06 | 1400000 | 32 | 2  | 64 |
| 82 | 195.843044 | 3.153284e+06 | 1700000 | 32 | 2  | 64 |
| 83 | 194.252660 | 3.182887e+06 | 1800000 | 32 | 2  | 64 |
| 84 | 192.513354 | 3.205121e+06 | 1900000 | 32 | 2  | 64 |
| 85 | 190.204815 | 3.218855e+06 | 2000000 | 32 | 2  | 64 |
| 86 | 186.930691 | 3.223006e+06 | 2100000 | 32 | 2  | 64 |
| 87 | 182.741915 | 3.217534e+06 | 2200000 | 32 | 2  | 64 |

The *getPathPts* method (not shown) uses the trade-off zone vertices returned by *getPath* to construct a Polygon object using the Python *shapely.geometry.polygon* library [88]. The Pareto front is updated to include all points that intersect with the trade-off zone Polygon object.

Listing 5.3 shows the stencil Pareto front DataFrame with the addition of all points that intersect the trade-off zone as plotted in Figure 4.9, step 5. These points are statistically equivalent to points that lie directly on the Pareto front when error limits of the trade-off data are considered.

### 5.1.5 Results Reporting

The HPCPlot module provides Python APIs for generating analysis metrics and plots using HPCProbe experiment data, the HPCModel module, and the Python matplotlib library [89]. Experimental observation data is loaded from YAML files generated by the HPCProbe module. Model predictions are generating using the HPCModel module. Observed and predicted data sets are manipulated using pandas DataFrames.

The YAML experiment data files required for analysis can be specified as a list of YAML file names and loaded into a Plot DataFrame as follows.

```
>>> import hpcplot as hplt
>>> ymls = ['stencil-f-e-h-64-4.yml']
>>> pd = hplt.PlotDf(ymls, basePredictors=True)
```

YAML files for separate energy and performance runs are combined as a list of a list of file names.

Model predictions can be generated using the HPCModel module. Training and test data selection dictionaries need to be assigned first. Training data selection is set for uniform sampling as Figure 4.5 shows. Test data selection is set for predictions across the full thread and frequency range at 64 nodes:

```
>>> import numpy as np
>>> import hpcmodel as hmdl
>>> train = {'custom': {
...     'nd': [64] * 12,
...     'tn': [8, 20, 32, 44] * 3,
...     'fq': np.repeat([1200000, 1900000, 2200000], 4)}
... }
>>> test = {
...     'nd': [64],
...     'tn': range(4, 45, 4),
...     'fq': range(1200000, 2200001, 100000)
... }
```

The list of response selection and definition dictionaries is assigned next:

```
>>> responses = [
```

```
...     {'col': 'ee', 'type': 'Energy', 'unit': 'MFlops/J',
...       'scale': 1000000.0},
...     {'col': 'ar', 'type': 'Performance', 'unit': 'MFlops/s',
...       'scale': 1000000.0}
... ]
```

Assign the dictionary defining trade-off zone limits for each response and if it needs to be maximised or minimised:

```
>>> lims = {
...     'ee': {'obs': [0.05, 0.0], 'for': [0.05, 0.0], 'pareto_max': True},
...     'ar': {'obs': [0.05, 0.0], 'for': [0.05, 0.0], 'pareto_max': True}
... }
```

Then generate a B-spline model prediction using default model settings as follows:

```
>>> rm = hmdl.SplineReg(pd.df, responses, pd.predictors,
...     train=train, test=test, lims=lims)
```

Appendix A provides further examples of using the HPCPlot and HPCModel APIs. Appendix B provides descriptions of plot and model initialisation parameters.

## Results Visualisation

HPCProbe implements two and three dimensional plots for visualising trade-off results using the Python matplotlib library. A basic 2D Pareto plot with trade-off zone can be generated for observed energy efficiency versus performance as follows:

```
>>> hplt.ParetoPlot(pd.df, 'ee', 'ar', poly=True, lims=True).plot()
```

The Pareto plot with trade-off zone can be generated for the B-spline model predicted data as follows:

```
>>> hplt.ParetoPlot(rm.df_test, 'ee', 'ar', poly=True, lims=True).plot()
```

## Evaluation Metrics

The HPCProbe API populates model evaluation metrics in a Python dictionary. API users can use this dictionary as the keyword arguments list for the Python format function to output the statistics of interest in the format required. The following code snippet provides a simple example:

```
>>> stats = rm.getStats()
>>> print(stats)
{'r1Rsqr': 1.0, 'r1Rmse': 4.3, 'nobs': 12.0, ...}
>>> print('R squared is {r1Rsqr}, RMS Error is {r1Rmse}'.format(**stats))
R squared is 1.0, RMS Error is 4.3
```

Evaluation tables presented in Chapter 6 use this approach to generate the required tables in  $\text{\LaTeX}$  format.

### 5.1.6 Orchestration

The *Experiment* class in the `hpcprobe.py` module orchestrates the following activities to automate energy tuning experiments:

1. Job submission to the HPC cluster using PBS.
2. Search space iteration using Nimrod/O.
3. Job output data post processing using CrayPAT.

HPC users access the HPC system using an SSH (Secure Shell) client on their workstation to connect to the HPC login node. The `hpcprobe.py` module is configured and launched on the login node. The Experiment configuration options for `hpcprobe.py` are loaded from an initialisation file or using command line parameters. An interactive mode (non-batch) can be used for debugging the run configuration.

### Job Submission

HPCProbe generates a PBS script to allocate the required HPC cluster resources and run the program to be tuned. HPC resources include the required number of nodes, number of cores per node, and the job maximum wall time. The job is submitted using the `qsub` command as follows:

```
$ qsub <pbs-script>
```

Our Cray ALPS laboratory environment provides the `aprun` command to specify application resources and placement, and launch a program on the HPC cluster. Application resources include the number of nodes for the run, number of MPI processes or ranks per node, number of cores per rank, core placement, number of the Hyper-Threads per core, and CPU frequency. Programs instrumented using `pat_build` are launched using the `aprun` command as follows:

```
$ aprun <aprun-options> <program>+pat <prg-options>
```

As discussed in section 5.1.1, programs not instrumented with `pat_build` can use `pat_run` instrumentation. The `pat_run` command can be used with `aprun` as follows:

```
$ aprun <aprun-options> pat_run <program> <prg-options>
```

Applications and system tools also often read configuration information from environment variables. For example, the `HPCProbe env` option is used to specify required CrayPAT performance counter events using an environment variable.

When program execution completes, CrayPAT automatically collates experiment data from the across the HPC cluster and writes the results to an experiment folder. A single instrumented run can generate a large amount of experiment data (over 10MB per run) so a clustered file system should be selected for the experiment folder using the `HPCProbe outputFolder` option. Using a login node local file system can significantly extend the time required to collect results, and negatively impact login node performance for other users.

## Search Space Iteration

`HPCProbe` generates a `Nimrod/O` schedule file to specify the required parameter sweeps to complete an experiment, the objective function evaluator to use, and the required optimisation method (or methods). Parameter sweeps include the range of CPU frequencies and OpenMP thread counts required for the experiment, specified using the `HPCProbe cpuFrequency` and `threadCount` options. The objective function evaluator takes the input parameters, performs the required evaluation, then outputs the objective. This implementation of `HPCProbe` uses the exhaustive search optimisation method, but `Nimrod/O` incorporates a number of standard non-linear methods.

`Nimrod/O` searches are launched as follows:

```
$ nimrodo -f <schedule-file>
```

Nimrod/O invokes the objective function evaluator to evaluate each of the required data points in the search space. The Nimrod/O local dispatcher uses the `aprun` command to start the kernel or application being evaluated with the required configuration. The HPCProbe `expExec` option is used to specify the application start command. The objective result, the energy used for example, is written to file when the job completes. Single objective optimisation methods can be used with multiple objectives by combining results as a weighted sum.

Execution monitoring detects non-zero completion codes or invalid objective results, and aborts the run. Results caching and concurrency controls are also available. HPCProbe restricts the concurrent jobs to one to ensure nodes are exclusively allocated for each evaluation run.

### **Job Post-Processing**

The HPCProbe `postExec` option is used to configure post-processing activities needed when submitted job completes. The CrayPAT `pat_report` utility is used to query experiment data generated from the program run as follows:

```
$ pat_report <pat-options> <exp-data>
```

The data aggregation capabilities of `pat_report` are an important aid in the processing of the experiment data. For example, to sum `PM_ENERGY:NODE` counters across all processing elements (MPI ranks) used for the program run, use the `-s aggr_pe` option as follows:

```
$ pat_report -s aggr_pe_PM_ENERGY:NODE=sum <exp-data>
```

Job post-processing also links experiment data folders to job submissions to provide a complete audit trail for experiments.

### **Fault Tolerance**

HPCProbe provides several features to improve the fault tolerance or resilience of tuning experiments. The mean time between failures (MTBF) of a HPC system will increase as the number of nodes increase, so techniques that avoid the need to rerun a job from the start when there is a failure can minimise the resulting time and resource waste. Service levels provided by laboratory environments may also be lower than production environments. A failure rate of one job in 1,000 means around one in 10 experimental sweeps will benefit from fault tolerance.

Experiments can be configured with a retry count using the `retries` setting to reattempt an activity when there is a failure. The experiment can continue if the activity is completed within the specified

number of retries. If an experiment is aborted, due to a node failure for example, HPCProbe can be configured to restart the experiment from the point of failure once the problem is resolved using the `continue` option. HPCProbe can also be configured to rerun a job with updated post-processing options using the `reget` option. This allows previously collected experiment data to be reused when there was a problem with the post-processing results.

## 5.2 Implementation Summary

This chapter presented implementation details for the HPCProbe framework. Aspects of the implementation that enable the proposed approach to be comprehensively evaluated are a focus. These aspects include highly configurable or tunable model implementations for system response prediction, allowing the proposed model performance to be fully studied. Fault tolerance features of the framework are another important aspect that improves the resource management efficiency of resource intensive tuning activities.

Implementation details are also presented for tuning innovations that are specific to HPCProbe. These innovations include statistical and machine learning models that can be trained to predict the system responses that are needed to identify energy efficiency and performance trade-off options. Another key contribution is the algorithms that implement the proposed trade-off zone approach. These algorithms allow HPCProbe to consider errors that are inherent in real-world multi-objective optimisation scenarios.

The presented implementation provides a number of technological advances that aim to improve the process of tuning parallel applications on HPC systems. It also aims to provide other performance researchers and analysts with a framework for developing future tools and products that further benefit HPC users.

Parts of the following publication have been incorporated in Chapter 6.

- [2] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, “Statistical and machine learning models for optimizing energy in parallel applications,” *The International Journal of High Performance Computing Applications*, 2019.

Chapter 6 Publication Contributor Statement

| Contributor  | Statement of contribution | % |
|--|---------------------------|---|
| As per Chapter 4 Publication Contributor Statement |                           |   |



# Chapter 6

## Evaluation

This chapter presents an evaluation of the architecture proposed in Chapter 4 using a series of case studies, including parallel software kernels used within scientific applications, mini-applications used for system benchmarking, and a full-scale weather modelling application. The aim is to demonstrate:

- That the proposed models and sampling strategy can accurately predict Pareto optimal trade-off options for energy use and performance.
- The predicted Pareto front provides significant trade-off ranges for energy use and performance, providing trade-off ranges.
- That the proposed method is generalisable across a range of software kernels and scientific applications.

### 6.1 Study Overview

This section provides a summary of the platform, parameters and phases of the experimental study. Access and set up details for the study source code and data set archive are provided in Appendix C.

#### 6.1.1 Platform

Experiments are conducted on a Cray XC system equipped as Table 6.1 shows. Runs use up to 86 exclusively allocated 44-core nodes, or 3,784 cores in total.

#### 6.1.2 Study Parameters

Table 6.2 lists configuration parameters and associated ranges used in the study. These parameters generate a full factorial design with 484 combinations.

Table 6.1: System Specification

| Component              | Specification                         |
|------------------------|---------------------------------------|
| CPU model              | Intel Xeon CPU E5-2699 v4 (Broadwell) |
| CPU clock              | 2.2 GHz                               |
| Sockets (NUMA Nodes)   | 2 per compute node                    |
| Cores                  | 22 per socket                         |
| Last Level Cache (LLC) | 55 MB per socket                      |
| Main memory (DRAM)     | 64 GB per socket                      |
| Memory bandwidth       | 76.8 GB/s max                         |

Table 6.2: Experimental Parameters

| Parameter               | Configuration                      |
|-------------------------|------------------------------------|
| Compute Nodes           | 20, 42, 64 and 86                  |
| MPI Ranks per Node      | 2                                  |
| OpenMP Threads per Rank | 2 to 22 incrementing by 2          |
| OpenMP Threads per Core | 1                                  |
| CPU Frequency Cap       | 1.2 to 2.2 GHz incrementing by 0.1 |
| Total CPU Cores         | 3,784                              |

### 6.1.3 Experiments

Study experiments are conducted in two main phases:

1. Model design and evaluation using kernels.
2. Model evaluation using applications.

The kernels and applications use the hybrid MPI/OpenMP programming model. Experiments allocate one MPI Rank per CPU socket and one OpenMP thread per CPU core. OpenMP threads are distributed uniformly across the available sockets and nodes using the Scatter thread placement policy. For each experiment, the sample combinations are collected as Table 6.2 lists, for 3,267 total tests for three kernels and four applications.

A simple Linear Regression model provides a naive baseline for assessing the performance of the proposed Basis Spline and Neural Network models. The naive model uses polynomial degree of three, random sampling (Figure 4.5), and no transform for data skew mitigation (section 4.3.3).

## 6.2 Kernels Study

This section demonstrates that the models can accurately predict Pareto-optimal energy and performance trade-off options with low cost for several scientific kernels that focus on specific computational idioms. The Parallel Research Kernels (PRK) [70] provide a collection of programs that cover

common patterns of communication, computation, and synchronisation encountered in parallel HPC applications. The PRK come with performance metric reporting, which allows us to focus on power consumption attributes to identify energy optimisation opportunities. Of the four available hybrid MPI/OpenMP kernels in PRK, this evaluation focuses on the three kernels that are most memory-bandwidth limited:

- **Stencil:** a kernel that performs a data-parallel stencil operation to a two-dimensional array.
- **Transpose:** a kernel that stresses communication and memory bandwidth.
- **Nstream:** an embarrassingly parallel kernel that computes memory bandwidth.

The **Synch-p2p** kernel is not evaluated as it performs a point-to-point synchronisation that is impacted by communication latency [70].

### 6.2.1 Stencil Kernel

The model response terms for stencil are energy efficiency in MFlops/J and performance in MFlops/s. The stencil radius is set to 2, grid size is 400k, and iterations are set to ensure that run time is at least 10 times the measurement sample rate.

The uniform, random, and Latin hypercube sampling methods from Section 4.3.2 are evaluated first using RMS error and  $R^2$  statistics, as described in Section 4.3.5. Uniform sampling results in the best fit with the fewest observations across the test cases. The method requires 12 samples or 10% of the search space for model training (4 core count  $\times$  3 frequency samples). The core count and frequency cap samples for uniform sampling are 8, 20, 32, and 44 cores, and 1.2, 1.9, and 2.2 GHz respectively. The model is then evaluated at node counts of 20, 42, 64 and 86, for a data set size of 484 samples.

Figure 6.1 includes error bars for the 95%  $t$ -distribution confidence interval for the mean of five samples. The error margins of about 5% consist of both measurement error and variability between runs caused by operating system jitter. A tool that uses the proposed models would calibrate measurement confidence intervals for each response variable. This eliminates the need for sample repetitions for each training sample to calculate their confidence intervals. The user would be alerted if the significance of model results is not within set limits.

Figure 6.2 shows the observed and model predicted Pareto fronts for stencil on 64 nodes. Points off the front are not Pareto optimal as points on the front always provide an improvement in one parameter with less impact on the other.

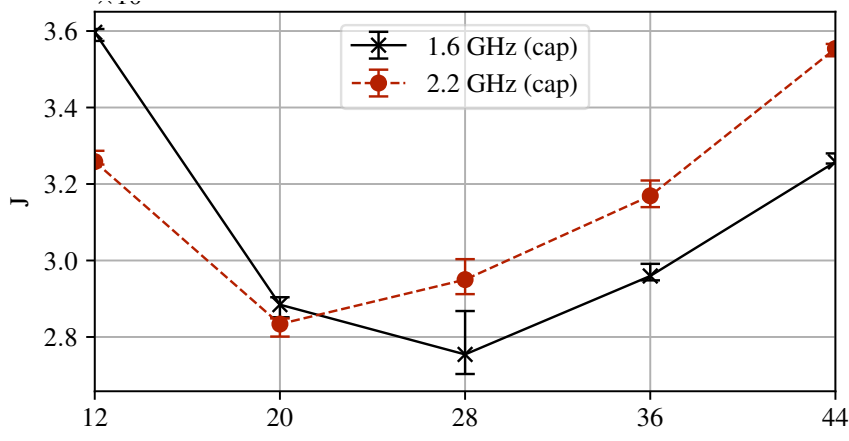


Figure 6.1: Stencil Measured Energy Efficiency

The interaction between thread count and frequency determines the shape of the Pareto front. Figure 6.2 shows that data points are grouped by thread count, and rotate as frequency varies. This rotation defines the shape of the Pareto front, which sets the energy versus performance trade-off ranges.

Table 6.3 shows the RMS Error between the observed and Basis Spline (BS) predicted values, the observed and Neural Network (NN) predicted values, and the observed and naive Linear Regression (LR) predicted values. The Total RMS Error between all 121 observed and predicted values is given by  $T_{RMSE}$ . The Pareto RMS Error between predicted Pareto points and observed values is given by  $P_{RMSE}$ .

The Pareto front section in Table 6.3 shows the error limit used to determine the measured and predicted Pareto front trade-off zones is 5.0%. It also shows measured and predicted Pareto point counts, and Pareto points grouped by search step distance. Coincident Pareto points have distance 0, non-coincident points within one search step (4 threads or 0.1 GHz) have distance 1, and so on. One search step is 9% in both dimensions of the  $11 \times 11$  search space.

Table 6.3 also shows *Baseline* performance and energy efficiency at maximum cores and threads, and their minima and maxima along the Pareto front,  $P_{min}$  and  $P_{max}$ .

The modelling methods are highly accurate for the Pareto front (for the BS models,  $P_{RMSE}$  is 2.2% for energy and 2.3% for performance and, for the NN models, it is 3.5% for energy and 3.1% for performance, and errors for the Pareto front are lower than overall error). Predicted and observed Pareto point counts are similar (31 observed, 37 BS predicted, and 25 NN predicted), and all non-coincident points are within one search step. Predicted and observed efficiency and performance gains are also consistent. Thus, the models can accurately guide this trade-off.

The surfaces in Figure 6.3 and 6.4 represent observed and predicted energy efficiency and perfor-

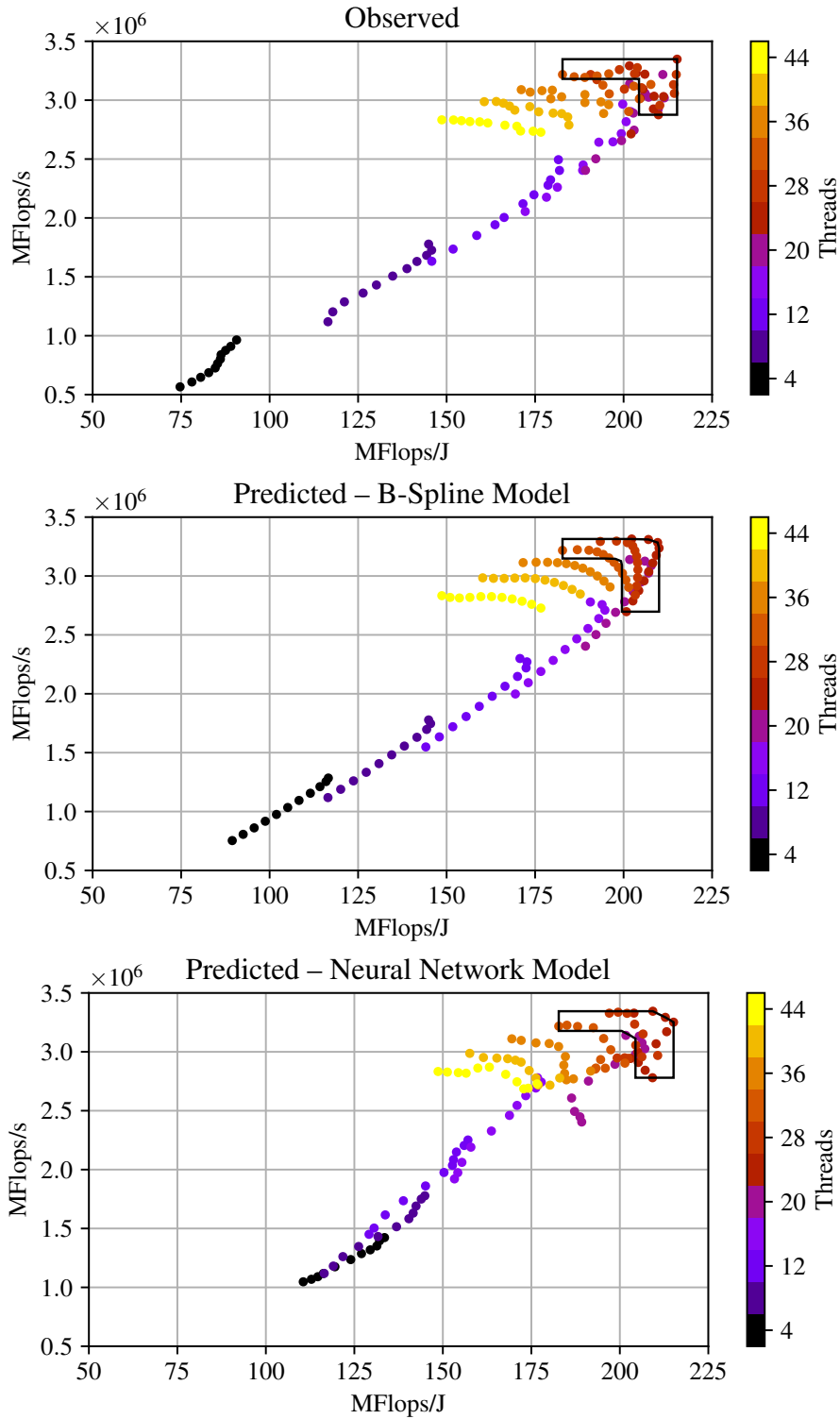


Figure 6.2: Stencil Pareto Front – Observed and Predicted

mance across the CPU frequency and thread count search space. Pareto points are plotted on these surfaces to show their context in the search space. The Pareto RMS Error  $P_{RMSE}$  provides a measure of the error between the between predicted and observed values.

Figure 6.4 shows that performance increases and then levels off as frequency and core count increase. The leveling-off marks the start of the trade-off zone, where energy efficiency starts to drop significantly, as Figure 6.3 shows, due to resource contention. Since energy efficiency and performance

Table 6.3: Stencil Results Summary

| Model fit             |       | Num obs  |                  | R <sup>2</sup>   | T <sub>RMSE</sub> |   | P <sub>RMSE</sub> |   |
|-----------------------|-------|----------|------------------|------------------|-------------------|---|-------------------|---|
| Energy                | BS    | 12       | 12               | 1.0              | 4.3               |   | 2.2               |   |
|                       | NN    | 12       | 12               | 1.0              | 9.2               |   | 3.5               |   |
|                       | LR    | 12       | 12               | 0.984            | 7.0               |   | 8.4               |   |
| Performance           | BS    | 12       | 12               | 1.0              | 4.2               |   | 2.3               |   |
|                       | NN    | 12       | 12               | 1.0              | 6.8               |   | 3.1               |   |
|                       | LR    | 12       | 12               | 0.991            | 5.5               |   | 7.4               |   |
| Pareto front          |       | Limit    | Points           | Dist: 0          | 1                 | 2 | 3                 | 4 |
| Observed              |       | 5.0      | 31               | 31               | 0                 | 0 | 0                 | 0 |
| BS forecast           |       | 5.0      | 37               | 31               | 6                 | 0 | 0                 | 0 |
| NN forecast           |       | 5.0      | 25               | 22               | 3                 | 0 | 0                 | 0 |
| LR forecast           |       | 5.0      | 26               | 17               | 8                 | 1 | 0                 | 0 |
| Energy                |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed              |       | 149M     | 183M             | 215M             | 22.9 to 44.7      |   |                   |   |
| Forecast<br>(Flops/J) | BS    | 149M     | 183M             | 210M             | 22.9 to 41.3      |   |                   |   |
|                       | Error | 0.0      | 0.0              | -2.4             | %                 |   |                   |   |
|                       | NN    | 149M     | 183M             | 215M             | 22.9 to 44.8      |   |                   |   |
|                       | Error | 0.0      | 0.0              | 0.0              | %                 |   |                   |   |
|                       | LR    | 100M     | 194M             | 222M             | 93.8 to 121.3     |   |                   |   |
|                       | Error | -32.6    | 6.2              | 3.0              | %                 |   |                   |   |
| Performance           |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed              |       | 2.83T    | 2.88T            | 3.35T            | 1.5 to 18.2       |   |                   |   |
| Forecast<br>(Flops/s) | BS    | 2.83T    | 2.70T            | 3.31T            | -4.8 to 17.0      |   |                   |   |
|                       | Error | 0.0      | -6.2             | -1.0             | %                 |   |                   |   |
|                       | NN    | 2.83T    | 2.78T            | 3.34T            | -1.9 to 18.1      |   |                   |   |
|                       | Error | 0.0      | -3.3             | -0.1             | %                 |   |                   |   |
|                       | LR    | 2.20T    | 3.09T            | 3.47T            | 40.6 to 57.8      |   |                   |   |
|                       | Error | -22.4    | 7.5              | 3.7              | %                 |   |                   |   |

diverge, tuning can increase energy efficiency significantly over a strategy that only minimises run time.

The smoothing effect of basis splines that is apparent in B-Spline predicted surfaces leads to a small increase in the predicted over the observed Pareto point count. For the Neural Network models, some slight of overfitting of the training data, as seen, for example, at lower thread counts, or in the 1.4 to 1.7 GHz range, produces a small decrease in the Pareto point count.

### 6.2.2 Transpose Kernel

The transpose model response terms are energy efficiency in MB/J and performance in MB/s. The transpose matrix order is set to 200k, blocking/tiling is disabled, and iterations are set to ensure run time is at least 10 times the measurement sample rate. Figure 6.5 shows the experimentally observed

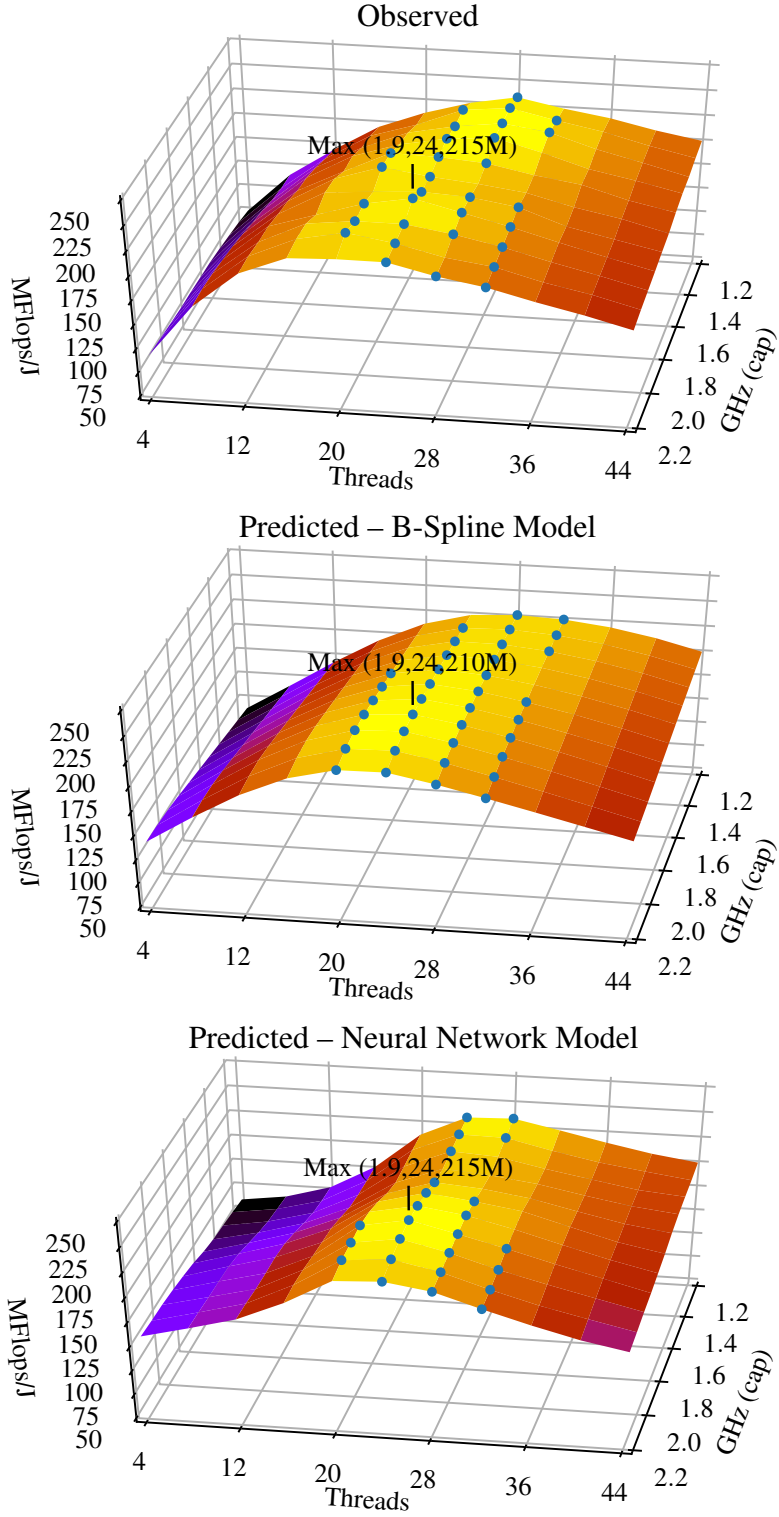


Figure 6.3: Stencil Energy Efficiency – Observed and Predicted

and model predicted Pareto fronts for 64 nodes.

Table 6.4 shows that the proposed methods are again highly accurate for the Pareto front (for the BS models,  $P_{RMSE}$  is 2.1% for energy and 1.4% for performance and, for the NN models, it is 3.0% for energy and 2.1% for performance). The Pareto front section shows similar observed and predicted Pareto point counts, with most of the non-conincident points within one search step of an overlapping

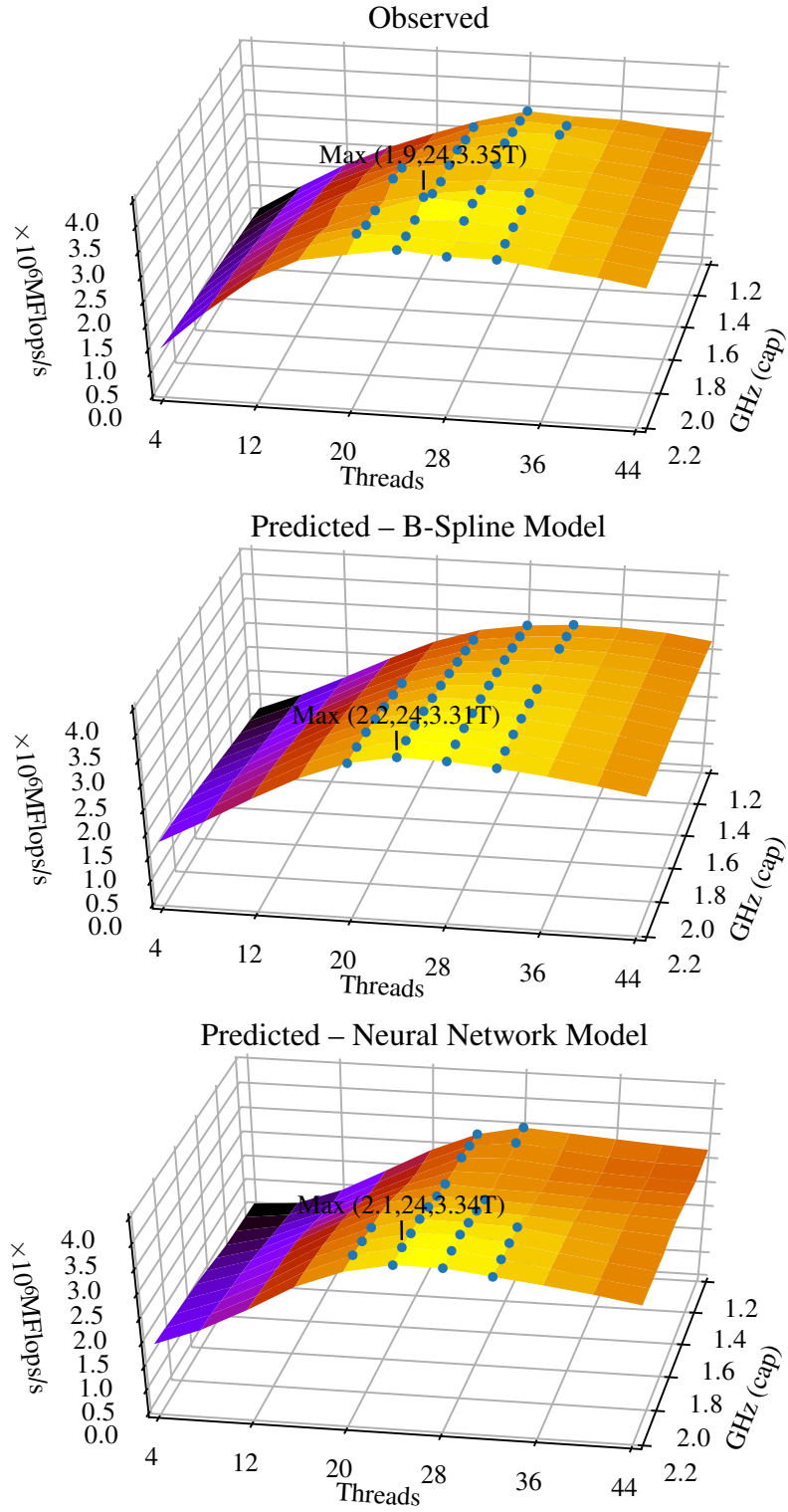


Figure 6.4: Stencil Performance – Observed and Predicted

point. The observed and predicted energy efficiency gains are similar (around 40%). The gain requires a trade-off in performance of 20%. The observed and predicted Pareto fronts agree that core and frequency tuning can increase energy efficiency by up to 40% in exchange for up to 20% performance loss.



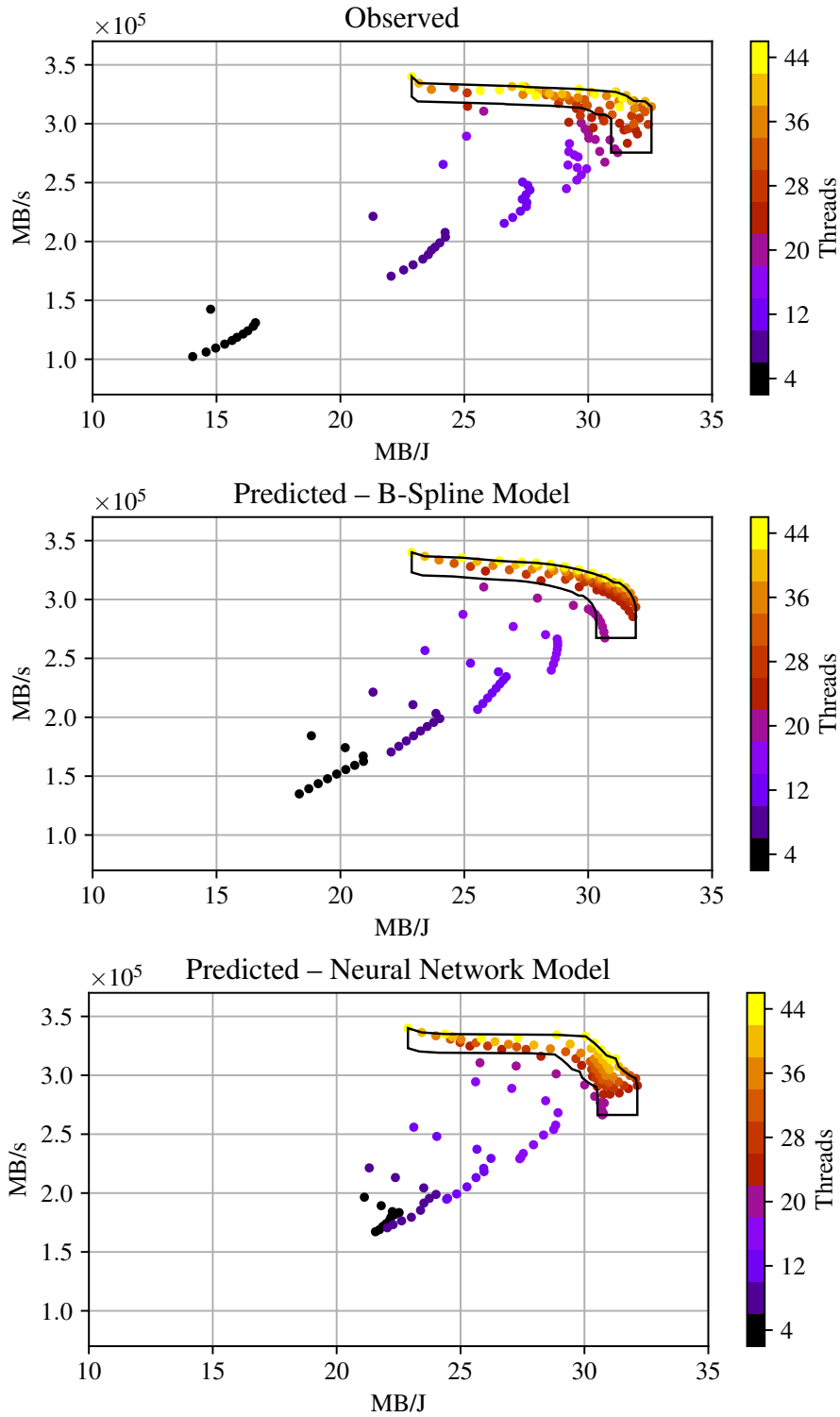


Figure 6.5: Transpose Pareto Front – Observed and Predicted

### 6.2.3 Nstream Kernel

As with transpose, the nstream model responses are efficiency in MB/J and performance in MB/s. The nstream vector length is set to 40G. Figure 6.6 shows the experimentally observed and model predicted Pareto fronts for 64 nodes.

Table 6.5 shows that, for the BS models, the Pareto front RMS Error  $P_{RMSE}$  is 3.6% for energy and

Table 6.4: Transpose Results Summary

| Model fit         |       | Num obs  |                  | R <sup>2</sup>   | T <sub>RMSE</sub> |   | P <sub>RMSE</sub> |   |
|-------------------|-------|----------|------------------|------------------|-------------------|---|-------------------|---|
| Energy            | BS    | 12       | 1.0              | 5.2              | 2.1               |   |                   |   |
|                   | NN    | 12       | 1.0              | 8.1              | 3.0               |   |                   |   |
|                   | LR    | 12       | 0.955            | 7.2              | 7.4               |   |                   |   |
| Performance       | BS    | 12       | 1.0              | 4.3              | 1.4               |   |                   |   |
|                   | NN    | 12       | 1.0              | 7.4              | 2.1               |   |                   |   |
|                   | LR    | 12       | 0.994            | 4.7              | 4.0               |   |                   |   |
| Pareto front      |       | Limit    | Points           | Dist: 0          | 1                 | 2 | 3                 | 4 |
| Observed          |       | 5.0      | 59               | 59               | 0                 | 0 | 0                 | 0 |
| BS forecast       |       | 5.0      | 71               | 59               | 10                | 2 | 0                 | 0 |
| NN forecast       |       | 5.0      | 71               | 59               | 9                 | 3 | 0                 | 0 |
| LR forecast       |       | 5.0      | 47               | 44               | 3                 | 0 | 0                 | 0 |
| Energy            |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed          |       | 22.9M    | 22.9M            | 32.6M            | 0.0 to 42.3       |   |                   |   |
| Forecast<br>(B/J) | BS    | 22.9M    | 22.9M            | 31.9M            | 0.0 to 39.5       |   |                   |   |
|                   | Error | 0.0      | 0.0              | -1.9             | %                 |   |                   |   |
|                   | NN    | 22.9M    | 22.9M            | 32.1M            | 0.0 to 40.5       |   |                   |   |
|                   | Error | 0.0      | 0.0              | -1.3             | %                 |   |                   |   |
|                   | LR    | 18.7M    | 24.1M            | 33.7M            | 28.6 to 80.1      |   |                   |   |
|                   | Error | -18.2    | 5.2              | 3.6              | %                 |   |                   |   |
| Performance       |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed          |       | 340G     | 275G             | 340G             | -19.0 to 0.0      |   |                   |   |
| Forecast<br>(B/s) | BS    | 340G     | 267G             | 340G             | -21.4 to 0.0      |   |                   |   |
|                   | Error | 0.0      | -2.9             | 0.0              | %                 |   |                   |   |
|                   | NN    | 340G     | 266G             | 340G             | -21.7 to 0.0      |   |                   |   |
|                   | Error | 0.0      | -3.3             | 0.0              | %                 |   |                   |   |
|                   | LR    | 296G     | 285G             | 341G             | -3.7 to 15.1      |   |                   |   |
|                   | Error | -12.9    | 3.5              | 0.2              | %                 |   |                   |   |

3.5% for performance and, for the NN models, it is 2.9% for energy and 3.5% for performance. The Pareto Front section shows similar observed and predicted Pareto point counts, with the majority of non-coincident points within one search step. The observed and predicted energy efficiency gains are again similar (around 40%). The performance impact is under 10%, which is close to measurement error limits. The observed and model predicted Pareto fronts provide consistent views that core and frequency tuning can increase energy efficiency up to 40% with little performance impact.

## 6.3 Applications Study

This section demonstrates that the models can accurately predict Pareto-optimal energy and performance trade-off options with low cost for more complex workloads. This evaluation applies the proposed energy modelling method to:

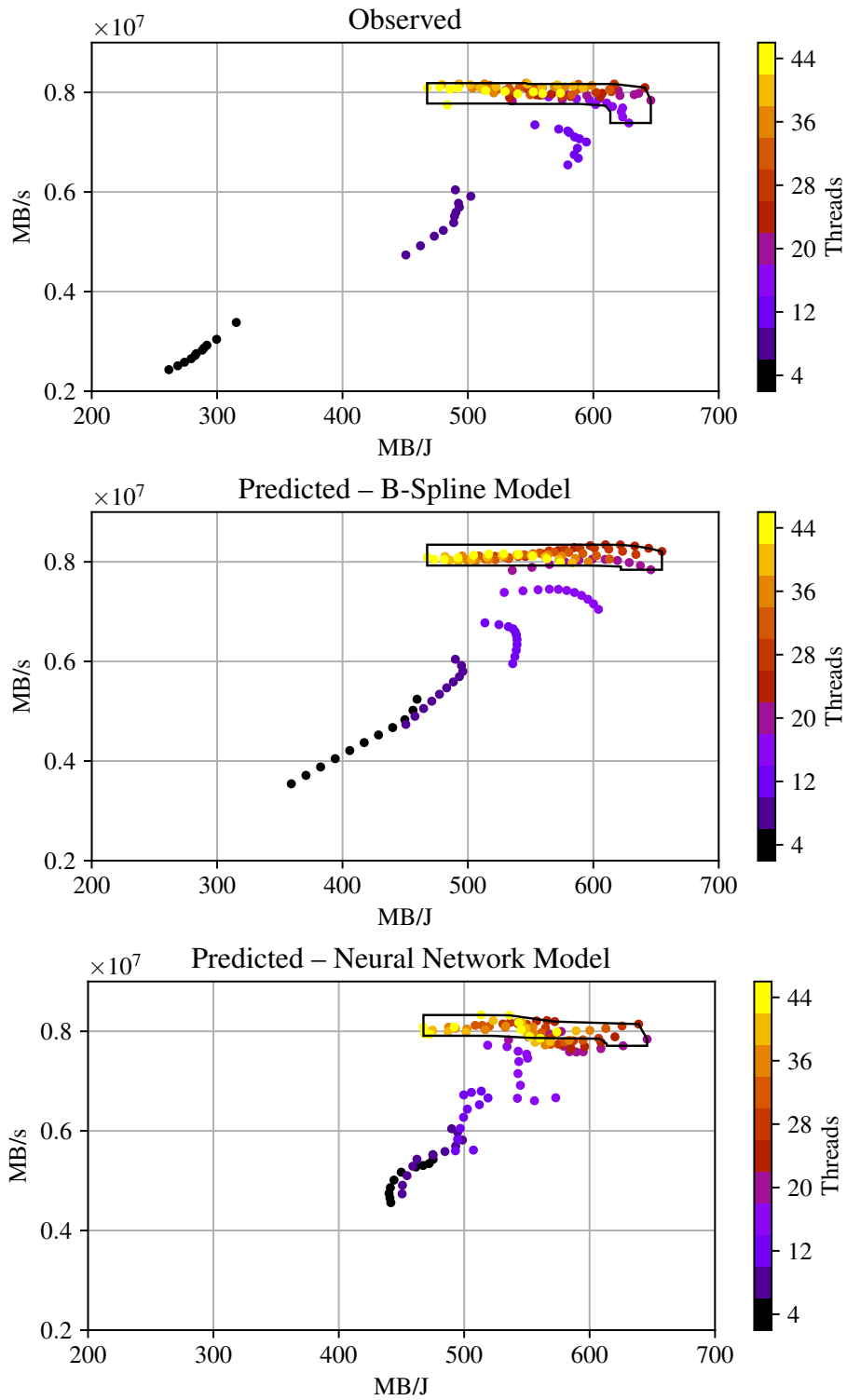


Figure 6.6: Nstream Pareto Front – Observed and Predicted

- **AMG:** a parallel algebraic multi-grid solver for linear systems [90].
- **LAMMPS:** a classical molecular dynamics simulator [91].
- **LULESH:** a mini-application for hydrodynamics modelling [92].
- **WRF:** an application suite used for mesoscale numerical weather forecasting [93].

Table 6.5: Nstream Results Summary

| Model fit         |       | Num obs  |                  | R <sup>2</sup>   | T <sub>RMSE</sub> |   | P <sub>RMSE</sub> |   |
|-------------------|-------|----------|------------------|------------------|-------------------|---|-------------------|---|
| Energy            | BS    | 12       |                  | 1.0              | 8.2               |   | 3.6               |   |
|                   | NN    | 12       |                  | 1.0              | 11.3              |   | 2.9               |   |
|                   | LR    | 12       | 0.967            |                  | 11.4              |   | 7.8               |   |
| Performance       | BS    | 12       |                  | 1.0              | 7.3               |   | 3.5               |   |
|                   | NN    | 12       |                  | 1.0              | 10.3              |   | 3.5               |   |
|                   | LR    | 12       | 0.982            |                  | 9.2               |   | 6.8               |   |
| Pareto front      |       | Limit    | Points           | Dist: 0          | 1                 | 2 | 3                 | 4 |
| Observed          |       | 5.0      | 87               | 74               | 11                | 2 | 0                 | 0 |
| BS forecast       |       | 5.0      | 75               | 74               | 1                 | 0 | 0                 | 0 |
| NN forecast       |       | 5.0      | 54               | 53               | 1                 | 0 | 0                 | 0 |
| LR forecast       |       | 5.0      | 42               | 42               | 0                 | 0 | 0                 | 0 |
| Energy            |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed          |       | 467M     | 467M             | 646M             | 0.0 to 38.2       |   |                   |   |
| Forecast<br>(B/J) | BS    | 467M     | 467M             | 655M             | 0.0 to 40.0       |   |                   |   |
|                   | Error | 0.0      | 0.0              | 1.4              | %                 |   |                   |   |
|                   | NN    | 467M     | 467M             | 646M             | 0.0 to 38.2       |   |                   |   |
|                   | Error | 0.0      | 0.0              | 0.0              | %                 |   |                   |   |
|                   | LR    | 361M     | 539M             | 660M             | 49.3 to 83.1      |   |                   |   |
|                   | Error | -22.8    | 15.2             | 2.3              | %                 |   |                   |   |
| Performance       |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed          |       | 8.09T    | 7.39T            | 8.19T            | -8.7 to 1.2       |   |                   |   |
| Forecast<br>(B/s) | BS    | 8.09T    | 7.84T            | 8.34T            | -3.1 to 3.1       |   |                   |   |
|                   | Error | 0.0      | 6.2              | 1.9              | %                 |   |                   |   |
|                   | NN    | 8.09T    | 7.71T            | 8.33T            | -4.7 to 2.9       |   |                   |   |
|                   | Error | 0.0      | 4.4              | 1.7              | %                 |   |                   |   |
|                   | LR    | 6.66T    | 7.42T            | 8.88T            | 11.5 to 33.4      |   |                   |   |
|                   | Error | -17.7    | 0.5              | 8.5              | %                 |   |                   |   |

### 6.3.1 AMG Application

AMG, which is well-known for its main memory bandwidth demands, contains microkernels that resemble the PRK kernels studied in Section 6.2. AMG first performs compressed sparse row (CSR) matrix vector multiplication. Optimising the coarsening process then includes matrix *transpose*. Finally, as the core of AMG, the algebraic multi-grid mesh relaxation process uses a 27-point *stencil*.

The AMG reported performance metric is Solve Figure of Merit (FOM) which is a measure of computations per second. An Energy FOM can be derived using the Solve FOM, total time and total energy, such that  $EFOM = FOM \times \text{time} / \text{energy}$ . In contrast to the fixed problem sizes of the PRKs, the AMG problem size scales as the processor topology scales (that is, weak scaling).

The node, MPI process, and OpenMP thread counts must fit within AMG processor topology restric-

tions. Thus, 20, 40, 60, and 80 nodes retain the same rank and thread count settings used for the PRK. The AMG problem size per processor is 256, which results in wall clock times of at least 30 seconds for each measurement.

### Single Node Count Predictions

Figure 6.7 shows the observed and predicted AMG Pareto fronts for 60 nodes. Table 6.6 shows that the models perform similarly with AMG, as seen with the kernels in Section 6.2. For the BS models,  $P_{RMSE}$  is 3.2% for energy and 3.8% for performance and, for the NN models, it is 3.9% for energy and 4.6% for performance. The observed and predicted energy efficiency improvements are similar (around 45%) and incur about a 15% performance drop. The observed and model predicted Pareto fronts provide consistent views that core and frequency tuning can reduce energy usage up to 45% but at up to 15% performance loss.

Table 6.6: AMG Results Summary

| Model fit          |       | Num obs  |                  | R <sup>2</sup>   | T <sub>RMSE</sub> |   | P <sub>RMSE</sub> |   |
|--------------------|-------|----------|------------------|------------------|-------------------|---|-------------------|---|
| Energy             | BS    | 12       |                  | 1.0              | 5.3               |   | 3.2               |   |
|                    | NN    | 12       |                  | 1.0              | 9.3               |   | 3.9               |   |
|                    | LR    | 12       | 0.959            |                  | 8.8               |   | 10.6              |   |
| Performance        | BS    | 12       |                  | 1.0              | 5.0               |   | 3.8               |   |
|                    | NN    | 12       |                  | 1.0              | 8.8               |   | 4.6               |   |
|                    | LR    | 12       | 0.988            |                  | 6.5               |   | 8.8               |   |
| Pareto front       |       | Limit    | Points           | Dist: 0          | 1                 | 2 | 3                 | 4 |
| Observed           |       | 5.0      | 75               | 66               | 9                 | 0 | 0                 | 0 |
| BS forecast        |       | 5.0      | 66               | 66               | 0                 | 0 | 0                 | 0 |
| NN forecast        |       | 5.0      | 48               | 48               | 0                 | 0 | 0                 | 0 |
| LR forecast        |       | 5.0      | 36               | 36               | 0                 | 0 | 0                 | 0 |
| Energy             |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed           |       | 3.40T    | 3.40T            | 4.89T            | 0.0 to 43.8       |   |                   |   |
| Forecast<br>(EFOM) | BS    | 3.40T    | 3.40T            | 4.92T            | 0.0 to 45.0       |   |                   |   |
|                    | Error | 0.0      | 0.0              | 0.8              | %                 |   |                   |   |
|                    | NN    | 3.40T    | 3.40T            | 5.01T            | 0.0 to 47.5       |   |                   |   |
|                    | Error | 0.0      | 0.0              | 2.6              | %                 |   |                   |   |
|                    | LR    | 2.45T    | 4.06T            | 4.92T            | 65.5 to 100.4     |   |                   |   |
|                    | Error | -27.8    | 19.5             | 0.7              | %                 |   |                   |   |
| Performance        |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed           |       | 56.2P    | 47.6P            | 56.7P            | -15.3 to 0.9      |   |                   |   |
| Forecast<br>(FOM)  | BS    | 56.2P    | 49.4P            | 57.1P            | -12.2 to 1.6      |   |                   |   |
|                    | Error | 0.0      | 3.7              | 0.7              | %                 |   |                   |   |
|                    | NN    | 56.2P    | 47.4P            | 57.9P            | -15.6 to 3.0      |   |                   |   |
|                    | Error | 0.0      | -0.4             | 2.1              | %                 |   |                   |   |
|                    | LR    | 46.6P    | 48.2P            | 60.7P            | 3.5 to 30.2       |   |                   |   |
|                    | Error | -17.1    | 1.4              | 6.9              | %                 |   |                   |   |

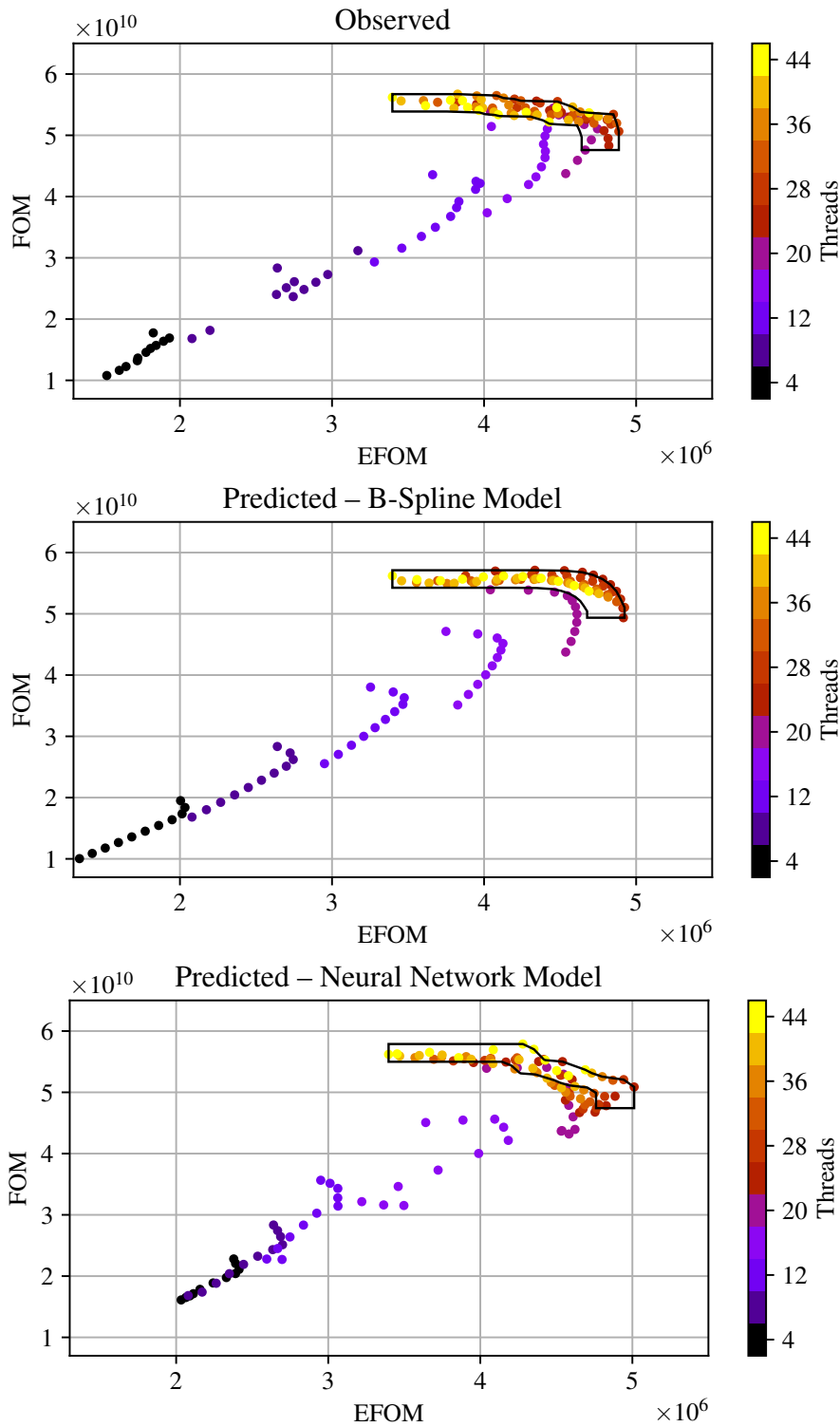


Figure 6.7: AMG Pareto Front – Observed and Predicted

### Multiple Node Count Predictions

The models can predict trade-off options for node counts for which we do not have training data. Figure 6.8 shows the observed and predicted Pareto fronts for AMG for 48 nodes (thick outline), and across 40, 48 and 60 nodes (thin outline).

The model training data now includes measurements across multiple node counts in the search space.

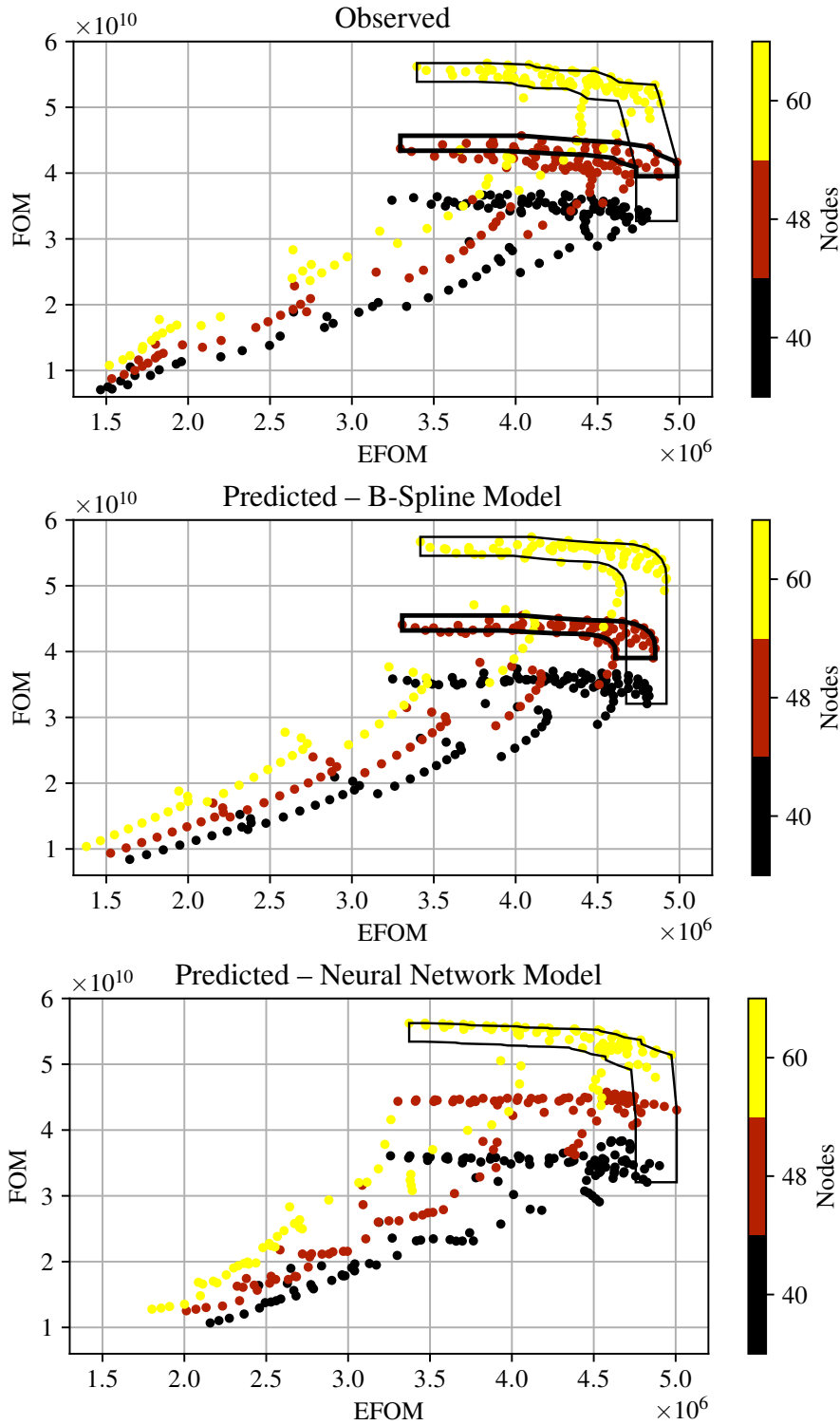


Figure 6.8: AMG Pareto Front for 40, 48 and 60 Nodes – Observed and Predicted

The previous examples use 12 samples at the required node count. This case uses 12 samples at 20, 40, 60, and 80 nodes, or 48 samples in total. The extra samples significantly expand the search space coverage of the model. Trade-off predictions at intervening node counts can now be made with similar accuracy.

The runs take 12 hours and 54 minutes to collect the complete 484 AMG test samples, which corresponds to a mean time to collect each sample of 1 minute and 35 seconds. The time to collect 48

training samples is 1 hour and 14 minutes, or 9.6% of the time to collect the full 484 samples. The process can also be speeded up by collecting sample data in parallel. For example, our 86 node cluster allows 20 and 40 node runs to be allocated in parallel.

Table 6.7 shows that, for the BS models,  $P_{RMSE}$  is 1.4% for energy and 1.6% for performance model and, for the NN models, it is 8.1% for energy and 9.0% for performance. The results show that tuning can reduce energy usage up to 45% but at around 10% performance loss.

Table 6.7: AMG Results Summary for 48 Nodes

| Model fit          |          | Num obs          |                  | R <sup>2</sup> | T <sub>RMSE</sub> |   | P <sub>RMSE</sub> |  |
|--------------------|----------|------------------|------------------|----------------|-------------------|---|-------------------|--|
| Energy             | BS       | 48               | 0.990            | 4.7            | 1.4               |   |                   |  |
|                    | NN       | 48               | 1.0              | 10.5           | 8.1               |   |                   |  |
|                    | LR       | 48               | 0.951            | 10.1           | 12.0              |   |                   |  |
| Performance        | BS       | 48               | 0.998            | 4.9            | 1.6               |   |                   |  |
|                    | NN       | 48               | 1.0              | 9.5            | 9.0               |   |                   |  |
|                    | LR       | 48               | 0.989            | 7.4            | 8.1               |   |                   |  |
| Pareto front       | Limit    | Points           | Dist: 0          | 1              | 2                 | 3 | 4                 |  |
| Observed           | 5.0      | 36               | 29               | 6              | 1                 | 0 | 0                 |  |
| BS forecast        | 5.0      | 60               | 29               | 15             | 9                 | 5 | 2                 |  |
| NN forecast        | 5.0      | 64               | 27               | 19             | 9                 | 6 | 3                 |  |
| LR forecast        | 5.0      | 36               | 17               | 12             | 5                 | 2 | 0                 |  |
| Energy             | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %        |                   |   |                   |  |
| Observed           | 3.30T    | 3.30T            | 4.99T            | 0.0 to 51.3    |                   |   |                   |  |
| Forecast<br>(EFOM) | BS       | 3.31T            | 3.31T            | 4.85T          | 0.0 to 46.7       |   |                   |  |
|                    | Error    | 0.3              | 0.3              | -2.7           | %                 |   |                   |  |
|                    | NN       | 3.30T            | 3.30T            | 5.01T          | 0.0 to 51.5       |   |                   |  |
|                    | Error    | 0.2              | 0.2              | 0.4            | %                 |   |                   |  |
|                    | LR       | 2.38T            | 4.05T            | 4.94T          | 70.3 to 107.7     |   |                   |  |
|                    | Error    | -27.9            | 22.8             | -1.0           | %                 |   |                   |  |
| Performance        | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %        |                   |   |                   |  |
| Observed           | 43.7P    | 39.5P            | 45.7P            | -9.6 to 4.5    |                   |   |                   |  |
| Forecast<br>(FOM)  | BS       | 44.1P            | 39.0P            | 45.5P          | -11.4 to 3.2      |   |                   |  |
|                    | Error    | 0.7              | -1.2             | -0.4           | %                 |   |                   |  |
|                    | NN       | 44.4P            | 41.1P            | 45.8P          | -7.3 to 3.1       |   |                   |  |
|                    | Error    | 1.5              | 4.0              | 0.2            | %                 |   |                   |  |
|                    | LR       | 35.2P            | 38.8P            | 47.7P          | 10.2 to 35.5      |   |                   |  |
|                    | Error    | -19.5            | -1.8             | 4.5            | %                 |   |                   |  |

### 6.3.2 LAMMPS Application

We now study energy and performance trade-offs for LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator), a full-scale scientific application. The LAMMPS model responses are cumulative functions rather than rate functions as with the PRKs and AMG, so the B-spline model equations 4.11 and 4.12 apply. LAMMPS solve time is measured in seconds as the performance met-



ric, and total energy in joules is adopted as a cumulative metric for energy use. This study uses the Lennard-Jones LAMMPS benchmark which simulates an atomic fluid with Lennard-Jones potential. The problem size is 32,000,000 atoms run for 6,000 timesteps on 80 nodes. Figure 6.9 shows the observed and predicted Pareto fronts, which appear at the lower left of the data set as the objective is now to minimise solve time and energy use.

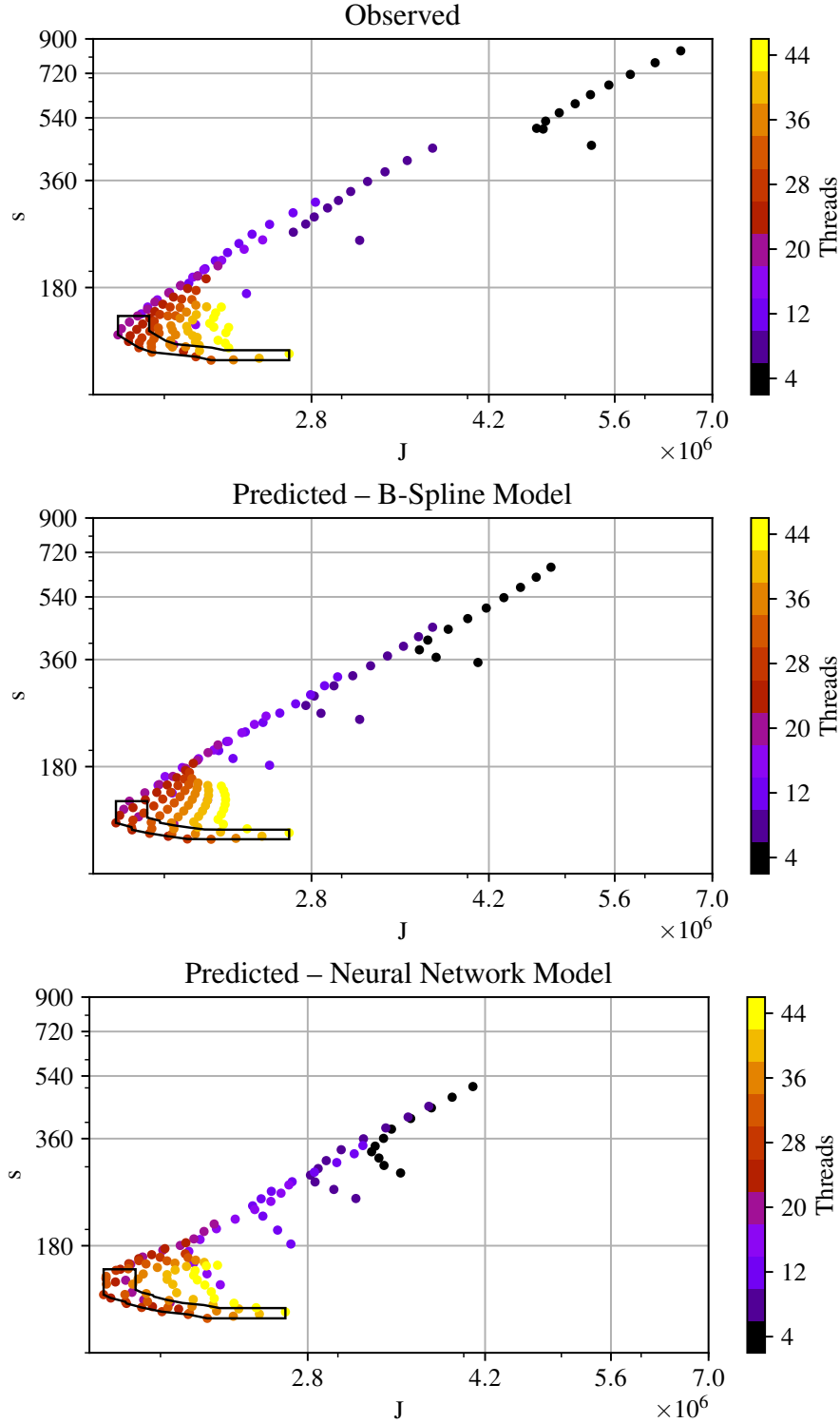


Figure 6.9: LAMMPS Pareto Front – Observed and Predicted

Table 6.8 shows that the total RMS error of the predicted values  $T_{RMSE}$  ranges from 14.2% to 28.4%. Figure 6.9 shows that this error mostly occurs at low thread counts, far from the Pareto front. Table 6.8

shows that the models remain highly accurate for the Pareto front, with  $P_{RMSE}$  for the BS models of 2.1% for energy and 3.8% for performance model and for the NN models of 4.7% for energy and 4.3% for performance. The observed and predicted Pareto fronts show that core and frequency tuning can increase energy efficiency up to 30% but at up to 30% performance loss.

Table 6.8: LAMMPS Results Summary

| Model fit       |       | Num obs  |                  | R <sup>2</sup>   | T <sub>RMSE</sub> |   | P <sub>RMSE</sub> |   |
|-----------------|-------|----------|------------------|------------------|-------------------|---|-------------------|---|
| Energy          | BS    | 12       | 1.0              | 14.2             | 2.1               |   |                   |   |
|                 | NN    | 12       | 1.0              | 21.8             | 4.7               |   |                   |   |
|                 | LR    | 12       | 0.975            | 20.0             | 27.7              |   |                   |   |
| Performance     | BS    | 12       | 1.0              | 15.7             | 3.8               |   |                   |   |
|                 | NN    | 12       | 1.0              | 28.4             | 4.3               |   |                   |   |
|                 | LR    | 12       | 0.986            | 28.4             | 49.7              |   |                   |   |
| Pareto front    |       | Limit    | Points           | Dist: 0          | 1                 | 2 | 3                 | 4 |
| Observed        |       | 5.0      | 27               | 23               | 4                 | 0 | 0                 | 0 |
| BS forecast     |       | 5.0      | 23               | 23               | 0                 | 0 | 0                 | 0 |
| NN forecast     |       | 5.0      | 33               | 26               | 4                 | 3 | 0                 | 0 |
| LR forecast     |       | 5.0      | 6                | 4                | 2                 | 0 | 0                 | 0 |
| Energy          |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed        |       | 2.66M    | 2.66M            | 1.80M            | 0.0 to -32.4      |   |                   |   |
| Forecast<br>(J) | BS    | 2.66M    | 2.66M            | 1.79M            | 0.0 to -32.7      |   |                   |   |
|                 | Error | 0.0      | 0.0              | -0.5             | %                 |   |                   |   |
|                 | NN    | 2.66M    | 2.66M            | 1.76M            | 0.0 to -34.0      |   |                   |   |
|                 | Error | 0.0      | 0.0              | -2.4             | %                 |   |                   |   |
|                 | LR    | 3.49M    | 1.49M            | 1.39M            | -57.5 to -60.1    |   |                   |   |
|                 | Error | 31.3     | -44.2            | -22.5            | %                 |   |                   |   |
| Performance     |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed        |       | 117      | 150              | 112              | 27.7 to -4.1      |   |                   |   |
| Forecast<br>(s) | BS    | 117      | 144              | 112              | 22.7 to -4.1      |   |                   |   |
|                 | Error | 0.0      | -3.9             | 0.0              | %                 |   |                   |   |
|                 | NN    | 117      | 155              | 112              | 31.8 to -4.1      |   |                   |   |
|                 | Error | 0.0      | 3.2              | 0.0              | %                 |   |                   |   |
|                 | LR    | 250      | 77.8             | 61.0             | -68.9 to -75.6    |   |                   |   |
|                 | Error | 113.2    | -48.0            | -45.7            | %                 |   |                   |   |

### 6.3.3 LULESH Application

LULESH (Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics) solves the Sedov blast wave problem, which models a shock front expanding in three dimensions from a point blast. Similar to AMG, the LULESH reported performance metric is Solve FOM where  $FOM = elements \times iterations/time$ . An Energy FOM is again derived using total time and energy in joules, such that  $EFOM = FOM \times time/energy$ .

The LULESH code uses a three dimensional mesh partitioned into domains. The number of domains

must be the cube of an integer, with one domain per MPI rank. The ranks used are 8, 27, 64, and 125 across 4, 14, 32, and 63 nodes respectively, with the same OpenMP thread count and frequency sweep as used for the other test cases.

Figure 6.10 shows the Pareto fronts for 63 nodes, 125 domains with  $31^3$  elements per domain, or 3,723,875 total elements. As Table 6.9 shows, the models are highly accurate for the Pareto front, with  $P_{RMSE}$  of 2.8% for energy and 2.1% for performance for the BS models and 3.3% for energy and 4.1% for performance for the NN models.

Table 6.9: LULESH Results Summary

| Model fit          |       | Num obs  |                  | R <sup>2</sup>   | T <sub>RMSE</sub> |   | P <sub>RMSE</sub> |   |
|--------------------|-------|----------|------------------|------------------|-------------------|---|-------------------|---|
| Energy             | BS    | 12       |                  | 1.0              | 4.7               |   | 2.8               |   |
|                    | NN    | 12       |                  | 1.0              | 7.7               |   | 3.3               |   |
|                    | LR    | 12       | 0.987            |                  | 5.0               |   | 4.7               |   |
| Performance        | BS    | 12       |                  | 1.0              | 3.6               |   | 2.1               |   |
|                    | NN    | 12       |                  | 1.0              | 8.2               |   | 4.1               |   |
|                    | LR    | 12       | 0.999            |                  | 2.3               |   | 3.5               |   |
| Pareto front       |       | Limit    | Points           | Dist: 0          | 1                 | 2 | 3                 | 4 |
| Observed           |       | 5.0      | 25               | 21               | 3                 | 1 | 0                 | 0 |
| BS forecast        |       | 5.0      | 23               | 21               | 2                 | 0 | 0                 | 0 |
| NN forecast        |       | 5.0      | 32               | 24               | 5                 | 2 | 1                 | 0 |
| LR forecast        |       | 5.0      | 20               | 18               | 2                 | 0 | 0                 | 0 |
| Energy             |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed           |       | 57.0M    | 57.0M            | 65.3M            | 0.0 to 14.6       |   |                   |   |
| Forecast<br>(EFOM) | BS    | 57.0M    | 57.0M            | 64.9M            | 0.0 to 13.8       |   |                   |   |
|                    | Error | 0.0      | 0.0              | -0.6             | %                 |   |                   |   |
|                    | NN    | 57.0M    | 57.0M            | 64.4M            | 0.0 to 12.8       |   |                   |   |
|                    | Error | 0.0      | 0.0              | -1.5             | %                 |   |                   |   |
|                    | LR    | 53.0M    | 53.0M            | 68.4M            | 0.0 to 29.1       |   |                   |   |
|                    | Error | -7.1     | -7.1             | 4.6              | %                 |   |                   |   |
| Performance        |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed           |       | 862G     | 587G             | 893G             | -31.9 to 3.7      |   |                   |   |
| Forecast<br>(FOM)  | BS    | 862G     | 598G             | 863G             | -30.6 to 0.1      |   |                   |   |
|                    | Error | 0.0      | 1.9              | -3.4             | %                 |   |                   |   |
|                    | NN    | 862G     | 462G             | 862G             | -46.4 to 0.0      |   |                   |   |
|                    | Error | 0.0      | -21.3            | -3.5             | %                 |   |                   |   |
|                    | LR    | 823G     | 603G             | 834G             | -26.7 to 1.2      |   |                   |   |
|                    | Error | -4.4     | 2.8              | -6.7             | %                 |   |                   |   |

### 6.3.4 WRF Application

The Weather Research and Forecasting (WRF) model is a mesoscale numerical weather prediction system that is widely used for both operational forecasting and research. WRF supports parallelisation using distributed memory, shared memory, or hybrid distributed/shared memory models.

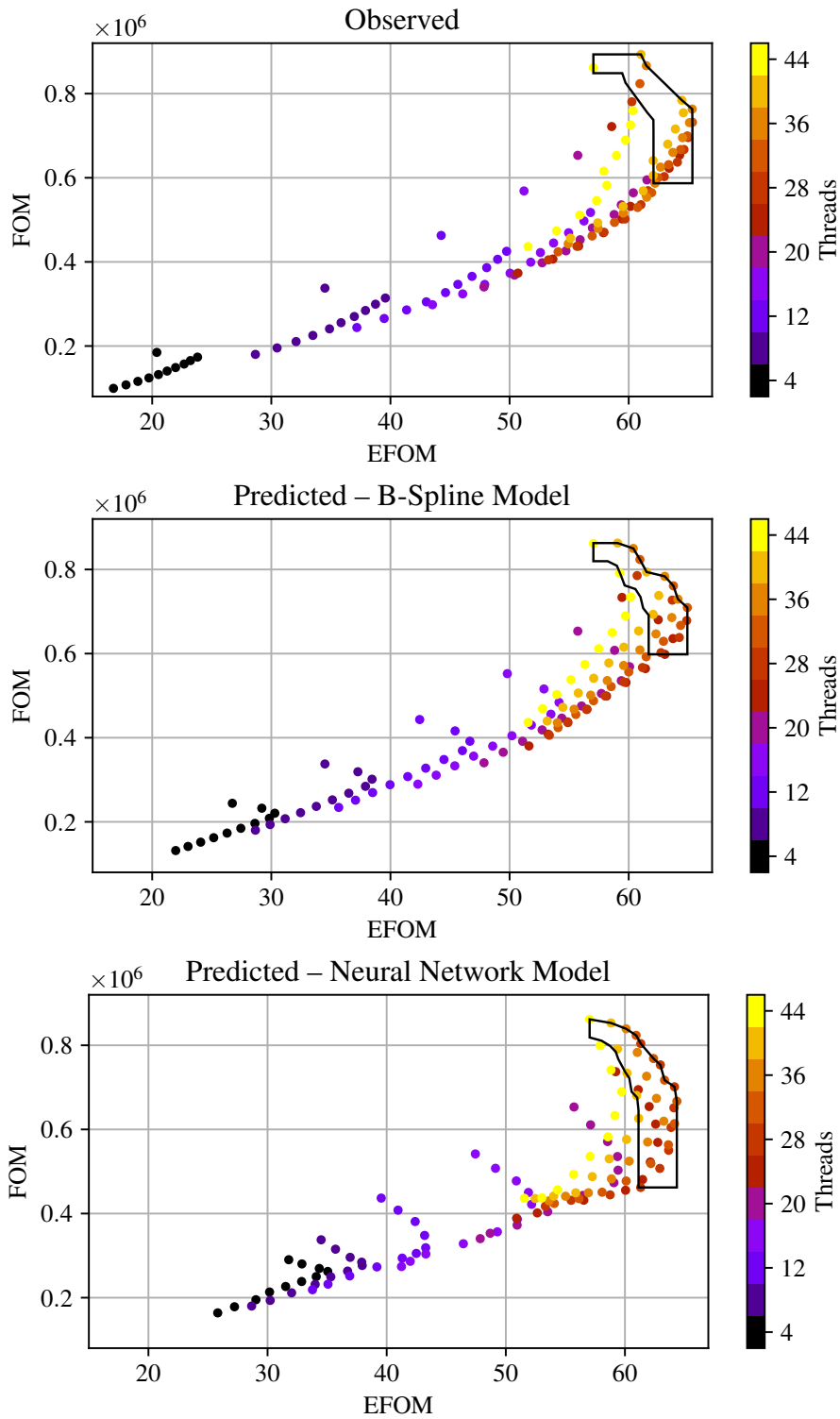


Figure 6.10: LULESH Pareto Front – Observed and Predicted

With distributed/shared memory, the model domain decomposes into patches and tiles for hybrid MPI/OpenMP processing. Patches are distributed to MPI tasks, and tiles within patches are distributed to OpenMP threads. Marginal performance gains can be achieved by tuning patch and tile parameters for the MPI task and OpenMP thread topology, typically using low thread counts (one to four threads). Using one thread per MPI task defaults to one tile per patch which provides acceptable performance with minimal run to run variability. Varying MPI ranks with a fixed thread count also provides a new test case for the proposed BS and NN models, where thread count and MPI ranks per node are fully

correlated. The earlier case studies hold MPI ranks per node constant.

The WRF Preprocessing System (WPS) utilities, `ungrib.exe`, `geogrid.exe`, and `metgrid.exe`, run once on a single HPC node to unpack model parameters, prepare the model domain, and interpolate meteorological data to model grid.

The WRF programs, `real.exe` and `wrf.exe` run on the full HPC cluster. Model initial and boundary conditions are set once using `real.exe`. Training and test data for the energy efficiency and performance models are captured over multiple runs of `wrf.exe` which performs the weather model simulation using the prepared inputs.

Figure 6.11 shows a 12 hour total precipitation forecast generated with WRF for Hurricane Katrina in the Gulf of Mexico on 28 August 2005. The domain grid size is  $3,500 \text{ km} \times 2,500 \text{ km}$  with grid resolution of  $5 \text{ km} \times 5 \text{ km}$ . The simulation duration is 12 hours with time step resolution of 9 seconds. The observed model execution times for `wrf.exe` using this configuration on 64 nodes range between 11 and 60 minutes.

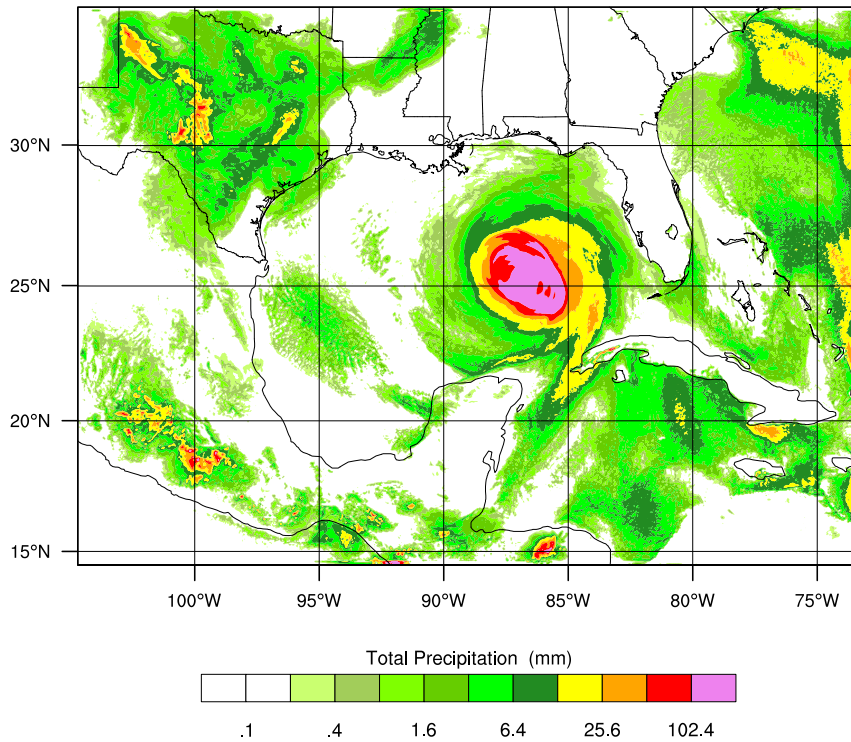


Figure 6.11: WRF Total Precipitation

Figure 6.12 shows the Pareto fronts for 64 nodes, with 2 to 22 OpenMP threads per node (2 to 22 MPI ranks per node with 1 OpenMP thread per rank). The WRF performance and energy efficiency metrics for optimisation are model *timesteps per second* and *timesteps per joule*.

Table 6.10 shows that the total RMS error of the predicted values  $T_{RMSE}$  ranges from 3.1% to 6.9%. For the BS models,  $P_{RMSE}$  is 3.2% for energy and 1.5% for performance and, for the NN models, it is

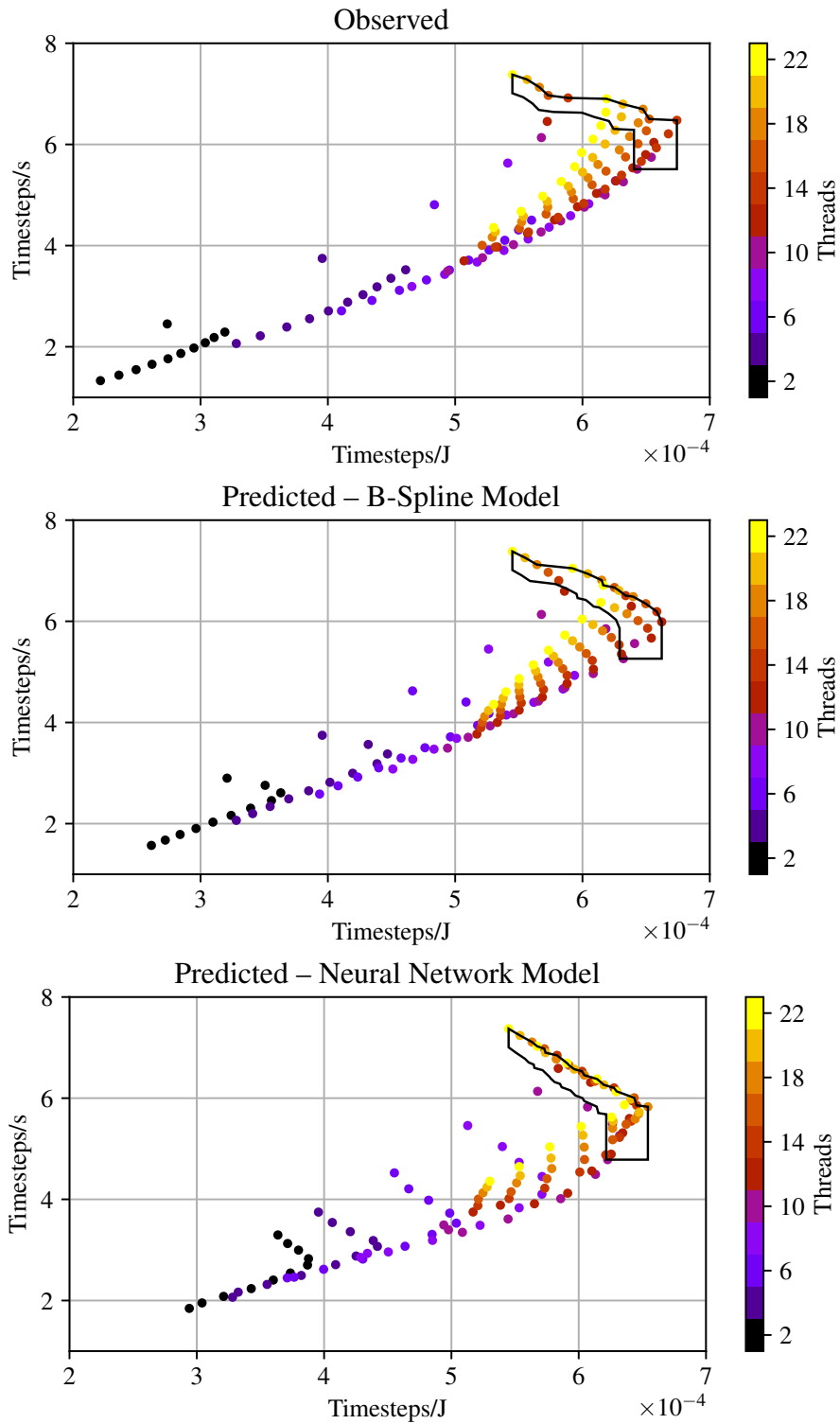


Figure 6.12: WRF Pareto Front – Observed and Predicted

5.4% for energy and 1.2% for performance. The observed and predicted Pareto fronts show that core and frequency tuning can increase energy efficiency more than 20% but at around 30% performance loss.

Table 6.10: WRF Results Summary

| Model fit                 |       | Num obs  |                  | R <sup>2</sup>   | T <sub>RMSE</sub> |   | P <sub>RMSE</sub> |   |
|---------------------------|-------|----------|------------------|------------------|-------------------|---|-------------------|---|
| Energy                    | BS    | 12       | 12               | 1.0              | 4.2               |   | 3.2               |   |
|                           | NN    | 12       | 12               | 1.0              | 6.9               |   | 5.4               |   |
|                           | LR    | 12       | 12               | 0.964            | 6.0               |   | 6.5               |   |
| Performance               | BS    | 12       | 12               | 1.0              | 3.1               |   | 1.5               |   |
|                           | NN    | 12       | 12               | 1.0              | 6.5               |   | 1.2               |   |
|                           | LR    | 12       | 12               | 0.995            | 3.2               |   | 3.0               |   |
| Pareto front              |       | Limit    | Points           | Dist: 0          | 1                 | 2 | 3                 | 4 |
| Observed                  |       | 5.0      | 22               | 20               | 2                 | 0 | 0                 | 0 |
| BS forecast               |       | 5.0      | 26               | 20               | 6                 | 0 | 0                 | 0 |
| NN forecast               |       | 5.0      | 42               | 21               | 9                 | 5 | 5                 | 2 |
| LR forecast               |       | 5.0      | 22               | 17               | 4                 | 1 | 0                 | 0 |
| Energy                    |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed                  |       | 545μ     | 545μ             | 674μ             | 0.0 to 23.7       |   |                   |   |
| Forecast<br>(Timesteps/J) | BS    | 545μ     | 545μ             | 662μ             | 0.0 to 21.5       |   |                   |   |
|                           | Error | 0.0      | 0.0              | -1.8             | %                 |   |                   |   |
|                           | NN    | 545μ     | 545μ             | 654μ             | 0.0 to 20.1       |   |                   |   |
|                           | Error | 0.0      | 0.0              | -2.9             | %                 |   |                   |   |
|                           | LR    | 496μ     | 496μ             | 698μ             | 0.0 to 40.7       |   |                   |   |
|                           | Error | -9.0     | -9.0             | 3.5              | %                 |   |                   |   |
| Performance               |       | Baseline | P <sub>min</sub> | P <sub>max</sub> | Range %           |   |                   |   |
| Observed                  |       | 7.38     | 5.51             | 7.38             | -25.3 to 0.0      |   |                   |   |
| Forecast<br>(Timesteps/s) | BS    | 7.38     | 5.26             | 7.38             | -28.7 to 0.0      |   |                   |   |
|                           | Error | 0.0      | -4.5             | 0.0              | %                 |   |                   |   |
|                           | NN    | 7.37     | 4.78             | 7.37             | -35.1 to 0.0      |   |                   |   |
|                           | Error | -0.1     | -13.2            | -0.1             | %                 |   |                   |   |
|                           | LR    | 6.85     | 5.67             | 7.06             | -17.1 to 3.2      |   |                   |   |
|                           | Error | -7.3     | 2.9              | -4.3             | %                 |   |                   |   |

## 6.4 Study Observations

The results show that the models perform well for the studied kernels and applications, and significantly outperform the naive model. Table 6.11 and 6.12 summarise key metrics across the eight test cases (three PRKs, single-node AMG, multi-node AMG, LAMMPS, LULESH, and WRF).

Table 6.11 shows that the mean Pareto RMS Error,  $P_{RMSE}$ , for energy across the test cases 2.6% for the Basis Spline models and 4.3% for the Neural Network models. Both results are superior to the naive Linear Regression model, which has mean Pareto RMS Error of 10.6%.

The mean Total RMS Error,  $T_{RMSE}$ , is around 6.4% for the Basis Spline models, and significantly higher for the Neural Network models at 10.6%. The mean RMS Errors for performance are similar. With our small training data set the Neural Network forecasts tend to overfit the observed data, closely

Table 6.11: Models RMSE Comparison – 8 Test Cases

| Model       |                | Metric            | Mean  |
|-------------|----------------|-------------------|-------|
| Energy      | B-Spline       | T <sub>RMSE</sub> | 6.36  |
|             |                | P <sub>RMSE</sub> | 2.57  |
|             | Neural Network | T <sub>RMSE</sub> | 10.59 |
|             |                | P <sub>RMSE</sub> | 4.32  |
|             | Linear Reg     | T <sub>RMSE</sub> | 9.43  |
|             |                | P <sub>RMSE</sub> | 10.64 |
| Performance | B-Spline       | T <sub>RMSE</sub> | 6.01  |
|             |                | P <sub>RMSE</sub> | 2.51  |
|             | Neural Network | T <sub>RMSE</sub> | 10.75 |
|             |                | P <sub>RMSE</sub> | 4.00  |
|             | Linear Reg     | T <sub>RMSE</sub> | 8.40  |
|             |                | P <sub>RMSE</sub> | 11.42 |

Table 6.12: Models Pareto Points Comparison – 8 Test Cases

| Model      | Mean points |      | For by distance |     |     |     |     |
|------------|-------------|------|-----------------|-----|-----|-----|-----|
|            | Obs         | For  | 0               | 1   | 2   | 3   | 4   |
| B-Spline   | 45.2        | 47.6 | 40.4            | 5.0 | 1.4 | 0.6 | 0.2 |
| Neural Net | 45.2        | 46.1 | 35.0            | 6.2 | 2.8 | 1.5 | 0.6 |
| Linear Reg | 45.2        | 29.4 | 24.4            | 3.9 | 0.9 | 0.2 | 0.0 |

fitting observed values that are near the training data, but the error increases for values further away. The observed time to solution is significantly longer for the Neural Network models. Although this model is less efficient, its iterative, automated approach requires less intuition and domain knowledge than the Basis Spline models. Domain knowledge and intuition are concerns at model construction time, so runtime users of the models are not impacted.

As Table 6.12 shows, the mean number of observed Pareto points in the trade-off zone is 45.2 points. The Basis Spline models forecast a mean of 47.6 points, roughly two more than the observed points. The Neural Network models forecast a mean of 46.1 points, 0.9 more than the observed points. So, the models exhibit 2-5% error in the prediction of the number of Pareto points. The naive Linear Regression model forecast of 29.4 points exhibits 34% error.

For the Basis Spline models, on average  $(40.4 + 5.0)/47.6 = 95\%$  of predicted Pareto points are co-incident or within one search step of the observed points. For the Neural Network models, this metric is slightly lower at  $(35.0 + 6.2)/46.1 = 89\%$ . The percentage is similar for the Linear Regression model at  $(24.4 + 3.9)/29.4 = 96\%$ , but negated by the 34% error in the predicted number of Pareto points.

Differences between test cases, such as strong versus weak scaling, rate versus cumulative performance and energy use metrics, and fixed versus correlated MPI ranks, further validate the generality of the method. However, several new challenges arise when complex workloads are modelled. Com-



plex applications typically require many system resources, so the cost to obtain training samples must be amortised over many real application runs. The required amortisation period is minimised by limiting training data to a small fraction of the search space, but the approach is only useful if those costs result in overall benefits. In practice, production applications are run many times with the same data dimensions and can accrue benefits that offset training costs. A weather model such as WRF that runs daily is a good example.

Complex interactions between applications and systems may lead to irregular energy or performance response curves, particularly if application data/processing dimensions align poorly with the system topology. Irregular response curves impact the statistical significance of the regression results, so users can be alerted that further analysis is needed when the results are not within desired limits.

## 6.5 3-Fold Cross Validation

Cross validation shows that the Basis Spline and Neural Network models do not overfit a subset of the data. Model performance for smaller training data sets is important in this case so three folds are used. This provides a reasonable balance between training data set size (67%) and data set partitions (three). With 3-fold cross validation, the data set is randomly partitioned into three equal sized subsets. Two are used as training data with model predictions tested against the remaining subset. The process repeats three times to test model accuracy across the full data set.

Table 6.13 shows cross validation results for the energy efficiency models. The data set for WRF has 121 samples, so the three folds consist of two folds with 40 samples and one with 41 samples. The data sets for the other test cases have 484 samples using two folds with 161 samples and one with 162 samples. Table 6.13 includes the percentage of predicted values from each fold that are below 20%, 10% and 5% and the RMS Error for the fold. The relatively even RMS Error results across the folds demonstrates that the energy efficiency models do not overfit any subset of the data. Table 6.14 shows cross validation results for the performance models using the same data sets. The consistent RMS Error results across the folds show that the performance models also do not overfit any subset of the data.

The training data is 67% of the full data set for each cross validation fold, compared to the 10% used to predict Pareto fronts. The NN models benefit significantly from the increased training data, achieving mean Total RMS Error of around 3% across the study cases, compared to around 5% for the BS models. This result provides further support for the observation that the NN models exhibit a level of overfitting with small training data sets. Practical models must consider the balance between training data requirements and accuracy. Adding training data may improve accuracy, but the higher training costs must be recoverable from benefits achieved in live runs of the application.

Table 6.13: 3-Fold Cross Validation Summary – Energy Efficiency Models

| Kernel      | Fold | B-Spline |       |      |        | Neural Network |       |       |        |
|-------------|------|----------|-------|------|--------|----------------|-------|-------|--------|
|             |      | 20%      | 10%   | 5%   | RMSE % | 20%            | 10%   | 5%    | RMSE % |
| Stencil     | 1    | 100.0    | 99.4  | 85.2 | 3.6    | 100.0          | 100.0 | 96.3  | 2.4    |
|             | 2    | 100.0    | 100.0 | 91.9 | 4.1    | 100.0          | 100.0 | 99.4  | 2.0    |
|             | 3    | 100.0    | 96.3  | 85.1 | 3.4    | 100.0          | 98.8  | 96.3  | 2.3    |
| Transpose   | 1    | 100.0    | 93.8  | 81.5 | 4.5    | 100.0          | 100.0 | 98.1  | 1.9    |
|             | 2    | 100.0    | 97.5  | 87.6 | 4.5    | 100.0          | 100.0 | 99.4  | 1.9    |
|             | 3    | 99.4     | 94.4  | 80.7 | 4.7    | 100.0          | 98.8  | 96.9  | 1.8    |
| Nstream     | 1    | 100.0    | 92.0  | 67.3 | 5.9    | 100.0          | 100.0 | 100.0 | 1.3    |
|             | 2    | 100.0    | 96.9  | 80.7 | 6.8    | 100.0          | 100.0 | 100.0 | 1.4    |
|             | 3    | 98.1     | 87.6  | 72.7 | 6.1    | 100.0          | 98.8  | 97.5  | 1.5    |
| AMG         | 1    | 100.0    | 95.1  | 82.7 | 3.5    | 100.0          | 100.0 | 92.6  | 2.1    |
|             | 2    | 100.0    | 98.1  | 93.2 | 4.0    | 100.0          | 97.5  | 93.2  | 2.7    |
|             | 3    | 100.0    | 96.9  | 91.3 | 3.6    | 100.0          | 98.1  | 95.7  | 2.6    |
| LAMMPS      | 1    | 100.0    | 87.7  | 75.3 | 7.7    | 100.0          | 88.9  | 77.2  | 7.3    |
|             | 2    | 100.0    | 87.6  | 68.9 | 8.4    | 100.0          | 95.0  | 80.7  | 7.5    |
|             | 3    | 100.0    | 96.3  | 85.7 | 10.3   | 100.0          | 96.9  | 83.9  | 9.3    |
| LULESH      | 1    | 100.0    | 94.4  | 79.0 | 4.3    | 100.0          | 100.0 | 94.4  | 2.3    |
|             | 2    | 100.0    | 99.4  | 84.5 | 4.3    | 100.0          | 100.0 | 97.5  | 1.7    |
|             | 3    | 99.4     | 91.9  | 80.1 | 4.0    | 100.0          | 96.3  | 94.4  | 2.1    |
| WRF         | 1    | 100.0    | 97.6  | 75.6 | 4.6    | 100.0          | 100.0 | 95.1  | 1.8    |
|             | 2    | 97.5     | 95.0  | 87.5 | 4.7    | 100.0          | 97.5  | 90.0  | 2.5    |
|             | 3    | 100.0    | 100.0 | 95.0 | 3.1    | 100.0          | 100.0 | 100.0 | 1.9    |
| Mean RMSE % |      |          |       |      | 5.06   | 2.88           |       |       |        |

## 6.6 Conclusion

This evaluation shows that the proposed method can accurately predict optimal performance and energy efficiency settings for parallel kernels and for more complex application workloads. The statistical and machine learning model results show that both approaches provide good overall fit between predicted and measured values. The models can accurately identify trade-offs between performance and energy. The approach requires measurement samples for a small fraction of the search space, that can be run in parallel when sufficient resources are available.

Table 6.14: 3-Fold Cross Validation Summary – Performance Models

| Kernel      | Fold | B-Spline |       |       |        | Neural Network |       |       |        |
|-------------|------|----------|-------|-------|--------|----------------|-------|-------|--------|
|             |      | 20%      | 10%   | 5%    | RMSE % | 20%            | 10%   | 5%    | RMSE % |
| Stencil     | 1    | 100.0    | 93.2  | 80.2  | 4.2    | 100.0          | 100.0 | 93.2  | 2.6    |
|             | 2    | 100.0    | 96.9  | 83.9  | 5.4    | 100.0          | 100.0 | 98.8  | 2.5    |
|             | 3    | 99.4     | 91.3  | 80.7  | 4.1    | 99.4           | 95.7  | 91.9  | 3.3    |
| Transpose   | 1    | 100.0    | 96.9  | 89.5  | 3.9    | 100.0          | 100.0 | 98.1  | 2.8    |
|             | 2    | 100.0    | 98.1  | 87.6  | 4.1    | 100.0          | 100.0 | 98.1  | 2.1    |
|             | 3    | 99.4     | 92.5  | 82.0  | 4.1    | 100.0          | 99.4  | 93.2  | 2.1    |
| Nstream     | 1    | 100.0    | 90.1  | 66.0  | 5.7    | 100.0          | 100.0 | 96.3  | 1.9    |
|             | 2    | 100.0    | 95.7  | 76.4  | 7.5    | 100.0          | 100.0 | 100.0 | 1.7    |
|             | 3    | 97.5     | 86.3  | 69.6  | 6.1    | 100.0          | 99.4  | 96.3  | 1.8    |
| AMG         | 1    | 100.0    | 96.3  | 78.4  | 3.2    | 100.0          | 98.1  | 91.4  | 2.5    |
|             | 2    | 100.0    | 98.1  | 89.4  | 4.3    | 100.0          | 97.5  | 92.5  | 2.5    |
|             | 3    | 100.0    | 96.9  | 87.0  | 3.4    | 100.0          | 97.5  | 95.7  | 2.6    |
| LAMMPS      | 1    | 100.0    | 85.2  | 76.5  | 8.6    | 100.0          | 95.1  | 87.7  | 6.0    |
|             | 2    | 100.0    | 85.1  | 72.7  | 8.6    | 100.0          | 94.4  | 83.9  | 7.1    |
|             | 3    | 100.0    | 96.9  | 82.6  | 9.4    | 100.0          | 98.1  | 93.8  | 5.6    |
| LULESH      | 1    | 100.0    | 94.4  | 77.8  | 4.1    | 100.0          | 94.4  | 81.5  | 3.8    |
|             | 2    | 100.0    | 100.0 | 85.7  | 4.4    | 100.0          | 95.0  | 83.9  | 5.0    |
|             | 3    | 99.4     | 92.5  | 78.9  | 3.4    | 99.4           | 88.2  | 77.6  | 3.3    |
| WRF         | 1    | 100.0    | 100.0 | 90.2  | 2.6    | 100.0          | 100.0 | 97.6  | 1.7    |
|             | 2    | 100.0    | 95.0  | 85.0  | 3.6    | 100.0          | 95.0  | 95.0  | 1.5    |
|             | 3    | 100.0    | 100.0 | 100.0 | 1.6    | 100.0          | 100.0 | 100.0 | 1.3    |
| Mean RMSE % |      |          |       |       | 4.87   | 3.03           |       |       |        |



# Chapter 7

## Conclusion and Future Directions

This chapter concludes the dissertation with a summary of the work undertaken and key contributions. HPC platforms and applications form a highly complex system that can behave in unexpected ways. At the start of this work, new stochastic models and trade-off analysis techniques appeared to be a promising research direction. However, achieving significant advances over existing techniques was not certain. The presented achievements are the result of a considerable research effort.

The last part of the chapter proposes a number of potential directions for future research in HPC energy efficiency optimisation, based on experience gained in the course of this work.

### 7.1 Thesis Summary

Chapter 1 introduces key challenges associated with performance and energy efficiency tuning. Optimisation techniques need to deal with extensive configuration spaces, due to the large number of platform and application specific parameters that influence performance and energy efficiency. Existing tools can also exclude key HPC user groups, such as scientific application users. Tuning costs must be minimised to allow a reasonable payback period on tuning effort, while maintaining an acceptable level of accuracy. It is also important to avoid excluding trade-off options that may be relevant to users.

A number of potential research directions for addressing these challenges are evident from the Chapter 2 literature survey. Development of new system models can be informed by further analysis of tuning control accessibility and the level of control provided. Complexities associated with white-box modelling techniques may be avoided if stochastic models can be developed to predict trade-off options at low cost. Improved trade-off analysis techniques for managing measurement and modelling error could provide a more complete set of trade-off options and enable better evaluation of observed and predicted results.

The tuning controls investigation presented in Chapter 3 establishes key parameter and search space constraints for scoping model designs. Parameter resolution and sensitivity findings guide model predictor selection and sampling strategies. The results provide convincing evidence that there are genuine trade-off opportunities available, subject to new modelling techniques that can predict trade-off options.

A new and innovative energy tuning architecture that extends traditional performance tuning infrastructure is proposed in Chapter 4. Key new features of the architecture include: stochastic models that tolerate small and noisy training data; a statistical validation scheme for trade-off range predictions; and an error management technique for multi-objective optimisation analysis. The modular design and pluggable interfaces allow integration with existing instrumentation, measurement, and profiling tools.

The HPCProbe framework provides a prototype implementation of the proposed architecture, as Chapter 5 describes. A key focus of HPCProbe is enabling a comprehensive study of the new models and error management techniques when applied in real-world multi-objective optimisation scenarios. HPCProbe implements configurable B-spline and machine learning models, along with algorithms for analysing trade-off options using the proposed trade-off zone technique as an extension of traditional Pareto analysis.

Chapter 6 presents a comprehensive evaluation of the new models and analysis techniques using HPCProbe. Model performance is evaluated using a range of metrics across three parallel kernels and four applications running at scale on a homogeneous HPC cluster. Table 6.11 shows the models achieve an average RMS error of less than 5% for predicted Pareto points for both energy and performance. Model training is attained using sample measurements from a small fraction of the search space.

The proposed models are capable of accurately reproducing the actual relationships between energy and performance based on the collected training samples. Appropriate sampling captures the unique features of each platform and application combination. This is the intuition of the work.

## 7.2 Key Contributions

This thesis provides a framework for accurately and efficiently predicting performance and energy usage trade-off options for scientific applications. Specifically, this work presents the following contributions:

1. An analysis of the energy usage and performance responses of a representative set of kernels, that arise from interactions between software and hardware. Response transitions and associ-

ated tuning sensitivity changes are identified for each kernel.

2. Energy and performance trade-off option selection using basic HPC job scheduling parameters. To use the desired trade-off option, parallel application users select the predicted optimal job scheduling parameters. Other parameters that cannot be controlled by the user are treated as constraints. Significant tuning opportunities are demonstrable within these constraints.
3. A practical method for predicting Pareto efficient trade-off options from few input measurements. Prediction methods must minimise the required optimisation effort so its cost can be recovered quickly by the effort saved in more efficient live runs.
4. Multi-objective models that accurately predict the energy efficiency and performance responses of parallel applications. B-spline piecewise polynomial and deep neural network machine learning models are presented that provide accurate trade-off predictions using a small set of training measurements.
5. A trade-off zone approach that improves Pareto optimisation in the presence of measurement and/or modelling error. This innovation ensures users are presented with a complete set of trade-off options that includes other points that are statistically equivalent to points on the Pareto front. The approach is generalisable to all Pareto analysis of noisy data sets.
6. HPCProbe, a prototype implementation of the proposed energy tuning framework. This framework implements the described advances using a modular and extensible design. HPCProbe is available to interested researchers, as described in Appendix C.

## 7.3 Future Research Directions

This section highlights future potential research directions within the HPC energy efficiency optimisation domain.

### 7.3.1 Iterative Application Training

The evaluation activities reported in Chapter 6 use the same job configuration for training and test runs. There were indications, however, that iterative or time-stepped applications can be accurately trained with reduced iteration counts.

Researching the practicality of further reducing training costs by reducing application iteration/time-step configurations has the potential to provide significant new reductions in model training costs. If the model for an application requiring 10,000 time-steps for live runs could be trained in 1,000 time-steps, the training effort may be reduced by a factor of 10.

### 7.3.2 Heterogeneous Systems

Heterogeneous or hybrid HPC architectures use multi-core CPUs in combination with accelerators, such as GPUs, FPGAs, or Intel Phi coprocessors. Highly parallel GPUs are typically equipped with thousands of specialised cores. GPUs can offer significant improvements in energy efficiency over multi-core CPUs [3]. The November 2018 TOP500 list [5] shows the importance of these systems, with approaching 30%, or 138 of the 500 supercomputers on the list using hybrid CPU/GPU or CPU/coprocessor architectures.

Although HPCProbe is able to capture GPU performance and energy use metrics as described in Appendix A, further work is required to identify and implement model extensions to enable response predictions for hybrid systems. GPU parallelisation and processor frequency controls are expected to be important, as is workload partitioning between CPU and GPU cores.

### 7.3.3 Search Optimisation

Energy efficiency and performance measurements often plateau when resource saturation occurs. Measurements errors can in turn overshadow the small gradient changes in plateau regions, such that the whole region must be considered statistically indistinguishable. For example, noise induced gradient changes may cause a gradient descent/ascent algorithm to terminate at local minima/maxima before reaching the global minima/maxima.

The trade-off zone method presented in this thesis is applied to measured and model generated data so that each data point is evaluated against all the statistically equivalent trade-off values.

Tools using search optimisation techniques, such as Nimrod/O [29], may also benefit from the trade-off zone approach. Extending the search goal to include all points within error limits will help avoid local optima and improve the completeness of trade-off data sets generated by tools using search optimisation.

### 7.3.4 Policy Development

New tuning policies are needed to guide users in the selection of the most appropriate trade-off option. Table 6.12 shows the Pareto optimal trade-off points constitute over one third of the search space on average for the case studies presented in this work.

Figure 2.2 places trade-off points along the Pareto front in power, balanced, and performance optimised clusters [18]. This approach guides users in selecting trade-off points that favour energy efficiency or performance, or make both equally important. After selecting a suitable cluster, there can be more options again to choose from within the cluster. These points may be essentially equivalent in performance and energy usage terms, so other forms of guidance are needed.



Resource usage is also used to guide trade-off decisions [15], providing further input for policy development. For example, thrifty, greedy, and balanced resource allocation policies may aim to minimise, maximise, or balance resource usage, such as core, socket, and node allocations. This policy development effectively extends the optimisation problem from bi-objective to three or more objectives. The additional objectives add new trade-off metrics that help to distinguish points that are difficult to separate on performance or energy use alone.

### 7.3.5 Uncertainties Propagation

The system responses evaluated for tuning may be functions of other measured inputs, as equation 7.1 shows for energy efficiency in Flops/J.

$$\text{energy efficiency} = \frac{\text{Flops/s} \times \text{time}}{\text{energy}} \quad (7.1)$$

Confidence intervals for each measured input can be derived using the  $t$ -distribution, as described in section 4.2.2. The confidence interval for the function needs to be derived by combining the confidence intervals for each measured input.

Although not widely used in computer performance analysis, uncertainties propagation techniques allow limits information from each input to be combined to derive limits for the function. Like most related work that considers error limits, this work assumes uniform percentage limits across responses, based on the available measured data. This provides sufficient accuracy for the case studies, but there appears to be an opportunity to improve the robustness of performance studies in general by adopting uncertainties propagation techniques.



# Bibliography

- [1] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, “A bottleneck-centric tuning policy for optimizing energy in parallel programs,” *Advances in Parallel Computing*, vol. 32, pp. 265–276, 2018.
- [2] ———, “Statistical and machine learning models for optimizing energy in parallel applications,” *The International Journal of High Performance Computing Applications*, 2019.
- [3] C. Jin, B. R. de Supinski, D. Abramson, H. Poxon, L. DeRose, M. N. Dinh, M. Endrei, and E. R. Jessup, “A survey on software methods to improve the energy efficiency of parallel computing,” *The International Journal of High Performance Computing Applications*, vol. 31, no. 6, pp. 517–549, 2017.
- [4] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, “Energy efficiency modeling of parallel applications,” in *Proceedings of SC18, the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE, 2018, pp. 17:1–17:13.
- [5] TOP500 project, “The top500 list,” 2018. [Online]. Available: <http://www.top500.org/>
- [6] J. J. Dongarra, “Performance of various computers using standard linear equations software,” University of Tennessee, Tech. Rep., 2014. [Online]. Available: <http://www.netlib.org/benchmark/performance.ps>
- [7] T. Haiden, M. Janousek, J. Bidlot, L. Ferranti, F. Prates, F. Vitart, P. Bauer, and D. Richardson, “Evaluation of ECMWF forecasts, including the 2016 resolution upgrade,” European Centre for Medium Range Weather Forecasts, Tech. Rep. 792, 2016.
- [8] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, “United states data center energy usage report,” Lawrence Berkeley National Laboratory, Berkeley, CA, Tech. Rep. LBNL-1005775, 2016.
- [9] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, “Dark silicon and the end of multicore scaling,” in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 365–376.

- [10] U.S. Energy Information Administration, “Average monthly bill – residential,” 2017. [Online]. Available: [https://www.eia.gov/electricity/sales\\_revenue\\_price/pdf/table5\\_a.pdf](https://www.eia.gov/electricity/sales_revenue_price/pdf/table5_a.pdf)
- [11] T. R. Scogland, C. P. Steffen, T. Wilde, F. Parent, S. Coghlan, N. Bates, W.-c. Feng, and E. Strohmaier, “A power-measurement methodology for large-scale, high-performance computing,” in *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*. ACM, 2014, pp. 149–159.
- [12] M. Avgerinou, P. Bertoldi, and L. Castellazzi, “Trends in data centre energy consumption under the European code of conduct for data centre energy efficiency,” *Energies*, vol. 10, no. 10, p. 1470, 2017.
- [13] ABC News Online, “Iceland will soon use more energy mining bitcoins than powering its homes,” 2018. [Online]. Available: <https://www.abc.net.au/news/2018-02-14/energy-spent-mining-bitcoins-to-overtake-iceland27s-domestic-p/9446916>
- [14] Brisbane Times, “Data centre power use greater than Woolworths, Coles combined,” 2018. [Online]. Available: <https://www.brisbanetimes.com.au/business/the-economy/data-centre-power-use-greater-than-woolworths-coles-combined-20180922-p505df.html>
- [15] P. Gschwandtner, J. J. Durillo, and T. Fahringer, “Multi-objective auto-tuning with Insieme: Optimization and trade-off analysis for time, energy and resource usage,” in *European Conference on Parallel Processing*. Springer, 2014, pp. 87–98.
- [16] M. Mezmaiz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, A. Y. Zomaya, and D. Tuytens, “A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems,” *Journal of Parallel and Distributed Computing*, vol. 71, no. 11, pp. 1497–1508, 2011.
- [17] C. J. Petrie, T. A. Webster, and M. R. Cutkosky, “Using Pareto optimality to coordinate distributed agents,” *AI EDAM*, vol. 9, no. 4, pp. 269–281, 1995.
- [18] J. Michanan, R. Dewri, and M. J. Rutherford, “Understanding the power-performance tradeoff through Pareto analysis of live performance data,” in *International Green Computing Conference*. IEEE, 2014, pp. 1–8.
- [19] P. E. Bailey, A. Marathe, D. K. Lowenthal, B. Rountree, and M. Schulz, “Finding the limits of power-constrained application performance,” in *Proceedings of SC15, the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 79.
- [20] T. Peachey, M. Riley, D. Abramson, and J. Stewart, “A simplex-like search method for bi-objective optimization,” in *Proceedings of the 3rd International Conference on Engineering Optimization*. Federal University of Rio de Janeiro, 2012, pp. 1–10.
- [21] R. Sen and D. A. Wood, “Pareto governors for energy-optimal computing,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 1, p. 6, 2017.

- [22] P. Balaprakash, A. Tiwari, and S. M. Wild, “Multi objective optimization of HPC kernels for performance, power, and energy,” in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 2013, pp. 239–260.
- [23] S. Cho and R. G. Melhem, “On the interplay of parallelization, program performance, and energy consumption,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 3, pp. 342–353, 2010.
- [24] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos, “Online strategies for high-performance power-aware thread execution on emerging multiprocessors,” in *IEEE 20th International Parallel and Distributed Processing Symposium*. IEEE, 2006, pp. 8–pp.
- [25] V. W. Freeh, N. Kappiah, D. K. Lowenthal, and T. K. Bletsch, “Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs,” *Journal of Parallel and Distributed Computing*, vol. 68, no. 9, pp. 1175–1185, 2008.
- [26] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, “Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs,” in *Proceedings of SC06, the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE, 2006, pp. 14–14.
- [27] M. Sourouri, E. B. Raknes, N. Reissmann, J. Langguth, D. Hackenberg, R. Schöne, and P. G. Kjeldsberg, “Towards fine-grained dynamic tuning of HPC applications on modern multi-core architectures,” in *Proceedings of SC17, the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 41.
- [28] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snively, “Auto-tuning for energy usage in scientific applications,” in *European Conference on Parallel Processing*. Springer, 2011, pp. 178–187.
- [29] D. Abramson, A. Lewis, and T. Peachey, “Nimrod/O: a tool for automatic design optimisation using parallel and distributed systems,” in *Algorithms And Architectures For Parallel Processing: ICA3PP 2000*. World Scientific, 2000, pp. 497–508.
- [30] S. F. Rahman, J. Guo, and Q. Yi, “Automated empirical tuning of scientific codes for performance and power consumption,” in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*. ACM, 2011, pp. 107–116.
- [31] Y. C. Lee and A. Y. Zomaya, “Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling,” in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2009, pp. 92–99.

- [32] V. W. Freeh and D. K. Lowenthal, “Using multiple energy gears in MPI programs on a power-scalable cluster,” in *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 2005, pp. 164–173.
- [33] C.-h. Hsu and W.-c. Feng, “A power-aware run-time system for high-performance computing,” in *Proceedings of the 19th Annual International Conference on Supercomputing*. IEEE Computer Society, 2005, p. 1.
- [34] R. Ge, X. Feng, W.-c. Feng, and K. W. Cameron, “CPU MISER: A performance-directed, run-time system for power-aware clusters,” in *International Conference on Parallel Processing*. IEEE, 2007, pp. 18–18.
- [35] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, “Bounding energy consumption in large-scale MPI programs,” in *Proceedings of the 21st Annual International Conference on Supercomputing*. ACM, 2007, p. 49.
- [36] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, “Adagio: Making DVS practical for complex HPC applications,” in *Proceedings of the 23rd Annual International Conference on Supercomputing*. ACM, 2009, pp. 460–469.
- [37] K. Kandalla, E. P. Mancini, S. Sur, and D. K. Panda, “Designing power-aware collective communication algorithms for InfiniBand clusters,” in *39th International Conference on Parallel Processing*. IEEE, 2010, pp. 218–227.
- [38] A. Venkatesh, A. Vishnu, K. Hamidouche, N. Tallent, D. Panda, D. Kerbyson, and A. Hoisie, “A case for application-oblivious energy-efficient MPI runtime,” in *Proceedings of SC15, the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2015, pp. 1–12.
- [39] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O’Reilly, and S. Amarasinghe, “Opentuner: An extensible framework for program autotuning,” in *Proceedings of the 23rd international conference on Parallel architectures and compilation*. ACM, 2014, pp. 303–316.
- [40] J. Li and J. F. Martinez, “Dynamic power-performance adaptation of parallel computation on chip multiprocessors,” in *The Twelfth International Symposium on High-Performance Computer Architecture*. IEEE, 2006, pp. 77–87.
- [41] R. Miceli, G. Civario, A. Sikora, E. César, M. Gerndt, H. Haitof, C. Navarrete, S. Benkner, M. Sandrieser, L. Morin, and F. Bodin, “AutoTune: A plugin-driven approach to the automatic tuning of parallel applications,” in *International Workshop on Applied Parallel Computing*. Springer, 2012, pp. 328–342.
- [42] O. Sarood, A. Langer, A. Gupta, and L. Kale, “Maximizing throughput of overprovisioned HPC data centers under a strict power budget,” in *Proceedings of SC14, the International Conference*

*for High Performance Computing, Networking, Storage and Analysis.* IEEE Press, 2014, pp. 807–818.

- [43] D. Abramson, R. Sasic, J. Giddy, and B. Hall, “Nimrod: a tool for performing parametrised simulations using distributed workstations,” in *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing.* IEEE, 1995, pp. 112–121.
- [44] W. Sudholt, K. K. Baldridge, D. Abramson, C. Enticott, S. Garic, C. Kondric, and D. Nguyen, “Application of grid computing to parameter sweeps and optimizations in molecular modeling,” *Future Generation Computer Systems*, vol. 21, no. 1, pp. 27–35, 2005.
- [45] S. D. Olabarriaga, A. J. Nederveen, and B. O’Nuallain, “Parameter sweeps for functional MRI research in the virtual laboratory for e-science project,” in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid.* IEEE Computer Society, 2007, pp. 685–690.
- [46] D. Abramson, A. Lewis, T. Peachey, and C. Fletcher, “An automatic design optimization tool and its application to computational fluid dynamics,” in *Proceedings of SC01: the 2001 ACM/IEEE Conference on Supercomputing.* IEEE, 2001, pp. 47–47.
- [47] D. H. Woo and H.-H. S. Lee, “Extending Amdahl’s law for energy-efficient computing in the many-core era,” *Computer*, vol. 41, no. 12, 2008.
- [48] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, “A roofline model of energy,” in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing.* IEEE, 2013, pp. 661–672.
- [49] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, and M. Püschel, “Applying the roofline model,” in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).* IEEE, 2014, pp. 76–85.
- [50] J. Hofmann and D. Fey, “An ECM-based energy-efficiency optimization approach for bandwidth-limited streaming kernels on recent Intel Xeon processors,” in *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing.* IEEE Press, 2016, pp. 31–38.
- [51] S. Song, C.-Y. Su, R. Ge, A. Vishnu, and K. W. Cameron, “Iso-energy-efficiency: An approach to power-constrained parallel computation,” in *2011 IEEE International Parallel & Distributed Processing Symposium.* IEEE, 2011, pp. 128–139.
- [52] L. Ramapantulu, B. M. Tudor, D. Loghin, T. Vu, and Y. M. Teo, “Modeling the energy efficiency of heterogeneous clusters,” in *2014 43rd International Conference on Parallel Processing.* IEEE, 2014, pp. 321–330.

- [53] G. Mitra, A. Haigh, A. Varghese, L. Angove, and A. P. Rendell, “Split wisely: When work partitioning is energy-optimal on heterogeneous hardware,” in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2016, pp. 781–788.
- [54] A. Varghese, J. Milthorpe, and A. P. Rendell, “Performance and energy analysis of scientific workloads executing on LPSoCs,” in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2017, pp. 113–122.
- [55] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. R. de Supinski, and M. Schulz, “A regression-based approach to scalability prediction,” in *Proceedings of the 22nd Annual International Conference on Supercomputing*. ACM, 2008, pp. 368–377.
- [56] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, “Methods of inference and learning for performance modeling of parallel applications,” in *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 2007, pp. 249–258.
- [57] C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, C.-Y. Su, and K. Cameron, “Power-aware predictive models of hybrid (MPI/OpenMP) scientific applications on multicore systems,” *Computer Science-Research and Development*, vol. 27, no. 4, pp. 245–253, 2012.
- [58] B. Johnston, G. Falzon, and J. Milthorpe, “Opencl performance prediction using architecture-independent features,” in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2018, pp. 561–569.
- [59] K. Yoshii, K. Iskra, R. Gupta, P. Beckman, V. Vishwanath, C. Yu, and S. Coghlan, “Evaluating power-monitoring capabilities on IBM Blue Gene/P and Blue Gene/Q,” in *2012 IEEE International Conference on Cluster Computing*. IEEE, 2012, pp. 36–44.
- [60] L. DeRose, B. Homer, D. Johnson, S. Kaufmann, and H. Poxon, “Cray performance analysis tools,” in *Tools for High Performance Computing*. Springer, 2008, pp. 191–199.
- [61] S. Desrochers, C. Paradis, and V. M. Weaver, “A validation of DRAM RAPL power measurements,” in *Proceedings of the Second International Symposium on Memory Systems*. ACM, 2016, pp. 455–470.
- [62] J. H. Laros, P. Pokorny, and D. DeBonis, “Powerinsight-a commodity power measurement capability,” in *2013 International Green Computing Conference Proceedings*. IEEE, 2013, pp. 1–6.
- [63] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, “Powermon: Fine-grained and integrated power monitoring for commodity computer systems,” in *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*. IEEE, 2010, pp. 479–484.



- [64] Intel Corp, *System Programming Guide, volume 3B-2 of Intel 64 and IA-32 Architectures Software Developer's Manual*, Intel Corp, 2016.
- [65] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE transactions on computers*, vol. 100, no. 9, pp. 948–960, 1972.
- [66] D. Molka, D. Hackenberg, R. Schöne, and W. E. Nagel, "Cache coherence protocol and memory performance of the Intel Haswell-EP architecture," in *2015 44th International Conference on Parallel Processing*. IEEE, 2015, pp. 739–748.
- [67] J. Hofmann, G. Hager, G. Wellein, and D. Fey, "An analysis of core-and chip-level architectural features in four generations of Intel server processors," in *International Supercomputing Conference*. Springer, 2017, pp. 294–314.
- [68] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.
- [69] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Mobile Computing*. Springer, 1994, pp. 449–471.
- [70] R. F. Van der Wijngaart and T. G. Mattson, "The parallel research kernels," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2014, pp. 1–6.
- [71] Intel Corp, "Intel VTune Amplifier," 2019. [Online]. Available: <https://software.intel.com/en-us/vtune>
- [72] Cray Inc, *Cray Performance Measurement and Analysis Tools User Guide (6.5.2)*, Cray Inc, 2017. [Online]. Available: [https://pubs.cray.com/pdf-attachments/attachment?pubId=00471748-DB&attachmentId=pub\\_00471748-DB.pdf](https://pubs.cray.com/pdf-attachments/attachment?pubId=00471748-DB&attachmentId=pub_00471748-DB.pdf)
- [73] Linux Kernel Organization, "Linux perf tools," 2019. [Online]. Available: <https://perf.wiki.kernel.org/>
- [74] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with PAPI-C," in *Tools for High Performance Computing 2009*. Springer, 2010, pp. 157–173.
- [75] S. Patil and D. J. Lilja, "Statistical methods for computer performance evaluation," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 1, pp. 98–106, 2012.
- [76] A. Loy, L. Follett, and H. Hofmann, "Variations of Q–Q plots: The power of our eyes!" *The American Statistician*, vol. 70, no. 2, pp. 202–214, 2016.
- [77] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.
- [78] L. Schumaker, *Spline functions: Basic theory*. Cambridge University Press, 2007.

- [79] G. Wilkinson and C. Rogers, “Symbolic description of factorial models for analysis of variance,” *Applied Statistics*, pp. 392–399, 1973.
- [80] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.
- [81] R. J. Carroll, D. Ruppert, C. M. Crainiceanu, and L. A. Stefanski, *Measurement error in nonlinear models: A modern perspective*. Chapman and Hall/CRC, 2006.
- [82] R. Buyya, D. Abramson, and J. Giddy, “Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid,” in *Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, vol. 1. IEEE, 2000, pp. 283–289.
- [83] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [84] S. Seabold and J. Perktold, “Statsmodels: Econometric and statistical modeling with Python,” in *Proceedings of the 9th Python in Science Conference*, vol. 57. SciPy society Austin, 2010, p. 61.
- [85] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [86] T. Givargis, F. Vahid, and J. Henkel, “System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip,” in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*. IEEE, 2001, pp. 25–30.
- [87] W. McKinney, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51–56.
- [88] S. Gillies *et al.*, “Shapely: manipulation and analysis of geometric objects,” [toblerity.org](http://toblerity.org), 2007–. [Online]. Available: <https://github.com/Toblerity/Shapely>
- [89] J. D. Hunter, “Matplotlib: A 2D graphics environment,” *Computing in Science and Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

- [90] J. Park, M. Smelyanskiy, U. M. Yang, D. Mudigere, and P. Dubey, “High-performance algebraic multigrid solver optimized for multi-core based distributed parallel systems,” in *Proceedings of SC15, the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2015, pp. 1–12.
- [91] S. Plimpton, “Fast parallel algorithms for short-range molecular dynamics,” *Journal of Computational Physics*, vol. 117, no. 1, pp. 1–19, 1995. [Online]. Available: <http://lammps.sandia.gov>
- [92] I. Karlin, J. Keasler, and R. Neely, “Lulesh 2.0 updates and changes,” Lawrence Livermore National Laboratory, Tech. Rep. LLNL-TR-641973, August 2013.
- [93] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, Z. Liu, J. Berner, W. Wang, J. G. Powers, M. G. Duda, D. M. Barker, and X.-Y. Huang, “A description of the advanced research WRF version 4,” National Center for Atmospheric Research, Tech. Rep. NCAR/TN-556+STR, 2019.



# Appendix A

## HPCProbe User Manual

The HPCProbe framework is comprised of the HPCProbe, HPCModel, and HPCPlot modules. User documentation for each module is provided here.

### A.1 HPCProbe Module

This section provides user documentation for the HPCProbe module. If the HPCProbe packages are installed, HPCProbe command line arguments help can also be viewed using:

```
$ python hpcprobe.py --help
```

API documentation is available in section B.1.

#### A.1.1 Command Line Options

The following options are only available as `hpcprobe.py` command line parameters.

`-h, --help`

Show the help message and exit.

`-i <init file>`

Read experiment configuration settings from the specified initialisation file and start the experiment.

Example: `python hpcprobe.py -i \`  
`$HOME/work/exp.stencil-f-e-h-20-4/stencil.ini`

-j <job id>

Suffix for the experiment results file name, results/job-<job id>. The results for each measurement run are stored here.

### A.1.2 Initialisation File Options

The following options can be set in the initialisation file or as command line parameters.

expName = <exp name>

Set the name for the experiment.

Example: expName = exp.stencil-f-e-h-20-4

Command line: -n <exp name>

expExec = <executeable>

Set the experiment executable file and required command line options. The threadCount, cpuFrequency, and execParam experiment parameters are substituted for \$th, \$fq, and \$p occurrences in this setting. Environment variables are expanded, unless prefixed with \\$.

Example: expExec = \$HOME/work/bin/stencil\_mpi\_omp+pat \\${OMP\_NUM\_THREADS} 5 \$p

Command line: -x <executeable>

workFolder = <work folder>

Set the working folder for the experiment executable. Use this option if the executable expects to find required supporting files in its working directory.

Example: workFolder = \$HOME/work/WRF/test/em\_real

Command line: --wf <work folder>

outputFolder = <output folder>

Set the output folder for experiment. An experiment results folder with the name specified in expName is created in the outputFolder.

Example: outputFolder = \$HOME/work

Command line: -o <output folder>

threadCount = <thread count>|<start,end,step>

Set the total thread count to use across all nodes. Use a single count for non-batch mode

experiments, or a start count, end count, and step size for batch experiments.

Example: `threadCount = 80,880,80` (for 4 to 44, step 4, threads across 20 nodes)

Command line: `-t <thread count>|<start,end,step>`

`cpuFrequency = <cpu frequency>|<start,end,step>`

Set the CPU frequency cap to use in kHz. Use a single frequency for non-batch mode experiments, or a start frequency, end frequency, and step size for batch experiments.

Example: `cpuFrequency = 1200000,2200000,100000` (for 1.2 GHz to 2.2 GHz, step 100 MHz)

Command line: `-f <cpu frequency>|<start,end,step>`

`execParam = <exec param>|<start,end,step>`

Set a variable parameter for the experiment executable. Use a single value for non-batch mode experiments, or a start value, end value, and step value for batch experiments.

Example: `execParam = 200000,200000,200000` (for 200,000 fixed across runs in batch mode)

Command line: `-p <exec param>|<start,end,step>`

`walltime = <walltime>`

Set the target wall time for the job. Use the PBS default if omitted.

Example: `walltime = 08:00:00` (8 hours)

Command line: `-w <walltime>`

`account = <account>`

Set the target account string for the job. If provided, a `#PBS -A <account>` statement is added to the PBS job.

Command line: `-a <account>`

`queue = <queue>`

Set the target queue for the job. Let PBS select the queue if omitted.

Command line: `-q <queue>`

`env = <n1=v1,n2=v2,...,nn=vn>`

Set environment variables required for the job.

Example: env = PAT\_RT\_PERFCTR=\

"PM\_ENERGY:NODE,PM\_POWER:NODE,UNITS:POWER,UNITS:ENERGY" (to select CrayPAT performance counters)

Command line: -e <n1=v1,n2=v2,...,nn=vn>

-m <cmd1,cmd2,...,cmdn>

List of subcommands for managing module files. Each subcommand is run using module <cmd>.

Example: modules = load perftools (runs module load perftools)

Command line: modules = <cmd1,cmd2,...,cmdn>

repeatCount = <repeat count>

Set a repeat count for each run so statistical data can be generated. The statistics include arithmetic mean, geometric mean, harmonic mean, maximum, median, minimum, standard deviation, *t* confidence interval, and variance. Only odd repeat counts are permitted to ensure the median value corresponds to an actual sample.

Example: repeatCount = 5

Command line: -r <repeat count>

nodes = <node count>

Set the target node count for the experiment.

Example: nodes = 20

Command line: --nodes <node count>

ppn = <processors per node>

Set the target processors per node for the experiment. This is equivalent to the total number of cores across all CPU sockets for a node.

Example: ppn = 44

Command line: --ppn <processors per node>

limits = <resource limits>

Set resource limits for the job. If provided, a #PBS -l <resource limits> statement is added to the PBS job.

Command line: -l <resource limits>

cpuBinding = <processing elements to cores binding>



Set target processing elements to cores binding. If provided, add to the Cray ALPS aprun command line parameters. See the `--cc <...>` argument in the aprun man page.

Command line: `--cpu-binding <processing elements to cores binding>`

`totalPEs = <total processing elements>`

Set target total processing elements (MPI ranks), to override the default setting of `nodePEs × minimum node count`.

Command line: `--total-pes <total processing elements>`

`nodePEs = <processing elements per node>`

Set target processing elements (MPI ranks) per node.

Example: `nodePEs = 2`

Command line: `--node-pes <processing elements per node>`

`peThreads = <threads per processing element>`

Set target OpenMP threads per processing element (MPI rank), to override the default setting calculated from `ppn` and `nodePEs`.

Command line: `--pe-threads <threads per processing element>`

`nodeSockets = <cpu sockets per node>`

Set the target CPU sockets per node.

Default: 2

Command line: `--node-sockets <cpu sockets per node>`

`socketRings = <core rings per cpu socket>`

Set target core rings per CPU socket, for Intel Cluster-on-Die experiments.

Default: 2

Command line: `--socket-rings <core rings per cpu socket>`

`batch = True|False`

Enable batch mode using Nimrod/O to automate the experiment through the provided `threadCount`, `cpuFrequency`, and `execParam` parameter ranges.

Command line: `--batch` (disabled if omitted)

`batchContinue = True|False`

Continue with an incomplete batch mode run. Searches through a partial YAML summary file to locate the last result, then continues the experiment from that point. Attempts to continue a run that exited before completion.

Command line: `-c` (disabled if omitted)

`omp = True|False`

If enabled, set the OpenMP `OMP_NUM_THREADS` environment variable to the thread count per MPI rank, calculated using the total thread and rank counts.

Command line: `--omp` (disabled if omitted)

`base2 = True|False`

Set thread count to base 2 exponent  $\times$  ppn. Allows exponential node count steps such as 1, 2, 4, 8, 16, 32, 64, etc.

Command line: `--base2` (disabled if omitted)

`reget = True|False`

Re-get results collected in a prior run. Attempts to re-analyse results collected in a previous run. Use when debugging results parser changes. Node allocation settings are also overridden to use one node only.

Command line: `--reget` (disabled if omitted)

`retries = <count>`

Retries count if `execExec` or `postExec` fail. Try to continue the experiment when intermittent failures occur.

Default: 0

Command line: `--retries <count>`

`threadPlacement = C|S|R`

Compact/Scatter/Ring thread placement. Compact allocates all cores in one socket before moving to another, while scatter utilises all sockets uniformly. Ring allocates all CPU core rings uniformly.

Default: S

Command line: `--tp c|s|r`

`nodePlacement = C|S`

Compact/Scatter node placement. Compact allocates all cores in one node before moving to another, while scatter utilises all nodes uniformly.

Default: S

Command line: `--np c|s`

`postExec = <executable>`

Post experiment executable file. Performs post processing of the experiment data generated by the executable run.

Example: `postExec = mv "%(dataPath)s" "%(jobResults)s"; \`

`pat_report "%(jobResults)s" (move the experiment data to the results folder and run pat_report)`

Command line: `--post-exec <executable>`

`accWaitLi = <acc wait line num>`

Include performance counter group data reported for the GPU/accelerator synchronisation wait at the given source code line number in the executable.

Example: `accWaitLi = 273` (to parse counters reported under group

`USER / main.ACC_SYNC_WAIT@li.273)`

Command line: `--acc-wait <acc wait line num>`

`accCopyLi = <acc copy line num>`

Include performance counter group data reported for the GPU/accelerator copy at the given source code line number in the executable.

Example: `accCopyLi = 273` (to parse counters reported under group

`USER / main.ACC_COPY@li.273)`

Command line: `--acc-copy <acc copy line num>`

`execLoopLi = <loop line num>`

Include performance counter group data reported for the loop at the given source code line number in the executable.

Example: `execLoopLi = 273` (to parse counters reported under group

`USER / main.LOOP@li.273)`

Command line: `--loop <loop line num>`

`execRegion = <region num>`

Include performance counter group data reported for the region at the given source code line number in the executable. Regions are defined within the experiment executable using CrayPAT API calls:

```
PAT_region_begin(<region num>, <region label>)
```

```
PAT_region_end(<region num>)
```

Example: `execRegion = 2` (to parse counters reported under group

```
USER / #2.<...>)
```

Command line: `--region <region num>`

`runtime = <runtime>`

Override the detected runtime. The `LOADEDMODULES` environment variable is used to detect `cray_mpi`, `open_mpi`, or `intel_mpi` runtimes, for selecting runtime specific job submission parameters.

Command line: `--runtime <runtime>`

### A.1.3 Counter Regular Expressions File

The HPCProbe counter data parser is configurable using the counter regular expressions YAML file. This file must be in the same folder as `hpcprobe.py` with the file name `hpcprobe.yml`. The top-level YAML file entries are for parsing output from running `expExec` (`run`) and `postExec` (`post`). The group 1-n second-level entries represent sections in the output text to be parsed. The YAML file outline is:

`run:`

```
<group 1>:
```

```
  groups: [{ name: ..., regex ... }, ...]
```

```
  counters: [{ name: ..., regex ... }, ...]
```

`post:`

```
<group 2>:
```

```
  groups: [...]
```

```
  counters: [...]
```

```
<group n>:
```

```
  groups: [...]
```

```
  counters: [...]
```

The groups entries define the list of regular expressions for extracting required subsections from a

---

### Listing A.1: Sample HPCProbe Initialisation File

---

```
# MPI+OPENMP stencil execution on 2D grid
[Experiment]
batch = True
cpuFrequency = 1200000,2200000,100000
env = PAT_RT_PERFCTR="PM_ENERGY:NODE,PM_POWER:NODE,UNITS:POWER,UNITS:ENERGY"
execParam = 200000,200000,200000
expExec = $HOME/work/bin/stencil_mpi_omp+pat \${OMP_NUM_THREADS} 5 $p
expName = exp.stencil-f-e-h-20-4
modules = load perftools
nodePEs = 2
nodePlacement = S
nodes = 20
omp = True
outputFolder = $HOME/work
postExec = mv "%(dataPath)s" "%(jobResults)s"; pat_report "%(jobResults)s"
ppn = 44
retries = 2
threadCount = 80,880,80
threadPlacement = S
```

---

section of text.

- The optional `param` key allows use of an input parameter in the group regular expression, for example `execLoopLi`.

The `counters` entries define the list of regular expressions for extracting the required counters from groups. The list of counter names is populated using the match groups defined in the counter regular expression.

- The optional `findall` key gets all matches in the text section.
- The optional `hide` key excludes a counter from the results summary file.

## A.1.4 Sample Initialisation File

Listing A.1 shows a sample initialisation file for HPCProbe. An initialisation file consists of an `Experiment` section followed by configuration key/value pairs. This sample combines a number of the example settings from section A.1.2 to specify an experiment using the stencil kernel on 20 nodes. The experiment can be started as section A.1.1 describes.

## A.1.5 Experiment Results Summary File

The results summary for an experiment is written to the `summary.yml` file located in the experiment folder, `<outputFolder>/<expName>`. For the sample in Listing A.1, the summary file location is:

```
$HOME/work/exp.stencil-f-e-h-20-4/summary.yml
```

The summary file contains a list of results for each measurement run in the experiment in YAML format. The results for each run have keys for parameters, objectives, and (if repeatCount is greater than 1) stats, as follows:

```
- objectives: {...}
  parameters: {...}
  stats: {...}
- objectives: {...}
  parameters: {...}
  stats: {...}
- ...
```

The parameters field uses key/value pairs to define the experiment configuration for a measurement run. The objectives field captures metrics from the run as key/value pairs. The stats field includes generated statistical data for each objective. The initialisation and results summary files provide a record of completed experiments.

## A.2 HPCModel Module

This section provides user documentation for the HPCModel module. API documentation is available in section B.2.

The BS and NN forecasts and trade-off analysis for the Chapter 6 evaluation are generated using the HPCModel module. The HPCModel calls for stencil kernel predictions and trade-off analysis are described, as used in 6.2.1.

### A.2.1 B-Spline Regression Model

Start by importing the required libraries.

```
>>> import json
>>> import numpy as np
>>> import shapely.geometry
>>> import hpcplot as hplt
>>> import hpcmodel as hmdl
```

Load the required YAML files into a pandas DataFrame. The kernel id is needed for indexing Kernel definitions. It can be found in the DataFrame 'kl' column.

```
>>> ymls = ['stencil-f-e-h-64-4.yml']
>>> pd = hplt.PlotDf(ymls, basePredictors=True)
>>> knl = hplt.Kernels.idsByNum[pd.df['kl'].unique()[0]]
```

Set up train and test data selectors. The train data selector will use samples at thread counts of 8, 12, 32, and 44, at frequency caps of 1.2, 1.9, and 2.2 GHz. The test data selector will generate a  $1 \times 11 \times 11$  product of the given node, thread, and frequency ranges for testing the model.

```
>>> train = {'custom': {
...     'nd': [64] * 12,
...     'tn': [8, 20, 32, 44] * 3,
...     'fq': list(np.repeat([1200000, 1900000, 2200000], 4))}
... }
>>> test = {
...     'nd': [64],
...     'tn': range(4, 45, 4),
...     'fq': range(1200000, 2200001, 100000)
... }
```

Use the Kernel definitions to set up required meta data for each response. The responses list includes the response column id, type (Energy/Performance), units, and scale ( $10^6$  for MFlops/s for example). The lims dictionary specifies observed and predicted error limits for each response, and if the response needs to be maximised or minimised.

```
>>> resps = hplt.Kernels.tradeoffs[knl]
>>> responses = [
...     { 'col':resp,
...       'type':hplt.Columns.respType[resp],
...       'unit':hplt.Kernels.getUnits(knl, resp),
...       'scale':hplt.Columns.scale[resp]
...     } for resp in resps
... ]
>>> lims = {
...     resp:dict(
```

```
...     list(hplt.Kernels.lims[knl][resp].items()) +
...     list({'pareto_max':hplt.Columns.paretoMax[resp]}.items())
... ) for resp in resps
... }
```

Create the BS model and generate response predictions for the test data. The options used add the *log* transform, disable scaling of predictors and responses, include second-order model term interactions, set three degrees of freedom and polynomial degree of two, and remove the interaction term between frequency and node count.

```
>>> rmBS = hmdl.SplineReg(pd.df, responses, pd.predictors,
...   train=train, test=test, transform='log', scalePredictors=False,
...   scaleResponses=False, scaleFirst=False, order=2, dfree=3, degree=2,
...   rmterms=['^bs([fqnd])*[nd]*[nd]*$'], lims=lims
... )
```

Removal of the interaction term can be seen by inspecting `rmBS.interacts`. The fourth term in the following list matches a regular expression passed using the `rmterms` argument so it will be removed.

```
>>> print(rmBS.interacts)
[
  'bs(fq,df=3,degree=2)',
  'bs(nd,df=3,degree=2)',
  'bs(tn,df=3,degree=2)',
  'bs(fq,df=3,degree=2):bs(nd,df=3,degree=2)',
  'bs(fq,df=3,degree=2):bs(tn,df=3,degree=2)',
  'bs(nd,df=3,degree=2):bs(tn,df=3,degree=2)'
]
```

The `df_test` DataFrame includes test data with observed and model predicted data for the responses.

```
>>> print(rmBS.df_test[['nd', 'tn', 'fq', 'ee_obs', 'ee', 'ar_obs', 'ar']])
```

|    | nd | tn | fq      | ee_obs    | ee        | ar_obs       | ar           |
|----|----|----|---------|-----------|-----------|--------------|--------------|
| 0  | 64 | 4  | 1200000 | 74.700296 | 89.438958 | 5.659213e+05 | 7.534808e+05 |
| 1  | 64 | 4  | 1300000 | 78.038206 | 92.522248 | 6.071655e+05 | 8.064514e+05 |
| .. | .. | .. | ...     | ...       | ...       | ...          | ...          |



```

119  64  44  2100000  151.949680  150.983203  2.831927e+06  2.818231e+06
120  64  44  2200000  148.647866  148.647866  2.832907e+06  2.832907e+06

```

[121 rows x 7 columns]

### A.2.2 Neural Network Model

The NN model uses the same set up process as the BS model. Once the BS model has run, the NN model can run using the same inputs.

Create the NN model and generate response predictions for the test data. The options used add the *log* transform, enable scaling of predictors and responses after the transform, disable the scikit-learn scaler and min-max scaling, set three hidden layers with 10, 20, and 10 hidden units, and 2,000 training steps.

```

>>> rmNN = hmdl.DnnReg(pd.df, responses, pd.predictors,
...   train=train, test=test, transform='log', scalePredictors=True,
...   scaleResponses=True, scaleFirst=False, scaleSkllearn=False,
...   scaleMinMax=False, h_units=[10, 20, 10], steps=2000, lims=lims
... )

```

The `df_test` DataFrame contains test data with observed and model predicted data for the responses.

```

>>> print(rmNN.df_test[['nd', 'tn', 'fq', 'ee_obs', 'ee', 'ar_obs', 'ar']])

```

|     | nd | tn | fq      | ee_obs     | ee         | ar_obs       | ar           |
|-----|----|----|---------|------------|------------|--------------|--------------|
| 0   | 64 | 4  | 1200000 | 74.700296  | 110.565677 | 5.659213e+05 | 1.047328e+06 |
| 1   | 64 | 4  | 1300000 | 78.038206  | 112.865555 | 6.071655e+05 | 1.068696e+06 |
| ..  | .. | .. | ...     | ...        | ...        | ...          | ...          |
| 119 | 64 | 44 | 2100000 | 151.949680 | 151.280692 | 2.831927e+06 | 2.828063e+06 |
| 120 | 64 | 44 | 2200000 | 148.647866 | 148.647861 | 2.832907e+06 | 2.832910e+06 |

[121 rows x 7 columns]

### A.2.3 Pareto Trade-Off Analysis

Pareto trade-off analysis results and model statistics can be accessed using the BS and NN `getStats()` method. Use `json.dumps` to pretty print the statistics dictionary. Use the required key to access a specific statistic, such as `r1PfRmse` and `r2PfRmse` for the Pareto front RMS error for each response.

```

>>> statsBS = rmBS.getStats()
>>> print(json.dumps(statsBS, indent=2, sort_keys=True))
{
  "nobs": 12.0,
  "pfForErr": 5,
  "pfFor01": 57.40794587557292,
  ...
}
>>> print(statsBS['r1PfRmse'])
2.156893824238247
>>> print(statsBS['r2PfRmse'])
2.3333525720331236

```

The `getStats()` method also populates a `paretoFronts` dictionary with observed and forecast Pareto data. The `pf` key provides a `DataFrame` with points lying along the extended Pareto front. The `pts` key accesses a `DataFrame` with points lying within the trade-off zone. The `poly` key provides a polygon object defining the trade-off zone boundaries. *Baseline*,  $P_{min}$ , and  $P_{max}$  values are also provided for each response (<r1> and <r2>), as the Chapter 6 results summary tables show, such as Table 6.3.

```

>>> print(rmBS.paretoFronts)
{
  'obs': {
    'pf': <DataFrame object>,
    'pts': <DataFrame object>,
    'poly': <Polygon object>,
    <r1>: {
      'base': <float>,
      'min': <float>,
      'max': <float>
    },
    <r2>: {
      'base': <float>,
      'min': <float>,
      'max': <float>
    }
  },
  'for': {
    ...

```

```

    }
}

```

Alternatively, the `ParetoFront` class can be used to perform Pareto trade-off analysis directly. This example uses the observed `PlotDf` `DataFrame` `pd.df`, kernel id `kn1`, responses list, and `lims` dictionary from section A.2.1.

Set up trade-off column ids, column ids for search space coordinates of Pareto points, their maximums (the *Baseline* value coordinates), and if trade-off columns need to be maximised or minimised, and the trade-off data set with the required `pd.df` columns.

```

>>> x_col, y_col = lims.keys()
>>> xc_col = 'fq'
>>> yc_col = 'tn'
>>> pe_col = 'pe'
>>> nd_col = 'nd'
>>> xc_max = max(pd.df[xc_col])
>>> yc_max = max(pd.df[yc_col])
>>> pe_max = max(pd.df[pe_col])
>>> nd_max = max(pd.df[nd_col])
>>> pf_max = (lims[x_col]['pareto_max'], lims[y_col]['pareto_max'])
>>> pf_df = pd.df[[x_col, y_col, xc_col, yc_col, pe_col, nd_col]]

```

The first two columns in `pf_df` contain the data points for trade-off analysis. The remaining columns provide the search space context for each point.

Initialise `ParetoFront` with the trade-off data set, maximise/minimise status, include the trade-off zone polygon calculation, and the error limits for each trade-off column.

```

>>> paretoFront = hmdl.ParetoFront(
...     pf_df, pmax=pf_max, poly=True,
...     lims=[lims[x_col]['obs'], lims[y_col]['obs']]
... )

```

Calculate the extended Pareto front points, the trade-off zone points, and the trade-off zone polygon.

```

>>> pf = paretoFront.get()

```

```
>>> pts = paretoFront.getPathPts()
>>> polySeq = paretoFront.getPath().vertices
>>> poly = shapely.geometry.polygon.Polygon(polySeq[0:-1])
```

The *Baseline* value for the first trade-off response occurs at the maximum search space coordinates, such as maximum CPU frequency and thread count. Replace `x_col` with `y_col` for the second response.

```
>>> base = pf_df.loc[
...     (pf_df[xc_col] == xc_max) &
...     (pf_df[yc_col] == yc_max) &
...     (pf_df[pe_col] == pe_max) &
...     (pf_df[nd_col] == nd_max), x_col
... ]
>>> r1Base = base.iloc[0]
>>> print(r1Base)
148.6478662167381
```

Calculate  $P_{min}$  and  $P_{max}$  values for the first trade-off response. Substitute `x_col` with `y_col` for the second response.

```
>>> if lims[x_col]['pareto_max']:
...     r1Min = min(pf[x_col])
...     r1Max = max(pf[x_col])
... else:
...     r1Min = max(pf[x_col])
...     r1Max = min(pf[x_col])
...
>>> print(r1Min)
182.74191461270135
>>> print(r1Max)
215.11328301839225
```

The *Baseline*,  $P_{min}$ , and  $P_{max}$  align with the observed values shown in Table 6.3. The BS and NN analysis can be performed by replacing `pd.df` with the model predicted data, `rmBS.df_test` or `rmNN.df_test`.

## A.3 HPCPlot Module

This section provides user documentation for the HPCPlot module. API documentation is available in section B.3.

### A.3.1 Pareto Plots

Figure 6.2 is generated using the ParetoPlot class. The *Observed* plot uses the observed data loaded into `pd.df` in section A.2.1. The 'ee' and 'ar' DataFrame column ids are plotted against the *x*-axis and *y*-axis. The remaining options set *x*-axis and *y*-axis limits or plot ranges, the 'tn' column id groups data point colours by thread count, a colour bar is added to indicate thread counts by colour, the trade-off zone polygon is plotted, and error limits are applied to the Pareto analysis.

```
>>> pp = hplt.ParetoPlot(pd.df, 'ee', 'ar',
...   x_lim=[50, 225], y_lim=[5e5, 35e5], s_id='tn',
...   colorbar=True, poly=True, lims=True
... )
>>> pp.plot()
```

The *Predicted – B-Spline Model* plot uses the same ParetoPlot initialisation parameters, except `pd.df` is replaced by the BS model predicted data, `rmBS.df_test`.

Similarly, the *Predicted – Neural Network Model* plot uses the NN model predicted data, `rmNN.df_test`, instead of `pd.df`.

### A.3.2 Surface Plots

Figure 6.3 and Figure 6.4 are generated using the SurfPlot class. As with Pareto plots, the *Observed* plot uses `pd.df`, the *Predicted – B-Spline Model* plot uses `rmBS.df_test`, and the *Predicted – Neural Network Model* plot uses `rmNN.df_test`. The 'gh', 'tn', and 'ee' DataFrame columns are plotted against the *x*-axis, *y*-axis, and *z*-axis. The remaining options set *z*-axis limits, the column and values for legend groups, the plot rotation elevation and azimuth, and the plot title.

```
>>> sp = hplt.SurfPlot(pd.df, 'gh', 'tn', 'ee',
...   z_lim=[50, 250], z2_id='nd', z2_vals=[64],
...   z3_id=None, rotate=[40, 10], title='Observed')
>>> sp.plot()
```

The performance surface plots use the same approach, except the column id for the  $z$ -axis changes from 'ee' to 'ar'.

# Appendix B

## HPCProbe API

The HPCProbe API documentation is provided in this section. If the HPCProbe packages are installed, this documentation can also be viewed using the Python pydoc utility, as follows:

```
$ pydoc [hpcprobe|hpcmodel|hpcplot]
```

### B.1 HPCProbe Module

This section provides API documentation for the HPCProbe module. User documentation is available in section A.1.

#### B.1.1 Name

hpcprobe - Automates HPC performance analysis experiments using PBS, Nimrod/O and Cray-PAT.

#### B.1.2 Description

The hpcprobe module captures experiment data reported by the program itself at runtime, and by the configured post-processing utility for analysing experiment data written by the program instrumentation at program runtime completion. The regular expressions library is used to parse performance data output from the program and post-processing utility. The regular expression counter parser configuration is loaded from a YAML configuration file. The experiment ResultSet is written to file in YAML format.

### B.1.3 Classes

builtins.object

Experiment

ResultSet

class **Experiment**(builtins.object)

Configure and run energy tuning experiments, read and parse job results, and perform statistical analysis of the results.

Specifications for parsing command line and/or configuration file parameters for the experiment are defined in **paramSpec**:

**name** : parameter name

**arg** : command line argument, or if omitted the parameter is only available in the configuration file

**metavar** : command line argument name for help

**help** : command line argument brief description for help

**type** : parameter type (eg. 'int', 'float', 'boolean', None is string)

**opts** : list of parameter options:

**noInit** : parameter is not in configuration file

**path** : expand OS environment variables in parameter value

**upper** : convert parameter value to upper case

**setattr** : set Experiment attribute to parameter value

Methods defined here:

**\_\_init\_\_**(self)

Initialise experiment to the default settings.

**analyseResults**(self, resultsList)

Do statistical analysis of the results when there is more than one sample.

**config**(self, initFile=None)

Configure the experiment using the counter regex file, command line arguments and/or optional initialisation file. The counter regex file has the same name and location as the Python module, but with the 'yaml' file extension, eg. hpcprobe.yaml.

**getResults**(self)

Get experiment results by parsing data reported by the program at runtime and by the configured post-processing utility after program runtime.



**run**(self)

Run the configured experiment.

Data and other attributes defined here:

**coreRounding** = 1

**execRounding** = 1

**expLogDefault** = 'exp.log'

**expLogFormat** = '%(levelname)s:%(funcName)s:%(message)s'

**freqRounding** = 100000

**jobLogFormat** = '%(message)s'

**jobLogName** = 'job'

**logLevel** = 20

**nimrodoExhShd** = 'parameter th integer range from {0} to {1}  
step ...m\...

**nimrodoPathDefault** = '\$HOME/local/bin'

**nimrodoResults** = 'final'

**paramSpec** = [{'arg': 'n', 'help': 'experiment name',  
'metavar': '<exp ...

**parser** = None

**pbsJob** = '#!/bin/bash\n\n#PBS -V\n#PBS -N {0}\n\n# #PBS -d {1}\n#...i\ne...

**pbsJobOmp** = 'export OMP\_NUM\_THREADS={0}\n\n# echo OMP\_NUM\_THREADS=\$OMP\_NUM...

**percentLimitDefault** = 0.2

**walltimePbsDefault** = '08:00:00'

class **ResultSet**(builtins.object)

Captures input parameters and output objectives for each experiment run.

Methods defined here:

**\_\_init\_\_**(self, regexList, inData=None)

Intialise the ResultSet including regex configuration for parsing counter data, and logger configuration. The inData argument provides the list of experiment parameters.

**\_\_str\_\_**(self)

Provides a YAML string representation of results.

**append**(self, regexList, inLog)

Public accessor to parse counter data in the given log file and append it to the results.

## B.1.4 Functions

### **checkPythonVer()**

Check the minimum required Python version is in use. Print help on setting up current version in user home directory if not.

### **float\_representer(dumper, data)**

Custom float representer to use 5 decimal point floats and NaN in generated YAML files. The data parameter contains the float being represented.

## B.1.5 Data

```
__contact__ = 'mark.endrei@uq.edu.au'
```

```
__copyright__ = 'Copyright 2019, The University of Queensland'
```

## B.1.6 File

```
./lib/python3.6/site-packages/hpcprobe.py
```

## B.2 HPCModel Module

This section provides API documentation for the HPCModel module. User documentation is available in section A.2.

### B.2.1 Name

hpcmodel - Models for predicting HPC system performance and energy responses.

### B.2.2 Description

The hpcmodel module provides basic polynomial, B-spline, and deep learning predictive model implementations that extend a common Model base class. It also provides a number of vectorised data transforms and scalers that can improve model performance. Transforms for mitigating normal data distribution deviations include log, exponential and root transforms. Data scalers for improving machine learning convergence include min-max normalisation and standardisation.

### B.2.3 Classes

builtins.object

Model

DnnReg

PolyReg

SplineReg

ParetoFront

SiPfx

Transform

class **DnnReg**(Model)

Deep neural network regressor model implemented using scikit-learn and TensorFlow. Regressor networks allow prediction of continuous values, like performance or energy usage, so DnnReg uses the DNNRegressor class. The model is initialised using a constant random seed to ensure consistent, repeatable results from run to run. Scalers are provided to improve model convergence when inputs and outputs that have different scales and ranges.

Method resolution order:

DnnReg

Model

builtins.object

Methods defined here:

```
__init__(self, df, responses, predictors, train=None,
          test=None, transform=None, scalePredictors=None,
          scaleResponses=None, scaleSkllearn=None,
          scaleMinMax=None, scaleFirst=None, scaleLims=None,
          lims=None, h_units=None, steps=None, saveSteps=
          None)
```

Initialise the model parameters, including:

**h\_units** : list defining the number of hidden units per network layer

**steps** : number of training steps

**saveSteps** : changes the step count for model convergence logging

See the Model class for other parameter descriptions.

```
fitModel(self, response)
```

Fit the neural network model to the training data for the given response.

```
getInputFn(self, df, response, num_epochs=None, shuffle=False)
```

Provide the model with the required feature names and values (predictors or input parameters) and the label name and values (response or output parameter).

```
getTrainingLosses(self, weight=0.6)
```

Get training loss data for plotting the model convergence curve (loss versus training steps). Weight is used for smoothing the curve.

```
predict(self, response)
```

Predict the given system response for the test data using the trained neural network model.

```
smooth(self, vals, weight=0.6)
```

Filter data using exponential smoothing with the provided weight (or smoothing factor), where 0.0 is no smoothing and 1.0 is maximum smoothing.

Data and other attributes defined here:

```
h_units = [10, 20, 10]
```

```
steps = 2000
```

Methods inherited from Model:

```
checkAllPredictors(), getDfColRange(), getDfFiltered(), getDfLhs(), getDfSelect(),  
getErrorQt(), getInterpRmsError(), getPcErr(), getPcRng(), getRmsError(), getRsq(),  
getStats(), getUnitStepPts(), kFoldValidate(), lhsRemap(), setParetoFronts(), setRe-  
sponses(), setStats(), setTestDf(), setTrainingDf()
```

Data and other attributes inherited from Model:

```
custom, lhs, obs, uniform
```

```
class Model(builtins.object)
```

Base class for HPC models.

Methods defined here:

```
__init__(self, df, responses, predictors, train=None,  
         test=None, transform=None, scalePredictors=None,  
         scaleResponses=None, scaleSklearn=None,  
         scaleMinMax=None, scaleFirst=None, scaleLims=None,  
         lims=None)
```

Initialise the model parameters, including:

```
df : DataFrame containing full data set
```

**predictors** : df column ids for model predictors or inputs  
**responses** : df column ids for model responses or outputs  
**train** : selector for training data subset from df  
**test** : selector for test data subset from df  
**lims** : error limits for predictors, defined in hpcplot.Kernels  
transform, scaler configuration : see hpcmodel.Transform

**checkAllPredictors**(self, predictors)

Check the given predictors are all in the known list of predictors.

**getDfColRange**(self, df, col)

Get value range information for the given DataFrame column, including min, max, step, and length.

**getDfFiltered**(self, df, filters)

Filter the DataFrame rows using the provided dictionary, eg to select matching values from 'nd', 'tn', 'fq' columns use:

```
{'nd':[64],'tn':[8,20,32,44],'fq':[12e5,17e5,22e5]}
```

**getDfLhs**(self, selects)

Select training data using Latin Hypercube Sampling.

**getDfSelect**(self, selects)

Select training and test data from the DataFrame data set using the Cartesian product of the given column values.

**getErrorQt**(self, a\_list, b\_list)

Calculate the counts of coincident points in the a and b lists, and the points within 1, 2, 3, and 4 units of measure.

**getInterpRmsError**(self, forecast, obs, axis=0, resps=None, num=50)

Interpolate the forecast and observed data to the same line space and the calculate RMS error between the two.

**getPcErr**(self, v1, v2, rnd=1)

Get the percentage error between v1 and v2, rounded to the given number of decimal places.

**getPcRng**(self, vbase, vmin, vmax)

Get the percentage range between a baseline value and the given minimum and maximum values.

**getRmsError**(self, forecast, obs)

Calculate the RMS error between a set of points with the same intervals. Return RMS value of the forecast data, RMS error between the forecast and observed data, and RMS error as a percentage of the forecast RMS.

**getRsq**(self, y, y\_pred)

Calculate the R-squared statistic by dividing the residual sum of squares by the total sum of squares.

**getStats**(self)

Get statistics formatted for model results summary reporting.

**getUnitStepPts**(self, pts)

Scale points to have a unit step between unique values, eg. so 4 to 44 step 4 threads becomes 0 to 10 step 1.

**kFoldValidate**(self, response, k=3)

Perform k-fold validation of model for the required number of folds, k, and predicted response using the full dataset.

**lhsRemap**(self, x\_lhs, x\_min, x\_max, x\_step)

Remap Latin Hypercube Sampling output values ranging from 0.0 to 1.0 to actual values using the given min, max, and step.

**setParetoFronts**(self)

Set observed and forecast Pareto front points for the data set that are within error limits.

**setResponses**(self, responses)

Set the primary and secondary/trade-off responses to be modelled. The responses argument is a list of dictionaries with the following keys:

**col** : column id

**type** : 'Energy' or 'Performance' response

**unit** : column units

**scale** : multiplier for units

The **type**, **unit**, and **scale** keys are defined in hpcplot.Columns.

**setStats**(self)

Set up the model statistics dictionary with the following structure:

```
'resp': {  
    <col_id_1>: {  
        'nobs': ., 'rsq': ., 'rms': ., 'rmse': ., 'rmsepc': .,  
        'pf': {'rms': ., 'rmsepc': ., 'rmse': .},  
        'for': {'base': ., 'min': ., 'max': .},
```

```

    'obs': {.}},
    <col_id_2>: {.}},
    'pf': {
        'obs': {'lim': ., 'pf': <df>, 'pts': ., 'qt': [.],
        'xylen': [.], 'overlap': .},
        'for': {.}}

```

Each response has a 'resp' column id key with statistics including:

**nobs** : number of observations

**rsq** : R-squared statistic

**rms**, **rmse**, **rmsepc** : response RMS value, RMS Error and percentage RMS error between observed and forecast values

**pf** : Pareto points statistics for response (within trade-off zone)

rms, rmse, **rmsepc** : response RMS value, RMS Error and percentage RMS error between observed and forecast values on the front

Statistic for the observed and forecast Pareto fronts include:

**lim** : percentage error limit

**pf** : DataFrame with Pareto points in the search space context (frontier only)

**pts** : trade-off zone points count

**qt** : point counts grouped by search step distance, first count is for coincident points, second count is for points one search step apart, and so on

**xylen** : search space dimensions, eg. frequency x threads x ranks x nodes

**overlap** : percentage overlap of trade-off polygon area with observed/forecast polygon

**setTestDf**(self, test)

Set up a DataFrame with the required predictor/input parameters for testing the model, eg.

```
{'nd':[64], 'tn':range(4,45,4), 'fq':range(1200000,2200001,100000)}
```

**setTrainingDf**(self, train)

Set up a DataFrame with the required predictors and responses for training the model, eg. for uniform sampling:

```
{'uniform':{'nd':[64], 'tn':range(8,45,12), 'fq':[1200000,1900000,2200000]}}
```

or for Latin Hypercube Sampling:

```
{'lhs':{'nd':[64], 'tn':None, 'fq':None}}
```

Data and other attributes defined here:

**custom** = 'custom'

**lhs** = 'lhs'

```
obs = '_obs'  
uniform = 'uniform'
```

```
class ParetoFront(builtins.object)
```

Calculate the Pareto front with error limits.

Methods defined here:

```
__init__(self, pts, pmax=None, lims=None, poly=True)
```

Initialise the Pareto front configuration parameters, including the x and y trade-off data columns, and if the columns need to be minimised or maximised.

```
get(self, lims=True)
```

Get the Pareto optimal points with/without error limits.

```
getLimits(self)
```

Extend the Pareto front outer limits horizontally and vertically to encompass all points that are off the front but within the error limits for the respective axes.

```
getPath(self)
```

Create a path/polygon object from vertices and codes to define the trade-off zone.

```
getPathPts(self)
```

Get the points enclosed by the trade-off zone path/polygon object.

```
getPoly(self)
```

Get a polygon enclosing the Pareto points within the given error limits of the front.

```
scaleAndTransform(self, x, y, x_scale, y_scale, x_off, y_off,  
                    inner=True)
```

Scale and transform the front by required scale and offset to define a trade-off zone that factors in error limits, rather than a simple frontier that does not.

```
setFront(self, poly)
```

Get the Pareto optimal points, maintaining the search space x and y coordinates if provided (eg. threads and frequency).

```
class PolyReg(Model)
```

Basic polynomial model using ordinary least squares regression. Increasing the polynomial degree parameter allows the model to fit more complex responses, while increasing the risk of overfitting the data.

Method resolution order:



PolyReg  
Model  
builtins.object

Methods defined here:

```
__init__(self, df, responses, predictors, train=None,  
         test=None, transform=None, scalePredictors=None,  
         scaleResponses=None, scaleSkllearn=None,  
         scaleMinMax=None, scaleFirst=None, scaleLims=None,  
         lims=None, degree=None)
```

Initialise the model parameters, including the polynomial degree. See the Model class for other parameter descriptions.

```
fitModel(self, response)
```

Fit the basic polynomial model to the training data for the given response.

```
predict(self, response)
```

Predict the given system response for the test data using the trained polynomial model.

Data and other attributes defined here:

```
degree = 3
```

Methods inherited from Model:

```
checkAllPredictors(), getDfColRange(), getDfFiltered(), getDfLhs(), getDfSelect(),  
getErrorQt(), getInterpRmsError(), getPcErr(), getPcRng(), getRmsError(), getRsqr(),  
getStats(), getUnitStepPts(), kFoldValidate(), lhsRemap(), setParetoFronts(), setRe-  
sponses(), setStats(), setTestDf(), setTrainingDf()
```

Data and other attributes inherited from Model:

```
custom, lhs, obs, uniform
```

```
class SiPfx(builtins.object)
```

Add an SI unit prefix and scale the number and significant figures to match the units.

Static methods defined here:

```
add(num)
```

Add an SI unit prefix and scale the number to match.

```
class SplineReg(Model)
```

B-spline piecewise polynomial model that fits an ordinary least squares model using the specified model formula and training data. The model formula uses plain-text Wilkinson notation. Methods are provided for manipulating formula terms and interactions between terms.

Method resolution order:

```
SplineReg
Model
builtins.object
```

Methods defined here:

```
__init__(self, df, responses, predictors, train=None,
          test=None, transform=None, scalePredictors=None,
          scaleResponses=None, scaleSkllearn=None,
          scaleMinMax=None, scaleFirst=None, scaleLims=None,
          lims=None, order=None, dfree=None, degree=None,
          lintervals=None, excterms=None, rmterms=None)
```

Initialise the model parameters, including:

**order** : interaction terms order, eg. 2 includes second order interactions

**dfree** : piecewise spline degrees of freedom

**degree** : polynomial degree, eg. 3 is cubic polynomial

**lintervals** : any terms that should be linear rather than B-spline terms

**excterms** : any terms that should be excluded

**rmterms** : any terms that should be removed, eg. a specific interaction term

See the Model class for other parameter descriptions.

```
addInteracts(self, df=None, rnd=False)
```

Add first, second, or third order interaction terms to the list of interactions. Optionally set term limits (which set model bounds) to the given data ranges from df, with rounding down/up.

```
calculateVif(self, X, thresh=100)
```

Remove predictors that have a variance inflation factor above the given threshold.

```
fitModel(self, response)
```

Fit the B-spline model to the training data for the given response.

```
getInteract(self, terms)
```

Combine the given terms into an interaction term containing linear and B-spline terms.

```
predict(self, response)
```

Predict the given system response for the test data using the trained B-spline model.

**removeInteracts(self)**

Remove any interaction terms that match regular expressions in the removeTerms list.

**setFormula(self, response)**

Add the required response term to the left hand side to complete the model formula.

**setFormulaRhs(self, all\_interacts=False)**

Initialise the right hand side of the formula for the B-spline model. Adds the required terms to the plain-text formula using the given model initialisation.

Data and other attributes defined here:

**bsDegFree** = 8

**bsDegree** = 3

**excterms** = []

**intOrder** = 1

**linterms** = []

**rmterms** = []

Methods inherited from Model:

checkAllPredictors(), getDfColRange(), getDfFiltered(), getDfLhs(), getDfSelect(), getErrorQt(), getInterpRmsError(), getPcErr(), getPcRng(), getRmsError(), getRsqr(), getStats(), getUnitStepPts(), kFoldValidate(), lhsRemap(), setParetoFronts(), setResponses(), setStats(), setTestDf(), setTrainingDf()

Data and other attributes inherited from Model:

custom, lhs, obs, uniform

class **Transform**(builtins.object)

Transforms for improving data distribution alignment with normal distribution, including the reverse or inverse transform. Also includes scalers for improving machine learning convergence.

Methods defined here:

```
__init__(self, transform, predictors=None, responses=None,
          scalePredictors=None, scaleResponses=None,
          scaleSkllearn=None, scaleMinMax=None, scaleFirst=
          None, scaleLims=None)
```

Initialise the transform parameters, setting the transform and/or scaler to use, and DataFrame predictor and response columns to be transformed.

**transform** : 'log', 'log2', 'log10', 'exp', 'bxcx' (Box-Cox), or 'sqrt'.

**predictors, responses** : list of predictor, response column ids to be transformed or scaled (Default: None)

**scalePredictors, scaleResponses** : Scale predictors, responses (Default: True)

**scaleSklearn** : Apply scikit-learn min-max scaler if True, bespoke if False (Default: True)

**scaleMinMax** : Apply min-max scaler if True, standard scaler if False (Default: True)

**scaleFirst** : Apply scaler before applying transform if True, the reverse if False (Default: True)

**scaleLims** : Tuple (nobs, min, max, std) to apply a percentage to min, max, standard deviation values if the sample count is less than nobs – to improve scaler performance for small sample sizes (Default: (20, 0.5, 1.05, 0.5))

**\_\_str\_\_**(self)

Convert the transform to a string for use in the plain-text B-Spline model formula.

**boxcox\_fwd**(self, x)

Box-Cox transform. Not used as a bug in scipy makes results unreliable.

**fwd**(self, df)

Perform the forward transform.

**getAdjMinMax**(self, rng, std=False)

Adjust the sample minimum and maximum to better match the scaling ranges of the test data.

**inv**(self, df)

Perform the reverse/inverse transform operation to undo the transform on the model outputs.

**scale**(self, df, func)

Scale the required DataFrame columns using the selected scaler function.

**scaleInv**(self, resp, func)

Perform the reverse/inverse scaler operation to undo the scaling of the model outputs.

**setScaleParams**(self, df)

Set the scaler parameters using the adjusted sample minimum and maximum.

Data and other attributes defined here:

**scaleFirst** = True

**scaleLims** = (20, 0.5, 1.05, 0.5)

**scaleMinMax** = True

```
scalePredictors = True  
scaleResponses = True  
scaleSklern = True
```

## B.2.4 Data

```
__contact__ = 'mark.endrei@uq.edu.au'  
__copyright__ = 'Copyright 2019, The University of Queensland'  
formatter = <logging.Formatter object>  
handler = <StreamHandler <stderr> (NOTSET)>  
level = 40  
logger = <Logger hpcmodel (ERROR)>
```

## B.2.5 File

```
./lib/python3.6/site-packages/hpcmodel.py
```

# B.3 HPCPlot Module

This section provides API documentation for the HPCPlot module. User documentation is available in section A.3.

## B.3.1 Name

hpcplot - Wrapper classes for plotting experiment data collected with the hpcprobe module.

## B.3.2 Description

The hpcplot module is used for reporting and visualising optimisation results with matplotlib. The Plot class is the base class for generating 2D and 3D graphical representations of the results. The Plot class has associated experiment details, including a list of experiment ResultSets. Experiment ResultSet data is read, selected, and merged into a plot DataFrame by the PlotDf class. Data manipulation operations such as filtering, slicing, and vector arithmetic can then be used on plot data.

### B.3.3 Classes

builtins.object

Axis

Columns

Experiment

Kernels

Plot

ClusterPlot

HistogramPlot

LinePlot

ParetoPlot

QQPlot

SpearRhoPlot

SurfPlot

PlotDf

class **Axis**(builtins.object)

Plot axis definitions, with one instance needed for each axis, eg. one for the x-axis and one for the y-axis for a line plot.

Methods defined here:

**\_\_init\_\_**(self, col=None, vals=[[ ]], lim=None)

Initialise the axis parameters, including:

**col** : column id containing plot data for the axis (Default: None)

**vals** : list of value lists, one for each line/curve to plot (Default: [[ ]])

**lim** : tuple specifying axis minimum and maximum limits (min, max) (Default: None)

**\_\_str\_\_**(self)

Return str(self).

**getCols**(self)

Get axis column as list of column names.

**setTickLim**(self)

Fit ticks and limits to the axis data values.

class **ClusterPlot**(Plot)

Plot hierarchical clustering dendrograms.

Method resolution order:

ClusterPlot

Plot

builtins.object

Methods defined here:

```
__init__(self, ymls, response=None, title=None, show=True,
          eps_name=None)
```

Initialise the plot parameters, including:

**response** : column id for the response to be plotted

Other parameters can be set directly using the following attributes:

**rho\_method** : see pandas.DataFrame.corr() (Default: 'spearman')

**rho2** : use rho squared rather than rho if True (Default: True)

**dist\_method** : see scipy.cluster.hierarchy.linkage() (Default: 'complete')

See the Plot class for other parameter descriptions.

**plotPd**(self)

Called by the Plot class to plot the dendrogram.

Methods inherited from Plot:

cfgFig(), getLabels(), getListVal(), getYmls(), label3dPt(), plot(), plotFig(), resetLineColors(), setCbarTicks(), setLineColors(), setLinestyles(), setMarkerfacecolors(), setMarkers(), setMarkersizes(), setTickFormatter(), toTex()

Data and other attributes inherited from Plot:

cmapDefault, epsExt, lineColorLenDefault

class **Columns**(builtins.object)

Static column definitions for plot data, including:

**id** : short name

**name** : full YAML field name

**label** : axis label for column

**unit** : column units

**scale** : multiplier for units, eg. MFlops multiplier is 1e6

**pred** : column is a predictor variable for regression modelling

**base** : used to filter base predictors required as model inputs versus all predictors

**resp** : column is a response variable for regression modelling

**type** : 'Energy' or 'Performance' response

**pareto\_max** : optimise for minimum values if False

The following attributes/helpers are available (using list comprehension):

**names** : dictionary mapping id to name

**labels** : dictionary mapping id to axis label

**units** : dictionary mapping id to units

**respType** : dictionary mapping id to response type

**paretoMax** : dictionary mapping id to pareto\_max

**idsByName** : dictionary mapping name to id

**labelsByName** : dictionary mapping name to label

**responses** : list of all responses

**predictorsBase** : list of all base predictors

**predictors** : list of all predictors

**nameList** : list of all column names

**scale** : dictionary mapping id to scale

Static methods defined here:

**getStatsName**(col\_id, stat)

Get the column name for statistics data in YAML file for the given column id and statistic name.

**updateColumnKey**(col\_id, key, val)

Find the column definition for the given column id and update given key with given value.

Data and other attributes defined here:

```
defs = [{'id': 'ar', 'label': 'Perf', 'name':  
        'objectives.appRate', 'r...
```

```
idsByName = {'derived.energyEfficiency': 'ee',  
             'derived.fomPerJ': 'fe'...
```

```
labels = {'ar': 'Perf', 'cc': 'Comp intensity', 'cr':  
          'Comp intensity'...
```

```
labelsByName = {'derived.energyEfficiency': 'EE',  
                'derived.fomPerJ': '...
```

```
nameList = ['objectives.appRate',  
            'objectives.totalCompIntensCyc', 'ob...
```

```
names = {'ar': 'objectives.appRate', 'cc':  
         'objectives.totalCompIntens...
```



```

paretoMax = {'ar': True, 'cc': True, 'cr': True, 'de':
              True, 'dp': Tru...

predictors = ['ep', 'kl', 'l3', 'nl', 'nr', 'rs', 'sb',
               'sc', 'rb', 't...

predictorsBase = ['fq', 'nd', 'tn']

respType = {'ar': 'Performance', 'cc': None, 'cr': None,
             'de': None, '...

responses = ['ar', 'ee', 'fe', 'fs', 'st', 'td', 'te',
               'tj', 'ts', 'tt...

scale = {'ar': 1000000.0, 'cc': 1.0, 'cr': 1.0, 'de':
           1.0, 'dp': 1.0, ...

statsPrefix = 'stats'

units = {'ar': 'MFlops/s', 'cc': 'Ops/cyc', 'cr':
           'Ops/ref', 'de': 'J'...

```

class **Experiment**(builtins.object)

Integration for YAML experiment results files written by the hpcprobe module.

Provided attributes include:

- res/repName** : regular expression string/pattern for experiment name/s in YAML file/s
- res/repIter** : regular expression for kernel iteration parameter
- res/repYml** : regular expression for experiment YAML file name/s
- knlsMap** : maps experiment kernel name to a kernel enumeration column
- rangeMap** : maps experiment l/h to lo True/False column

Static methods defined here:

```

getYmls(knls, ids=[4], lo=False, hi=False, energy=False,
          perf=False)

```

Get a list of matching YAML file names for the experiment from the current directory.

YAML file names need to use the following format:

```
<kernel name>-f-<elp>-<llh>-<node count>-<id>.yaml
```

The required YAML files are selected using the available method arguments:

- knls** : list of kernel ids
- energy** : 'e' or energy usage results
- performance** : 'p' or performance results
- lo** : 'l' or low range results
- hi** : 'h' or high range results

**ids** : list of YAML ids

Data and other attributes defined here:

```
knlsMap = {'kl': {'amg': 5, 'dgemm': 4, 'hacc': 7,  
                'lammps': 6, 'lules...
```

```
rangeMap = {'lo': {'h': False, 'l': True}}
```

```
repIter = re.compile('^.* [\$]OMP_NUM_THREADS  
                    (?P<it>[0-9]+)\\b.*$')
```

```
repName = re.compile('^exp.(?P<kl>sparse|stencil|  
                    transpose...ulesh|wrf...
```

```
repYml = re.compile('^(?P<kl>[^-]*)-f-(?P<cr>[ep])-  
                    (?P<lo>[lh])-(?P<nd>...
```

```
resIter = r'^.* [\$]OMP_NUM_THREADS (?P<it>[0-9]+)\\b.*$'
```

```
resName = '^exp.(?P<kl>sparse|stencil|transpose|nstream|  
                    dge...|lulesh|...
```

```
resYml = '^(?P<kl>[^-]*)-f-(?P<cr>[ep])-(?P<lo>[lh])-  
                    (?P<nd>[0-9]{2})-...
```

class **HistogramPlot**(Plot)

Plot population distribution histograms.

Provided attributes include:

**linestyles** : default list of line styles

**markers** : default list of line markers

**x\_lim** : default x-axis limits in standard deviations

Method resolution order:

HistogramPlot

Plot

builtins.object

Methods defined here:

```
__init__(self, ymls, response, xform=None, bins=None,  
          title=None, show=True, eps_name=None)
```

Initialise the plot parameters, including:

**response** : response column id for plot

**xform** : Model.Transform to apply to response (Default: None)

**bins** : numpy.histogram bins (Default: None – uses numpy default of 10 bins)

See the Plot class for other parameter descriptions.

**plotPd**(self)

Called by the Plot class to plot the histogram.

Data and other attributes defined here:

**linestyles** = ['-', ':']

**markers** = [' ', 'o']

**x\_lim** = [-3.3, 3.3]

Methods inherited from Plot:

cfgFig(), getLabels(), getListVal(), getYmls(), label3dPt(), plot(), plotFig(), resetLineColors(), setCbarTicks(), setLineColors(), setLinestyles(), setMarkerfacecolors(), setMarkers(), setMarkersizes(), setTickFormatter(), toTex()

Data and other attributes inherited from Plot:

cmapDefault, epsExt, lineColorLenDefault

class **Kernels**(builtins.object)

Static properties for test software or kernels, including:

**id** : short name

**name** : full name

**tradeoff** : trade-off column ids for this kernel, eg. performance tradeoff with energy

**limits** : data error limits for each of the kernel trade-off columns

**<col\_id\_1>** :

**obs** : observation/measurement limits

**for** : model forecast/prediction limits

**<col\_id\_2>** : ...

**unit** : kernel specific units that override column units

**scale** : kernel specific axis plot scales for columns

The following attributes/helpers are available (using list comprehension):

**knList** : list of all kernels

**nameList** : list of all kernel names

**names** : dictionary mapping id to name

**tradeoffs** : dictionary mapping id to tradeoff

**lims** : dictionary mapping id to limits

**labels** : dictionary mapping id to labels

**units** : dictionary mapping id to units

**nums** : dictionary mapping id to an enumeration id

**scale** : dictionary mapping id to scale

**numsByName** : dictionary mapping name to an enumeration id

**idsByNum** : dictionary mapping enumeration id to name

Static methods defined here:

**getLabel**(knl\_id)

Get the kernel label using kernel definitions, if not found use the kernel name.

**getLims**(knl\_id, col\_ids, obs=True)

Get the kernel observed/forecast limits for the given column id.

**getUnits**(knl\_id, col\_id)

Get the kernel units using the kernel definitions, if not found use the column definitions.

Data and other attributes defined here:

```
defs = [{'id': 'sp', 'limits': {'ar': {'for': [0.05, 0.0], 'obs': [0.0...
idsByNum = {0: 'sp', 1: 'st', 2: 'tr', 3: 'ns', 4: 'dg',
5: 'am', 6: '...
knlList = ['sp', 'st', 'tr', 'ns', 'dg', 'am', 'lm',
'ha', 'lu', 'wr']
labels = {'am': 'AMG', 'dg': 'DGEMM', 'ha': 'HACC',
'lm': 'LAMMPS', 'l...
lims = {'am': {'fe': {'for': [0.05, 0.0], 'obs':
[0.05, 0.0]}}, 'fs': {...
nameList = ['sparse', 'stencil', 'transpose', 'nstream',
'dgemm', 'amg...
names = {'am': 'amg', 'dg': 'dgemm', 'ha': 'hacc',
'lm': 'lammmps', 'lu...
nums = {'am': 5, 'dg': 4, 'ha': 7, 'lm': 6, 'lu': 8,
'ns': 3, 'sp': 0,...
numsByName = {'amg': 5, 'dgemm': 4, 'hacc': 7, 'lammmps': 6,
'lulesh': ...
scale = {'am': {}, 'dg': {}, 'ha': {'te': 'log', 'tt':
'log'}, 'lm': {...
timesteps = {'am': None, 'dg': None, 'ha': None, 'lm':
None, 'lu': Non...
tradeoffs = {'am': ('fe', 'fs'), 'dg': ('ee', 'ar'), 'ha':
('te', 'tt')...
units = {'am': {'ep': 'nz'}, 'dg': {}, 'ha': {'ep':
'Runs'}, 'lm': {'e...
```

class **LinePlot**(Plot)

Plot line data.

Provided attributes include:

**markerList** : default list of line markers

Method resolution order:

LinePlot

Plot

builtins.object

Methods defined here:

```
__init__(self, ymls, x_id, y_id, y2_id=None, z_ids=None,
          y_err=False, z_vals=None, s_id=None, legend=False,
          colorbar=False, title=None, show=True, eps_name=
          None)
```

Initialise the plot parameters, including:

**x\_id, y\_id** : column ids for x-axis and y-axis data

**y2\_id** : column id for twin y-axis data (Default: None)

**z\_ids** : column id for legend groups (Default: None)

**y\_err** : enable error bars when data includes confidence intervals (Default: False)

**z\_vals** : column values for legend groups (Default: None)

**s\_id** : column id for marker color groups (Default: None)

See the Plot class for other parameter descriptions.

**plotPd**(self)

Called by the Plot class to plot the lines.

Data and other attributes defined here:

**markerList** = ['x', 'o', '+', 's']

Methods inherited from Plot:

cfgFig(), getLabels(), getListVal(), getYmls(), label3dPt(), plot(), plotFig(), resetLineColors(), setCbarTicks(), setLineColors(), setLinestyles(), setMarkerfacecolors(), setMarkers(), setMarkersizes(), setTickFormatter(), toTex()

Data and other attributes inherited from Plot:

cmapDefault, epsExt, lineColorLenDefault

class **ParetoPlot**(Plot)

Plot Pareto fronts.

Provided attributes include:

**ptsLinestyle** : line style for data set points (Default: ' ')  
**ptsMarker** : line markers for data set points (Default: '.')  
**ptsSize** : line markers size for data set points (Default: None)  
**ptsFcolor** : line markers face color for data set points (Default: None)  
**frtLinestyle** : line style for Pareto front (Default: '-')  
**frtMarker** : line markers for Pareto front (Default: 'o')  
**frtSize** : line markers size for Pareto front (Default: 12)  
**frtFcolor** : line markers face color for Pareto front (Default: 'none')  
**demoLineColors** : line colors for trade-off zone demonstration plot

Method resolution order:

ParetoPlot  
Plot  
builtins.object

Methods defined here:

```
__init__(self, ymls, x_id, y_id, x_lim=None, y_lim=None,  
          z_id=None, z_vals=None, s_id=None, poly=False,  
          lims=False, obs=True, demo=False, clusters=None,  
          relax=False, iterate=False, colorbar=False,  
          legend=False, title=None, show=True, eps_name=  
          None)
```

Initialise the plot parameters, including:

**x\_id, y\_id** : column ids for x-axis and y-axis data  
**x\_lim, y\_lim** : x-axis and y-axis limits (Default: None)  
**z\_id** : column id for legend groups (Default: None)  
**z\_vals** : column values for legend groups (Default: None)  
**s\_id** : column id for marker color groups (Default: None)  
**poly** : plot trade-off zone polygon (Default: False)  
**lims** : apply error limits to Pareto analysis (Default: False)  
**obs** : use observed or forecast error limits (Default: True is observed)  
**demo** : color trade-off zone construction steps (Default: False)  
**clusters** : add Pareto point cluster labels (Default: None)  
    <label\_1>: [<start Pareto point>, <end Pareto point>] <label\_n2>: [<start  
    Pareto point>, <end Pareto point>]  
**relax** : plot error limits boxes around Pareto points (Default: False)  
**iterate** : extend Pareto set by repeating analysis with Pareto points excluded  
(Default: False)

See the Plot class for other parameter descriptions.

**plotPd**(self)

Called by the Plot class to plot the Pareto front.

Data and other attributes defined here:

**demoLineColors** = ['b', 'r']

**frtFcolor** = 'none'

**frtLinestyle** = '-'

**frtMarker** = 'o'

**frtSize** = 12

**ptsFcolor** = None

**ptsLinestyle** = '.'

**ptsMarker** = '.'

**ptsSize** = None

Methods inherited from Plot:

cfgFig(), getLabels(), getListVal(), getYmls(), label3dPt(), plot(), plotFig(), resetLineColors(), setCbarTicks(), setLineColors(), setLinestyles(), setMarkerfacecolors(), setMarkers(), setMarkersizes(), setTickFormatter(), toTex()

Data and other attributes inherited from Plot:

cmapDefault, epsExt, lineColorLenDefault

class **Plot**(builtins.object)

Base class for plotting DataFrame data using matplotlib. By default, plots use the 'gnu-plot' colormap and are saved in PDF format.

Methods defined here:

```
__init__(self, df_ymls, x_lim=None, y_lim=None, legend=False, colorbar=False, title=None, surf=False, wire=False, show=True, eps_name=None)
```

Initialise the plot parameters, including:

**df\_ymls** : DataFrame or list of YAML files containing plot data

**x\_lim** : x-axis limits (Default: None)

**y\_lim** : y-axis limits (Default: None)

**legend** : add legend to plot if True (Default: False)

**colorbar** : add a color bar to the plot if True (Default: False)

**title** : plot title (Default: None)

**surf** : 3d surface plot (Default: False)

**wire** : 3d wire plot (Default: False)

**show** : display plot on screen (Default: True)

**eps\_name** : save plot if file name is not None (Default: None)

**cfgFig**(self)

Configure the plot legend, colorbar, axes scale, limits, ticks, tick labels, axis labels, twin axis, z-axis if for a 3d plot, etc.

**getLabels**(self)

Get text labels for the plot using label items from the z-axis data.

**getListVal**(self, vals)

Get a value for a list that is expected to be all the same value. If list values are not all the same return "??"

**getYmls**(self)

Get the list of YAML file dependencies for the plot.

**label3dPt**(self, x, y, z, z\_lim, l, z\_id=None)

Label a point on a 3d plot, including its coordinates.

**x, y, z** : coordinates of point

**z\_lim** : vertical limit for label anchor

**l** : label text

**z\_id** : id of z column used to add SI prefix to label

**plot**(self)

Use the plotPd method from the child class to plot data.

**plotFig**(self, part=False)

Plot the figure

**part** : do not show or save the figure, overriding the current save and show settings. This is used when the plot will be further manipulated after calling this method.

**resetLineColors**(self, axis=None)

Reset the line color cycle to the default setting

**setCbarTicks**(self, ticks, labels)

Set the colorbar ticks and labels

**setLineColors**(self, axis=None, clen=None)

Set the line color cycle using the plot colormap setting.

**setLinestyles**(self, linestyles)



Initialise the cycle object for line styles.

**setMarkerfacecolors**(self, markerfacecolors)

Initialise the cycle object for line marker face colors

**setMarkers**(self, markers, marker=None, idx=-1)

Initialise the cycle object for line markers.

**setMarkersizes**(self, markersizes)

Initialise the cycle object for line marker sizes.

**setTickFormatter**(self, ax\_plt, ax\_lims, ax\_cfg)

The axis tick formatter needs to be reset if setting a log scale.

**ax\_plt** : matplotlib axis to be set

**ax\_lims** : axis limits

**ax\_cfg** : axis definition

**toTex**(self, txt)

Convert text for L<sup>A</sup>T<sub>E</sub>X compatibility if usetex setting is active in matplotlibrc file.

Data and other attributes defined here:

**cmapDefault** = 'gnuplot'

**epsExt** = '.pdf'

**lineColorLenDefault** = 10

class **PlotDf**(builtins.object)

Allows hpcprobe experiment results to be read, selected, and merged into a DataFrame for plotting. Data manipulation operations such as filtering, slicing, and vector arithmetic can then be used on plot data.

Provided attributes include:

**userTimeThreshold** : alert if user time is less than this percentage of total time, as the results may be affected by high job initialisation or clean-up times

**knls** : list of kernels found in the data

**predictors** : list of predictors found in the data

**responses** : list of responses found in the data

Methods defined here:

**\_\_call\_\_**(self)

PlotDf() returns the DataFrame.

**\_\_init\_\_**(self, df\_ymls, basePredictors=False)

Load the plot DataFrame. The `df_ymls` argument can be a list of YAML file/s to load data from, or a DataFrame to use. The `basePredictors` argument should be True if plot data is being used with `hpcmodels`

**checkUserTime**(self, tt\_col=None, tt=None, pc=None)

Check the minimum user time is over the given percent threshold of the total time as the accuracy of the energy efficiency calculation is impacted if user time is not close to the total time.

**getColsBy**(self, cols, by\_col)

Get column data grouped by values in another column.

**getProdDf**(self, cols)

Generate plot DataFrame containing the required combinations of columns and values eg. `cols = {'nd':[20], 'tn':range(4,45,4), 'gh':numpy.arange(1.2,2.2,.1)}`

**getReshapedCol**(self, col)

Reshape a (1 x n) list into a (m x len/m) list where m is repeat length of the column values.

**getReshapedCols**(self, cols)

Reshape multiple (1 x n) lists into (m x len/m) lists, grouped horizontally.

**getYmls**(self)

Get the list of YAML files used to load the DataFrame.

**readYaml**(self)

Load the DataFrame using data read from the configured list of YAML files. YAML files in a simple list `[yaml1,yaml2,..]` are appended to the DataFrame. Otherwise, pairs of YAML files `[[yaml1,yaml2],[yaml3,yaml4],..]` are merged and appended. This enables energy and performance data merging of energy and performance data that was collected in separate runs.

**setDerivedCols**(self)

Set/add columns derived from other columns, such as energy efficiency. Also rename all columns in column definitions using their short name.

**setFilterCol**(self, col, col\_filt, label=None)

Add a DataFrame column with the name given by `col_filt` that applies a Savitzky-Golay smoothing filter to address noise in the `col` column.

**setPredictors**(self, base)

Set the predictors attribute using the given base argument to select base only or all predictors.

**setYErr**(self, col, stat, err\_id)

Set yerr1 and yerr2 columns in required format for errorbar plot.

Data and other attributes defined here:

**userTimeThreshold** = 0.95

class **QQPlot**(Plot)

Plot quantile-quantile plots for comparing probability distributions.

Provided attributes include:

**linestyles** : default list of line styles

**markers** : default list of line markers

**x\_lim** : default x-axis limits in standard deviations

Method resolution order:

QQPlot

Plot

builtins.object

Methods defined here:

```
__init__(self, ymls, response, xform=None, title=None,
          show=True, eps_name=None)
```

Initialise the plot parameters, including:

**response** : response column id for plot

**xform** : Model.Transform to apply to response (Default: None)

See the Plot class for other parameter descriptions.

**plotPd**(self)

Called by the Plot class to plot the Q-Q plot.

Data and other attributes defined here:

**linestyles** = [' ', '-.', ':']

**markers** = ['x', ' ', ' ', ' ']

**x\_lim** = [-3.3, 3.3]

Methods inherited from Plot:

cfgFig(), getLabels(), getListVal(), getYmls(), label3dPt(), plot(), plotFig(), resetLineColors(), setCbarTicks(), setLineColors(), setLinestyles(), setMarkerfacecolors(), setMarkers(), setMarkersizes(), setTickFormatter(), toTex()

Data and other attributes inherited from Plot:

cmapDefault, epsExt, lineColorLenDefault

class **SpearRhoPlot**(Plot)

Plot the Spearman's rho association between predictors and the selected response.

Method resolution order:

SpearRhoPlot

Plot

builtins.object

Methods defined here:

```
__init__(self, ymls, response, sort=False, rho2=False,  
         title=None, show=True, eps_name=None)
```

Initialise the plot parameters, including:

**response** : column id for the response to be plotted

**sort** : sort the responses by rho values if True (Default: False)

**rho2** : use rho squared rather than rho if True (Default: False)

See the Plot class for other parameter descriptions.

**plotPd**(self)

Called by the Plot class to plot the Spearman's rho associations.

Methods inherited from Plot:

cfgFig(), getLabels(), getListVal(), getYmls(), label3dPt(), plot(), plotFig(), resetLineColors(), setCbarTicks(), setLineColors(), setLinestyles(), setMarkerfacecolors(), setMarkers(), setMarkersizes(), setTickFormatter(), toTex()

Data and other attributes inherited from Plot:

cmapDefault, epsExt, lineColorLenDefault

class **SurfPlot**(Plot)

Plot three dimensional surface data. This class can be used to show Pareto points in context of search space dimensions selected for the x-axis and y-axis.

Provided attributes include:

**pLinestyle** : default line style **pMarker** : default line markers

Method resolution order:

SurfPlot

Plot

builtins.object

Methods defined here:

```
__init__(self, ymls, x_id, y_id, z_id, z_lim=None, z2_id=None, z2_vals=None, z3_id=None, pareto=True, poly=True, lims=True, obs=True, legend=False, title=None, rotate=None, show=True, eps_name=None)
```

Initialise the plot parameters, including:

**x\_id, y\_id, z\_id** : column ids for x, y and z axis data

**z\_lim** : z-axis limits (Default: None)

**z2\_id** : column id for legend groups (Default: None)

**z2\_vals** : column values for legend groups (Default: None)

**z3\_id** : trade-off column id (Default: None)

**pareto** : plot Pareto points for z\_id versus z3\_id on the surface (Default: True)

**poly** : plot trade-off zone polygon points (Default: True)

**lims** : apply error limits to Pareto analysis (Default: True)

**obs** : use observed or forecast error limits (Default: True is observed)

**rotate** : rotate plot using given (elev,azim) tuple (Default: None)

See the Plot class for other parameter descriptions.

**plotPd**(self)

Called by the Plot class to plot the surface.

Data and other attributes defined here:

**pLinestyle** = ' '

**pMarker** = '.'

Methods inherited from Plot:

cfgFig(), getLabels(), getListVal(), getYmls(), label3dPt(), plot(), plotFig(), resetLineColors(), setCbarTicks(), setLineColors(), setLinestyles(), setMarkerfacecolors(), setMarkers(), setMarkersizes(), setTickFormatter(), toTex()

Data and other attributes inherited from Plot:

cmapDefault, epsExt, lineColorLenDefault

## B.3.4 Functions

**run()**

HPCPlot API usage demonstration.

## B.3.5 Data

**\_\_contact\_\_** = 'mark.endrei@uq.edu.au'

```
__copyright__ = 'Copyright 2019, The University of Queensland'
```

```
logger = <Logger hpcplot (WARNING)>
```

### **B.3.6 File**

```
./lib/python3.6/site-packages/hpcplot.py
```

# Appendix C

## Artefacts Archive

This appendix provides instructions for setting up the required tools, launching experiments, and analysing the results, as presented in this thesis.

### C.1 Description

Instructions on how to obtain the artefacts archive and the associated hardware and software dependencies are provided in this section.

#### C.1.1 How Software can be Obtained

The source code and data set archive is available in The University of Queensland's institutional repository, UQ eSpace, using [doi.org/10.14264/uql.2019.697](https://doi.org/10.14264/uql.2019.697). To request access, send an email to [data@library.uq.edu.au](mailto:data@library.uq.edu.au). Select the **Artefacts\_archive** ZIP file.

#### C.1.2 Hardware Dependencies

The software has been tested on Intel Haswell and Broadwell servers running CrayPE version 2.5.

#### C.1.3 Software Dependencies

Build instructions and dependencies are provided in the artefacts archive `README.md` for each program under the following folders:

- `tools/nimrodo`
- `tools/hpcprobe`

Third party source code is available as follows:

- Parallel Research Kernels version 2.17 commit d52f82f – [github.com/ParRes/Kernels](https://github.com/ParRes/Kernels)
- Algebraic multi-grid benchmark version 1.2 commit 09fe8a7 – [github.com/LLNL/AMG](https://github.com/LLNL/AMG)
- LAMMPS version 14 May 2016 – [lammps.sandia.gov](http://lammps.sandia.gov)
- LULESH version 2.0 commit b56882b – [github.com/LLNL/LULESH](https://github.com/LLNL/LULESH)
- WRF version 4.0.1 commit f729619 – [github.com/wrf-model/WRF](https://github.com/wrf-model/WRF)
  - WPS version 4.0.1 commit 398505c – [github.com/wrf-model/WPS](https://github.com/wrf-model/WPS)

### C.1.4 Archive Contents

Figure C.1 shows the structure of the artefacts archive with a brief description of the contents of each folder.












| Folder  | Description                       |
|---|-----------------------------------|
|  exp         | Experiment artefacts              |
|  evaluation | Chapter 6 evaluation artefacts    |
|  config    | Experiment initialisation files   |
|  results   | Experiment results summary files  |
|  tools     | Tools artefacts                   |
|  amg       | AMG build information             |
|  hpcprobe  | HPCProbe source code              |
|  lulesh    | LULESH build information          |
|  nimrodo   | Nimrod/O source code              |
|  prk       | Parallel Research Kernels patches |
|  wrf       | WRF build information             |

Figure C.1: Archive Contents

### C.1.5 Datasets

All experiment initialisation files and YAML results summary files are available in the data set archive under the exp/evaluation folder. File names include the kernel name and node count. All model responses and data plots are generated from these data sets. HPC node counts for the full data set include:

- 20, 42, 64, and 86 for Stencil;
- 20, 42, 64, and 86 for Transpose;
- 20, 42, 64, and 86 for Nstream;
- 20, 40, 48, 60, and 80 for AMG;
- 20, 40, 60, and 80 for LAMMPS;



- 4, 14, 32, and 63 for LULESH; and
- 64 for WRF.

## C.2 Installation

To build and install the required tools and test codes:

1. Extract the artefacts archive ZIP file
2. Build Nimrod/O as per README.md instructions from the data set
3. Build HPCProbe tools as per README.md instructions from the data set
4. Clone the PRK ParRes/Kernels GitHub repository
  - Check tools/prk/README.md for instructions
5. Build the required kernels, MPI/OpenMP stencil, transpose, and nstream
  - Edit Kernels/common/make.defs to suit your environment. For CrayPE, set
    - MPICC=cc
    - CC=cc
    - DEFAULT\_OPT\_FLAGS:=-hpic -dynamic
  - Make the required kernels, for example
    - cd Kernels/MPIOENMP/Stencil
    - make stencil
  - Instrument the kernels with CrayPAT, for example
    - pat\_build -w stencil

The job post processing command option mentioned in section A.1.2 can be used to collect power measurements using another mechanism

6. Clone the LLNL/AMG GitHub repository
7. Build AMG

- Edit Makefile.include to suit, in our case
  - CC = cc
  - INCLUDE\_CFLAGS = -O2 -DTIMER\_USE\_MPI -DHYPRE\_USING\_OPENMP \
    - h omp -DHYPRE\_HOPSCOTCH -DHYPRE\_USING\_PERSISTENT\_COMM \
    - DHYPRE\_BIGINT -hpic -dynamic
  - INCLUDE\_LFLAGS = -lm -h omp

- Make and instrument AMG
  - make
  - pat\_build -w test/amg

## 8. Download LAMMPS

## 9. Build LAMMPS

- Edit MAKE/Makefile.mpi to suit
- Make and instrument LAMMPS
  - make -j8 mpi
  - pat\_build -w lmp\_mpi

## 10. Clone the LLNL/LULESH GitHub repository

## 11. Build LULESH

- Edit Makefile to suit, in our case
  - MPICXX = CC -static -homp -DUSE\_MPI=1
  - CXXFLAGS = -O2 -hipa4
  - LDFLAGS = -O2 -hipa4
  - Comment out MPI\_INC and MPI\_LIB
- Make and instrument LULESH
  - module load fftw
  - make
  - pat\_build -w lulesh2.0

## 12. Clone the wrf-model/WRF GitHub repository

- Check tools/wrf/README.md for instructions

## 13. Build WRF

- ./configure
  - Compiler choice: 51. (dm+sm) INTEL (ftn/icc): Cray XC  
dm+sm is distributed and shared memory, or MPI/OpenMP
  - Nesting option: 1 basic
- Make WRF
  - make

## 14. Clone the wrf-model/WPS GitHub repository

- Check tools/wrf/README.md for instructions

## 15. Build WPS

- `./configure`
  - Compiler choice: 40. Cray XC, Intel (dmpar\_NO\_GRIB2)
- Make WPS
  - `make`

## C.3 Experiment Workflow

The following steps are used to run experiments and analyse the results:

1. Create the experiment initialisation file. The initialisation files for the Chapter 6 experiments are available in the artefacts archive under the `exp/evaluation/config` folder.
2. Launch the experiment using the following command:

```
hpcprobe.py -i <init file>
```

3. Analyse the experiment results using the Python API provided by the HPCModel and HPCPlot modules.

Analysis data and plots are generated from the `summary.yml` files output from `hpcprobe.py`. Section A.2 and section A.3 provide examples showing how the plots and results summary data in this thesis are generated using the HPCModel and HPCPlot modules.

## C.4 Evaluation and Expected Result

Chapter 6 describes the model evaluation process in detail.

There are several log files to monitor that an experiment is progressing as expected:

1. `log/exp.log` for overall experiment logs
2. `log/job.out` for Nimrod/O logs
3. `log/qsub.out` for PBS logs
4. `log/job-<id>.out` for each measurement run log

The scripts used for generating the experimental and model data log to standard output. The logging level can be set via the command line with the `--info` or `--debug` arguments.

## C.5 Experiment Customisation

Customisation options for the HPCProbe tools include:

- Measurement data parsing using regular expressions defined in the `hpcplot.yml` file. See section A.1.3.
- Experiment configuration using initialisation files. Configurable options include PBS job submission details, software module requirements, job post processing command details, and repeat count for measurement confidence intervals. See section A.1.2.
- Results analysis using Python scripts with the HPCModel and HPCPlot modules. The HPCModel module implements the proposed models. Plots can be generated from experiment `summary.yml` files using the HPCPlot module. See section A.2 and section A.3.

For further details, please refer to the user documentation in Appendix A and API documentation in Appendix B.