Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations

Graduate Studies

12-2016

Statistical Techniques to Model and Optimize Performance of Scientific, Numerically Intensive Workloads

Steena Dominica Steven Monteiro
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd

Part of the Computer Sciences Commons

Recommended Citation

Steven Monteiro, Steena Dominica, "Statistical Techniques to Model and Optimize Performance of Scientific, Numerically Intensive Workloads" (2016). *All Graduate Theses and Dissertations*. 5228.
https://digitalcommons.usu.edu/etd/5228

This Dissertation is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

UtahStateUniversity
MERRILL-CAZIER LIBRARY

STATISTICAL TECHNIQUES TO MODEL AND OPTIMIZE PERFORMANCE OF SCIENTIFIC, NUMERICALLY INTENSIVE WORKLOADS

by

Steena Dominica Steven Monteiro

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

Amanda Lee Hughes, Ph.D.
Major Professor

Chad Mano, Ph.D.
Committee Member

Dan Watson, Ph.D.
Committee Member

Nicholas Dickenson, Ph.D.
Committee Member

Breanne Litts, Ph.D.
Committee Member

Ilya Sharapov, Ph.D.
Committee Member

Daniel Wong, Ph.D.
Committee Member

Mark R. McLellan, Ph.D.
Vice President for Research and
Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2016

Copyright © Steena Dominica Steven Monteiro 2016

All Rights Reserved

ABSTRACT

Statistical Techniques to Model and Optimize Performance of Scientific, Numerically Intensive Workloads

by

Steena Dominica Steven Monteiro, Doctor of Philosophy
Utah State University, 2016

Major Professor: Amanda Lee Hughes, Ph.D.
Department: Computer Science

The ability to estimate how well a computer will perform when executing large scientific computations (or workloads) is important for hardware designers, software developers, supercomputing centers, and end users. However, accurately projecting computer performance is difficult due to differences in computer system specifications, workload execution, and the type of datasets. Using statistical learning techniques, this dissertation models, optimizes, and projects computer performance by evaluating workloads from different domains on different processors. The key contribution of this work is the design and evaluation of two statistical performance modeling techniques.

The first technique, Statistical Techniques for Optimizing and Modeling Performance of blocked sparse matrix vector multiplication (STOMP), builds statistical models to predict and optimize blocked sparse matrix vector multiplication across different sparse matrices prior to their execution. Predictions from STOMP achieve 93.62% accuracy on diverse sparse matrices. STOMP's block-size selection technique produces a performance improvement of up to 75% on different matrices.

The second technique, Statistical Techniques for Analyzing Metrics and Predicting Performance of workloads (STAMPP) analyzes hardware metric space and predicts speedup in performance on a set of workloads across generations of processors. STOMP eliminates redundancy in performance monitoring by using statistical techniques to identify important hardware metrics and relevant workloads that underline performance trends. With its suite of statistical models, STAMPP predicts performance speedup of new workloads prior to their execution on a new processor with high accuracy.

(111 pages)

PUBLIC ABSTRACT

Statistical Techniques to Model and Optimize Performance of Scientific, Numerically Intensive Workloads

Steena Dominica Steven Monteiro

Projecting performance of applications and hardware is important to several market segments—hardware designers, software developers, supercomputing centers, and end users. Hardware designers estimate performance of current applications on future systems when designing new hardware. Software developers make performance estimates to evaluate performance of their code on different architectures and input datasets. Supercomputing centers try to optimize the process of matching computing resources to computing needs. End users requesting time on supercomputers must provide estimates of their application's run time, and incorrect estimates can lead to wasted supercomputing resources and time. However, application performance is challenging to predict because it is affected by several factors in application code, specifications of system hardware, choice of compilers, compiler flags, and libraries.

This dissertation uses statistical techniques to model and optimize performance of scientific applications across different computer processors. The first study in this research offers statistical models that predict performance of an application across different input datasets prior to application execution. These models guide end users to select parameters that produce optimal application performance during execution. The second study offers a suite of statistical models that predict performance of a new application on a new processor. Both studies present statistical techniques that can be generalized to analyze, optimize, and predict performance of diverse computation- and data-intensive applications on different hardware.

For my husband, Forrest Iandola.

ACKNOWLEDGMENTS

Thank you to my adviser, Amanda Lee Hughes, for her support in helping me reach the finish line. Thank you to my supervisory committee for their feedback. Thank you to Daniel Wong for advice on publishing a chapter from this dissertation.

Thank you to Lawrence Livermore National Laboratory for support through the Lawrence Graduate Scholar Program (2011–2015).

It would not have been possible to complete this dissertation without support from Intel. I would like to thank Ilya Sharapov for his mentorship. Thank you to my manager, John Kreatsoulas, for his constant encouragement.

I thank my husband, Forrest Iandola, for his optimism and humor through the Ph.D. process. He is my best friend and it is only right that I dedicate this dissertation to him.

Steena Monteiro

CONTENTS

Page

ABSTRACT iii

PUBLIC ABSTRACT v

ACKNOWLEDGMENTS vii

LIST OF TABLES x

LIST OF FIGURES xii

CHAPTER

1 INTRODUCTION 1
- Importance of Performance Modeling and Performance Projections 1
- Two Statistical Techniques for Predicting Workload Performance 2
- Dissertation Overview 4

2 STOMP: STATISTICAL TECHNIQUES FOR OPTIMIZING AND MODELING PERFORMANCE OF BLOCKED SPARSE MATRIX VECTOR MULTIPLICATION 6
- Abstract 6
- Introduction 7
- Related Work 8
- Understanding Blocked SpMV Performance 11
- Experimental Environment 13
- Predicting SpMV Performance 14
- SpMV Performance Prediction Results 20
- Guiding Optimal Block Size Selection 27
- Evaluating STOMP on Knight's Landing 31
- STOMP's Performance Prediction on Knight's Landing 33
- Conclusion 41

3 STAMPP: STATISTICAL TECHNIQUES FOR ANALYZING METRICS AND PREDICTING PERFORMANCE OF WORKLOADS 43
- Abstract 43
- Introduction 43
- Related Work 47
- Experiment Environment 51
- Metric Analysis using Statistical Techniques 54
- Workload Characterization 60
- STAMPP Prediction Models 63
- Statistical Performance Prediction Model Suite 64

STAMPP Performance Predictions 69
Summarizing STAMPP's Prediction Results 71
Conclusion 72

4 CONCLUSION 75
Performance Modeling: Performance Prediction and Performance Optimization . 75
Future Work 79

REFERENCES 81

APPENDICES 90
Appendix A Permission-to-use Letter 91

CURRICULUM VITAE 93

LIST OF TABLES

Table Page

2.1 SpMV arithmetic intensity for different block sizes 13

2.2 Bai matrix group 14

2.3 Mixed matrix group 15

2.4 Boeing-Chen-FIDAP matrix group 15

2.5 SPARSITY's matrix set 16

2.6 STOMP SpMV performance prediction errors across different block sizes . . 26

2.7 Distribution of STOMP SpMV performance prediction errors on Sandy Bridge 27

2.8 STOMP SpMV speedup on Boeing-Chen-FIDAP matrices 30

2.9 STOMP SpMV speedup on Bai group of matrices 30

2.10 STOMP SpMV speedup on mixed group of matrices 31

2.11 STOMP SpMV speedup evaluation on SPARSITY's matrices 32

2.12 STOMP versus SPARSITY 32

2.13 Evaluating STOMP's block size predictions on all matrix groups 33

2.14 STOMP SpMV performance prediction errors on Knight's Landing 35

2.15 Distribution of STOMP SpMV performance prediction errors on Knight's Landing 36

2.16 STOMP SpMV speedup on Boeing-Chen-FIDAP on Knight's Landing. . . . 41

2.17 STOMP SpMV speedup on Bai matrices on Knight's Landing 41

2.18 STOMP SpMV speedup on mixed group of matrices on Knight's Landing . 42

2.19 Evaluating STOMP's block size predictions on Knight's Landing 42

3.1 SPEC CPU 2006 benchmarks 46

3.2 SPEC OMP 2012 benchmarks 47

3.3 SPEC MPI 2007 benchmarks 48

3.4 Machine specifications 51

3.5 Hardware Metrics Collected on Sandy Bridge, Haswell, and Broadwell . . . 53

3.6 Important metrics from VIF 60

3.7 Important metrics from VIF further reduced by MIC 61

3.8 STAMPP performance model suite 64

3.9 Prediction errors from performance projections 69

LIST OF FIGURES

Figure Page

1.1 Dissertation problem and solution space. 5

2.1 Blocking a sparse matrix. 12

2.2 Roofline model of serial blocked SpMV on Sandy Bridge. 12

2.3 SpMV performance model training. 17

2.4 Predicting SpMV performance of a new matrix. 17

2.5 NNZE vs. time for *in-2004*. 18

2.6 NNZB vs. time for *in-2004*. 18

2.7 Comparing prediction errors from NNZE and NNZB models for Boeing-Chen-FIDAP group of matrices. 21

2.8 NNZB model prediction error distribution with respect to matrix specifications for Boeing-Chen-FIDAP group. 22

2.9 Comparing prediction errors from NNZE and NNZB models for Bai group of matrices. 23

2.10 NNZB model prediction error distribution with respect to matrix specifications for Bai group. 24

2.11 Comparing prediction errors from NNZE and NNZB models for mixed group of matrices. (Miscellaneous group with greatest diversity.) 25

2.12 NNZB model prediction error distribution with respect to matrix specifications for mixed group of matrices. 26

2.13 Performance prediction errors from inter-matrix group cross-validation (block size 2). 27

2.14 Comparing predictions from NNZE and NNZB models for Boeing-Chen-FIDAP matrices on Knight's Landing. 34

2.15 NNZB model prediction error distribution with respect to matrix specifications for Boeing-Chen-FIDAP on Knight's Landing. 35

2.16 Comparing predictions from NNZE and NNZB models for Bai matrices on Knight's Landing. 36

2.17 NNZB model prediction error distribution with respect to matrix specifications for Bai matrices on Knight's Landing. 37

2.18 Comparing predictions from NNZE and NNZB models fo matrices from mixed group on Knight's Landing. 38

2.19 NNZB model prediction error distribution with respect to matrix specifications for mixed matrix group on Knight's Landing. 39

2.20 Comparing predictions from NNZE and NNZB models across ten trials on KNL. 40

3.1 Distribution of serial SPEC CPU metrics on Sandy Bridge. 55

3.2 Maximal information coefficients for SPEC CPU on Sandy Bridge. 58

3.3 Variable inflation factor for SPEC CPU on Sandy Bridge. 59

3.4 Hierarchical clustering of serial SPEC CPU on Sandy Bridge. 59

3.5 Workload characterization. 62

3.6 SPEC CPU siblings for serial SPEC OMP on Haswell. 65

3.7 SPEC CPU siblings for parallel SPEC OMP on Haswell. 66

3.8 SPEC CPU siblings for serial SPEC MPI on Sandy Bridge. 67

3.9 SPEC CPU siblings for parallel SPEC MPI on Sandy Bridge. 68

3.10 Serial SPEC CPU speedup from Sandy Bridge to Haswell. 68

3.11 Serial SPEC OMP prediction errors from baseline and STAMPP on Broadwell. 71

3.12 Parallel SPEC OMP prediction errors from baseline and STAMPP on Broadwell. 72

3.13 Serial SPEC MPI prediction errors from baseline and STAMPP on Haswell. 73

3.14 Parallel SPEC MPI prediction errors from baseline and STAMPP on Haswell. 74

ACRONYMS

BDW Broadwell. This is the codename for CPU microarchitecture developed by Intel in 2014. Broadwell is the tock of Haswell in Intel's Tick-Tock process and has a 14 nm process.

CPU Central processing unit. This is commonly called a processor and contains computing elements and cache hierarchy.

DRAM Dynamic Random Access Memory. This piece of memory stores data that a processor will require for program executions.

ELLPACK This is a high-level package that solves elliptic boundary problems. ELLPACK is implemented as a Fortran preprocessor.

FLOPs Floating Point Operations. This is used to measure peak compute capability of a processor.

GPU Graphics processing unit. This processor accelerates display of images and processing of image data.

GNU GNU is not Unix. This project comprises free software such as operating system and compiler tools. The operating system is like Unix, but differs in that it is free and does not contain Unix code.

Hit This occurs when data requested by the processor is found in a particular level of cache or memory.

HPC High Performance Computing

HSW Haswell. This is the codename for CPU microarchitecture developed by Intel in 2013. Haswell has a 22 nm process.

KNL Knight's Landing. Co-processor developed by Intel in the Xeon Phi line and released in 2016.

L1 Level one cache. It is the primary cache built onto the processor chip and is private to each core. It is the smallest and fastest cache in the memory hierarchy.

L2 Level two cache. Unlike the L1 cache, the L2 cache is located outside of CPU cores. L2 cache provides data to the L1 cache. L2 caches are larger and slower than the L1 cache.

L3 Level three cache. L3 caches feed L2 caches. They are larger and slower than L1 or L2 caches.

Miss This occurs when data requested by the processor is not found in a particular level of cache or memory.

MPI Message Passing Interface. This is a library that is used to facilitate and optimize inter-processor communication and communication patterns between parallel processes.

OpenMP Open Multi-Processing. This is a library that is used for shared memory programming.

PETSc Parallel Extensible Toolkit for Scientific Computing. This is a library that provides a multitude of scalable routines for mathematical operations used in scientific applications.

SNB Sandy Bridge. This is Intel's codename for their 32-nm process microarchitecture launched in 2005.

SpMV Sparse matrix vector multiplication. Mathematical operation consisting of multiplication of a sparse matrix with a dense vector to produce a dense vector.

TLB Translation Lookaside Buffer. This stores recent address translations from virtual memory to physical memory.

UMA Uniform Memory Access. This represents a shared memory architecture.

CHAPTER 1

INTRODUCTION

1.1 Importance of Performance Modeling and Performance Projections

The ability to efficiently estimate computer performance when executing large scientific computations (or workloads) is important for hardware designers, software developers, supercomputing centers, and end users. Hardware designers need to project and compare application performance across existing and proposed designs so they can make decisions about future processor specifications. Software developers need to estimate computer performance so they can optimize software code for different architectures and datasets. High performance supercomputing centers harness the power of thousands of processors to run numerically intensive algorithms. To efficiently schedule these processors, supercomputing centers need accurate estimates of computing performance. Finally, end users must estimate job run times before requesting time on a supercomputer; an incorrect estimation will lead to inefficient mapping between available resources and scheduled computation.

However, accurately predicting computer performance on scientific workloads is challenging for several reasons. First, the system specification of a computer affects its performance. For example, different memory configurations on have different effects on the run time of a workload. Benign features, such as hardware and software prefetching [1] or hyperthreading [2], designed to help performance can also cause decreases in performance [1] [3]. Second, the execution of a workload can vary greatly. Most workloads exhibit different phases during their execution [4]. Scientific workloads typically exhibit the following different phases during their runtimes: variable state during data loading and initializing, intermittent steady state during computation, and a result collation and write back stage. Thus, workload performance differs depending on particular workload state.

Third, performance of data-intense applications can vary based on the amount of data they have to compute, the data‘s layout in memory, and data access patterns. These reasons make accurate prediction of computer performance a challenging problem.

In this dissertation, we use statistical learning techniques to address this problem and improve performance modeling and prediction of different applications. Statistical learning techniques have shown promise in several fields in their ability to process and provide insight into large amounts of empirical data in fields such as financial forecasting [5], [6], and [7], climate change research [8], cancer research efforts [9], and electrical profiling [10]. The strength of statistical algorithms lies in their ability to detect important data patterns and reuse past data for making predictions. By using statistical learning processes for performance projections and optimizations, applications and hardware are made less opaque to end users. This work uses statistical techniques and prediction models to provide insight into workload behavior on different CPU architectures by examining trends across different hardware metrics and application input data. Our statistical models leverage performance data from prior workload executions to model workload behavior and make performance predictions.

1.2 Two Statistical Techniques for Predicting Workload Performance

This dissertation presents two studies that use statistical analyses to model performance. The first study—Statistical Techniques for Optimizing and Modeling Performance on blocked sparse matrix vector multiplication (STOMP)—predicts performance of a workload and optimizes its performance across different input data sets by guiding optimal parameter selection. Sparse matrix vector multiplication (SpMV) is a mathematical kernel that forms the core of several code bases such as Internet search engines, recommendation systems, and linear solvers. Because of the inherent sparse structure of the matrix, SpMV exhibits irregular memory accesses, which in turn degrades performance. In addition, SpMV is rarely performed just once in an application, thus becoming the primary bottleneck in the parent algorithm. STOMP addresses this common bottleneck with performance models that incorporate matrix sparsity patterns to predict the optimal parameters for a new, unseen sparse matrix. During the inference phase, STOMP's performance prediction models produced a high average accuracy (93.62%) across three groups of matrices on the Sandy Bridge architecture. We evaluated STOMP's parameter selection in terms of the speedup it produces over an exhaustive search algorithm and found that STOMP produces speedup of up to 75%. STOMP also produces an speedup of at least 49% over default SpMV parameter options.

We also compared performance benefits produced from STOMP's parameter selection algorithm to that produced from SPARSITY [11], a well-known SpMV framework. SPARSITY produced an average speedup of 31.65% while STOMP produced an average speedup of 50.96% on the same set of sparse matrices. The research questions that this study addresses are:

- Given diversity of sparse matrices, is it possible to accurately predict performance of a new sparse matrix prior to SpMV operation?
- Is there a way to characterize sparse matrices during an SpMV operation, such that this characterization remains valid across sparse matrices with diverse and unpredictable nonzero patterns?
- Can we reuse performance information from previous workload executions to tune a new workload prior to its execution?
- Can statistical prediction techniques be ported across different processor architectures?

Although STOMP is evaluated on SpMV, it shows applicability across different data-driven workloads where performance is significantly influenced by input data. Chapter 2 discusses STOMP in detail and elaborates on its contributions.

The second study—Statistical Techniques for Analyzing Metrics and Predicting Performance of workloads (STAMPP)—predicts performance of diverse workloads across different

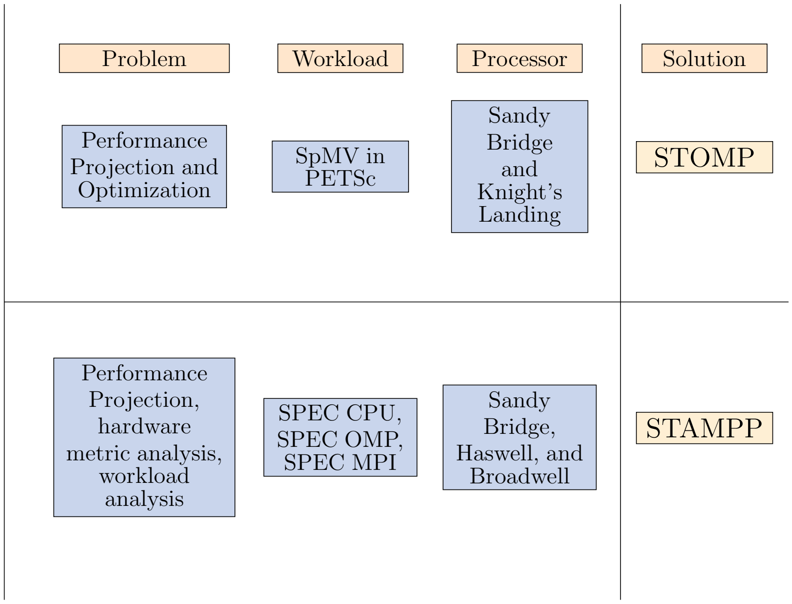
CPU architectures. This study offers techniques to analyze hardware metric space, identify sibling workloads, and formulate a suite of performance models to predict performance of new workloads on a target system. By analyzing metric space and eliminating redundant metrics, we found that we could trim hardware metric space by up to 53%. We then identified sets of benchmarks (sibling workloads) that could serve as performance proxies for a test workload. Findings show that the suite of performance models we built on these reduced metrics sets and benchmarks produced high prediction accuracy across different sets of Intel CPU architectures. The research questions this study addresses are:

- Are performance prediction models portable across different architectures?
- Is there a benefit to monitoring hardware components (cache and memory) multiple times?
- Can certain sets of workloads be used as performance proxies for a different workload?

Although STAMPP is evaluated on CPU processors, its general methodology is applicable across different processor architectures with relevant performance data. Chapter 3 contains the details of this study.

1.3 Dissertation Overview

Fig. 1.1 shows the problem space that this dissertation explores and the solutions that it contributes. Following this introductory chapter, the dissertation is divided into three additional chapters. Chapter 2 describes the development, deployment, and evaluation of STOMP. Chapter 3 describes metric analysis, sibling benchmark identification, and performance prediction techniques and evaluation in STAMPP. Chapter 4, the concluding chapter, examines the contributions of this dissertation to performance modeling and optimization and presents directions for future work.



Problem	Workload	Processor	Solution
Performance Projection and Optimization	SpMV in PETSc	Sandy Bridge and Knight's Landing	STOMP
Performance Projection, hardware metric analysis, workload analysis	SPEC CPU, SPEC OMP, SPEC MPI	Sandy Bridge, Haswell, and Broadwell	STAMPP

Fig. 1.1. Dissertation problem and solution space.

CHAPTER 2

STOMP: STATISTICAL TECHNIQUES FOR OPTIMIZING AND MODELING PERFORMANCE OF BLOCKED SPARSE MATRIX VECTOR MULTIPLICATION

2.1 Abstract

Sparse matrix vector multiplication (SpMV) is the core compute routine for several scientific and commercial codebases. Because of its extremely irregular memory accesses (low temporal locality), indirect memory referencing (low spatial locality), low arithmetic intensity, and the nonzero pattern and nonzero density of the matrix, SpMV achieves a mere 10% of peak system performance. Because sparse matrices have extremely varied nonzero patterns and densities, performance of SpMV is hard to predict. Blocking sparse matrices increases arithmetic intensity and spatial locality during SpMV operations, thereby improving SpMV performance. However, selection of an incorrect block size can produce performance degradation as high as 70%. In this study, we describe the *STOMP*[1] approach of using statistical techniques to predict run time of SpMV in PETSc for new matrices with mean accuracy of 93.52% on Sandy Bridge and 91.92% on Knight's Landing. We use these statistical prediction models to guide block size selection to achieve up to 100% of optimal performance, comparable to that attained through exhaustive block size search. Our block size selection results produce an average of 55.56% speedup over default SpMV options. On the same set of matrices used in the SPARSITY SpMV framework, STOMP yields a 50.96% speedup while SPARSITY yields a 31.62% speedup over the same default.

---

[1] Monteiro, S., Iandola, F., Wong, D., STOMP: Statistical Techniques for Optimizing and Modeling Performance of blocked sparse matrix vector multiplication, in Proceedings of 28th International Symposium of Computer Architecture and High Performance Computing, Los Angeles, California, October 2016. Following chapter content was not included in the publication: roofline model, evaluation of STOMP on KNL, additional error statistics on Sandy Bridge and Knight's Landing, and plots showing error distribution according to matrix specifications.

2.2 Introduction

Sparse matrix vector multiplication (SpMV) is a key computational block that underpins calculations across domains such as linear algebra solvers, Internet search engines, and recommendation systems. SpMV is represented as:

$$y = A \times x \tag{2.1}$$

Where A is a sparse matrix, and x and y are dense vectors.

SpMV is a memory-bound kernel on several architectures since its meager computational intensity is dominated by memory accesses. SpMV is known to operate at several orders of magnitude lower than the peak floating-point capability of a system. Because of its irregular memory access, techniques typically used to hide memory latency from users do not succeed for SpMV. For instance, hardware prefetchers that hide latency by bringing in streams of data from memory to cache are ineffective when dealing with irregular memory accesses. SpMV performance is further influenced by the sparsity and the nonzero pattern of the matrix on which it operates. To exacerbate the problem, SpMV is rarely executed just once within an algorithm, causing an underperforming SpMV code to become a performance bottleneck in large code bases. Iterative solvers such as BiCGSTAB [12] and GMRES [13] and algorithms such as PageRank's power method execute SpMV multiple times until they achieve convergence. Data mining algorithms such as linear regression and PageRank [14] work on extremely large matrix datasets to classify, predict, or rank data points. Section 2.3 examines different SpMV optimization techniques in detail.

Performance degradation in SpMV can be attributed to certain primary effects [15], [16]. These effects explain why common optimization strategies are ineffective for SpMV.

1. *Irregular memory access of dense vector* $\boldsymbol{x}$: In dense matrix vector multiplications, all elements of vector x are used by each element across a row of the matrix. In SpMV, few elements of the vector are accessed irregularly thwarting any possibility of substantial reuse of elements in vector x.

2. *High memory intensity*: SpMV is a memory bound code because of its high number of memory accesses in comparison to its arithmetic operations. Low floating point operations (FLOPs) guarantee that SpMV will never quite achieve peak compute performance.

3. *Indirect memory references for matrix* ***A***: Because of the sparse distribution of elements in A, sparse matrix data structures only store pointers to nonzero elements by using pointer arrays. This many-layered indirect memory access eliminates any performance benefit that could be derived from hardware prefetchers.

4. *Loop overheads*: Most SpMV algorithms operate row-wise on a matrix. The unequal distribution of nonzeros in the matrix makes each row a different size. Compiler efforts to unroll loop iterations will not work because iteration count differs across rows and is also not provided at compile time. Additional loop instruction overhead further degrades SpMV performance.

In this work, we explore blocked SpMV performance in PETSc [17], [18] on a variety of sparse matrices from the University of Florida sparse matrix collection [19] with focus on matrices derived from the Boeing [20], Chen [21], FIDAP [22], Law [23], Bai [24] groups in addition to a set of matrices derived from very specific domains such as Amazon's book similarity network, a financial portfolio, web crawlers, etc.

2.3 Related Work

2.3.1 SpMV data layouts and algorithms

Researchers have developed several sparse storage formats to facilitate efficient memory access and smaller storage requirements during SpMV operations. These formats include coordinate format (COO) [25], compressed sparse format (CSR) [26], compressed sparse column format (CSC) [27], diagonal format (DIAG), ELLPACK (ELL) [28], Blocked ELLPACK (BELL) [29], jagged diagonal (JDS) [30], skyline (SKY) [31], blocked row column (BRC) [32], and blocked compressed sparse row (BCSR). In their work [33], Byun et al., describe the specifics of each of these structures. COO stores both the row and column index value for each nonzero element. CSR is partial to row order and therefore does not explicitly store row indices like COO. ELL and BELL are suited for matrices that have the same number of nonzero elements in every row. BRC is designed especially for high efficiency on GPUs. Because a sparse matrix is abstracted as a series of blocks, BCSR requires less storage for row pointers and column indices than CSR. However, explicit padding of zeroes in each block adds to the number of FLOPs and amount of storage, leading to non-essential computations and storage.

In addition to new data layouts, a number of other algorithmic optimizations for SpMV have been proposed. When optimizing SpMV algorithms, a worthwhile goal is to minimize cache misses and to use the memory hierarchy more effectively. Toward this goal, a number of approaches for reordering matrix rows or columns have been proposed, including: Approximate Minimum degree algorithm [34], Reverse Cuthill-McKee (RCM) [35], and the Traveling Salesman Problem (TSP) [36]. One challenge is that these reordering techniques are often more computationally intensive than the SpMV operation, so these techniques are primarily useful in cases where the reordering time can be amortized by multiplying the matrix by many different vectors.

2.3.2 SpMV implementations

A number of highly efficient sparse linear algebra libraries have been developed, including pOSKI [33], SPARSITY [11], and PETSc [17] [18]. PETSc has been widely adopted, with over 2000 citations and a number of headline use-cases in computational fluid dynamics, biology, and other scientific applications. In this work, we conduct experiments on a modified version of the PETSc library. Many of the aforementioned PETSc-based applications stand to benefit from our work.

2.3.3 Importance of block size selection

The University of Florida sparse matrix collection [19] provides thousands of different sparse matrices taken from different applications. This matrix collection also provides a visualization of the sparsity pattern in each matrix. From these visualizations, it is easy to see that a general-purpose SpMV implementation will face an enormous range of matrix sizes and sparsity patterns. Vuduc [37], Karakasis et al. [38], and others have shown that the optimal settings of parameters such as *block size* vary depending on the matrix. Vuduc [37] reports that BCSR with optimal block sizes yields a speedup of up to 4x over CSR across diverse matrices and architectures. However, Vuduc also finds that the "wrong" block size can lead to slowdowns compared to CSR. While our focus is on the popular BCSR data layout, the block size selection is a recurring problem across a number of sparse formats such as BELL and blocked row column (BRC). In summary, selecting the appropriate block size is a prerequisite to achieving maximum efficiency on blocked SpMV calculations.

2.3.4 Predicting and modeling SpMV execution time

In our view, the most sensible way to select the right block size for a new matrix is to accurately predict the execution time for each block size, and then simply select the block size that we predict will execute most quickly. Thus, the crux of this problem is *automatically predicting SpMV execution time.* A number of heuristics and strategies have been proposed to predict the execution time of sparse matrix computations.

Load balancing is a common problem in parallel SpMV, where it is important to assign roughly the same amount of work (in terms of execution time) to each thread, processor, or server. Of course, to do this it is necessary to estimate the execution time of a matrix or submatrix. As part of distributed SpMV load-balancing logic, Liu and Vinter [39], Grigori and Li [40], and pOSKI [33] all use the following heuristic for predicting SpMV execution time. The heuristic is: counting the number of nonzero elements (NNZE) per row, and dividing NNZE evenly across parallel work units. Thus, it seems that the "conventional wisdom" is that the NNZE in a matrix (or a group of rows within a matrix) is a good indicator considering how much time will be required to execute SpMV on a particular matrix. With this in mind, we will use NNZE as a starting point in our experiments throughout this paper, and we will also propose an alternative metric that we find leads to higher accuracy in predicting the SpMV execution time on a given matrix.

2.3.5 Complementary approaches

While we focus on block size, Sedaghati et al. [41] use machine learning techniques to select the best matrix storage structure (e.g. CSR, ELL, etc.) for a sparse matrix after training and testing on several models. Guo et al. [42] analyze and predict SpMV performance on GPUs for CSR, ELL, COO, and HYB SpMV kernels. The method uses several GPU architecture features, possibly limiting its applicability on CPU architectures.

Our work uses statistical models trained on SpMV performance data from PETSc over a set of matrices. We use these models to predict SpMV performance of a new, unseen matrix before it is executed. We use these trained statistical models to predict optimal block size for a new sparse matrix, prior to its SpMV execution. Long term, we see potential for unifying our prediction and selection techniques for block size with these approaches to optimally configure linear algebra libraries without requiring a massive brute-force grid search over all possible configurations for each new matrix.

2.4 Understanding Blocked SpMV Performance

Blocking sparse matrices upgrades SpMV performance through improved spatial locality. PETSc provides tuned blocked SpMV routines for block sizes of 2×2 to 7×7. Blocks are always square. Fig. 2.1 is an example of blocking a sparse matrix with 8 rows and 8 columns using block size 2 (2×2 square blocks). As seen in Fig. 2.1, a blocked matrix is characterized by the number of blocked rows it contains (here, 4). Blocked rows in turn can be characterized by the number of nonzero blocks that they contain. A nonzero block is one that contains at least one original nonzero entry from the sparse matrix.

By re-arranging the matrix into blocks, sequences of bytes are brought into cache from memory making memory accesses more efficient. After blocking the matrix, each square block is padded with zeroes in places where the sparse matrix has no entries. Because of this nonzero padding, the upper bound on the number of FLOPs per block is $2 \times b\_size^2$ This is derived by examining the number of arithmetic operations needed while multiplying one block of a matrix with a dense vector. For b_size rows of a block, there are b_size multiplications, $(b\_size\text{-}1)$ additions (from adding results of matrix vector multiplication), and b_size additions (from adding results to vector y).

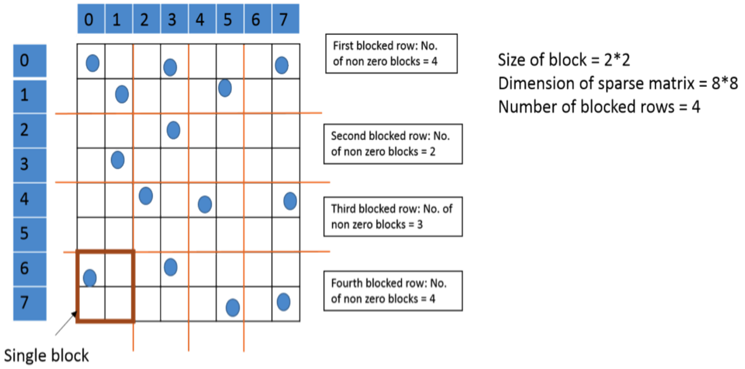

0
1
2
3
4
5
6
7
First blocked row: No. of non zero blocks = 4
Second blocked row: No. of non zero blocks = 2
Third blocked row: No. of non zero blocks = 3
Fourth blocked row: No. of non zero blocks = 4
Size of block = 2*2
Dimension of sparse matrix = 8*8
Number of blocked rows = 4
Single block

Fig. 2.1. Blocking a sparse matrix.

The upper bound on the number of bytes needed for each block is $(b\_size^2+2\times b\_size)\times 8$ Where, b_size^2 is the total number of operands in a block and $2 \times b\_size$ are operands from b_size sections of vectors x and y. The flop-to-byte ratio, a measure of bandwidth and compute utilization, is higher for larger block sizes in SpMV. A higher ratio indicates a compute bound kernel while a lower value indicates a memory bound kernel. For block size = 1 (no blocking), blocked SpMV is more memory bound with a very poor arithmetic intensity. The roofline model [43] calculated on Sandy Bridge in Fig. 2.2 shows arithmetic intensity of serial blocked SpMV across different block sizes. Table 2.1 details arithmetic intensity of blocked SpMV.

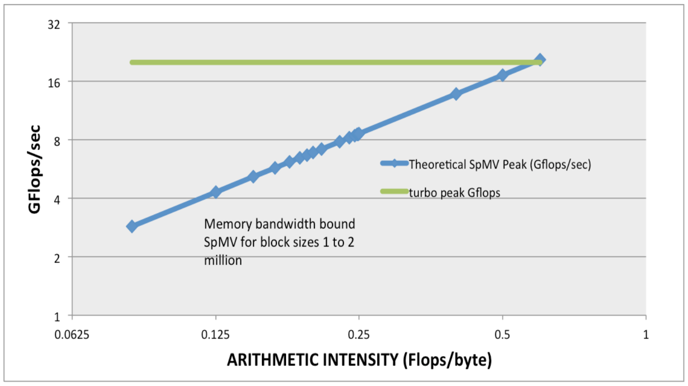


Fig. 2.2. Roofline model of serial blocked SpMV on Sandy Bridge.

Table 2.1. SpMV arithmetic intensity for different block sizes

Block size	Flop count	Bytes	Arithmetic intensity
1	2	24	0.09
2	8	64	0.12
3	18	120	0.15
4	32	192	0.16
5	50	280	0.17
6	72	384	0.18
7	98	504	0.19

However, increasing block size does not guarantee a significant increase in compute intensity. Even for purely theoretical block sizes of 1 to 2 million (for arithmetic intensity greater than 0.19), SpMV still remains memory bound. With an increase in block size, overheads from transferring large blocks from memory to caches also increase. This overhead propagates and manifests in a higher SpMV run time. Large block sizes have a large number of wasted computations due to additional FLOPs from zero padding. A low number of essential FLOPs per block ensure that SpMV does not reach the peak FLOP capability of a machine.

2.5 Experimental Environment

We investigate SpMV performance in PETSc (version 3.5.3) [18]. We compiled PETSc with GNU compilers, and we executed our experiments on an Intel Xeon E5-2670 multicore CPU. This CPU has 16 cores and an L1 cache size of 32K, L2 size of 64K, and an L3 size of 20480K. PETSc provides tuned SpMV kernel code for block sizes 2 through 7. We instrument each tuned MatMultSeqBAIJ [44] routine inside PETSc source code to collect timing information for SpMV operations on each row of the matrix that it operates on. Our dataset comprises sparse matrices from the University of Florida sparse matrix collection [19]. Our dataset contains a range of matrices with nonzero patterns and dimensions representative of scientific and commercial applications (Amazon, DBLP, financial portfolios, etc.). Tables 2.2, 2.3, and 2.4 describe our dataset of matrices in detail.

2.6 Predicting SpMV Performance

2.6.1 Algorithms for predicting performance

As introduced earlier in the paper, an important question is, *“before performing SpMV with a given sparse matrix, can we estimate how long the SpMV takes to execute?”* Our approach is to apply a straightforward statistical learning technique—linear regression—to make this prediction given a number of other matrices to be analyzed offline.

We execute SpMV on our dataset of matrices listed in Tables 2.2, 2.3, and 2.4. Each matrix is multiplied once with a dense vector that is randomly populated at run time. For each matrix execution, we collect timing information for each blocked row as the SpMV code works its way through all the blocked rows.

Our statistical model is a linear regression model that uses matrix structure characteristics to predict matrix run time. Conventional wisdom suggests using the *number of nonzero* elements (NNZE) to quantify SpMV work. NNZE is the metric used by others such as [40] and [33] for estimating computational cost during SpMV operations (e.g. for dividing matrices over multiple servers/threads). The following works appear to believe that “number of nonzero elements” is the key metric to consider when partitioning sparse matrices across multiple processors for blocked SpMV and related algorithms:

Table 2.2. Bai matrix group

Matrix	Order	Nonzeros	Density%	Application
Af23560	23560	460598	0.08	Airfoil
Cdde1	961	4681	0.5	Differential equation
Ck656	656	3884	0.9	Eigen values
Cryg10000	10000	49699	0.05	Crystal growth
Cryg2500	2500	12349	0.19	Crystal growth
Mhd3200a	4800	102252	0.44	Multiple Eigen values
Olm5000	5000	19996	0.08	Olmstead flow model
Rdb5000	5000	29600	0.12	Brusselator model
Tols340	340	340	1.89	Tolosa
Tub100	100	396	3.96	Tubular reactor model

Table 2.3. Mixed matrix group

Matrix	Order	Nonzeros	Density%	Application
Pwtk	217918	11524432	0.02	Tunnel sim.
Linverse	11999	53998	0.03	Inverse
Finan512	74752	596992	0.01	Financial
Conf5-49152	49152	1916928	0.08	QCD
Conf5-3072	3072	119808	1.26	QCD
Cantilever	62451	4007383	0.10	FEM
G3circuit	1585498	7660826	0.0003	Circuit sim.
Amazon-2008	735323	5158388	0.0009	Book network
Dblp-2010	326186	1615400	0.001	Bibliography
In-2004	1382908	16917053	0.0008	Webcrawl .in
Indo-china-2004	7414866	194109311	0.0003	Webcrawl .in .cn

Table 2.4. Boeing-Chen-FIDAP matrix group

Matrix	Order	Nonzeros	Density%	Application
Bcsstk36	23052	143140	0.21	Shock absorber
Bcsstk37	25503	1140977	0.17	Track ball
Bcsstk38	8032	355460	0.55	Airplane component
Crystk03	24696	1751178	0.28	FEM crystal
Ex13	2568	75628	1.15	Axisymmetric flow
Ex35	19716	227872	0.05	2D turbulent flow
Ex40	7740	456188	0.76	3D die swell
Ex8	3096	90841	0.94	Developing flow
Pkustk01	22044	979380	0.20	Civil engineering
Pkustk02	10800	810000	0.69	Civil engineering
Pkustk03	63336	3130416	0.07	Civil engineering
Pkustk04	55590	4218660	0.14	Civil engineering
Pkustk05	37164	2205144	0.16	Civil engineering
Pkustk06	43164	2571768	0.14	Civil engineering
Pkustk07	16860	2418804	0.85	Civil engineering
Pkustk09	33960	1583640	0.14	Civil engineering
Pkustk10	80676	4308984	0.07	Civil engineering
Pkustk11	87804	5217912	0.07	Civil engineering
Pkustk12	94653	7512317	0.08	Civil engineering
Pkustk13	94893	6616827	0.07	Machine element

- When performing blocked SpMV on multiple processors, Liu et al. partition the matrix into groups of rows with approximately equal numbers of nonzeros [45]. In the same vein, Liu and Vinter use the number of nonzeros per row to partition rows across OpenCL work items for blocked SpMV [39].

Table 2.5. SPARSITY's matrix set

Matrix	Order	Nonzeros	Density%	Application
dense_1000	10000	1000000	100	Dense matrix
lhr10	10672	232633	0.20	Light hydrocarbon
raefsky3	21200	1488768	0.33	Fluid/structure
goodwin	7320	324784	0.61	Fluid mechanics
bcsstk35	30237	1450163	0.39	Automobile frame
bayer02	13935	63679	0.03	Chemical process
venkat01	62424	1717792	0.04	Flow simulation
bayer10	13436	94926	0.05	Chemical process
crystk02	13965	968583	0.50	FEM Crystal
coater2	9540	207308	0.23	Coating flows
crystk03	24696	1751178	0.29	FEM Crystal
finan512	74752	596992	0.01	Financial modeling
nasarb	54870	2698463	0.09	Shuttle booster
onetone2	36057	227628	0.02	Harmonic balance
3dtube	45330	3213332	0.16	3D pressure tube
pwtk	36519	326107	0.02	Structural engineering
ct20stif	52329	2698463	0.10	CT20 Engine block
vibrobox	12328	342828	0.23	Vibroacoustics
bai	23560	484256	0.09	Airfoil eigenvalue
wang4	26068	177196	0.03	Semiconductor devices
raefksy4	19779	1328611	0.34	Buckling problem
lnsp3937	3937	25407	0.16	Fluid flow
ex11	16614	1096948	0.40	3D steady flow
sherman5	3312	20793	0.19	Oil reservoir
rdist1	4134	94408	0.55	Chemical processes
osreg1	2205	14133	0.29	Oil reservoir
vavasis3	41092	1683902	0.10	2D PDE problem
saylr4	3564	22316	0.18	Airfoil
rim	22560	1014951	0.20	FEM fluid mechanics
shyy161	76480	329762	0.01	Viscous flow
memplus	17758	126150	0.04	Circuit simulation
wang3	26064	177168	0.03	Semiconductor devices
gemat11	4929	33185	0.14	Power flow
mcfe	765	24382	4.17	Astrophysics

- In blocked sparse LU factorization, Grigori and Li use the number of nonzeros as the sole metric for estimating matrix density [40].

- clSpMv framework uses the number of nonzeros as a criterion for automatically selecting between BCSR and COO storage formats [46].

- The pOSKI parallel sparse linear algebra framework performs autotuning on several

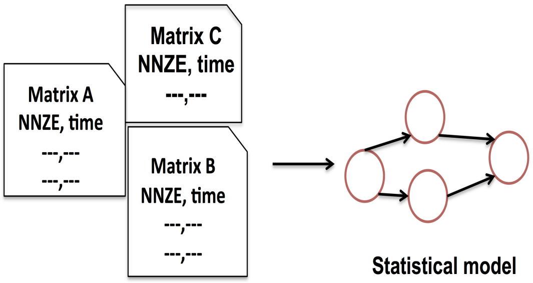


Fig. 2.3. SpMV performance model training.

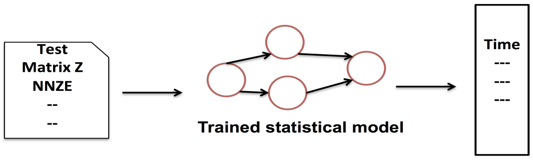


Fig. 2.4. Predicting SpMV performance of a new matrix.

aspects of computation (e.g. choice of blocked sparse storage format and choice of computational kernels), yet it only has one approach to matrix partitioning: dividing up matrices based on the number of nonzeros per row [33].

However, we think that NNZE may not be an ideal feature for predicting the computational cost of *blocked* CSR SpMV. Our intuition is that SpMV is typically memory bound, so much of the cost lies in data movement. The same overhead is paid to load a 2×2 block regardless of whether it has one nonzero or four nonzeros. Therefore, we propose characterizing a matrix in terms of its *number of nonzero blocks* (NNZB). Additionally, examining blocked SpMV time per blocked row of a matrix shows that a matrix characterized with NNZE has a noisier relationship with execution time than with NNZB. Noisy non-linear relationships digress from linear prediction models and require expensive statistical models to predict run time. For each blocked row in matrix *in-2004*, Fig. 2.5 and Fig. 2.6 show relationships between NNZE counts and NNZB counts, and SpMV time. The large divergence of data points from the linear fit line in the NNZE plot, compared to the smaller divergence in the NNZB plot, hints that NNZE might not be the best feature for predicting SpMV run time.

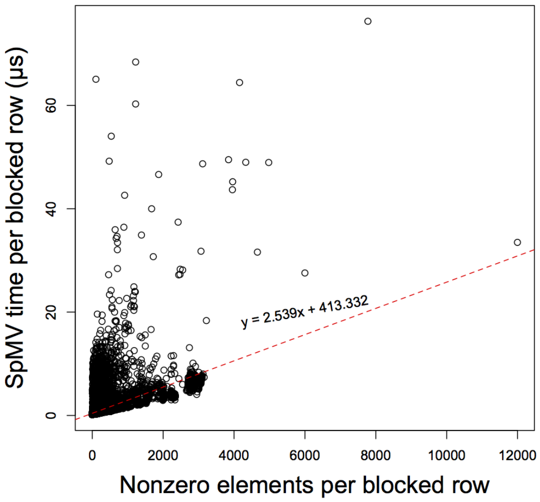


Fig. 2.5. NNZE vs. time for *in-2004*.

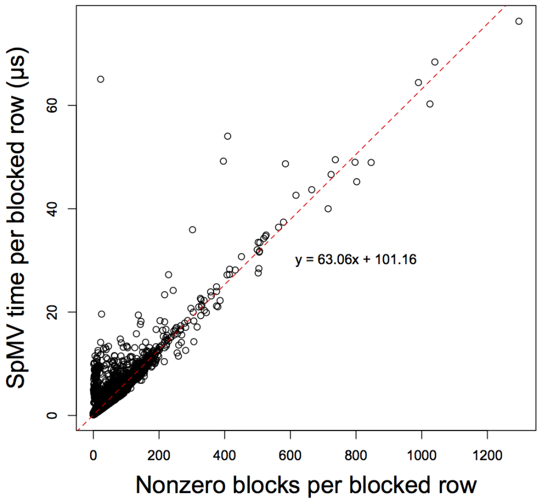


Fig. 2.6. NNZB vs. time for *in-2004*.

We conducted a Pearson correlation analysis to examine interactions between NNZE and NNZB across the three matrix groups across the range of block sizes. The overall trend is that the correlation strength decreases as the block size increases. There are some matrices where this decrease is not monotonic. Very sparse matrices, such as *amazon-2008 and* G3-circuit, have correlations as low as 0.26 between their NNZE and NNZB values.

2.6.2 Training SpMV performance prediction models

We train our performance prediction models on a set of matrices that have previously been used for an SpMV operation. Fig. 2.3 diagrammatically represents this training process. Because of widely different run times from different block sizes, each model is trained to predict run times corresponding to one block size. The linear regression models are of the following form

$$\begin{cases} Y_{ib=2} &= \beta x_{ib=2} + \epsilon_{ib=2} \\ Y_{ib=3} &= \beta x_{ib=3} + \epsilon_{ib=3} \\ \quad + & \vdots \\ Y_{ib=7} &= \beta x_{ib=7} + \epsilon_{ib=7} \end{cases}$$

where the Y_i are vectors of run times for different matrices i and the x_i are NNZB counts per blocked row of a matrix for NNZB performance models. x_i are NNZE counts per blocked row of a matrix for NNZE performance models. During model training and construction, the model uses NNZE (or NNZB) and time data from *N-1* matrices to predict run time of the N$^{\text{th}}$ matrix. Each block-size specific SpMV gets its own trained model.

2.6.3 Predicting SpMV performance of a matrix

To predict the run time of SpMV on a new matrix *prior* to its execution, the models require a vector containing the number of NNZE (or NNZB) counts within the matrix. The list of NNZE (or NNZB) per blocked row is obtained by parsing the matrix structure via a script. Predictions, i.e. run time $\hat{Y}_{b=i}$, for each block size are obtained using block-specific regression models.[2] For instance, performance prediction for matrix with a block size 2 can be represented as

$$\hat{Y}_{b=2} = \beta x_{test\_matrix\_b=2} + \epsilon_{b=2}$$

where $x_{test\_matrix\_b=2}$ is the vector of nonzero block counts derived from the test matrix structure. β and ϵ are coefficients derived from the model training process.

Prediction accuracy of a matrix's total run time is measured using the relative percentage error metric.

$$\text{Relative percentage error} = \frac{|\sum Y_{b=i} - \sum \hat{Y}_{b=i}|}{\sum Y_{b=i}} \times 100$$

Is NNZB a more useful feature than NNZE for estimating the SpMV computation time for a new sparse matrix? We now present a series of experiments to evaluate this question. For each of these groups of matrices, we train and test our models using an N-1 cross validation process. We train on N-1 matrices from our dataset, and use the resulting model to predict run time of the Nth matrix. Fig. 2.4 shows this prediction process. The next three subsections examine prediction accuracy from the NNZE model and the NNZB model

---

[2] To be clear, $\hat{Y}$ is the predicted runtime, and Y is the actual (measured) runtime.

for groups of matrices listed in Tables 2.2, 2.3, and 2.4. We examine prediction results from block size 6 for the three groups of matrices.

2.7 SpMV Performance Prediction Results

2.7.1 Prediction results for Boeing, Chen, and FIDAP group

Fig. 2.7 compares prediction errors for the NNZB and NNZE models on block size 6. The majority of the matrices in this group have better predictions from using the NNZB model. The prediction errors from the NNZB model average 8.12% compared to 43.16% for the NNZE model. Matrices from civil engineering applications (pkustk*) matrices have high prediction accuracy from the NNZB model. Some matrices with smaller dimensions and higher density of nonzeros suffer from higher prediction errors—*bcsstk38* (8032 rows) and *ex40* (3096 rows). Table 2.7 lists additional statistics that provide insight into the distribution of prediction errors across the matrix group for block size 6. The predictions from the NNZE model comes close to the NNZB model for matrices with smaller dimensions such as *ex40* (7740 rows). Unlike the NNZE model, the NNZB model's prediction accuracy does not deteriorate when faced with matrices with larger dimensions—*pkustk11* (87804 rows), *pkustk10* (80676 rows), *pkustk13* (94893 rows). Fig. 2.8 visualizes distribution of errors based on matrix size and nonzero density percentage for block size 6. Table 2.6 summarizes prediction errors for different block sizes for NNZE and NNZB. The NNZB model produces the highest accuracy across all block sizes.

2.7.2 Prediction results from Bai group

Prediction errors from using the NNZE and NNZB models are shown in Fig. 2.9. The NNZB model yields an average error of 4.17% across all matrices in this group, while the NNZE model averages 20.96%. *Tub100*, the matrix with the highest nonzero density (3.95%) in the group, has a high error for both the NNZE and the NNZB models.

Table 2.7 lists additional statistics that provide insight into the distribution of prediction errors across the matrix group for block size 6. Fig. 2.10 visualizes distribution of

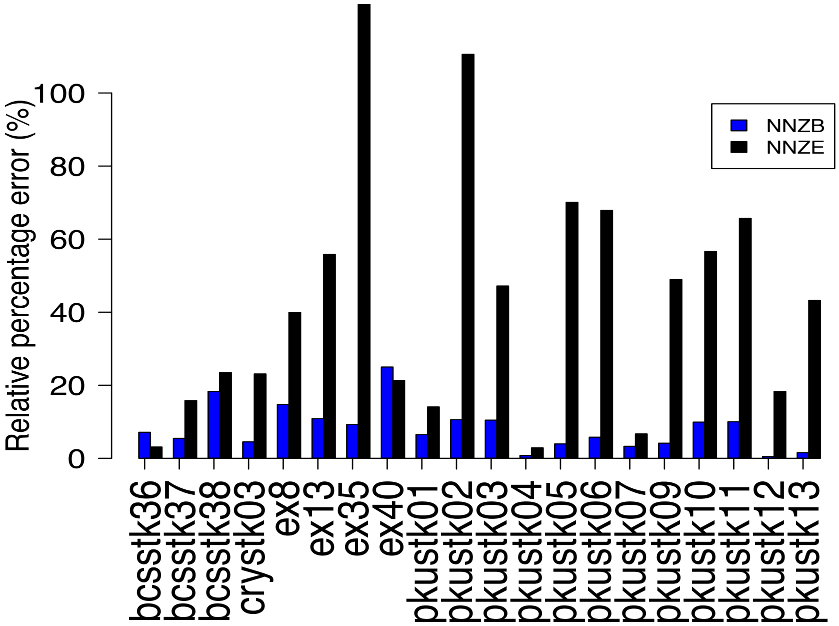


Fig. 2.7. Comparing prediction errors from NNZE and NNZB models for Boeing-Chen-FIDAP group of matrices.

errors based on matrix size and nonzero density percentage. A majority of the matrices have errors less than 5%. Table 2.6 lists the average prediction errors across all matrices in the Bai group and compares errors from the NNZE and the NNZB models. The NNZE model always performs poorly and its error increases quickly with block size.

2.7.3 Prediction results from the miscellaneous group of matrices

The miscellaneous group contains the most diverse set of matrices (Table 2.3) from key applications and domains that either use or produce sparse matrices as part of their computations. *Amazon-2008*, *in-2004*, *indochina-2004*, and *dblp* are very sparse graphs derived from crawling domains and websites. The key characteristics of these matrices are their unpredictable nonzero patterns and extremely low nonzero densities (0.0008% *in-2004*, 0.0003% *Indochina-2004*). Fig. 2.11 shows performance prediction results for this group from using the NNZE and the NNZB models. Overall, the NNZB model outperforms the

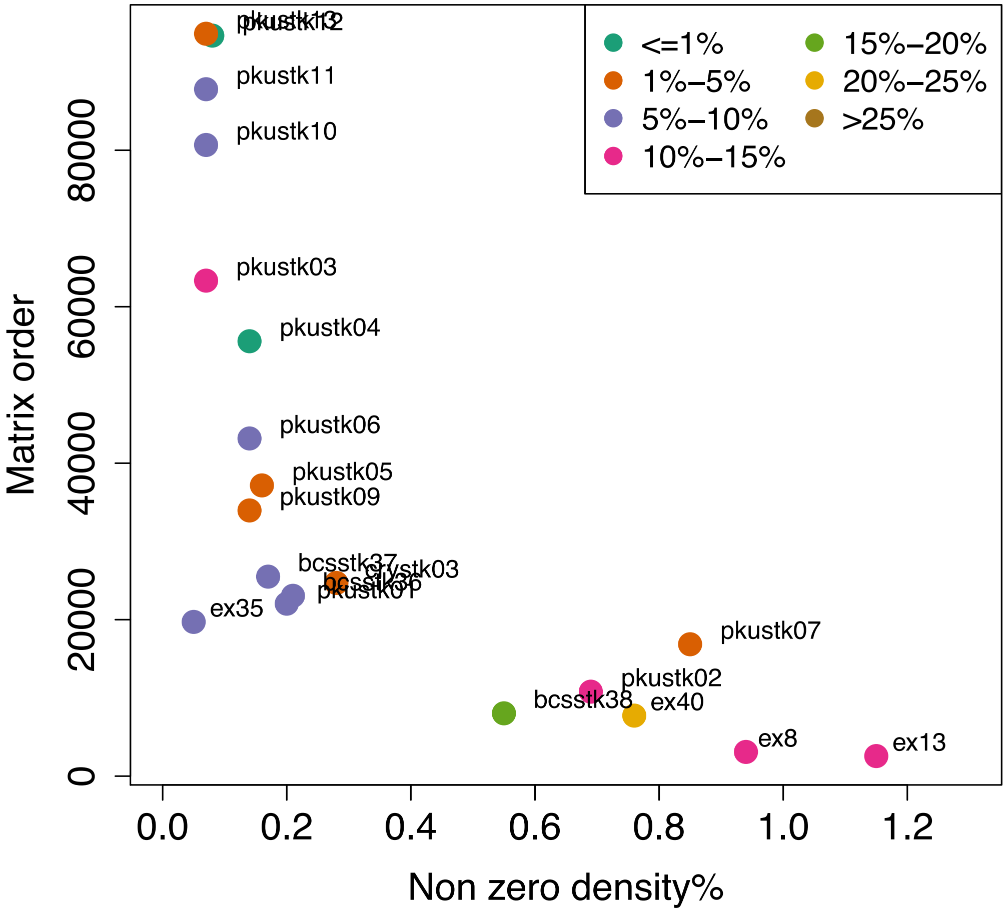


Fig. 2.8. NNZB model prediction error distribution with respect to matrix specifications for Boeing-Chen-FIDAP group.

NNZE model across a majority of the matrices. *Dblp* and *cantilever* show preference for the NNZE model for block size 6. However, errors from both NNZE and NNZB for these matrices lie below 10%. The average error for the NNZE model is 33.78% while the error from the NNZB model averages 11.11%. Table 2.6 shows average prediction errors from the NNZE and NNZB models over all matrices for different block sizes. An interesting observation is the 10% increase in errors for the NNZE model with the increase in block size. The error for the NNZB model stays constant at 11% across the three block sizes.

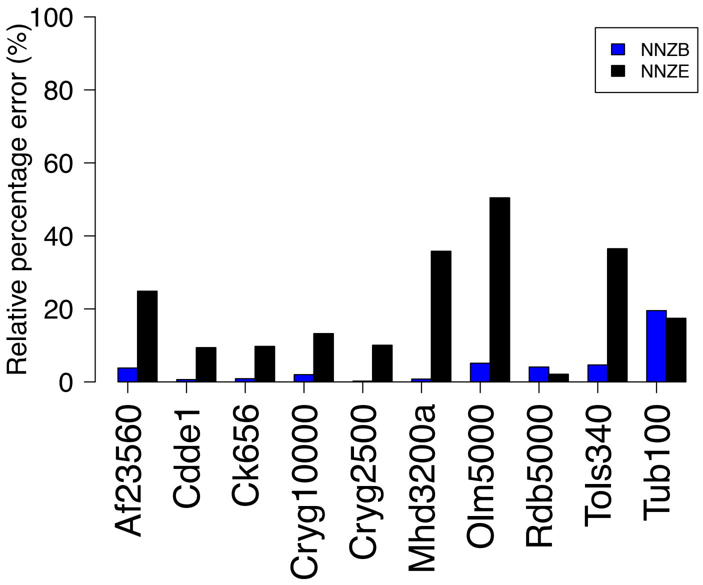


Fig. 2.9. Comparing prediction errors from NNZE and NNZB models for Bai group of matrices.

Table 2.7 lists additional statistics that provide insight into the distribution of prediction errors across the matrix group for block size 6. Fig. 2.12 visualizes distribution of errors based on matrix size and nonzero density percentage.

2.7.4 Cross-validation-based performance prediction across all matrix groups

To investigate performance accuracy over more diverse matrix sets, we conduct a stricter training and testing process using all matrices from all three groups. We sample without replacement two-thirds of the matrices from each group. This forms the randomized training set with representation from each matrix group to account for diverse nonzero patterns and densities. The test set comprises the remaining one-third of matrices from each matrix group. We repeat this process for ten trials. We train NNZB and NNZE models and make predictions on the test set for each trial. The results for each test matrix are averaged

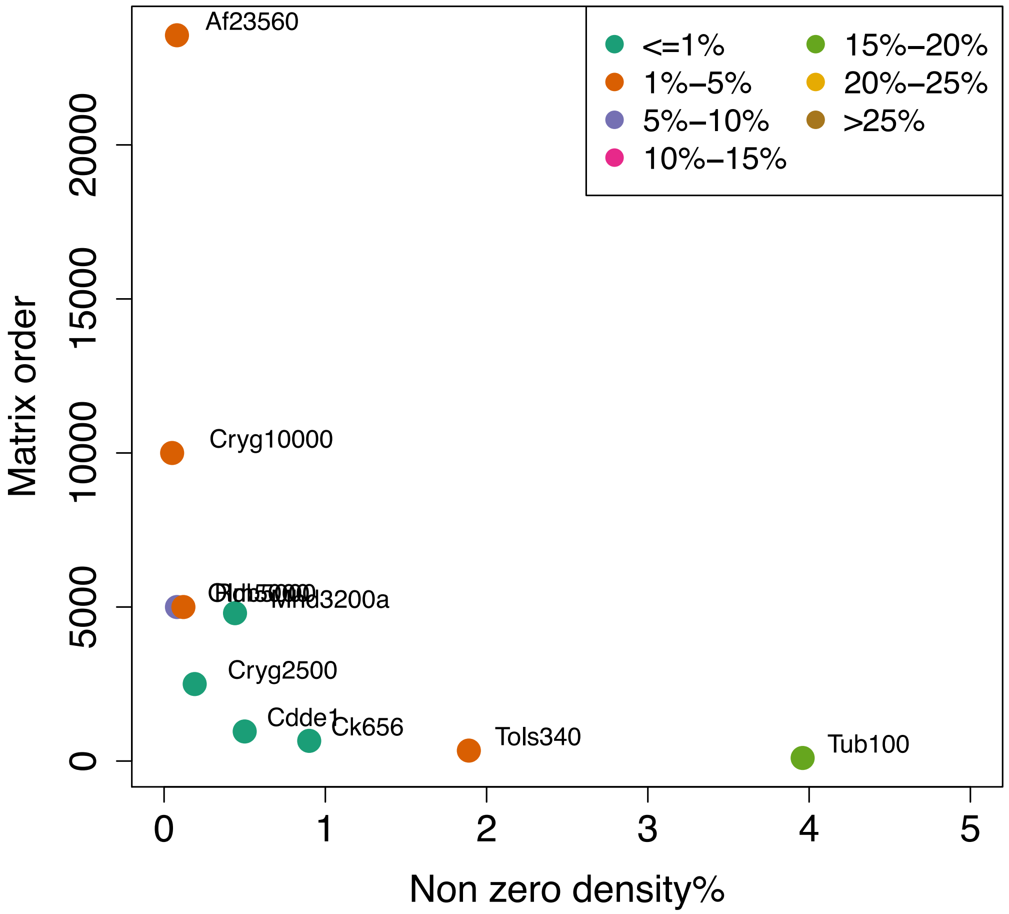


Fig. 2.10. NNZB model prediction error distribution with respect to matrix specifications for Bai group.

across the ten trials. Fig. 2.13 shows errors averaged across the ten trials for the NNZE and NNZB model for all matrices for block size 2. Because of the significant difference in errors between the two models, errors from the NNZE model are represented on a second Y-axis in Fig. 2.13.

A large number of denser matrices from our dataset show NNZB errors greater than 10%. Most matrices lying in the higher end of the nonzero density have NNZB errors greater than 10%; in particular, *cdde1* (density=0.5%), *ck656* (density=0.9%), *conf5-*

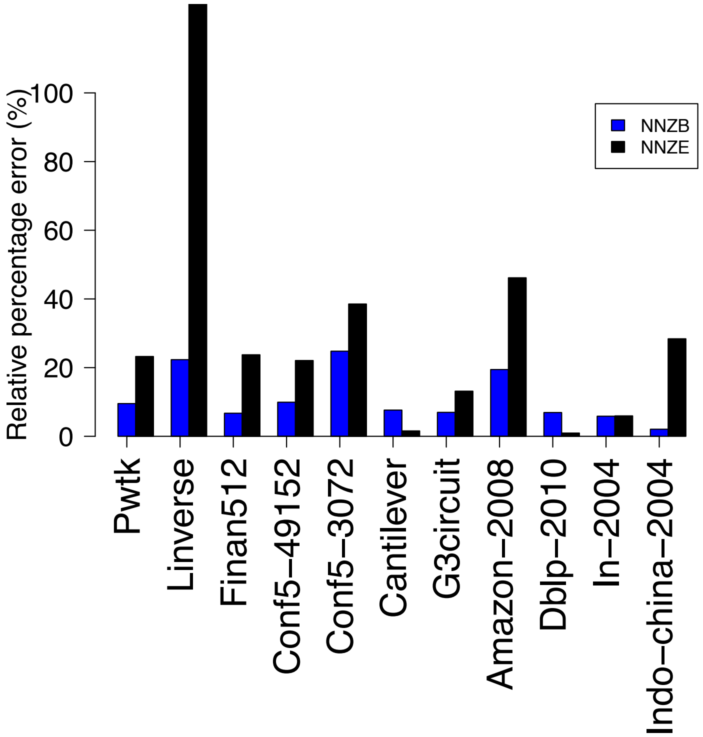


Fig. 2.11. Comparing prediction errors from NNZE and NNZB models for mixed group of matrices. (Miscellaneous group with greatest diversity.)

3072 (density=1.26%), *ex40* (density=0.76%), *mhd3200a* (density=0.44%), *pkustk02* (density=0.69%), and *tols340* (density=1.89%).

Since we randomly sample the matrix dataset, which is dominated by sparser matrices, the NNZB model at each trial does not contain sufficient training data to predict for denser matrices. *Amazon-2008* and *linverse* have high prediction errors similar to their prediction errors from models trained only on the mixed matrix group.

As shown in Fig. 2.13, our NNZB model outperforms the NNZE model for all matrices. The disadvantage of using NNZE to predict run time becomes clearer when using a diverse set of matrices with different nonzero patterns. The NNZE model averages a prediction error of 229.08% while the NNZB model averages 10.16%.

Table 2.6 lists prediction errors for additional block sizes. For block size 2, Table 2.7 shows additional statistics on prediction errors.

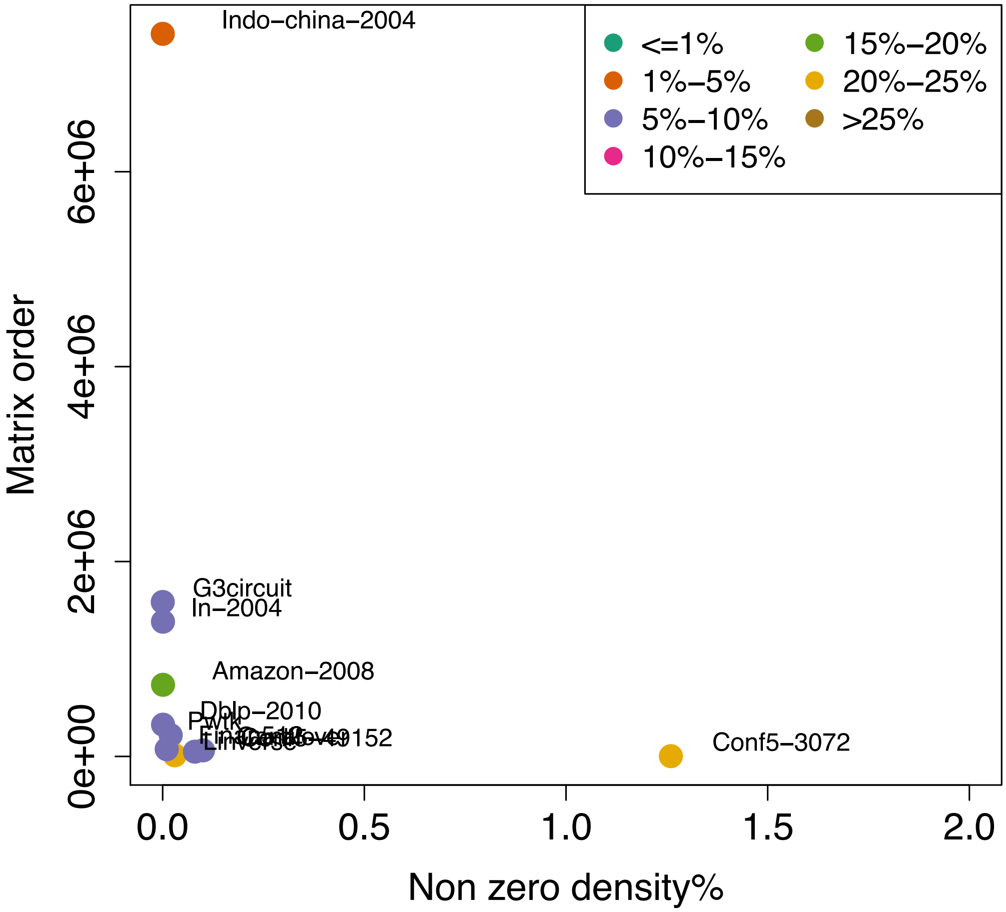


Fig. 2.12. NNZB model prediction error distribution with respect to matrix specifications for mixed group of matrices.

Table 2.6. STOMP SpMV performance prediction errors across different block sizes

	BOEING-CHEN-FIDAP		BAI		MIXED		TEN TRIALS	
Block size	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)
2	13.13	2.37	3.20	2.22	9.23	11.08	229.08	10.16
4	18.31	5.71	8.93	2.55	22.19	11.03	181.84	11.78
6	43.16	8.12	20.95	4.17	33.78	11.11	187.79	14.35

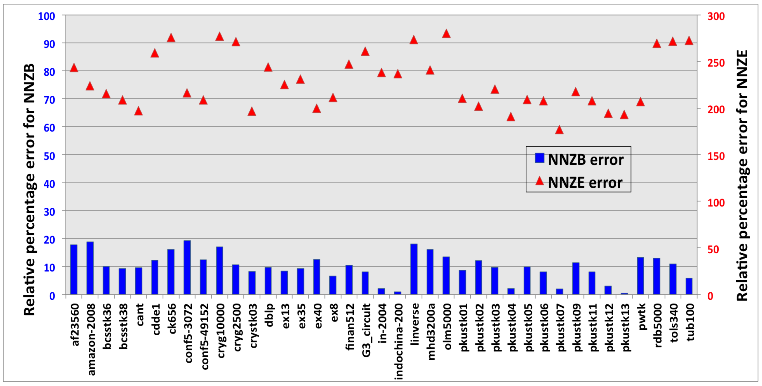


Fig. 2.13. Performance prediction errors from inter-matrix group cross-validation (block size 2).

Table 2.7. Distribution of STOMP SpMV performance prediction errors on Sandy Bridge

	BOEING-CHEN-FIDAP		BAI		MIXED		TEN TRIALS	
Statistic	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)
Minimum	2.84	0.47	2.12	0.23	0.95	2.06	176.84	0.48
Average	43.16	8.12	20.96	4.17	33.79	11.11	229.09	10.16
Maximum	128.82	24.98	50.44	19.53	167.90	24.79	280.06	19.33
Standard deviation	33.29	5.92	14.67	5.41	44.58	7.15	29.22	4.89

2.8 Guiding Optimal Block Size Selection

While working with sparse matrices, it is difficult to know what block size would deliver optimal performance. End users typically guess and use small, even-sized blocks. Another option is to conduct an exhaustive search, which is slow and wastes computational time, particularly if there are many matrices.

We use block-specific prediction models described in previous sections to guide block size selection of a matrix.

1. We extract the number of nonzero blocks for a new, test matrix for a range of block sizes.

2. These vectors of nonzero block counts are input to block-size specific performance

models described in Section 2.6 to obtain SpMV run times for the matrix for a particular block size. During the prediction process, each model (corresponding to one block size) produces a vector of run times.

3. Block size of the model that yields the minimum run time is selected as the optimum block size.

2.8.1 Evaluating model-guided block size on SpMV performance

To examine the optimality of predicted block size, we execute SpMV using predicted block size to determine performance gain over other block size choices. In this section, we present block prediction results from all matrices from all groups. We use two metrics to evaluate the quality of our predicted block selections—*speedup left on the table* and *speedup over default*.

Speedup left on the table: This is the relative percentage difference between the best SpMV performance obtained through exhaustive search over our block size range and that obtained from using block size predicted by STOMP. A lower number, close to zero, shows that there is no wasted performance optimization opportunity.

Speedup over default: As discussed earlier, PETSc leaves block size selection to the end user. PETSc defaults to using block size 1 in the event that the user does not specify a block size. Block size 1 is non-blocked SpMV. Speedup over default is the relative percentage difference between SpMV performance from using block size 1 (unblocked SpMV) and the performance obtained from using block size from our model. Tables 2.8, 2.9, and 2.10 examine speedup derived from using our performance models across the three matrix datasets. Table 2.13 shows average speedup results from all matrix groups used in our study.

In the Boeing-Chen-FIDAP group, *speedup over default* produced by STOMP lies between a minimum of 13.46% (pkustk13) and a maximum of 72.24% (pkustk09). Average *speedup over default* in this group is 53.35%. While all matrices in this group achieve optimal SpMV performance from STOMP's block size selection, three matrices leave a small percentage of speedup on the table—*bcsstk36* (1.81%), *bcsstk37* (0.03%), and *ex8* (2.12%). Average *speedup left on the table* is 0.20%. Table 2.8 lists speedup details on all matrices in this group.

In the Bai group, *speedup over default* produced by STOMP lies between a minimum of 56.40% (cdde1) and a maximum of 73.39% (olm5000). Average *speedup over default* in this group is 64.03%. Two leave a small percentage of speedup on the table—*mhd3200a* (0.89%), and *tub100* (13.7%). Average *speedup left on the table* is 1.46%. Table 2.9 lists speedup details on all matrices in this group.

In the diverse group of mixed matrices, *speedup over default* produced by STOMP lies between a minimum of 22.37% (amazon-2008) and a maximum of 75.53% (linverse). All matrices in this group achieve 100% of their optimal SpMV performance, leaving no speedup on the table. Table 2.10 lists speedup details on all matrices in this group.

2.8.2 Comparing STOMP to SPARSITY

The SPARSITY framework [11] attempts to select optimal block size by analyzing sparse matrix structure and machine profile in detail. The framework requires machine profiling to find optimal block sizes for a set of matrices. SPARSITY then uses a simple heuristic to find the best block size for a matrix. The heuristic selects a block with dimensions $r \times c$ for a matrix B that maximizes

$$\frac{\textit{Performance of dense matrix in } r \times c \textit{ sparse blocked format}}{\textit{Estimated fill overhead for } r \times c \textit{ blocking of } B}$$

This static heuristic selects a block size that minimizes the number of non-essential floating point operations (FLOPs) that arise from excessive nonzero padding.

To compare with SPARSITY, we evaluate STOMP's block-size selection on the set of matrices used in the SPARSITY study. This matrix set is listed in Table 2.5. Table 2.12 compares speedup obtained from STOMP and SPARSITY over the same set of matrices.

Table 2.8. STOMP SpMV speedup on Boeing-Chen-FIDAP matrices

Matrices	Speedup left on the table(%)	Speedup over default(%)
bcsstk36	1.81	54.75
bcsstk37	0.03	54.56
bcsstk38	0	52.01
crystk03	0	52.44
ex8	2.12	54.87
ex13	0	54.01
ex35	0	51.50
ex40	0	40.04
pkustk01	0	55.93
pkustk02	0	71.23
pkustk03	0	69.97
pkustk04	0	39.83
pkustk05	0	70.04
pkustk06	0	69.24
pkustk07	0	22.63
pkustk09	0	72.24
pkustk10	0	69.58
pkustk11	0	68.32
pkustk12	0	30.29
pkustk13	0	13.46

Table 2.9. STOMP SpMV speedup on Bai group of matrices

Matrices	Speedup left on the table(%)	Speedup over default(%)
af23560	0	62.12
cdde1	0	56.40
ck656	0	63.43
cryg10000	0	64.20
cryg2500	0	64.79
mhd3200a	0.89	60.84
olm5000	0	73.39
rdb5000	0	64.96
tols340	0	68.67
tub100	13.7	61.47

Table 2.12 compares performance benefits from SPARSITY and STOMP. Though coarse, this summation of results shows that statistical modeling provides several benefits over using rough, static heuristics:

1. On the set of matrices used in the SPARSITY study (listed in Table 2.5), STOMP produces an average speedup of 50.96% over default, unblocked SpMV while SPARSITY produces a speedup of 31.62%

Table 2.10. STOMP SpMV speedup on mixed group of matrices

Matrices	Speedup left on the table(%)	Speedup over default(%)
pwtk	0	54.28
linverse	0	75.53
finan512	0	33.44
conf5-49152	0	52.78
conf5-3072	0	57.17
cantilever	0	46.87
G3_circuit	0	52.68
amazon-2008	0	22.37
dblp	0	47.02
indochina-2004	0	47.01
in-2004	0	53.07

2. On matrices in Table 2.5 when compared to exhaustive search, STOMP leaves as little as 0.31% performance on the table. On the other hand, SPARSITY leaves behind 2.99%. Table 2.11 lists speedup from STOMP over each matrix used in SPARSITY.

3. To predict appropriate block dimensions of a matrix, SPARSITY requires execution of a dense matrix in 144 blocked configurations over 12 choices of row and column block dimensions. STOMP yields optimum performance using an extremely small set of matrices—a minimum of 11 (Bai group) and a maximum of 33 (SPARSITY's set of matrices). Statistical prediction models learn and adapt using information from smaller datasets leading to better predictions than those obtained from simple and static heuristics.

2.9 Evaluating STOMP on Knight's Landing

Knight's Landing (KNL) [47] is a recently released (July 2016) co-processor in Intel's Xeon Phi line. While KNL has a completely different architecture from processors in Intel's Xeon line, a stark difference is KNL's lower clock rate and higher number of cores. We used a self-bootable Intel ^{R} Xeon Phi^{TM} Processor 7210 node for our experiments. This KNL node has an L1 cache of 32K and an L2 cache of 1024K.

Table 2.11. STOMP SpMV speedup evaluation on SPARSITY's matrices

Matrices	Speedup left on the table(%)	Speedup over default(%)
dense_1000	3.05	66.14
raefksy3	0	65.78
bcsstk35	0	68.84
venkat01	0	70.19
crystk02	0	57.58
crystk03	0	56.69
nasarb	0	67.42
3dtube	0	53.80
ct20stif	0	44.88
bai	0	61.76
raefsky4	0	42.88
ex11	0	42.72
rdist1	2.09	57.26
vavasis3	0	31.69
rim	0	22.65
memplus	0	56.18
gemat11	0	46.10
lhr10	0	50.84
goodwin	0	31.57
bayer02	0	53.03
bayer10	0	54.72
coater2	4.09	30.75
finan512	0	35.97
onetone2	0	42.17
pwtk	0	54.09
vibrobox	0	5.42
wang4	0	58.24
lnsp3937	0	54.70
sherman5	0	69.85
osreg1	0	58.15
saylr4	0	52.75
shyy161	0	63.10
wang3	0	58.48
mcfe	2.05	16.37

Table 2.12. STOMP versus SPARSITY

Criteria	Matrices	STOMP	SPARSITY
Average speedup left on the table	SPARSITY matrices	0.31%	2.99%
Average Speedup over default SpMV	SPARSITY matrices	50.96%	31.62%

Table 2.13. Evaluating STOMP's block size predictions on all matrix groups

Matrix group	Speedup left on the table(%)	Speedup over default(%)
Bai	1.46	64.03
Boeing-Chen-FIDAP	0.20	53.35
Mixed set	0	49.29
SPARSITY's matrix set	0.31	50.96
Average across all groups	0.49	54.46

We executed SpMV in serial on the Boeing-Chen-FIDAP, Bai, and mixed matrix groups. Our experiment setup remains the same as in the Sandy Bridge study presented earlier. The following sections (i) evaluate STOMP's SpMV prediction accuracy on three matrix groups and (ii) evaluate the quality of STOMP's block size selection heuristic.

Similar to the study on Sandy Bridge, we evaluate STOMP's prediction accuracy for block size 6 in detail.

2.10 STOMP's Performance Prediction on Knight's Landing

2.10.1 Prediction results from the Boeing-Chen-FIDAP group of matrices

Fig. 2.14 shows prediction errors for the NNZB and the NNZE models for block size 6. A majority of matrices in this group benefit from using the NNZB model. The NNZB and NNZE models appear to compete in matrix *pkustk09*. Prediction errors for this group average 12.83% for the NNZB model and 45.68% for the NNZE model. Table 2.15 lists additional statistics that provide insight into the distribution of errors across the matrix group for block size 6. Fig. 2.15 visualizes distribution of errors based on matrix size and nonzero density percentage. Table 2.14 shows errors across additional block sizes. The NNZB model performs consistently better than the NNZE model across all block sizes.

2.10.2 Prediction results from the Bai group of matrices

Fig. 2.16 shows prediction errors for the NNZB and the NNZE models on block 6 for the Bai group. A majority of matrices in this group benefit from using the NNZB

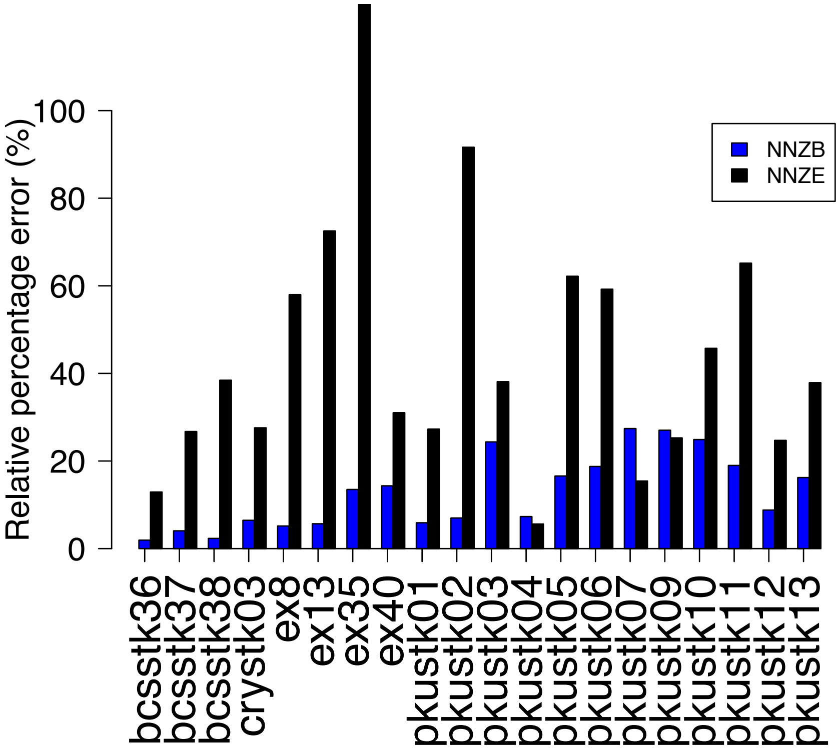


Fig. 2.14. Comparing predictions from NNZE and NNZB models for Boeing-Chen-FIDAP matrices on Knight's Landing.

model. The NNZE model beats the NNZB model for *tub100*. *Tub100* is the densest matrix (3.95%) in our entire collection that comprises matrices with nonzero densities of less than 0.1%. Since our prediction models are trained on very sparse matrices, predictions for denser matrices are not as accurate. Prediction errors for this group average 3.62% for the NNZB model and 19.65% for the NNZE model. Table 2.15 lists additional statistics that provide insight into the distribution of errors across the matrix group for block size 6. Fig. 2.17 visualizes distribution of errors based on matrix size and nonzero density percentage. While all errors lie under 10 %, *tub100* with its high nonzero density has the highest error at 19.41%. Similar to the study on Sandy Bridge, the NNZB model beats the NNZE model across all block sizes in this group, as shown in Table 2.14

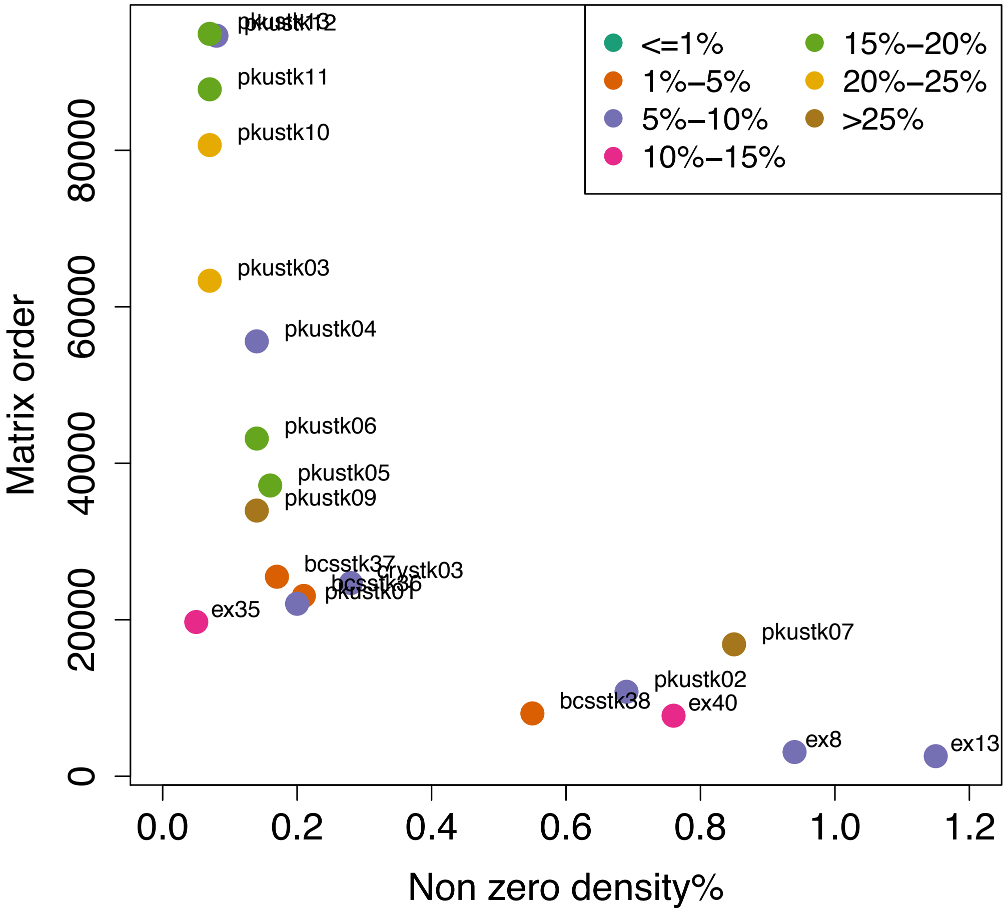


Fig. 2.15. NNZB model prediction error distribution with respect to matrix specifications for Boeing-Chen-FIDAP on Knight's Landing.

Table 2.14. STOMP SpMV performance prediction errors on Knight's Landing

	BOEING-CHEN-FIDAP		BAI		MIXED		TEN TRIALS	
Block size	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)
2	7.87	2.93	4.36	2.04	19.39	15.78	1461.1	17.10
4	16.48	5.36	9.70	5.94	15.85	10.54	1352.2	9.76
6	45.68	12.83	19.65	3.62	26.35	13.64	1195.7	15.03

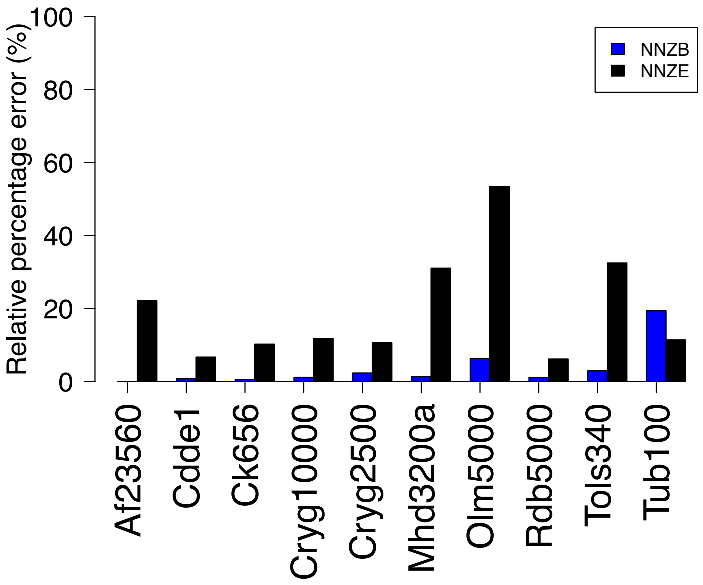


Fig. 2.16. Comparing predictions from NNZE and NNZB models for Bai matrices on Knight's Landing.

Table 2.15. Distribution of STOMP SpMV performance prediction errors on Knight's Landing

	BOEING-CHEN-FIDAP		BAI		MIXED		TEN TRIALS	
Statistic	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)	NNZE (%)	NNZB (%)
Minimum	5.62	1.93	6.21	0.0015	1.45	0.54	1082.2	0.02
Average	45.68	12.83	19.65	3.62	26.35	13.64	1195.7	15.03
Maximum	148.09	27.41	53.52	19.41	89.28	50.65	1287.3	62.65
Standard deviation	31.75	8.30	14.47	5.53	23.13	12.96	39.55	13.95

2.10.3 Prediction results from the miscellaneous group of matrices

Fig. 2.18 shows prediction errors for the NNZB and the NNZE models for block size 6 for the extremely sparse matrix group listed in Table 2.3. The key characteristic of this matrix group is the extreme sparseness and larger orders of its matrices. As summarized in Table 2.14 and visualized in Fig. 2.18, this group benefits from using the NNZB model.

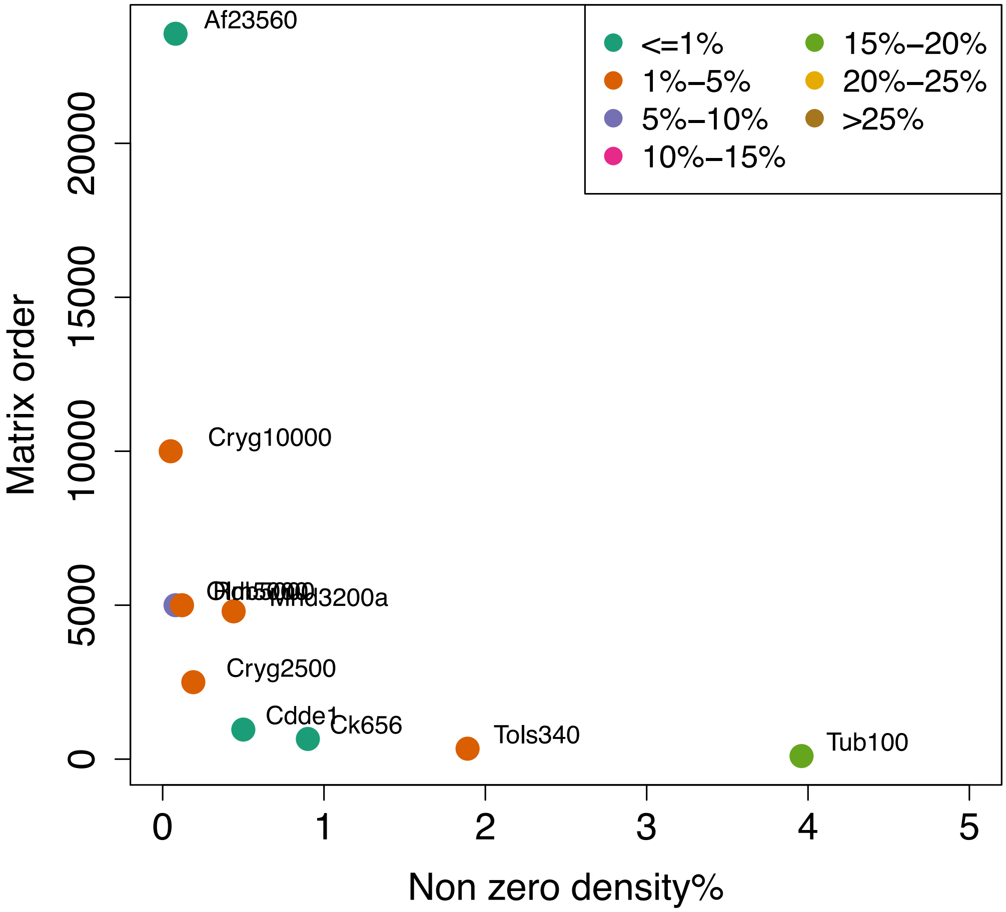


Fig. 2.17. NNZB model prediction error distribution with respect to matrix specifications for Bai matrices on Knight's Landing.

Similar to the study on Sandy Bridge, *dblp* and *cantilever* show preference for the NNZE model. Both the NNZB and the NNZE model have a hard time predicting *inverse* with prediction errors of 50.65% and 89.27%, respectively. Across all matrices, the NNZB model averages 13.64% and the NNZE model averages 26.35%. Table 2.15 lists additional statistics that provide insight into the distribution of prediction errors across the matrix group for block size 6. Fig. 2.19 visualizes distribution of errors based on matrix size and nonzero density percentage.

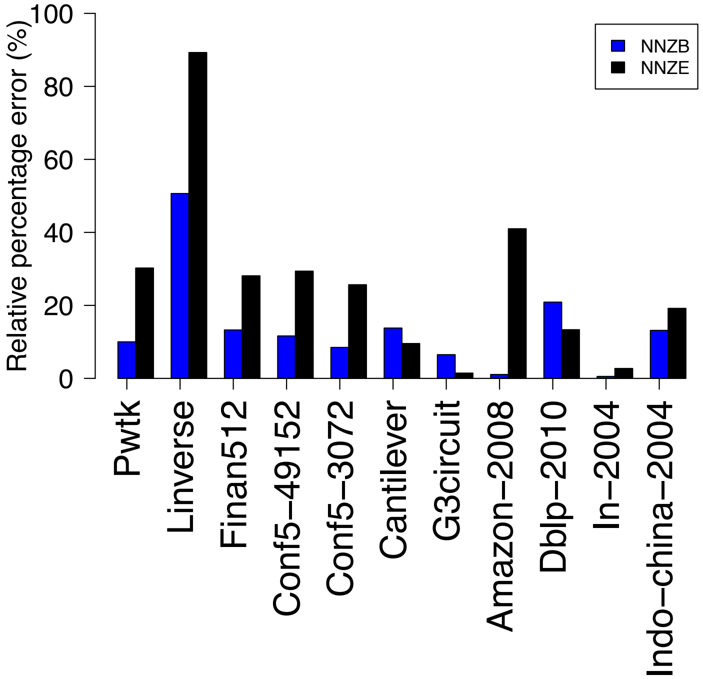


Fig. 2.18. Comparing predictions from NNZE and NNZB models fo matrices from mixed group on Knight's Landing.

2.10.4 Cross-validation-based performance prediction on KNL across all matrix groups

Similar to the study on Sandy Bridge, we investigate the accuracy of both the NNZE and NNZB models using a more rigorous testing and training process. Instead of building and testing prediction models for each matrix group, we pool matrices from the three matrix groups into a single group. Over ten trials, we sample two-thirds of the matrices to serve as the training set, and remaining one-third as the test set. This process is described in detail in Section 2.7.4.

Fig. 2.20 shows errors from the NNZE and NNZB models averaged across the ten training-testing trials for block size 2. Because errors from the NNZE model are orders of magnitude larger than those from NNZB, we use two different Y-axes that span different ranges. The general trend is that the NNZE model fails drastically when faced with diverse training data from different matrix groups.

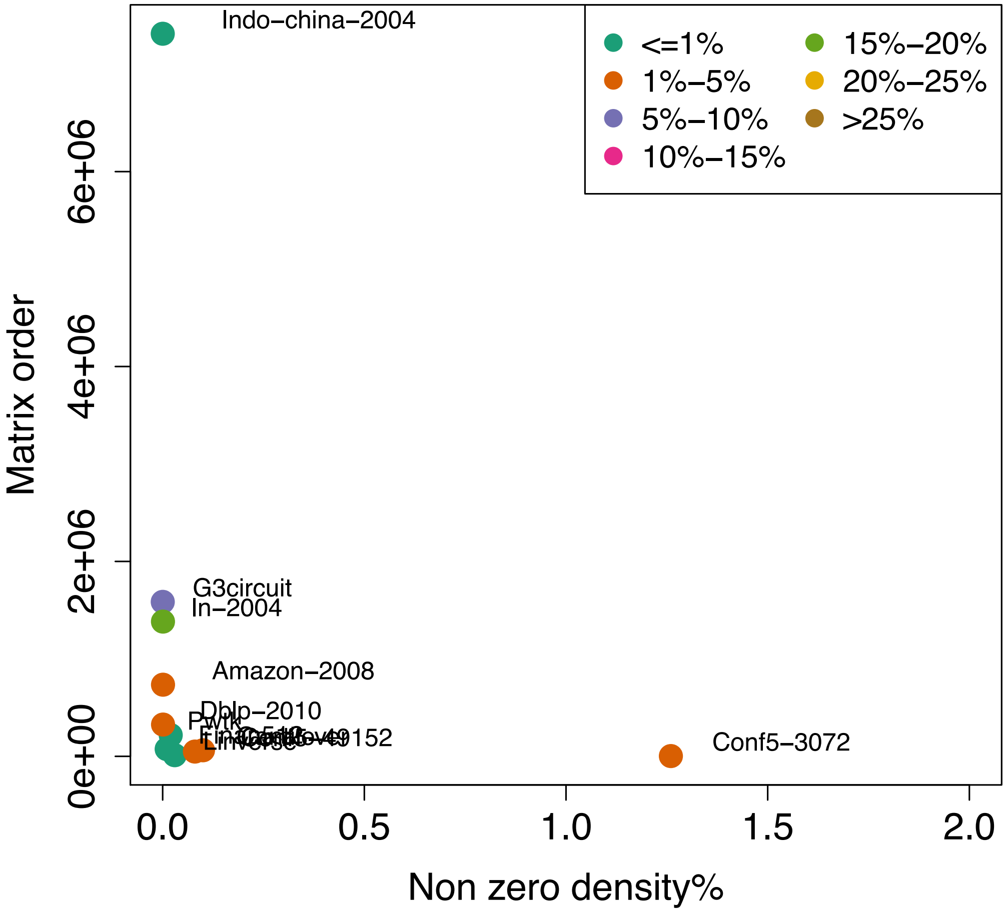


Fig. 2.19. NNZB model prediction error distribution with respect to matrix specifications for mixed matrix group on Knight's Landing.

2.10.5 STOMP's block size selection on KNL

In this section, we evaluate the quality of STOMP's block size selection algorithm on SpMV operations on KNL. Similar to the study on SNB, we use *speedup over default* and *speedup left on the table* as metrics for evaluating SpMV speedup from using STOMP's block selection process. Tables 2.16, 2.17, and 2.18 show speedup evaluations using these metrics for matrix groups Boeing-Chen-FIDAP, Bai, and the mixed group of matrices, respectively.

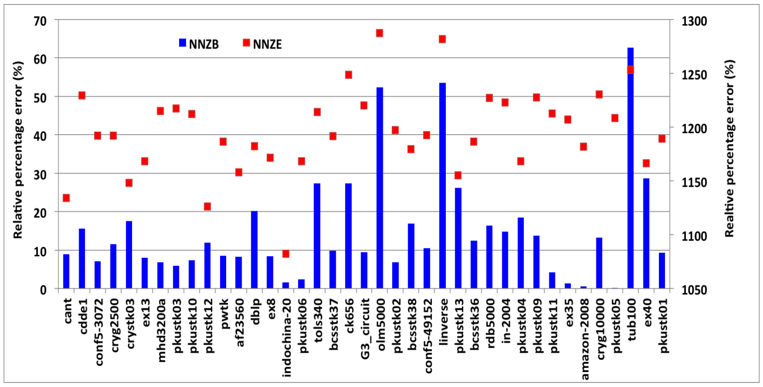


Fig. 2.20. Comparing predictions from NNZE and NNZB models across ten trials on KNL.

In the Boeing-Chen-FIDAP group (Table 2.16), *speedup over default* produced by STOMP lies between a minimum of 43.40% (ex40) and a maximum of 76.71% (pkustk03). Average *speedup over default* in this group is 64.64%. While a majority of the matrices in this group achieve optimal SpMV performance from STOMP's block size selection, there are a few matrices that leave a small percentage of speedup on the table—*bcsstk36* (4.18%), *bcsstk37* (2.10%), *crystk03* (0.08%), *ex8* (2.78%), *ex13* (4.83%), *ex40* (5.92%), and *pkustk01* (5.66%). Average *speedup left on the table* is 1.28%.

In the Bai group (Table 2.17), *speedup over default* produced by STOMP lies between a minimum of 54.76% (cdde1) and a maximum of 73.34% (olm5000). Average speedup over default in this group is 65.82%. With the exception of *cdde1* and *mhd3200a*, all matrices achieve optimal performance using STOMP's block size selection. Average *speedup left on the table* is 0.56%.

In the mixed matrix group (Table 2.18), *speedup over default* produced by STOMP, lies between a minimum of 41.84% (amazon-2008) and a maximum of 75.02% (linverse). Average speedup over default in this group is 57.53%. With the exception of *dblp* and *in-2004*, all matrices achieve optimal performance using STOMP's block size selection. Average *speedup left on the table* is 0.32%. Table 2.18 lists speedup details on all matrices in this group. Table 2.19 summarizes average speedup across all matrix groups on KNL.

Table 2.16. STOMP SpMV speedup on Boeing-Chen-FIDAP on Knight's Landing.

Matrices	Speedup left on the table(%)	Speedup over default(%)
bcsstk36	4.18	61.83
bcsstk37	2.10	61.98
bcsstk38	0	56.84
crystk03	0.08	63.82
ex8	2.78	57.93
ex13	4.83	56.41
ex35	0	55.36
ex40	5.92	43.40
pkustk01	5.66	62.52
pkustk02	0	74.98
pkustk03	0	76.71
pkustk04	0	64.08
pkustk05	0	75.56
pkustk06	0	75.48
pkustk07	0	65.60
pkustk09	0	76.19
pkustk10	0	75.38
pkustk11	0	75.42
pkustk12	0	56.56
pkustk13	0	56.69

Table 2.17. STOMP SpMV speedup on Bai matrices on Knight's Landing

Matrices	Speedup left on the table(%)	Speedup over default
af23560	0	64.13
cdde1	5.58	54.76
ck656	0	64.84
cryg10000	0	64.43
cryg2500	0	64.84
mhd3200a	0.89	65.07
olm5000	0	73.34
rdb5000	0	65.51
tols340	0	71.88
tub100	0	69.43

2.11 Conclusion

We presented the STOMP statistical approach for optimizing and modeling performance of blocked SpMV for a set of diverse sparse matrices from several different scientific and commercial domains. Using statistical models trained on data from matrices previously used in SpMV operations, we predicted run time of a new matrix with mean accuracy of 93.52% across matrices on Sandy Bridge and 91.91% on Knight's Landing. STOMP's block

Table 2.18. STOMP SpMV speedup on mixed group of matrices on Knight's Landing

Matrices	Speedup left on the table(%)	Speedup over default(%)
pwtk	0	64.74
linverse	0	75.02
finan512	0	39.71
conf5-49152	0	62.96
conf5-3072	0	61.94
cantilever	0	57.03
G3_circuit	0	58.21
amazon-2008	0	41.84
dblp	1.50	54.09
indochina-2004	0	56.87
in-2004	1.98	60.39

Table 2.19. Evaluating STOMP's block size predictions on Knight's Landing

Matrix group	Speedup left on the table(%)	Speedup over default(%)
Bai	0.56	65.82
Boeing-Chen-FIDAP	1.28	64.64
Mixed set	0.32	57.53
Average across all groups	0.72	62.66

selection technique produced a performance benefit of as high as 75% on Sandy Bridge and 76% on Knight's Landing. Our techniques produce an average performance improvement of 55.56% over default unblocked SpMV performance across all our matrices on Sandy Bridge and 62.66% on Knight's Landing. We compared the quality of STOMP's block selection process on Sandy Bridge with SPARSITY, a framework that defines heuristics for block-size selection. On the same set of matrices, STOMP yields a 50.96% speedup while SPARSITY yields a 31.62% speedup over the same default.

CHAPTER 3

STAMPP: STATISTICAL TECHNIQUES FOR ANALYZING METRICS AND PREDICTING PERFORMANCE OF WORKLOADS

3.1 Abstract

The ability to estimate application performance on new infrastructures is important for several hardware and software efforts in the HPC community. Estimating performance benefits through intensive hardware monitoring is expensive and often times infeasible. Hardware monitoring costs increase as measurement dimensions grow to include diverse hardware, performance counters, and workloads. STAMPP presents a set of high-level techniques to automate analysis of hardware metrics and workload space, and to predict performance of applications before they are executed on a target system. Using a suite of performance models that abstract performance trends, STAMPP achieves high performance prediction accuracy on individual benchmarks for performance projections across generations of recent Intel CPUs.

3.2 Introduction

Projecting performance of today's applications on future systems is a common goal in the hardware and software community. Hardware designers rely on performance projections to make design decisions for future hardware by analyzing performance profiles of current applications. Application developers leverage performance projections to understand application performance on a new system, evaluate performance bottlenecks, and determine opportunities to tune code. Performance projections can help supercomputing centers to accurately map computational needs of applications to available resources. Projecting performance of a set of applications on a new system prototype is a typical evaluation performed during design and delivery of new systems. At these early stages, access to the new target system is limited to simulations or to a small set of prototype nodes. Thus, it becomes important to make accurate performance projections using techniques that use the limited available data obtained from the target system. Limited access to the target system makes it important to be judicious while designing performance monitoring experiments.

Designing experiment space to predict application performance is challenging due to the following reasons:

1. *Impact of workload input data*: Performance of data-driven applications varies depending on the size and type of input data. Input data influences both memory access patterns across the memory hierarchy as well as layout of the data in the memory. Performance worsens and varies with increased irregularity of memory access and reduced data reuse. A jarring example of this effect is the stark difference in performance expectation of matrix multiplication on dense matrices versus sparse matrices.

2. *Effects of system configurations*: Benign features such as hyperthreading and hardware prefetching, devised to improve system utilization, can either improve or degrade actual application performance depending on individual code structure of the application and its memory use. For instance, Lee et al. [1] show that hardware prefetching does not induce uniform performance benefits in all applications. A study by Zhao et al. [3] that analyzes the effects of hyperthreading on a set of workloads shows that hyperthreading does not always guarantee a performance benefit.

3. *Phases in workload execution*: Applications seldom exhibit a steady state of system resource utilization during their lifetimes. Depending on whether an application is in a data setup phase (low CPU utilization, high cache activity) or an intense compute phase (high CPU utilization), performance can vary. Sherwood et al. [4] examine and show different phases that exist during workload execution.

4. *Monitoring hardware is expensive*: While it is important to understand hardware behavior across the platform—core, socket, cache, memory—it is important to bear in

mind cost of monitoring multiple metrics and the overhead involved. Excessive benchmarking and metric collection are expensive processes that require long simulations or infeasible amounts of time on a prototype system. More importantly, depending on the processor, only certain sets of compatible performance counters can be monitored simultaneously. This limitation further increases hardware monitoring time.

STAMPP presents a general methodology to analyze hardware metrics and workloads to project performance across different CPU architectures. STAMPP uses hardware responses collected on a system during workload execution while staying agnostic to the workload's instruction mix. By using quantitative statistical analyses, STAMPP reduces metric space by identifying important metrics and eliminating redundant ones. STAMPP analyzes and characterizes workloads across three benchmark suites. Performance models in STAMPP leverage hardware metric data on an older *base system* to make projections of a new, unseen workload on a new *target system*.

In demonstrating this performance modeling methodology, STAMPP uses data from the SPEC CPU 2006 [48], SPEC OMP 2012 [49], and SPEC MPI 2007 [50] benchmark suites across three generations of Intel CPUs—Sandy Bridge (SNB), Haswell (HSW), and Broadwell (BDW) in serial and parallel configurations. Table 3.1 describes the SPEC CPU 2006 dataset [51]. Table 3.2 describes the SPEC OMP 2012 dataset [52]. Table 3.3 describes the SPEC MPI 2007 dataset [53]. STAMPP makes the following contributions:

1. Predicts performance (speedup) of a new workload on a target system with high accuracy using a novel suite of performance models.

2. Demonstrates the use of statistical techniques to automatically identify important hardware metrics and reduce metric space by up to 53% to make successful performance projections.

3. Presents a heuristic to identify workloads that elicit similar hardware response on a system. Similar workloads are used as proxies of the test workload to predict performance.

Table 3.1. SPEC CPU 2006 benchmarks

Benchmark	Language	Application
perlbench	C	Perl Programming language
bzip2	C	Compression
gcc	C	C Compiler
mcf	C	Combinatorial optimization
gobmk	C	Artificial intelligence for Go
hmmer	C	Search gene sequence
sjeng	C	Chess
libquantum	C	Quantum computing
h264ref	C	Video compression
omnetapp	C++	Discrete event simulation
astar	C++	Path-finding algorithm
xalancbmk	C++	XML processing
bwaves	Fortran	Fluid dynamics
cactusADM	C/Fortran	Physics \General relativity
calculix	CFortran	Structural mechanics
dealII	C++	Finite element analysis
gamess	C/Fortran	Quantum chemistry
GemsFDTD	C/Fortran	Computational electromagnetics
gromacs	C/Fortran	Biochemistry \molecular dynamics
lbm	C	Fluid Dynamics
leslie3d	Fortran	Fluid Dynamics
milc	C	Physics/Quantum Chromodynamics
namd	C++	Biology Molecular Dynamics
povray	C++	Image ray tracing
soplex	C++	Linear Programming, Optimization
sphinx3	C	Speech recognition
tonto	Fortran	Quantum Chemistry
zeusmp	Fortran	Computational Fluid Dynamics

Table 3.2. SPEC OMP 2012 benchmarks

Benchmark	Language	Application
md	Fortran	Molecular Dynamics
bwaves	Fortran	Computational fluid dynamics
nab	C	Molecular modeling
Bt331	Fortran	Computational fluid dynamics
botsalign	C	Protein alignment
botsspar	C	Sparse LU
ilbdc	Fortran	Lattice Boltzmann
Fma3d	Fortran	Mechanical response simulation
swim	Fortran	Weather prediction
imagick	C	Image processing
Mgrid331	Fortran	Computational fluid dynamics
Applu331	Fortran	Computational fluid dynamics
smithwa	C	Optimal pattern matching
kdtree	C++	Sorting and searching

3.3 Related Work

Several techniques have been used to investigate performance projection in different ways over the years: using data from simulator runs, leveraging collections of benchmarks to approximate performance of a new application, and replicating application hot-spots through smaller proxy apps.

3.3.1 Performance projections through simulations

Simulations can be used to analyze and predict performance on a target system. Specific frameworks such as BigSim [54] leverage properties from the Charm++ runtime and require dedicated executions of a target application through a tiered simulation framework. The use of the framework is fitted to applications written in Charm++. It is unclear if this method works across different programming languages. Zheng et al. [55] use their BigSim simulation framework to predict performance of Charm++ applications. Among many

Table 3.3. SPEC MPI 2007 benchmarks

Benchmark	Language	Application
milc	C	Quantum chromodynamics
leslie3d	Fortran	Computational fluid dynamics
gemsFDTD	Fortran	Electromagnetics
Fds4	C\Fortran	Computational fluid dynamics
Pop2	C\Fortran	Ocean modeling
tachyon	C	Graphics: parallel ray tracing
lammps	C++	Molecular dynamics
Wrf2	C\Fortran	Weather prediction
Tera_tf	Fortran	3D Eulerian hydrodynamics
socorro	C\Fortran	Molecular dynamics
Zeusmp2	C\Fortran	Computational fluid dynamics
zeusmp	Fortran	Computational fluid dynamics
lu	Fortran	Computational fluid dynamics
dmilc	C	Quantum chromodynamics
dleslie	Fortran	Computational fluid dynamics
lGemsFDTD	Fortran	Electromagnetics
L2wrf2	C\Fortran	Weather prediction

inputs, this framework requires the end user to provide speedup estimates. This diminishes the value of a performance prediction framework.

MPI-Sim [56] is a library designed to emulate task and data parallel programs. The simulation specifically focuses on parallel programs and uses communication patterns to analyze performance of the NASA Advanced Supercomputing (NAS) benchmark suite. Riesen [57] presents a hybrid MPI simulator that simulates MPI calls within an application. A network simulator logs MPI message events from the executing application, which is used to project application performance on a new network.

These simulation techniques do not predict performance of an application before its execution. By using SPEC CPU as the set of representative data on the target machine, STAMPP starts out with accurate real-world knowledge of performance on the target machine. Our approach does not require any executions of the test workload on the target machine. More importantly, our techniques are not tied to a particular runtime framework or library and show general applicability across processor architectures.

3.3.2 Performance estimations through benchmark suites

Performance of a new workload can be abstracted using information from previously executed workloads. Similarly, compatible sets of workloads can serve as performance proxies for each other. Work by Phansalkar et al. [58] uses clustering analysis to find subsets of benchmarks from the SPEC CPU 2006 benchmark suite. Their goal is to evaluate benchmark redundancy in the SPEC CPU suite using fine grain routine profiling inside each benchmark to extract details from the instruction mix. Breslow et al. [59] show benefits of interleaving and co-locating pairs of jobs (HPC applications and benchmarks) on a shared set of nodes. The study uses simple heuristics to find pairs of workloads that can be ideally co-located. Pairs of workloads are examined exhaustively to determine if they are symbiotic pairs (performance of neither workload deteriorates), minor interference pairs (performance deteriorates by less than 5%), or non-symbiotic pairs (performance deteriorates by more than 5%). Although this work comprises finding sets of compatible workloads, the techniques do not use statistical analysis to trim their large dataset or predict performance or speedup. Hoste et al. [60] use a mixture of SPEC CPU benchmarks to predict performance of an application. The approach uses a series of normalization and genetic algorithms to match micro-architecture characteristics from an application of interest to CPU SPEC benchmarks across multiple architectures. The work uses multiple correlation analyses and a genetic algorithm to find proxies within the suite. The work does not use hardware metrics that are independent of instruction mix and does not include workloads other than the SPEC CPU benchmark suite. Sharkawi et al. [61] show effectiveness of using surrogate benchmarks from the SPEC CPU suite to project performance of HPC applications across architectures. Surrogate benchmarks are those that have lowest weighted error between their metrics and those of the HPC application on a machine. This work does not use statistical analysis, regression, or clustering to identify similar workloads and prune metric space. Jaleel [62] presents extensive characterizations of SPEC CPU 2000 and SPEC CPU 2006 benchmarks using instrumentation-driven simulations. The work investigates the suites memory access patterns and usage. The study shows the amount of overhead involved during detailed performance monitoring.

An important step while selecting sets of benchmarks to model a new workload, is assessing compatibility between the benchmark set and the test workload. It is also important to find a sufficient number of compatible benchmarks to model performance of a new workload, such the set abstracts critical performance traits of the test workload. STAMPP, through its workload characterization heuristic, finds sets of similar workloads that best describe performance of a new workload on a base system using statistical techniques.

3.3.3 Performance projections through application skeletons

Another popular approach for predicting performance is the use of skeleton codes to simulate application performance. Skeletons are simplified versions of an application that are either handwritten or synthetically produced to replicate performance characteristics of the original application. Skeleton codes, also called proxy applications or *proxy apps* for short, have received attention in recent years via the CORAL [63] and APEX [64] projects. Notable proxy apps such as MMCK and XSBench [65] are designed to simulate performance of important kernels within the OpenMC [66] code. Although proxy apps provide simpler implementations of core sections of a large application, they do not always capture all interactions between real code and hardware that occur during the execution of the application. Because they are simplified representations, they also do not always adequately capture data flow patterns within the original application. Crafting proxy apps can be tedious and challenging when the application, and corresponding libraries and compilers, evolve. A few approaches have investigated generating application skeletons automatically by extracting MPI communication patterns in [67] and [68].

An important problem in performance projection studies is *how to predict performance of a new workload on a new target system prior to execution with minimum hardware monitoring overhead.* STAMPP predicts performance of a new workload on a target machine using statistical prediction models. These models are trained on performance data from SPEC CPU benchmark executions on the base and target machine. STAMPP does not require the test workload to be executed on the *target machine*. STAMPP's performance prediction models operate on sibling benchmarks and a reduced set of hardware metrics.

3.4 Experiment Environment

3.4.1 Machine configurations

Our experiments are based on hardware metrics collected on recent generations of Intel systems: Sandy Bridge (SNB), Haswell (HSW), and Broadwell (BDW). All machines are two-CPU platforms varying in the number of cores per CPU and in the sizes of their last level cache (L3). Our SNB, IVB, HSW, and BDW systems comprise respectively 8, 14, and 22 cores and 20, 35, and 55 megabytes of unified L3 cache per processor. All systems had cores with separate 32KB L1I and L1D caches and a unified 256KB L2 cache. Table 3.4 summarizes the specifications of the systems that we used to run benchmarks and collect hardware metrics. The systems had 128 GB of memory but all the benchmarks used much lower capacity requirements.

3.4.2 Hardware metrics

To characterize workload behavior in this study, we track several cache statistics: miss

Table 3.4. Machine specifications

	Sandy Bridge	Haswell	Broadwell
Sockets	2	2	2
Cores	8	14	22
Total cores	24	28	44
L1 (I and D)	32KB	32KB	32KB
L2	256KB	256KB	256KB
L3	20MB	35MB	55MB

rate per instruction and per request for all three levels of cache as well as the bandwidth associated with data transfer between cache levels. For the L3 cache we additionally monitor hardware prefetch activity and average latency of the misses. We also track D-TLB misses, fraction of the cycles when cores are stalled, and a few parameters that characterize DRAM activity: transaction count, utilization, total aggregate bandwidth, burst bandwidth (average of top 95% samples), and the percentage of transactions that cross UMA domains. Table 3.5 lists the metrics we use and explains their meaning.

Monitoring cache and DRAM statistics have proven effective in utilizing hardware response to optimize codes across CPUs and general-purpose computing on graphics processing units (GPGPUs). For instance, Kim et al. in [69] use hardware counter data to construct models to optimize the fast multipole method on a GPU. Work in [70] classifies cache misses in workloads to construct cache data profile paths through workloads. By measuring cache usage and identifying cache misses, their work eliminates bottlenecks and provides a 18%-57% improvement in throughput. Dimitrov et al. [71] monitor and evaluate cache and DRAM activity across Hadoop workloads for different fundamental functions such as join, union, and sort operations to gauge benefit of prefetching and caching. Diamond et al. [72] use cache and DRAM statistics to identify multicore performance bottlenecks and optimize the high order method modeling environment (HOMME) [73] benchmark. Our measurements include a variety of micro-architectural measurements in the form of DRAM and cache metrics listed in Table 3.5.

A simple way to get preliminary insight into distributions of metric values is to create a boxplot, which is a concise visual representation of the spread of the data. Fig. 3.1 shows a boxplot chart for average values of metrics listed in Table 3.5 collected for SPEC CPU Speed runs on a Sandy Bridge system. Each hardware metric x is scaled as follows:

$$\frac{(x-mean(x))}{standard_{d}eviation(x)}$$

This scaling allows metrics with different magnitudes of values to be compared on the same plot. The boxplot indicates the median of the dataset, the minimum and maximum values (whiskers), percentage of values that form the top 25% (upper quartile) and the

Table 3.5. Hardware Metrics Collected on Sandy Bridge, Haswell, and Broadwell

Metric	Interpretation
StallCycPct	Mean percentage of all core cycles stalled
L1MissInst	Mean number of L1 D-cache demand load misses per 100 retired instructions
L1MissRqst	Mean percentage of all demand load requests to L1 D-cache that missed
L2toL1BW	Mean bandwidth for each HW thread from its L2 cache into its L1 D-cache
L2MissInst	Mean L2 demand load misses per 100 retired instructions. Does not include misses due to HW prefetches
L2MissRqst	Mean percentage of all demand data load requests to L2 cache that missed
L3toL2BW	Mean aggregate bandwidth from L3 to L2
L3MissInst	Mean L3 data demand load misses per 100 retired instructions
L3MissRqst	Mean percentage of all demand data load requests to L3 that missed
L3MissPFPct	Mean percentage of all L3 load misses that were due to HW prefetch
L3MissLat	Mean latency (ns) of load requests that missed L3
TLBMissInst	Total DTLB misses per 100 retired instructions
DRAMTrInst	Mean number of DRAM transactions (reads + writes) per 100 retired instructions
DRAMTrCyc	Mean number of DRAM transactions (reads + writes) per 100 core cycles.
DRAMUtil	Mean percentage of total available DRAM cycles that were used to move blocks into or out of DRAM
DRAMBW	Mean aggregate bandwidth of all DRAM transactions.
DRAMBrBW	Burst aggregate bandwidth of all DRAM transaction
UMAPct	Mean percentage of all Home Agent read transactions involving blocks in the same UMA domain as the requesting HW thread
Figure of merit	Application-specific unit of performance. Examples include, run time, bandwidth
FOMSpeedup	Ratio of figures of merit (FOM) between two machines

bottom 25% (lower quartile). Values that are lesser or greater than 3/2 times the lower or upper quartile are designated as low and high outliers.

There are a few consistent outliers in different metric classes. Common outliers include: *mcf*, *leslie3d*, *bwaves*, *milc*, *libquantum*, and *gromacs*. With the exception of *mcf*, the outlier benchmarks are fluid dynamics or molecular dynamic codes with very intensive memory accesses. The resulting high memory traffic from these applications is manifested in high DRAM metric values in the boxplot. *Mcf* is a scheduling code that requires at least 1700MB of memory, which is greater than SNBs cache sizes. This high memory requirement could account for the very high number of cache misses.

Ideally we would like to collect as much metric data as possible during an application's run time. However, metric collection does not come without its own overhead. This is discussed in detail in the proceeding section.

3.5 Metric Analysis using Statistical Techniques

Evaluating metric space is a pre-cursor to efforts that leverage cues from hardware to understand workload performance trends. While collecting all possible hardware metrics to get a 360-degree view into performance trends seems promising, there are several disadvantages to overmeasuring hardware space:

1. *Hardware monitoring is not free*: Metric collection is not achieved without a heavy price. Most CPU architectures do not support collection of more than a few events simultaneously. For instance, Sandy Bridge has only three fixed and four programmable registers on each of its hardware threads. Most metrics are derived from more than two events (such as misses per N instructions per N cycles), which depending on event compatibility, cannot be measured together. Co-processors from the Xeon Phi line, namely, Knight's Corner and Knight's Landing, have fewer programmable counters (two each) than their Xeon counterparts, making event collection even more expensive. The combination of incompatible events and few counters increase the number of workload executions required for metric collection.

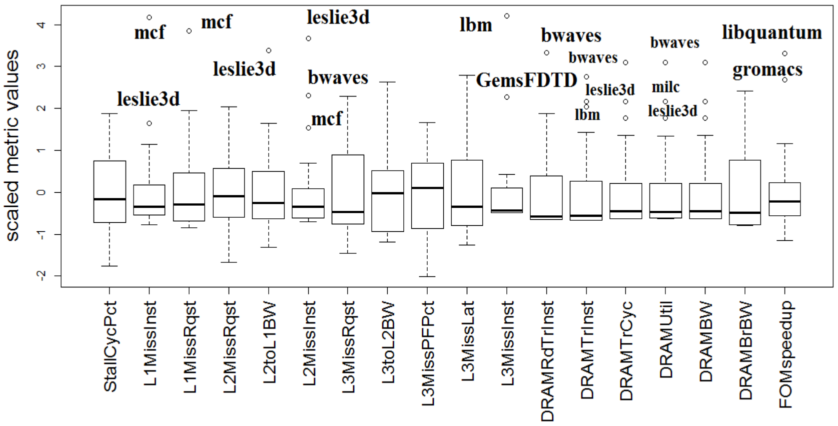


Fig. 3.1. Distribution of serial SPEC CPU metrics on Sandy Bridge.

2. *Metrics can be derived from others*: The same phenomenon can be measured multiple times at different levels of granularity at different stages of the pipeline. For instance, information on memory requests—frequency, number of cache misses across the cache hierarchy—can be gathered at different points of the platform, namely, cache, CHAs (cache homing agents), sockets, and the memory controller. However, measuring latency at only a single CHA and cache misses at only one cache level, will give an indication of miss rate or miss penalty across the cache hierarchy.

3. *Putting all your metrics in the proverbial one basket*: To keep pace with microarchitectural changes across processor generations, events and the corresponding hardware metrics have to evolve. An example being the evolution of events to count FLOPs and related metrics over multiple generations of Intel Xeon processors. In other situations, key hardware events might not even exist on a particular platform or might not provide desired information required to formulate a given metric. For instance, the Intel Xeon Phi coprocessor does not have appropriate events to calculate L2 hit rate. As a result, "estimated L2 latency impact" metric is used as a proxy for measuring L2 hit rate. Analyzing correlations between metrics allows performance analysis and projection frameworks to be flexible so that when a given metric is unavailable on a platform, the framework can replace the missing metric with a relevant metric.

4. *Cost of oversampling*: While not only time intensive, excessive monitoring at fine granularity produces large quantities of data that do not provide immediate insight into hardware behavior. This problem is exacerbated in data centers. To address this problem, metric analysis can be used to identify correlations and filter metrics that do not bring additional insight. Metric analysis can determine spaces of hardware that are oversampled and those that need additional monitoring.

The next subsections describe statistical techniques used in STAMPP to scan metric space to identify pertinent metrics.

3.5.1 Maximal information coefficient

In a large data set with several potentially correlated variables, one of the first steps of data exploration is to look for relationships between different pairs of variables. Relationships can be defined as simply as a correlation coefficient or a little more complex to take the form of functional relations (periodic, sinusoidal, polynomial, etc.). The disadvantage of using traditional correlation coefficients is that they are biased towards linear relationships between data points. This means that in a dataset, a stronger polynomial relationship between data points will be scored as having a lower correlation coefficient than a noisier linear relationship [74]. To address this problem, Reshef et al. [74] introduced the Maximal Information Coefficient (MIC) to measure the strength of a relationship between two variables in a way that is both general and equitable. Unlike the traditional Spearman [75] or Pearson correlation coefficients [76], MIC stays unbiased towards the type of the relationship and is equitable in scoring similarly noisy patterns regardless of relationship type.

We considered 17 metrics collected for the SPEC CPU benchmarks on a Sandy Bridge system in our MIC calculations. While all the metrics individually capture a unique response of the machine during benchmark execution, not all of them are equally important and unique. In fact, many of the metrics capture the same phenomenon and introduce redundancy. We show MIC values between pairs of metrics for serial SPEC CPU in Fig. 3.2.

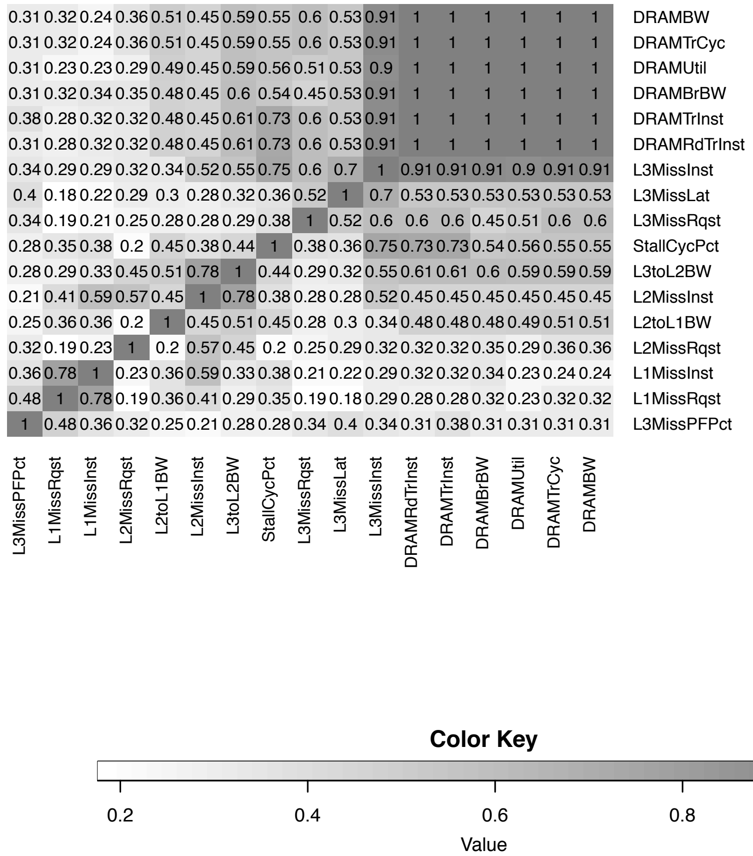
Values closer to one indicate a stronger relationship between the metrics. The large dark square on the upper right of the figure shows a strong relationship between all the DRAM metrics in our set. Other significant relationships include the following pairs of metrics—(StallCycPct, L3MissInst), (StallCycPct, DRAMRdTrInst), (StallCycPct, DRAMTrInst), and (L3toL2BW, L2MissInst).

3.5.2 Variable inflation factor

Multicollinearity exists when pairs or sets of predictor variables or features are highly correlated and can be expressed as a linear combination of each other. Ignoring multicollinearity can produce a model that overfits to the training data and will have limited predictive ability for new test data. In smaller dimensional data, multicollinearity can be detected visually by observing plots of variables plotted against each other. However, with a large number of variables this approach has limitations. Fig. 3.3 shows variable inflation factor (VIF) values for SNB hardware metrics derived from executing serial SPEC CPU. Using VIF, we successively picked the metric with the highest VIF value, then removed it and applied the VIF computation to the remaining set. Metrics that lie below 5, a threshold commonly used by the community, are considered as sufficiently unique and non-redundant. The VIF computation automates detecting multicollinearity. VIF is explained in detail in [77] and [78]. We can see that the set of metrics reduced using this method has only one DRAM-related metric indicating that the other DRAM metrics do not carry sufficient additional information. By using VIF we detect multicollinearity across sets of metrics as opposed to only pair-wise assessments in other correlation techniques.

3.5.3 Hierarchical clustering for metrics

Hierarchical clustering [79] algorithms use a tree structure, called a dendrogram, to illustrate sets of different clusters in the data set. The root node of a tree represents a single cluster containing all elements of the dataset. The leaf nodes are individual elements. Intermediate internal cluster nodes indicate clusters formed by merging leaf nodes bottom-up. Each horizontal level in the cluster tree is associated with a particular distance measure



	L3MissPFPct	L1MissRqst	L1MissInst	L2MissRqst	L2toL1BW	L2MissInst	L3toL2BW	StallCycPct	L3MissRqst	L3MissLat	L3MissInst	DRAMRdTrInst	DRAMTrInst	DRAMBrBW	DRAMUtil	DRAMTrCyc	DRAMBW
DRAMBW	0.31	0.32	0.24	0.36	0.51	0.45	0.59	0.55	0.6	0.53	0.91	1	1	1	1	1	1
DRAMTrCyc	0.31	0.32	0.24	0.36	0.51	0.45	0.59	0.55	0.6	0.53	0.91	1	1	1	1	1	1
DRAMUtil	0.31	0.23	0.23	0.29	0.49	0.45	0.59	0.56	0.51	0.53	0.9	1	1	1	1	1	1
DRAMBrBW	0.31	0.32	0.34	0.35	0.48	0.45	0.6	0.54	0.45	0.53	0.91	1	1	1	1	1	1
DRAMTrInst	0.38	0.28	0.32	0.32	0.48	0.45	0.61	0.73	0.6	0.53	0.91	1	1	1	1	1	1
DRAMRdTrInst	0.31	0.28	0.32	0.32	0.48	0.45	0.61	0.73	0.6	0.53	0.91	1	1	1	1	1	1
L3MissInst	0.34	0.29	0.29	0.32	0.34	0.52	0.55	0.75	0.6	0.7	1	0.91	0.91	0.91	0.9	0.91	0.91
L3MissLat	0.4	0.18	0.22	0.29	0.3	0.28	0.32	0.36	0.52	1	0.7	0.53	0.53	0.53	0.53	0.53	0.53
L3MissRqst	0.34	0.19	0.21	0.25	0.28	0.28	0.29	0.38	1	0.52	0.6	0.6	0.6	0.45	0.51	0.6	0.6
StallCycPct	0.28	0.35	0.38	0.2	0.45	0.38	0.44	1	0.38	0.36	0.75	0.73	0.73	0.54	0.56	0.55	0.55
L3toL2BW	0.28	0.29	0.33	0.45	0.51	0.78	1	0.44	0.29	0.32	0.55	0.61	0.61	0.6	0.59	0.59	0.59
L2MissInst	0.21	0.41	0.59	0.57	0.45	1	0.78	0.38	0.28	0.28	0.52	0.45	0.45	0.45	0.45	0.45	0.45
L2toL1BW	0.25	0.36	0.36	0.2	1	0.45	0.51	0.45	0.28	0.3	0.34	0.48	0.48	0.48	0.49	0.51	0.51
L2MissRqst	0.32	0.19	0.23	1	0.2	0.57	0.45	0.2	0.25	0.29	0.32	0.32	0.32	0.35	0.29	0.36	0.36
L1MissInst	0.36	0.78	1	0.23	0.36	0.59	0.33	0.38	0.21	0.22	0.29	0.32	0.32	0.34	0.23	0.24	0.24
L1MissRqst	0.48	1	0.78	0.19	0.36	0.41	0.29	0.35	0.19	0.18	0.29	0.28	0.28	0.32	0.23	0.32	0.32
L3MissPFPct	1	0.48	0.36	0.32	0.25	0.21	0.28	0.28	0.34	0.4	0.34	0.31	0.38	0.31	0.31	0.31	0.31

Fig. 3.2. Maximal information coefficients for SPEC CPU on Sandy Bridge.

that is used to merge the clusters. Fig. 3.4 shows a dendrogram from hierarchical clustering on metrics from serial execution of SPEC CPU on SNB. From bottom-up, the tree groups all DRAM metrics into a large cluster. Similarly, L3, L1, and L2 metrics are clustered together. Independent metrics such as L3MissLat and L3MissPFPct form singleton clusters, indicating their uniqueness. We use hierarchical clustering only as a means to visualize the metric space and to validate metrics filtered by VIF and MIC.

3.5.4 Summarizing metric analysis: principal and redundant hardware metrics

MIC, VIF, and hierarchical clustering are used to group metrics and filter redundant hardware metrics. MIC groups all DRAM metrics together with a high mutual MIC value

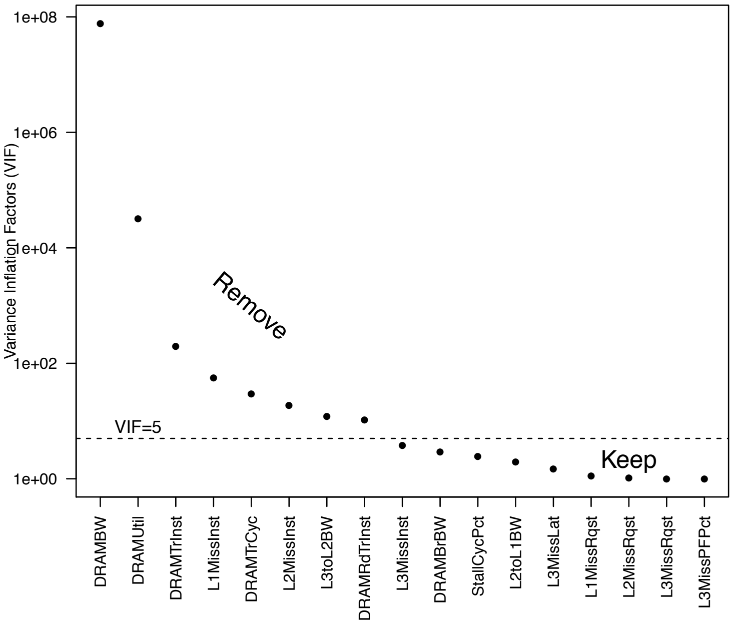


Fig. 3.3. Variable inflation factor for SPEC CPU on Sandy Bridge.

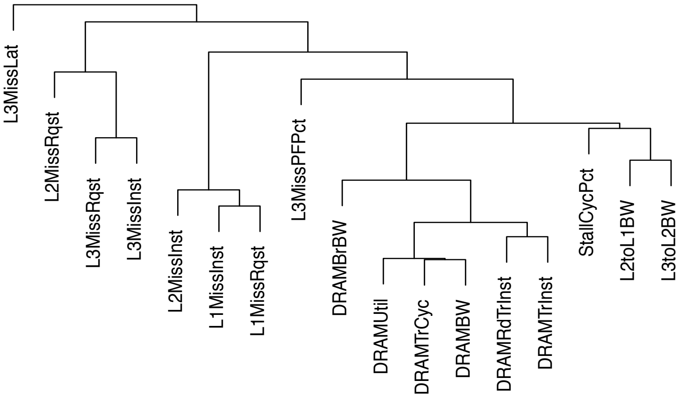


Fig. 3.4. Hierarchical clustering of serial SPEC CPU on Sandy Bridge.

(see Fig. 3.2). VIF also eliminates all DRAM metrics, save one, to produce a reduced set of non-collinear metrics shown in Fig. 3.3. Hierarchical clustering groups all DRAM metrics together to indicate that DRAM activity is clearly represented with just one DRAM metric. Since VIF and MIC give quantitative measures of redundancy in the metric space, we use these techniques to select non-redundant subsets of metrics. We validate the efficacy of these filtered metrics derived from using VIF and MIC through our performance prediction experiments in Section 3.7.

Tables 3.6 and 3.7 show non-redundant metrics derived from using VIF and VIF and combined with MIC from SPEC CPU executions on SNB and HSW in serial and parallel.

3.6 Workload Characterization

Executing sets of relevant proxy apps and benchmarks on a system to monitor hardware is a common process in most performance analysis efforts. However, it is possible that different workloads might elicit similar hardware response from the system. Although different workloads have distinct operations, they might share code patterns such as calls to the same library functions, the same nested loop patterns, etc. While analyzing benchmark space, we seek to identify workloads that invoke similar responses from cache and DRAM and can therefore serve as proxies for each other.

Table 3.6. Important metrics from VIF

machine	cores	StallCycPct	L1MissInst	L1MissRqst	L2MissRqst	L2toL1BW	L2MissInst	L3MissRqst	L3toL2BW	L3MissPFPct	L3MissLat	L3MissInst	DRAMRdTrInst	DRAMTrInst	DRAMTrCyc	DRAMUtil	DRAMBW	DRAMBrBW	DTLBMissInst	UMApct
SNB	1	✓		✓	✓	✓		✓		✓	✓	✓						✓	-	
SNB	8	✓			✓	✓	✓	✓		✓	✓	✓						✓	-	
HSW	1	✓		✓	✓		✓	✓	✓	✓								✓	✓	
HSW	28			✓	✓	✓	✓	✓		✓		✓						✓	✓	✓

Table 3.7. Important metrics from VIF further reduced by MIC

machine	cores	StallCycPct	L1MissInst	L1MissRqst	L2MissRqst	L2toL1BW	L2MissInst	L3MissRqst	L3toL2BW	L3MissPFPct	L3MissLat	L3MissInst	DRAMRdTrInst	DRAMTrInst	DRAMTrCyc	DRAMUtil	DRAMBW	DRAMBrBW	DTLBMissInst	UMApct
SNB	1	✓		✓	✓	✓		✓		✓	✓	✓						✓	-	
SNB	16	✓			✓	✓	✓	✓		✓	✓							✓	-	
HSW	1	✓		✓	✓		✓		✓	✓								✓	✓	
HSW	28			✓	✓	✓	✓	✓		✓									✓	

3.6.1 Identifying sibling workloads

We examine similarity between workloads by calculating Euclidean distance between each test workload (SPEC MPI or SPEC OMP) and workloads from the SPEC CPU suite. We use principal metrics identified using VIF and MIC to calculate the Euclidean distance matrix between test workload sets. A lower Euclidean distance indicates greater similarity between two workloads. The Euclidean distance calculation for a workload produces a vector of values that shows magnitude of similarity with the SPEC CPU suite. We step through this Euclidean distance vector in increments of one (2, 3, ..., max(distance)). All SPEC CPU workloads that fall within a step range are considered *siblings* of the test workload. We build linear regression models using SPEC CPU benchmarks collected at each Euclidean distance range. The number of siblings identified at each Euclidean distance step influences the quality and strength of the linear regression performance prediction models built at that distance. To mitigate the effects from overfitting and underfitting of SPEC CPU training data to the model, we use the Akaike Information Criteria (AIC) [80] to evaluate the quality of each trained linear regression model at each Euclidean distance. The Euclidean distance range corresponding to the linear regression model with the highest AIC value is the distance at which the search for *sibling* SPEC CPU workloads ends. This ideal distance is called the *AIC distance*. Fig. 3.5 visualizes this process. The SPEC CPU workloads that lie within this distance are termed as *sibling benchmarks* of the test workload.

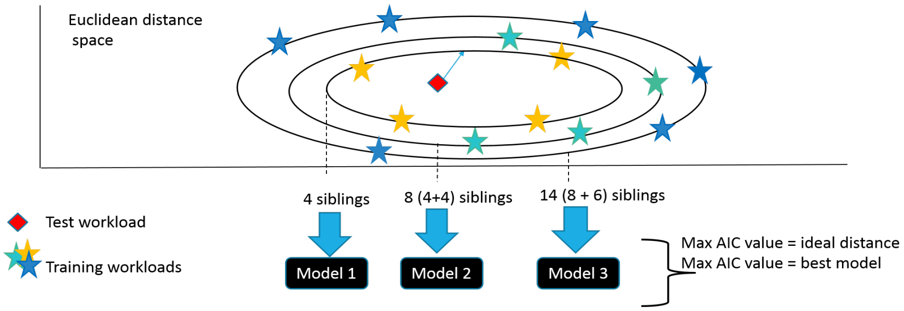


Fig. 3.5. Workload characterization.

3.6.2 Workload similarity between SPEC CPU and SPEC OMP and SPEC MPI

Fig. 3.6 and Fig. 3.7 show heatmaps of Eucilidean distances between SPEC OMP workloads on Haswell in serial and parallel, respectively. Fig. 3.8 and Fig. 3.9 show heatmaps of Eucilidean distances between SPEC MPI workloads on Sandy Bridge in serial and parallel, respectively. Darker cells indicate dissimilarity, while lighter colored ones indicate increased similarity between workloads. In all the heatmaps, SPEC MPI and SPEC OMP workloads are represented along the X axes while SPEC CPU workloads are along the Y axes.

Siblings for serial SPEC OMP

Fig. 3.6 quantifies similarity distances between SPEC CPU and SPEC OMP on Haswell in serial. With the exception of *Swim*, *mgrid331*, *botsspar*, and *applu331* in the SPEC OMP suite, most benchmarks show significant similarity with most SPEC CPU workloads.

Siblings for parallel SPEC OMP

Fig. 3.7 shows similarity distribution between SPEC CPU and SPEC OMP in parallel on Haswell. Unlike the rest of the SPEC OMP suite, *swim*, *mgrid331*, *botsspar*, and *ilbdc* stand out as being quite dissimilar from the set of SPEC CPU benchmarks. In the SPEC CPU set, *mcf*, *povray*, and *omnetpp* from SPEC CPU suite are dissimilar to most SPEC OMP workloads.

Siblings for serial SPEC MPI

Fig. 3.8 shows that most SPEC MPI benchmarks—*socorro*, *lesie3d*, *gemsFDTD*, *zeusmp*, *pop2*, *fds4*, *lu*, and *wrf2*—are uniformly dissimilar to the same set of SPEC CPU workloads—*hmmer*, *sjeng*, *lbm*, *bzip2*, *tonto*, *astar*, *h264ref*.

Siblings for parallel SPEC MPI

As shown in Fig. 3.9, there are varying degrees of similarity between the SPEC MPI and SPEC CPU components. In the SPEC CPU suite, *povray*, *sjeng*, *mcf* are mostly dissimilar to *gap*, *lu*, *socorro*, *FDTD*, *wrf2*, *leslie3d*, and *fds4*.

3.7 STAMPP Prediction Models

Performance prediction is the crux of performance analysis efforts. Prior to system deployment, performance projection is used at different stages to provide performance estimates at system delivery milestones. Building on our metric analysis and workload characterization work, we build a suite of performance models that utilize different amounts of data. We show that pruning metric space and benchmark space produce better performance predictions.

For performance analysis we treat one platform as a *base* system, which is typically a previous generation system, and the other as a *target*, which is a newer platform. Rather than attempting to predict absolute performance we will focus on predicting the speedup, or improvement in figure of merit (FOM) that a workload experiences when it is migrated from the base to the target system. Fig. 3.10 shows speedup of SPEC CPU benchmarks when migrated from SNB to HSW.

We collect SPEC CPU metric values on both base and target system, as well as the speedup values for each benchmark component, and use both the metrics and speedups as training data for the model. For the benchmarks in our test set we collect metric data only on the base machine and then project the speedup to estimate the test workloads' performance on the target system. We will use SPEC MPI and SPEC OMP sets in our analysis both in serial and parallel modes with the parallel mode using all cores on the system (with hyperthreading disabled). Table 3.8 briefly introduces our performance prediction models.

3.8 Statistical Performance Prediction Model Suite

The dataset for training our performance models comprises two dimensions—hardware metrics and SPEC CPU benchmarks. All our performance models are multiple linear regressions that use varying amounts of training data across metrics and benchmarks. Our models take the following form:

$$Y = \alpha_{i=1} X_{i=1} + \beta X_{i=2} + ... + \zeta X_{i=n} + \epsilon$$

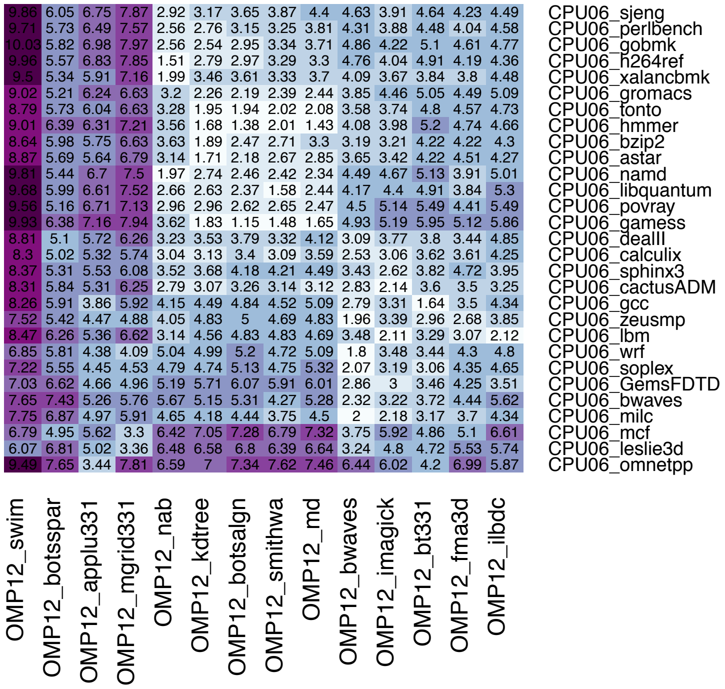
where, Y is the FOM speedup between the base and target machines; X_i are hardware metrics from SPEC CPU executions on the base machine; and α, β, and ζ are regression coefficients for each hardware metric. The next subsections describe STAMPP's performance models in detail.

3.8.1 CM-CB: Complete Metric and Complete Benchmark Model

This model is trained on all hardware metrics obtained from executing SPEC CPU benchmarks on the base machine.

Table 3.8. STAMPP performance model suite

Models	Metrics (M)	Benchmarks (B)	Heuristic for dimension reduction
CM-CB	**C**omplete	**C**omplete	–
RM-CB-VIF	**R**educed	**C**omplete	VIF
RM-CB-VIF-MIC	**R**omplete	**C**omplete	VIF and MIC
Mean baseline	NA	**C**omplete	–
EU-VIF	**R**educed	**R**educed	VIF and Euclidean distance
EU-VIF-MIC	**R**educed	**R**educed	VIF and MIC, and Euclidean distance



	OMP12_swim	OMP12_botsspar	OMP12_applu331	OMP12_mgrid331	OMP12_nab	OMP12_kdtree	OMP12_botsalgn	OMP12_smithwa	OMP12_md	OMP12_bwaves	OMP12_imagick	OMP12_bt331	OMP12_fma3d	OMP12_ilbdc
CPU06_sjeng	9.86	6.05	6.75	7.87	2.92	3.17	3.65	3.87	4.4	4.63	3.91	4.64	4.23	4.49
CPU06_perlbench	9.71	5.73	6.49	7.57	2.56	2.76	3.15	3.25	3.81	4.31	3.88	4.48	4.04	4.58
CPU06_gobmk	10.03	5.82	6.98	7.97	2.56	2.54	2.95	3.34	3.71	4.86	4.22	5.1	4.61	4.77
CPU06_h264ref	9.96	5.57	6.83	7.85	1.51	2.79	2.97	3.29	3.3	4.76	4.04	4.91	4.19	4.36
CPU06_xalancbmk	9.5	5.34	5.91	7.16	1.99	3.46	3.61	3.33	3.7	4.09	3.67	3.84	3.8	4.48
CPU06_gromacs	9.02	5.21	6.24	6.63	3.2	2.26	2.19	2.39	2.44	3.85	4.46	5.05	4.49	5.09
CPU06_tonto	8.79	5.73	6.04	6.63	3.28	1.95	1.94	2.02	2.08	3.58	3.74	4.8	4.57	4.73
CPU06_hmmer	9.01	6.39	6.31	7.21	3.56	1.68	1.38	2.01	1.43	4.08	3.98	5.2	4.74	4.66
CPU06_bzip2	8.64	5.98	5.75	6.63	3.63	1.89	2.47	2.71	3.3	3.19	3.21	4.22	4.22	4.3
CPU06_astar	8.87	5.69	5.64	6.79	3.14	1.71	2.18	2.67	2.85	3.65	3.42	4.22	4.51	4.27
CPU06_namd	9.81	5.44	6.7	7.5	1.97	2.74	2.46	2.42	2.34	4.49	4.67	5.13	3.91	5.01
CPU06_libquantum	9.68	5.99	6.61	7.52	2.66	2.63	2.37	1.58	2.44	4.17	4.4	4.91	3.84	5.3
CPU06_povray	9.56	5.16	6.71	7.13	2.96	2.96	2.62	2.65	2.47	4.5	5.14	5.49	4.41	5.49
CPU06_gamess	9.93	6.38	7.16	7.94	3.62	1.83	1.15	1.48	1.65	4.93	5.19	5.95	5.12	5.86
CPU06_dealII	8.81	5.1	5.72	6.26	3.23	3.53	3.79	3.32	4.12	3.09	3.77	3.8	3.44	4.85
CPU06_calculix	8.3	5.02	5.32	5.74	3.04	3.13	3.4	3.09	3.59	2.53	3.06	3.62	3.61	4.25
CPU06_sphinx3	8.37	5.31	5.53	6.08	3.52	3.68	4.18	4.21	4.49	3.43	2.62	3.82	4.72	3.95
CPU06_cactusADM	8.31	5.84	5.31	6.25	2.79	3.07	3.26	3.14	3.12	2.83	2.14	3.6	3.5	3.25
CPU06_gcc	8.26	5.91	3.86	5.92	4.15	4.49	4.84	4.52	5.09	2.79	3.31	1.64	3.5	4.34
CPU06_zeusmp	7.52	5.42	4.47	4.88	4.05	4.83	5	4.69	4.83	1.96	3.39	2.96	2.68	3.85
CPU06_lbm	8.47	6.26	5.36	6.62	3.14	4.56	4.83	4.83	4.69	3.48	2.11	3.29	3.07	2.12
CPU06_wrf	6.85	5.81	4.38	4.09	5.04	4.99	5.2	4.72	5.09	1.8	3.48	3.44	4.3	4.8
CPU06_soplex	7.22	5.55	4.45	4.53	4.79	4.74	5.13	4.75	5.32	2.07	3.19	3.06	4.35	4.65
CPU06_GemsFDTD	7.03	6.62	4.66	4.96	5.19	5.71	6.07	5.91	6.01	2.86	3	3.46	4.25	3.51
CPU06_bwaves	7.65	7.43	5.26	5.76	5.67	5.15	5.31	4.27	5.28	2.32	3.22	3.72	4.44	5.62
CPU06_milc	7.75	6.87	4.97	5.91	4.65	4.18	4.44	3.75	4.5	2	2.18	3.17	3.7	4.34
CPU06_mcf	6.79	4.95	5.62	3.3	6.42	7.05	7.28	6.79	7.32	3.75	5.92	4.86	5.1	6.61
CPU06_leslie3d	6.07	6.81	5.02	3.36	6.48	6.58	6.8	6.39	6.64	3.24	4.8	4.72	5.53	5.74
CPU06_omnetpp	9.49	7.65	3.44	7.81	6.59	7	7.34	7.62	7.46	6.44	6.02	4.2	6.99	5.87

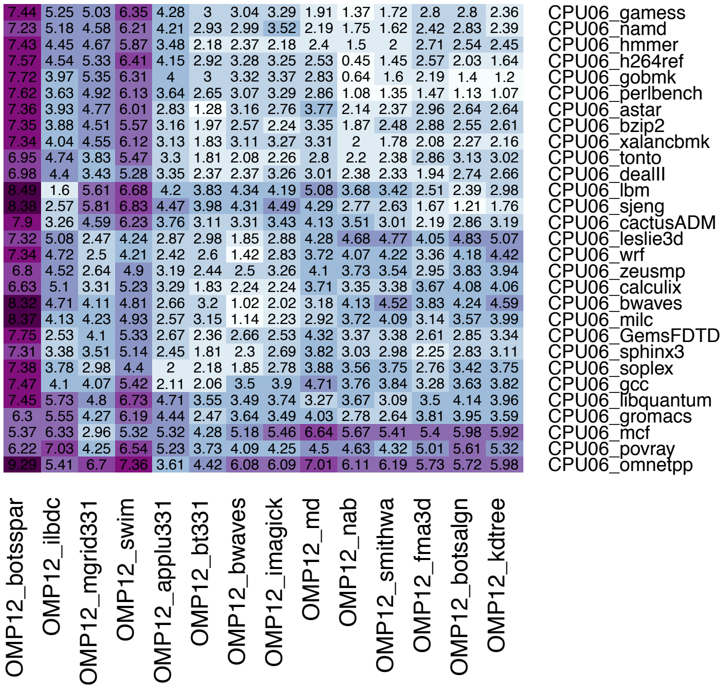
Fig. 3.6. SPEC CPU siblings for serial SPEC OMP on Haswell.

3.8.2 RM-CB-VIF: Reduced Metric and Complete Benchmark Model

This model uses principal metrics identified by VIF (Section 3.5.2) across the complete set of SPEC CPU benchmarks.

3.8.3 RM-CB-VIF-MIC: Reduced Metric and Complete Benchmark Model

This model uses both VIF and MIC to eliminate hardware metrics that are related both linearly and non-linearly across all the SPEC CPU benchmarks.

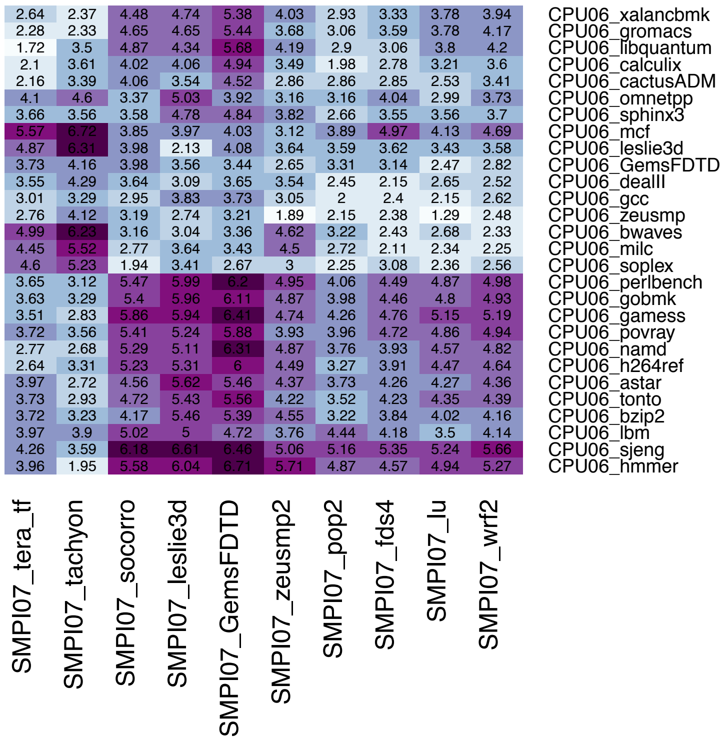


	OMP12_botsspar	OMP12_ilbdc	OMP12_mgrid331	OMP12_swim	OMP12_applu331	OMP12_bt331	OMP12_bwaves	OMP12_imagick	OMP12_md	OMP12_nab	OMP12_smithwa	OMP12_fma3d	OMP12_botsalgn	OMP12_kdtree
CPU06_gamess	7.44	5.25	5.03	6.35	4.28	3	3.04	3.29	1.91	1.37	1.72	2.8	2.8	2.36
CPU06_namd	7.23	5.18	4.58	6.21	4.21	2.93	2.99	3.52	2.19	1.75	1.62	2.42	2.83	2.39
CPU06_hmmer	7.43	4.45	4.67	5.87	3.48	2.18	2.37	2.18	2.4	1.5	2	2.71	2.54	2.45
CPU06_h264ref	7.57	4.54	5.33	6.41	4.15	2.92	3.28	3.25	2.53	0.45	1.45	2.57	2.03	1.64
CPU06_gobmk	7.72	3.97	5.35	6.31	4	3	3.32	3.37	2.83	0.64	1.6	2.19	1.4	1.2
CPU06_perlbench	7.62	3.63	4.92	6.13	3.64	2.65	3.07	3.29	2.86	1.08	1.35	1.47	1.13	1.07
CPU06_astar	7.36	3.93	4.77	6.01	2.83	1.28	3.16	2.76	3.77	2.14	2.37	2.96	2.64	2.64
CPU06_bzip2	7.35	3.88	4.51	5.57	3.16	1.97	2.57	2.24	3.35	1.87	2.48	2.88	2.55	2.61
CPU06_xalancbmk	7.34	4.04	4.55	6.12	3.13	1.83	3.11	3.27	3.31	2	1.78	2.08	2.27	2.16
CPU06_tonto	6.95	4.74	3.83	5.47	3.3	1.81	2.08	2.26	2.8	2.2	2.38	2.86	3.13	3.02
CPU06_dealII	6.98	4.4	3.43	5.28	3.35	2.37	2.37	3.26	3.01	2.38	2.33	1.94	2.74	2.66
CPU06_lbm	8.49	1.6	5.61	6.68	4.2	3.83	4.34	4.19	5.08	3.68	3.42	2.51	2.39	2.98
CPU06_sjeng	8.38	2.57	5.81	6.83	4.47	3.98	4.31	4.49	4.29	2.77	2.63	1.67	1.21	1.76
CPU06_cactusADM	7.9	3.26	4.59	6.23	3.76	3.11	3.31	3.43	4.13	3.51	3.01	2.19	2.86	3.19
CPU06_leslie3d	7.32	5.08	2.47	4.24	2.87	2.98	1.85	2.88	4.28	4.68	4.77	4.05	4.83	5.07
CPU06_wrf	7.34	4.72	2.5	4.21	2.42	2.6	1.42	2.83	3.72	4.07	4.22	3.36	4.18	4.42
CPU06_zeusmp	6.8	4.52	2.64	4.9	3.19	2.44	2.5	3.26	4.1	3.73	3.54	2.95	3.83	3.94
CPU06_calculix	6.63	5.1	3.31	5.23	3.29	1.83	2.24	2.24	3.71	3.35	3.38	3.67	4.08	4.06
CPU06_bwaves	8.32	4.71	4.11	4.81	2.66	3.2	1.02	2.02	3.18	4.13	4.52	3.83	4.24	4.59
CPU06_milc	8.37	4.13	4.23	4.93	2.57	3.15	1.14	2.23	2.92	3.72	4.09	3.14	3.57	3.99
CPU06_GemsFDTD	7.75	2.53	4.1	5.33	2.67	2.36	2.66	2.53	4.32	3.37	3.38	2.61	2.85	3.34
CPU06_sphinx3	7.31	3.38	3.51	5.14	2.45	1.81	2.3	2.69	3.82	3.03	2.98	2.25	2.83	3.11
CPU06_soplex	7.38	3.78	2.98	4.4	2	2.18	1.85	2.78	3.88	3.56	3.75	2.76	3.42	3.75
CPU06_gcc	7.47	4.1	4.07	5.42	2.11	2.06	3.5	3.9	4.71	3.76	3.84	3.28	3.63	3.82
CPU06_libquantum	7.45	5.73	4.8	6.73	4.71	3.55	3.49	3.74	3.27	3.67	3.09	3.5	4.14	3.96
CPU06_gromacs	6.3	5.55	4.27	6.19	4.44	2.47	3.64	3.49	4.03	2.78	2.64	3.81	3.95	3.59
CPU06_mcf	5.37	6.33	2.96	5.32	5.32	4.28	5.18	5.46	6.64	5.67	5.41	5.4	5.98	5.92
CPU06_povray	6.22	7.03	4.25	6.54	5.23	3.73	4.09	4.25	4.5	4.63	4.32	5.01	5.61	5.32
CPU06_omnetpp	9.29	5.41	6.7	7.36	3.61	4.42	6.08	6.09	7.01	6.11	6.19	5.73	5.72	5.98

Fig. 3.7. SPEC CPU siblings for parallel SPEC OMP on Haswell.

3.8.4 Mean Baseline Model

This is a simple average of speedup across all benchmarks in the SPEC CPU benchmark. Without introducing modeling techniques, the simplest way to get an estimate of speedup is to average speedup of benchmarks previously executed on the base and target machine. Fig. 3.10 shows the speedup plot of SPEC CPU between SNB and HSW. The baseline model would provide a decent prediction estimate only if the speedup of the new benchmark lies within a reasonable range of the mean speedup value.



	SMPI07_tera_tf	SMPI07_tachyon	SMPI07_socorro	SMPI07_leslie3d	SMPI07_GemsFDTD	SMPI07_zeusmp2	SMPI07_pop2	SMPI07_fds4	SMPI07_lu	SMPI07_wrf2
CPU06_xalancbmk	2.64	2.37	4.48	4.74	5.38	4.03	2.93	3.33	3.78	3.94
CPU06_gromacs	2.28	2.33	4.65	4.65	5.44	3.68	3.06	3.59	3.78	4.17
CPU06_libquantum	1.72	3.5	4.87	4.34	5.68	4.19	2.9	3.06	3.8	4.2
CPU06_calculix	2.1	3.61	4.02	4.06	4.94	3.49	1.98	2.78	3.21	3.6
CPU06_cactusADM	2.16	3.39	4.06	3.54	4.52	2.86	2.86	2.85	2.53	3.41
CPU06_omnetpp	4.1	4.6	3.37	5.03	3.92	3.16	3.16	4.04	2.99	3.73
CPU06_sphinx3	3.66	3.56	3.58	4.78	4.84	3.82	2.66	3.55	3.56	3.7
CPU06_mcf	5.57	6.72	3.85	3.97	4.03	3.12	3.89	4.97	4.13	4.69
CPU06_leslie3d	4.87	6.31	3.98	2.13	4.08	3.64	3.59	3.62	3.43	3.58
CPU06_GemsFDTD	3.73	4.16	3.98	3.56	3.44	2.65	3.31	3.14	2.47	2.82
CPU06_dealII	3.55	4.29	3.64	3.09	3.65	3.54	2.45	2.15	2.65	2.52
CPU06_gcc	3.01	3.29	2.95	3.83	3.73	3.05	2	2.4	2.15	2.62
CPU06_zeusmp	2.76	4.12	3.19	2.74	3.21	1.89	2.15	2.38	1.29	2.48
CPU06_bwaves	4.99	6.23	3.16	3.04	3.36	4.62	3.22	2.43	2.68	2.33
CPU06_milc	4.45	5.52	2.77	3.64	3.43	4.5	2.72	2.11	2.34	2.25
CPU06_soplex	4.6	5.23	1.94	3.41	2.67	3	2.25	3.08	2.36	2.56
CPU06_perlbench	3.65	3.12	5.47	5.99	6.2	4.95	4.06	4.49	4.87	4.98
CPU06_gobmk	3.63	3.29	5.4	5.96	6.11	4.87	3.98	4.46	4.8	4.93
CPU06_gamess	3.51	2.83	5.86	5.94	6.41	4.74	4.26	4.76	5.15	5.19
CPU06_povray	3.72	3.56	5.41	5.24	5.88	3.93	3.96	4.72	4.86	4.94
CPU06_namd	2.77	2.68	5.29	5.11	6.31	4.87	3.76	3.93	4.57	4.82
CPU06_h264ref	2.64	3.31	5.23	5.31	6	4.49	3.27	3.91	4.47	4.64
CPU06_astar	3.97	2.72	4.56	5.62	5.46	4.37	3.73	4.26	4.27	4.36
CPU06_tonto	3.73	2.93	4.72	5.43	5.56	4.22	3.52	4.23	4.35	4.39
CPU06_bzip2	3.72	3.23	4.17	5.46	5.39	4.55	3.22	3.84	4.02	4.16
CPU06_lbm	3.97	3.9	5.02	5	4.72	3.76	4.44	4.18	3.5	4.14
CPU06_sjeng	4.26	3.59	6.18	6.61	6.46	5.06	5.16	5.35	5.24	5.66
CPU06_hmmer	3.96	1.95	5.58	6.04	6.71	5.71	4.87	4.57	4.94	5.27

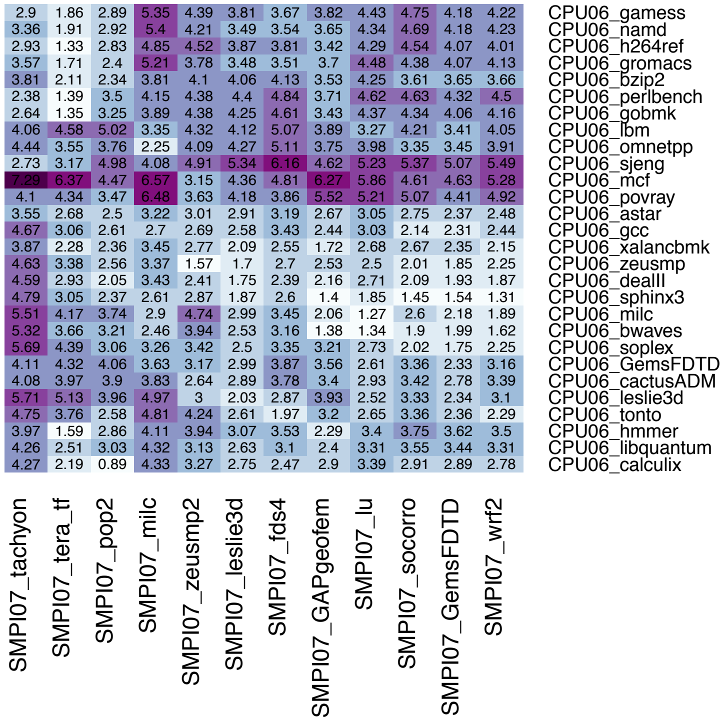
Fig. 3.8. SPEC CPU siblings for serial SPEC MPI on Sandy Bridge.

3.8.5 EU-VIF Model

This model is trained on *sibling benchmarks* identified through the Euclidean distance calculation. The model uses reduced metrics identified by using VIF.

3.8.6 EU-VIF-MIC Model

Similar to EU-VIF, this model is trained on *sibling benchmarks* in the SPEC CPU suite as identified during the workload characterization phase. Metrics used in the EU-VIF-MIC model are filtered first by using VIF, and then MIC.



	SMPI07_tachyon	SMPI07_tera_tf	SMPI07_pop2	SMPI07_milc	SMPI07_zeusmp2	SMPI07_leslie3d	SMPI07_fds4	SMPI07_GAPgeofem	SMPI07_lu	SMPI07_socorro	SMPI07_GemsFDTD	SMPI07_wrf2
CPU06_gamess	2.9	1.86	2.89	5.35	4.39	3.81	3.67	3.82	4.43	4.75	4.18	4.22
CPU06_namd	3.36	1.91	2.92	5.4	4.21	3.49	3.54	3.65	4.34	4.69	4.18	4.23
CPU06_h264ref	2.93	1.33	2.83	4.85	4.52	3.87	3.81	3.42	4.29	4.54	4.07	4.01
CPU06_gromacs	3.57	1.71	2.4	5.21	3.78	3.48	3.51	3.7	4.48	4.38	4.07	4.13
CPU06_bzip2	3.81	2.11	2.34	3.81	4.1	4.06	4.13	3.53	4.25	3.61	3.65	3.66
CPU06_perlbench	2.38	1.39	3.5	4.15	4.38	4.4	4.84	3.71	4.62	4.63	4.32	4.5
CPU06_gobmk	2.64	1.35	3.25	3.89	4.38	4.25	4.61	3.43	4.37	4.34	4.06	4.16
CPU06_lbm	4.06	4.58	5.02	3.35	4.32	4.12	5.07	3.89	3.27	4.21	3.41	4.05
CPU06_omnetpp	4.44	3.55	3.76	2.25	4.09	4.27	5.11	3.75	3.98	3.35	3.45	3.91
CPU06_sjeng	2.73	3.17	4.98	4.08	4.91	5.34	6.16	4.62	5.23	5.37	5.07	5.49
CPU06_mcf	7.29	6.37	4.47	6.57	3.15	4.36	4.81	6.27	5.86	4.61	4.63	5.28
CPU06_povray	4.1	4.34	3.47	6.48	3.63	4.18	3.86	5.52	5.21	5.07	4.41	4.92
CPU06_astar	3.55	2.68	2.5	3.22	3.01	2.91	3.19	2.67	3.05	2.75	2.37	2.48
CPU06_gcc	4.67	3.06	2.61	2.7	2.69	2.58	3.43	2.44	3.03	2.14	2.31	2.44
CPU06_xalancbmk	3.87	2.28	2.36	3.45	2.77	2.09	2.55	1.72	2.68	2.67	2.35	2.15
CPU06_zeusmp	4.63	3.38	2.56	3.37	1.57	1.7	2.7	2.53	2.5	2.01	1.85	2.25
CPU06_dealII	4.59	2.93	2.05	3.43	2.41	1.75	2.39	2.16	2.71	2.09	1.93	1.87
CPU06_sphinx3	4.79	3.05	2.37	2.61	2.87	1.87	2.6	1.4	1.85	1.45	1.54	1.31
CPU06_milc	5.51	4.17	3.74	2.9	4.74	2.99	3.45	2.06	1.27	2.6	2.18	1.89
CPU06_bwaves	5.32	3.66	3.21	2.46	3.94	2.53	3.16	1.38	1.34	1.9	1.99	1.62
CPU06_soplex	5.69	4.39	3.06	3.26	3.42	2.5	3.35	3.21	2.73	2.02	1.75	2.25
CPU06_GemsFDTD	4.11	4.32	4.06	3.63	3.17	2.99	3.87	3.56	2.61	3.36	2.33	3.16
CPU06_cactusADM	4.08	3.97	3.9	3.83	2.64	2.89	3.78	3.4	2.93	3.42	2.78	3.39
CPU06_leslie3d	5.71	5.13	3.96	4.97	3	2.03	2.87	3.93	2.52	3.33	2.34	3.1
CPU06_tonto	4.75	3.76	2.58	4.81	4.24	2.61	1.97	3.2	2.65	3.36	2.36	2.29
CPU06_hmmer	3.97	1.59	2.86	4.11	3.94	3.07	3.53	2.29	3.4	3.75	3.62	3.5
CPU06_libquantum	4.26	2.51	3.03	4.32	3.13	2.63	3.1	2.4	3.31	3.55	3.44	3.31
CPU06_calculix	4.27	2.19	0.89	4.33	3.27	2.75	2.47	2.9	3.39	2.91	2.89	2.78

Fig. 3.9. SPEC CPU siblings for parallel SPEC MPI on Sandy Bridge.

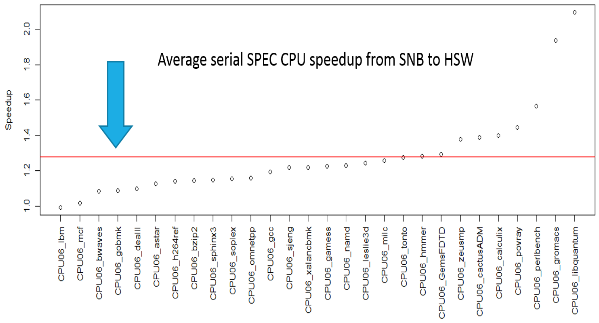


Fig. 3.10. Serial SPEC CPU speedup from Sandy Bridge to Haswell.

We evaluate the prediction quality of these models on two pairs of Intel CPU generations—Haswell and Broadwell, and Sandy Bridge and Haswell. Prediction errors are relative percentage errors calculated as

$$\frac{(RealSpeedup - PredictedSpeedup)}{RealSpeedup} \times 100$$

3.9 STAMPP Performance Predictions

We evaluate the quality of our prediction models, filtered primary hardware metrics, and sibling workloads across different CPU architectures.

3.9.1 Predicting serial SPEC OMP performance on Broadwell

We trained our models on serial SPEC CPU hardware data on Haswell, our *base* machine. Our *target* machine is Broadwell and we predict speedup of serial SPEC OMP from Haswell to Broadwell. The reduced metric models use principal metrics identified by using VIF and MIC and are listed in Tables 3.6 and 3.7. Fig. 3.11 shows prediction accuracy from the baseline model and from the model that best predicts each SPEC OMP benchmark. Overall, our suite of performance models yield better predictions than the standard baseline. *Swim* and *applu331* show significantly higher errors than the baseline model. Based on

Table 3.9. Prediction errors from performance projections

	Haswell to Broadwell (SPEC OMP)		Sandy Bridge to Haswell (SPEC MPI)	
Model	Serial (%)	Parallel (%)	Serial (%)	Parallel (%)
Baseline	11	14	12	16
CM-CB	14	11	36	255
RM-CB-VIF	11	10	19	14
RM-CB-VIF-MIC	11	11	17	14
EU-VIF	11	11	15	16
EU-VIF-MIC	10	11	12	14

our workload analysis, we see that these benchmarks are dissimilar to all the SPEC CPU benchmarks, and could lack representation in the training set. Table 3.9 shows average errors for each benchmark suite across our model suite. The CM-CB model has the highest error (14%) indicating that using all hardware metrics for performance prediction is not beneficial. In this prediction setup, the *EU-VIF-MIC* model provides the best accuracy at 10% compared to 11% accuracy from the baseline model.

3.9.2 Predicting parallel SPEC OMP performance on Broadwell

In this setting, we train our models on hardware metrics collected from executing SPEC CPU benchmarks on 28 Haswell cores. Table 3.9 shows that all STAMPP's models have lower prediction errors than the baseline model (14%). The *RM-CB-VIF* model yields the best accuracy with an average error of 10%, while the rest of the models have prediction errors of 11% each. Fig. 3.12 shows distribution of prediction errors from the baseline model and the model from STAMPP's suite that best predicts SPEC OMP benchmarks. Overall, prediction models from our suite yield lower prediction errors than the baseline model on all benchmarks. *Bt331* and *ilbdc* have prediction errors significantly different from the baseline model. However, prediction errors from STAMPP's models for *bt331* and *ilbdc* are lower than 5%. STAMPP's models and the baseline model have a hard time predicting performance of *botsalgn*, *fma3d*, and *applu331* with errors greater than 15%.

3.9.3 Predicting serial SPEC MPI performance on Haswell

To predict performance of serial SPEC MPI on Haswell, we trained our suite of models on hardware metrics obtained from executing SPEC CPU on SNB and SPEC CPU speedup from SNB to HSW. Fig. 3.13 shows distribution of prediction errors from the baseline model and the STAMPP prediction model that best predicts performance of SPEC MPI. *Tera_tf*, *wrf2*, *pop2*, and *tachyon* have higher prediction errors. From our workload analysis, we find that *socorro* and *lu* are very different from the SPEC CPU set. The misprediction of these outlier benchmarks contributes to the high average prediction error. In Table 3.9, both the baseline model and the EU-VIF-MIC model have prediction errors of 12%.

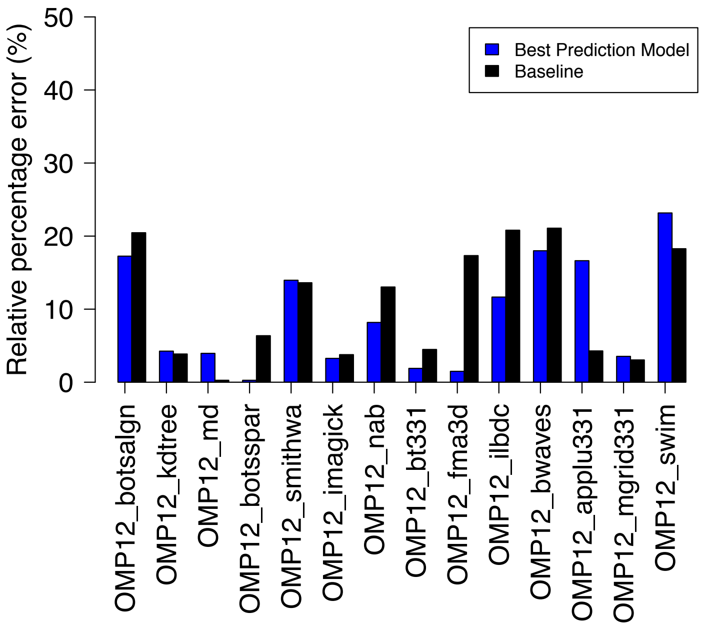


Fig. 3.11. Serial SPEC OMP prediction errors from baseline and STAMPP on Broadwell.

3.9.4 Predicting parallel SPEC MPI performance on Haswell

To predict parallel SPEC MPI performance on Haswell, we train our prediction models on hardware metrics obtained from parallel SPEC CPU execution on Sandy Bridge. Fig. 3.14 shows distribution of prediction errors from the baseline model and STAMPP. Similar to predictions on serial SPEC MPI, both the baseline and STAMPP have difficulty predicting *tera_tf*, *pop2*, and *socorro*. Because of these outliers, the average prediction error across the SPEC MPI suite is 14% while the baseline model averages a prediction error of 16%.

3.10 Summarizing STAMPP's Prediction Results

With the exception of a few outliers, our suite of performance models yield a higher prediction accuracy than the baseline model across both machine pairs. SPEC MPI outliers such as *socorro*, *pop2*, and *tera_tf* are characterized by heavy MPI traffic. Our metric collection focuses on recording responses from cache and DRAM and is agnostic to instruction

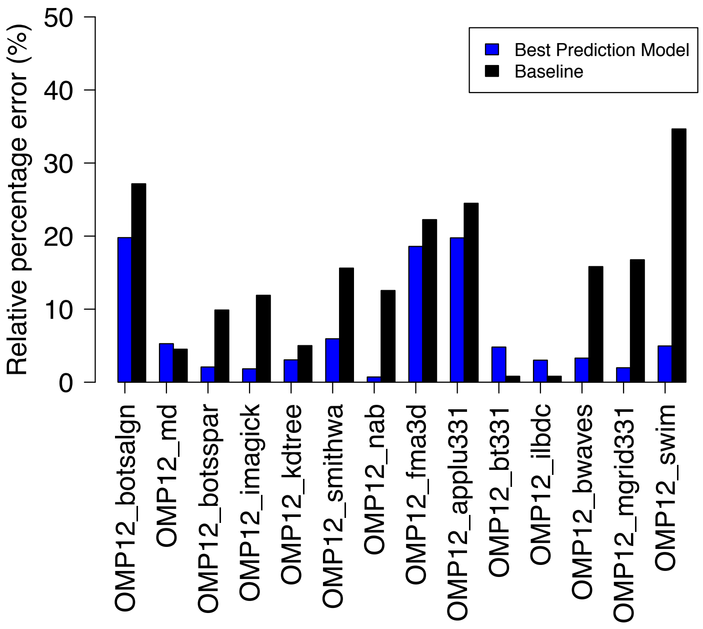


Fig. 3.12. Parallel SPEC OMP prediction errors from baseline and STAMPP on Broadwell.

or code flow. Parallel SPEC CPU executions are embarrassingly parallel and do not use either data or task decomposition. Because of this, models trained on parallel executions of SPEC CPU fail to predict certain SPEC MPI workloads. Speedup due to factors such as improved compilers and libraries on a new system is difficult to represent currently in STAMPP.

3.11 Conclusion

The ability to estimate performance of applications on new infrastructures is important for several hardware and software efforts in the HPC community. Measuring performance benefits through intensive hardware monitoring is expensive and often times unfeasible. Hardware monitoring costs increase as it extends to include diverse hardware, performance counters, and workloads. STAMPP presents a set of techniques to automate the analysis of hardware metrics and workload space, and the prediction performance of applications before

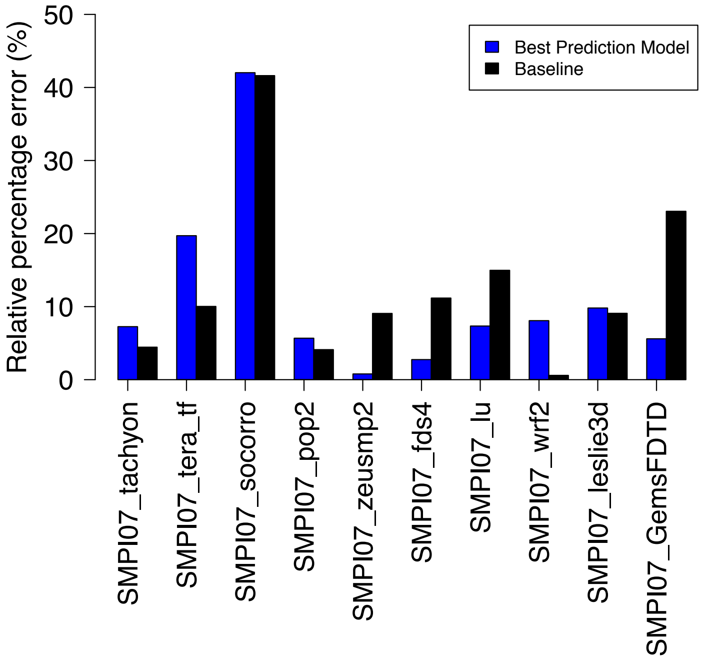


Fig. 3.13. Serial SPEC MPI prediction errors from baseline and STAMPP on Haswell.

they are executed on a target system. Using a suite of performance models, predictions from STAMPP achieve high performance prediction accuracy on individual benchmarks for performance projections across Haswell and Broadwell processors.

Acknowledgment

The author would like to thank Hugh Caffey for collecting hardware metrics for benchmarks used in this work.

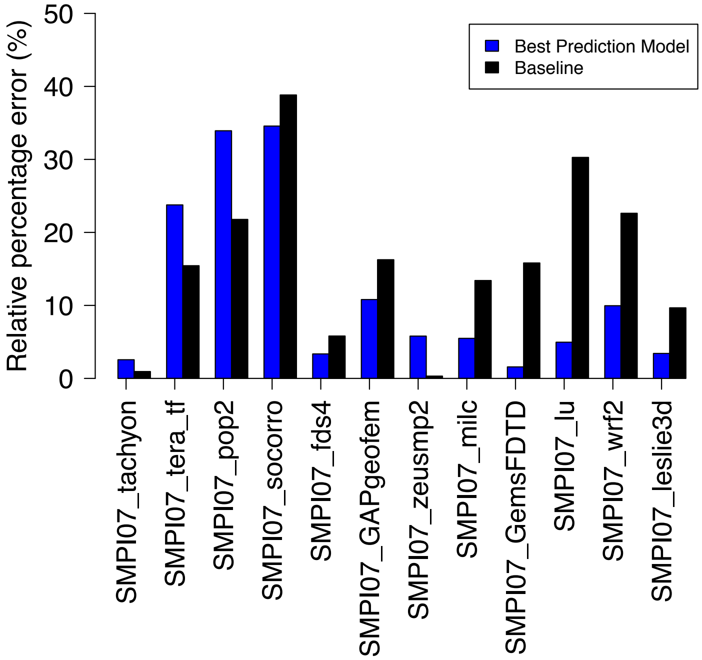


Fig. 3.14. Parallel SPEC MPI prediction errors from baseline and STAMPP on Haswell.

CHAPTER 4

CONCLUSION

4.1 Performance Modeling: Performance Prediction and Performance Optimization

Performance modeling, performance optimization, and performance projections are important efforts in the high performance computing (HPC) hardware and software community. However, optimizing performance is often challenging because workload performance is influenced by several factors such as compiler flags, memory settings, and the data used by the workload, to name a few. This dissertation demonstrates the effectiveness of using statistical data analysis techniques to model, optimize, and predict performance in both software (workload) and hardware problem spaces. The two studies presented in this dissertation demonstrate how statistical analyses and modeling can improve performance predictions in two situations: (1) predicting and optimizing workload performance across different input data sets by guiding optimal parameter selection and (2) predicting performance of diverse workloads across different CPU architectures. The contributions of each of these studies are described below.

4.1.1 Contributions to performance prediction and performance optimization for workloads

Performance of a workload can vary based on values of key parameters that influence performance. Sparse matrix vector multiplication (SpMV) is a performance bottleneck in several codes such as Internet search engine algorithms, data mining algorithms, and that use SpMV multiple times. Blocking a sparse matrix to improve memory locality during an SpMV operation is one of the easiest ways to optimize SpMV performance. However, selecting an incompatible and wrong block size can cause performance degradation. To predict the right block size for a new, unseen sparse matrix prior to SpMV, it is essential to predict accurate run time of the matrix prior to its SpMV operation. In our study—

Statistical Techniques for Optimizing and Modeling Performance of blocked sparse matrix vector multiplication (STOMP)—we developed statistical performance models to predict performance of a new matrix and guide block size selection. Using statistical models trained on data from matrices previously used in SpMV operations, we predicted run time of a new matrix with mean accuracy of 93.52% across matrices on the Sandy Bridge processor. STOMP's block selection technique produced a performance benefit of as high as 75% on the Sandy Bridge processor. Our techniques produced an average performance improvement of 50.32% over default unblocked SpMV performance across all our matrices. We compared the quality of STOMP's block selection process with SPARSITY, a framework that defines static heuristics for block-size selection. On the same set of matrices, STOMP yielded a 50.96% speedup while SPARSITY yielded a 31.62% speedup over the same default. We also evaluated STOMP's performance prediction capabilities on the new Knight's Landing co-processor. STOMP produced an average performance prediction accuracy of 91.92% on Knight's Landing across the same matrices used in the study on the Sandy Bridge processor. By re-using SpMV data from previously executed matrices, STOMP demonstrated success in making a-priori decisions to improve performance of a workload before its execution.

We now show how this work answers the research questions posed in Chapter 1:

1. *Given the diversity of sparse matrices, is it possible to accurately predict performance of a new sparse matrix prior to SpMV operation?* – Yes. By building statistical prediction models on SpMV data from a diverse set of sparse matrices, STOMP predicted SpMV performance with an average accuracy of 93.52% on the Sandy Bridge processor and 91.92% on the Knights Landing co-processor.

2. *Is there a way to characterize sparse matrices during an SpMV operation, such that this characterization remains valid across sparse matrices with diverse and unpredictable nonzero patterns?* – Much past work has used the number of nonzero elements (NNZE) in a sparse matrix to quantify work for different sparse operations. However, we have shown that the number of nonzero blocks (NNZB) is more useful in characterizing a matrix and abstracting SpMV performance. Prediction accuracy

from models using NNZBs is several orders of magnitude higher than that of NNZE models.

3. *Can we reuse performance information from previous workload executions to tune a new workload prior to its execution?* – Indeed, by utilizing SpMV performance information from different sparse matrices, we built performance models to predict performance of a new matrix prior to its SpMV execution. We then used these models to guide block size selection for the matrix. STOMP's block size selection speeds up performance of default SpMV by an average of 54.46%. STOMP's performance improvements are comparable to an expensive, exhaustive search over a range of block sizes. STOMP enables optimal SpMV performance through its block size selection process.

4. *Can statistical prediction techniques be ported across different processor architectures?* – Yes. By utilizing performance information from the processor of choice, STOMP can be used to predict SpMV performance on different processors. We have evaluated STOMP on two kinds of processors with promising results—Sandy Bridge (Intel Xeon line) and Knights Landing (Intel Xeon Phi line).

4.1.2 Contributions to performance prediction across Intel CPUs

When making performance predictions of workloads on a new processor, hardware architects are faced with large dimensional data resulting from monitoring different hardware across large sets of workloads. Large amounts of performance data seldom provide instantaneous insight into performance trends. Excessive performance monitoring is expensive and requires multiple executions of the workload on a processor. The number of workload executions increases as performance monitoring experiments include diverse workloads. Excessive hardware sampling leads to capturing the same phenomena in multiple ways and introduces redundancy. To remedy this, we used statistical techniques to reduce the amount of time dedicated to performance monitoring experiments before performance projection. We reduced the time dedicated to hardware monitoring by identifying important, non-redundant hardware metrics that provide insight into performance trends. In addition, we reduced the number of workloads used in performance monitoring experiments by identifying “sibling workloads” that can serve as proxies for other workloads. We then built a suite of statistical prediction models to predict performance of a new, unseen workload on a new machine prior to its execution.

Our study—Statistical Techniques for Analyzing Metrics and Predicting Performance of workloads (STAMPP)—demonstrates effectiveness of using statistical techniques and statistical models to analyze metric space and workloads, and to predict performance of a new workload on a new processor. STAMPP explores hardware metric space and predicts performance of the SPEC OMP and SPEC MPI suites of benchmarks across recent generations of Intel CPUs—Broadwell, Haswell, and Sandy Bridge in serial and parallel modes. STAMPP yielded high accuracy for predicting performance of individual benchmarks on these CPUs.

Although STAMPP is evaluated on Intel CPU generations, STAMPP’s methodology is generalized and processor independent, and can be harnessed to predict different performance measures, such as power, across different kinds of processors using relevant hardware metrics.

This STAMPP work has answered the following research questions pertaining to inter-architecture performance predictions:

1. *Are performance prediction models portable across different architectures?* – Predicting performance across different processors is challenging. STAMPP addresses this by training prediction models on performance information from both a newer and an older processor, but uses information from only the older processor to predict performance. By including performance metrics from the newer processor, albeit in short supply, STAMPP’s performance prediction models have real information (as opposed to synthetic or simulated) about hardware from the new processor.

2. *Is there a benefit to monitoring all hardware components (cache and memory) multiple times?* – No. Measuring all possible hardware components introduces redundancy

and significantly increases the cost of performance monitoring. Redundant hardware metric information also makes predicting performance more challenging. STAMPP demonstrates that performance prediction accuracy increases after pruning hardware metric data to eliminate non-essential performance information.

3. *Can certain sets of workloads be used as performance proxies for a different workload?* – Indeed, by examining hardware responses from a set of workloads, STAMPP's heuristic finds sets of workloads that can act as proxies for a new, test workload.

4.2 Future Work

With the increase in computational capabilities and the innovation of scalable computing infrastructure in recent years, statistical learning is continuing to span several market segments. This makes a very promising case for statistical learning in HPC; whether by using the power of HPC resources to speed up statistical algorithms or by using statistical algorithms to design, explore, and exploit performance potential of HPC resources. In addition to performance prediction, this dissertation envisions the use of statistical techniques to aid performance analysis in our work (STOMP and STAMPP) and other directions.

4.2.1 Extensions to STOMP

STOMP's block-selection technique shows promise for correcting load imbalance in parallel SpMV when executed on multiple processors. STOMP can be used to tune additional matrix data structures that rely on block size to attain optimal performance, such as blocked ELLPACK (BELL). In addition to SpMV, STOMP can be generalized to optimize additional linear algebra routines such as linear solvers, matrix transpose, convolutions, and mathematical routines involving combinations of multiple matrix-vector multiplications.

4.2.2 Extensions to STAMPP

A natural extension to STAMPP would be to replicate it using hardware metrics collected during the entire run time of a workload. Summary metrics collected at the end of execution do not provide detailed insight into variations of performance. Increasing diversity of workloads in the experiments will provide better representation for workloads dominated by certain features: MPI traffic, increased working set size, irregular memory accesses, etc. Analyzing an application's instruction mix together with hardware metrics could help with fine grain performance predictions across different phases of an application's run time.

4.2.3 Additional uses cases that could benefit from statistical analysis

- *Optimization opportunities in speeding up algorithms*: Statistical techniques can identify additional optimization opportunities in algorithms with diverse computational patterns.
- *Improving HPC toolsets*: By training statistical models on previously executed datasets, tools such as compilers and libraries can determine better optimization strategies for codes with specific computational patterns.
- *Resource allocation in supercomputing centers*: By using statistical techniques to analyze application executions on different datasets, schedulers in charge of resource allocation can override incorrect allocation estimates provided by end users.

REFERENCES

[1] J. Lee, H. Kim, and R. Vuduc, "When prefetching works, when it doesn't, and why," *ACM Transactions on Architecture and Code Optimization*, 2012.

[2] "Intel hyperthreading," http://www.intel.com/content/www/us/en/architecture-and-technology/hyperthreading/hyper-threading-technology.html, 2016, [Online; accessed September 26, 2016].

[3] Z. Zhao and K. Antypas, "Effects of hyper-threading on the NERSC workload on Edison," *Cray User Group 2013 (CUG2013)*, 2013.

[4] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and exploiting program phases," *IEEE Micro*, 2003.

[5] D. Grech and Z. Mazur, "Can one make any crash prediction in finance using the local hurst exponent idea?" *Physica A: Statistical Mechanics and its Applications*, vol. 336, no. 1, pp. 133–145, 2004.

[6] G. Zhang, M. Y. Hu, E. B. Patuwo, and D. C. Indro, "Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis," *European journal of operational research*, vol. 116, no. 1, pp. 16–32, 1999.

[7] E. Scalas, R. Gorenflo, F. Mainardi *et al.*, "Fractional calculus and continuous-time finance," EconWPA, Tech. Rep., 2004.

[8] J. Murphy, "Predictions of climate change over europe using statistical and dynamical downscaling techniques," *International Journal of Climatology*, vol. 20, no. 5, pp. 489–501, 2000.

[9] A. Dupuy and R. M. Simon, "Critical review of published microarray studies for cancer outcome and guidelines on statistical analysis and reporting," *Journal of the National Cancer Institute*, vol. 99, no. 2, pp. 147–157, 2007.

[10] A. Ihbal, H. S. Rajamani, R. A. Abd-Alhameed, and M. Jalboub, "Statistical predictions of electric load profiles in the uk domestic buildings," in *1st International Conference on Energy, Power and Control (EPC-IQ), 2010*, 2010, pp. 345–350.

[11] E.-J. Im, K. Yelick, and R. Vuduc, "Sparsity: Optimization framework for sparse matrix kernels," *International Journal of High Performance Computing Applications*, vol. 18, no. 1, pp. 135–158, 2004.

[12] H. A. V. der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of BiCG for the solution of nonsymmetric linear systems," *SIAM Journal on scientific and Statistical Computing*, pp. 631–644, 1992.

[13] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, 1986.

[14] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: bringing order to the web," *Stanford tech report*, 1999.

[15] A. Buttari, V. Eijkhout, J. Langou, and S. Filippone, "Performance optimization and modeling of blocked sparse kernels," *International Journal of High Performance Computing Applications*, vol. 21, no. 4, pp. 467–484, Nov. 2007.

[16] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris, "Understanding the performance of sparse matrix-vector multiplication," in *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2008, pp. 283–292.

[17] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object-oriented numerical software libraries," in *Modern software tools for scientific computing*, 1997, pp. 163–202.

[18] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, and V. Eijkhout, “PETSc users manual revision 3.5,” *Technical report, Argonne National Laboratory (ANL)*, 2014.

[19] T. A. Davis and Y. Hu, “The University of Florida sparse matrix collection,” *ACM Transactions on Mathematical Software (TOMS)*, 2011.

[20] “Boeing matrix group,” http://www.cise.ufl.edu/research/sparse/matrices/Boeing/index.html, 2015, [Online; accessed August 12, 2016].

[21] “Chen matrix group,” http://www.cise.ufl.edu/research/sparse/matrices/Chen/index.html, 2015, [Online; accessed August 12, 2016].

[22] “FIDAP matrix group,” http://www.cise.ufl.edu/research/sparse/matrices/FIDAP/index.html, 2015, [Online; accessed August 12, 2016].

[23] “LAW matrix group,” http://www.cise.ufl.edu/research/sparse/matrices/LAW/index.html, 2015, [Online; accessed August 12, 2016].

[24] “Bai matrix group,” http://www.cise.ufl.edu/research/sparse/matrices/Bai/index.html, 2015, [Online; accessed August 12, 2016].

[25] “COO,” https://software.intel.com/en-us/node/599837, 2016, [Online; accessed April 17, 2016].

[26] “CSR,” http://netlib.org/linalg/html_templates/node91.html, 2016, [Online; accessed April 17, 2016].

[27] “CSC,” http://netlib.org/linalg/html_templates/node92.html, 2016, [Online; accessed April 17, 2016].

[28] “ELL,” https://www.lanl.gov/Caesar/node223.html, 2016, [Online; accessed April 17, 2016].

[29] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," in *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2010, pp. 115–126.

[30] "JDS," http://netlib.org/linalg/html_templates/node95.html, 2016, [Online; accessed April 17, 2016].

[31] "SKY," http://www.netlib.org/utk/people/JackDongarra/etemplates/node378.html, 2016, [Online; accessed April 17, 2016].

[32] A. Ashari, N. Sedaghati, J. Eisenlohr, and P. Sadayappan, "An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on GPUs," in *Proceedings of the 28th ACM International Conference on Supercomputing*, 2014, pp. 273–282.

[33] J. Byun, R. Lin, J. W. Demmel, and K. A. Yelick, "pOSKI: Parallel optimized sparse kernel interface library," *Technical report, University of California, Berkeley*, 2012.

[34] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "A column approximate minimum degree ordering algorithm," *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 3, pp. 353–376, Sep. 2004.

[35] W.-H. Liu and A. H. Sherman, "Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices," *SIAM Journal on Numerical Analysis*, vol. 13, no. 2, pp. 198–213, 1976.

[36] A. Pinar and M. T. Heath, "Improving performance of sparse matrix-vector multiplication," in *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, 1999.

[37] R. W. Vuduc, "Automatic performance tuning of sparse matrix kernels," Ph.D. dissertation, University of California, Berkeley, 2003.

[38] V. Karakasis, G. Goumas, and N. Koziris, "A comparative study of blocking storage methods for sparse matrices on multicore architectures," in *IEEE International Conference on Computational Science and Engineering (CSE)*, 2009, pp. 247–256.

[39] W. Liu and B. Vinter, "CSR5: An efficient storage format for cross-platform sparse matrix-vector multiplication," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015.

[40] L. Grigori and X. S. Li, "A new scheduling algorithm for parallel sparse LU factorization with static pivoting," in *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, 2002, pp. 1–18.

[41] N. Sedaghati, T. Mu, L.-N. Pouchet, S. Parthasarathy, and P. Sadayappan, "Automatic selection of sparse matrix representation on GPUs," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 99–108.

[42] P. Guo, L. Wang, and P. Chen, "A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 1112–1123, 2014.

[43] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[44] "Blocked SpMV code in PETSc," http://www.mcs.anl.gov/petsc/petsc-current/src/mat/impls/baij/seq/baij2.c.html, 2016, [Online; accessed April 17, 2016].

[45] X. Liu, M. Smelyanskiy, E. Chow, and P. Dubey, "Efficient sparse matrix-vector multiplication on x86-based many-core processors," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, 2013, pp. 273–282.

[46] B.-Y. Su and K. Keutzer, "clSpMV: A cross-platform OpenCL SpMV framework on GPUs," in *Proceedings of the 26th ACM International Conference on Supercomputing*, 2012, pp. 353–364.

[47] "Knight's Landing," http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html, 2016, [Online; accessed September 17, 2016].

[48] J. L. Henning, "Performance counters and development of SPEC CPU2006," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 118–121, Mar. 2007.

[49] M. S. Müller, J. Baron, W. C. Brantley, H. Feng, D. Hackenberg, R. Henschel, G. Jost, D. Molka, C. Parrott, J. Robichaux, P. Shelepugin, M. van Waveren, B. Whitney, and K. Kumaran, "SPEC OMP2012 – an application benchmark suite for parallel systems using OpenMP," in *International Conference on OpenMP in a Heterogeneous World*, 2012.

[50] M. S. Müller, M. van Waveren, R. Lieberman, B. Whitney, H. Saito, K. Kumaran, J. Baron, W. C. Brantley, C. Parrott, T. Elken, H. Feng, and C. Ponder, "SPEC MPI 2007—an application benchmark suite for parallel systems using MPI," *Concurrency and Computation: Practice and Experience*, vol. 22, pp. 191–205, 2010.

[51] "SPEC CPU 2006 description," https://www.spec.org/cpu2006/Docs/, 2006, [Online; accessed August 12, 2016].

[52] "SPEC MPI 2007 description," https://www.spec.org/omp2012/Docs/, 2012, [Online; accessed August 12, 2016].

[53] "SPEC MPI 2007 description," https://www.spec.org/mpi2007/Docs/, 2007, [Online; accessed August 12, 2016].

[54] G. Zheng, G. Kakulapati, and L. Kale, "BigSim: a parallel simulator for performance prediction of extremely large parallel machines," in *Parallel and Distributed Processing Symposium*, 2004.

[55] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. Kale, "Simulation-based performance prediction for large parallel machines," *International Journal of Parallel Programming*, vol. 33, pp. 183–207, 2005.

[56] S. Prakash and R. L. Bagrodia, "MPI-SIM: Using parallel simulation to evaluate mpi programs," in *Proceedings of the 30th Conference on Winter Simulation.* IEEE Computer Society Press, 1998.

[57] R. Riesen, “A hybrid mpi simulator,” *IEEE International Conference on Cluster Computing (CLUSTER)*, 2006.

[58] A. Phansalkar, A. Joshi, and L. K. John, “Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite,” in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*, 2007, pp. 412–423.

[59] A. D. Breslow, L. Porter, A. Tiwari, M. Laurenzano, L. Carrington, D. M. Tullsen, and A. E. S. , “The case for colocation of high performance computing workloads,” *Concurr. Comput. : Pract. Exper.*, 2016.

[60] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. D. Bosschere, “Performance prediction based on inherent program similarity,” in *International Conference on Parallel Architectures and Compilation Techniques*, 2006, pp. 114–122.

[61] S. Sharkawi, D. DeSota, R. Panda, R. Indukuru, S. Stevens, V. Taylor, and X. Wu, “Performance projection of hpc applications using SPEC CPU 2006 benchmarks,” in *IEEE International Symposium on Parallel and Distributed Processing*, 2009, pp. 1–12.

[62] A. Jaleel, “Memory characterization of workloads using instrumentation-driven simulation,” http://www.glue.umd.edu/ajaleel/workload, 2010, [Online; accessed April 17, 2016].

[63] “CORAL: Collaboration Oak Ridge, Argonne, Livermore,” https://asc.llnl.gov/CORAL/, 2016, [Online; accessed September 26, 2016].

[64] “APEX: Alliance for Application Performance at Extreme Scale,” http://www.lanl.gov/projects/apex/, 2016, [Online; accessed September 26, 2016].

[65] J. Tramm, Siegel, A. R., T. Islam, and M. Schulz, “The development and verification of a performance abstraction for Monte Carlo reactor analysis,” in *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*, 2014.

[66] P. Romano, B. R. Herman, N. Horelik, A. Nelson, B. Forget, and K. Smith, “OpenMC: A state-of-the-art Monte Carlo code for research and development,” in *Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo*, 2013.

[67] R. Preissl, T. Köckerbauer, M. Schulz, D. Kranzlmüller, B. R. d. Supinski, and D. J. Quinlan, “Detecting patterns in MPI communication traces,” in *International Conference on Parallel Processing*, 2008, pp. 230–237.

[68] S. Sodhi and J. Subhlok, “Skeleton based performance prediction on shared networks,” in *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, April 2004, pp. 723–730.

[69] H. Kim, R. Vuduc, S. Baghsorkhi, J. Choi, and W.-m. Hwu, “Performance analysis and tuning for general purpose graphics processing units (GPGPU),” *Synthesis Lectures on Computer Architecture*, vol. 7, no. 2, pp. 1–96, 2012.

[70] A. Pesterev, N. Zeldovich, and R. T. Morris, “Locating cache performance bottlenecks using data profiling,” in *European conference on Computer systems*, 2010, pp. 335–348.

[71] M. Dimitrov, K. Kumar, P. Lu, V. Viswanathan, and T. Willhalm, “Memory system characterization of big data workloads,” in *IEEE International Conference on Big Data*, 2013, pp. 15–22.

[72] J. Diamond, M. Burtscher, J. D. McCalpin, B.-D. Kim, S. W. Keckler, and J. C. Browne, “Evaluation and optimization of multicore performance bottlenecks in supercomputing applications,” in *Performance Analysis of Systems and Software (ISPASS)*, 2011, pp. 32–43.

[73] “Higher order method modeling environment HOMME benchmark,” https://wiki.ucar.edu/display/tddbenchmark/HOMME, 2011, [Online; accessed November 22, 2016].

[74] D. A. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. Mcvean, P. J. Turnbaugh, S. L. Eric, M. Mitzenmacher, and P. C. Sabeti, “Detecting novel associations in large data sets,” *Science*, pp. 1518–1524, 2011.

[75] M. H. Kutner, *Applied linear statistical models*, Berlin, Heidelberg, 1996.

[76] J. Levin, “Elementary statistics in social research,” in *Pearson Education India*, 2006.

[77] E. R. Mansfield and B. P. Helms, “Detecting multicollinearity,” *The American Statistician*, 1982.

[78] “VIF stepwise variable selection,” https://beckmw.wordpress.com/2013/02/05/collinearity-and-stepwise-vif-selection/, 2016, [Online; accessed August 12, 2016].

[79] M. H. Dunham, “Introductory and advanced topics.” in *Pearson Education India*, 2006.

[80] H. Akaike, *Akaike’s Information Criterion.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 25–25.

APPENDICES

Appendix A

Permission-to-use Letter

DocuSign Envelope ID: 94FAC93A-9F2A-469C-91D4-C3E8389BEA84

To whom it may concern:

I, Forrest Iandola, am a co-author of the paper “STOMP: Statistical Techniques for Optimizing and Modeling Performance of blocked sparse matrix vector multiplication,” published in proceedings of the 28th International Symposium of Computer Architecture and High Performance Computing, Los Angeles, California, October 2016. I give my permission to Steena Monteiro to use this work in her dissertation.

DocuSigned by:
Forrest Iandola 11/15/2016
4DE4B4C3DC4340C...

Forrest Iandola, CEO DeepScale.

CURRICULUM VITAE

Steena Monteiro

Education

Ph.D., Computer Science. Utah State University, Logan, Utah. 2016.

M.S., Computer Science. Utah State University, Logan, Utah. 2008.

B.E., Information Technology. University of Mumbai, Mumbai, India. 2005.

Research Interests

performance analysis, high performance computing, performance modeling, performance tuning, statistical analysis, machine learning algorithms, parallel file systems

Research and Industry Experience

- High performance computing graduate intern, Intel Corporation, Santa Clara, CA, July 2015–present
- Lawrence graduate scholar, Lawrence Livermore National Laboratory, Livermore, CA, June 2011–June 2015
- High performance computing graduate intern, Intel Corporation, Dupont, WA, June 2014–October 2014
- Computation intern, Lawrence Livermore National Laboratory, Livermore, CA, May 2010–August 2010
- Ph.D. cybersecurity intern, Pacific Northwest National Laboratory, Richland, WA, May 2009–August 2009

Awards

- Lawrence Livermore National Laboratory's Lawrence Livermore Graduate Scholar Program scholarship (2011–2015)
- XSEDE/PRACE/RIKEN sponsored scholarship to attend the International Summer School on HPC Challenges in Computational Sciences, New York City, New York, 2013
- Broader Engagement travel award to attend Supercomputing, New Orleans, LA, 2014
- Broader Engagement travel award to attend Supercomputing, Salt Lake City, UT, 2012
- Computing Research Association-W sponsored grant for the Grad Cohort, Boston, MA, 2011
- Computer Science Department Outstanding Graduate Teaching Assistant Award 2010
- Broader Engagement travel award to attend Supercomputing, New Orleans, LA, 2010
- Microsoft's Golden Ticket Program at Redmond, WA, 2010
- Computing Research Association-W sponsored grant to attend the Grad Cohort, Seattle, WA, 2010
- Utah State University Graduate Student Senate Travel Award, Keystone, Colorado, 2008
- Microsoft-sponsored travel grant for ACM SRC, Keystone, CO, 2008
- IBM-sponsored Scholarship Grace Hopper conference, Keystone, CO, 2008
- NSF and ARO sponsored travel grant to attend SADFE, Oakland, CA, 2008
- First Google Workshop for Women Engineers, San Jose, CA, 2008
- Utah State University instate tuition award, 2007

- NSF-sponsored full travel scholarship to attend Grace Hopper Conference, Orlando, FL, 2007

Conference Publications

- Monteiro, S., Sharapov, I., and Naik, S., STAMPP: Statistical Techniques for Analyzing Metrics and Predicting Performance of workloads, International Symposium on Performance Analysis of Systems and Software 2017. (under review)
- Monteiro, S., Iandola, F., and Wong, D., STOMP: Statistical Techniques for Optimizing and Modeling Performance of blocked sparse matrix vector multiplication, in proceedings of the 28th International Symposium of Computer Architecture and High Performance Computing, Los Angeles, California, 2016.
- Monteiro, S. and Erbacher, R.F., Exemplifying Attack Identification and Analysis in a Novel Forensically Viable Syslog Model, 3rd IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering, Oakland, CA, May 2008, pp. 57-68.

Posters

- Monteiro, S., Bronevetsky, G., and Casas-Guix, M. Modeling Data-driven Application Behavior, Research Poster, Supercomputing '12, Salt Lake City, Utah, November 12-November 18, 2012
- Monteiro, S., Bronevetsky, G., and Casas-Guix, M. Modeling and Predicting Computational Behavior of Large-scale Data-driven Applications, ACM Student Research Competition, Grace Hopper Conference, Portland, Oregon, November 9-November 12, 2011
- Monteiro, S., Bronevetsky, G., and Casas-Guix, M., Modeling the Behavior of the Ceph Parallel File System for Classifying and Detecting Performance Faults, Lawrence Livermore Summer Scholar Poster Symposium, August 12, 2010

- Monteiro, S. and Erbacher, R.F., A Novel Authentication and Validation Mechanism for Attack Detection and Trackback in Syslogs for Forensic Analysis, ACM Student Research Competition, Grace Hopper Conference, Keystone, Colorado, 2008.
- Monteiro, S. and Erbacher, R.F., Authenticating and Validating Logs for Forensic Analysis," Grace Hopper Conference, Orlando, Florida, October 17-20, 2007.

Talks

- Modeling and Predicting Performance of MPIBLAST across Multiple Genome Sequences and Databases, International Summer School on HPC Challenges in Computational Sciences, New York City, NY, June 23–June 28, 2013.
- Modeling Data-driven Application Behavior, Early Doctoral Research Showcase, Supercomputing '12, Salt Lake City, UT, November 2012.
- Modeling and Predicting Computational Behavior of Large-scale Data-driven Applications at the NNSA/ASC Technical Program, Supercomputing '11, Seattle, WA, November 2011.
- Modeling the Behavior of the Ceph Parallel File System for Detecting and Classifying Performance Faults at the NNSA/ASC Technical Program, Supercomputing '10, New Orleans, LA, November 2010.

Journal Publications

- Monteiro, S. and Bryce, R. Code Inspections: A Web Crawler Exercise for Students, ACM Journal of Computing Sciences in Colleges (JCSC), December 2011, pp: 67-77.
- Monteiro, S. and Erbacher, R.F. An Authentication and Validation Mechanism for Analyzing Syslogs Forensically," ACM SIGOPS Operating Systems Review, Vol. 42, No. 3 , 2008, pp. 41-50.