ABSTRACT

Title of dissertation:	Locality Transformations and Prediction Techniques for Optimizing Multicore Memory Performance
	Abdel-Hameed Badawy, Doctor of Philosophy, 2013
Dissertation directed by:	Professor Donald Yeung Department of Electrical and Computer Engineering

Chip Multiprocessors (CMPs) are here to stay for the foreseeable future. In terms of programmability of these processors what is different from legacy multiprocessors is that sharing among the different cores (processors) is less expensive than it was in the past. Previous research suggested that sharing is a desirable feature to be incorporated in new codes. For some programs, more cache leads to more beneficial sharing since the sharing starts to kick in for large on chip caches. This work tries to answer the question of whether or not we can (should) write code differently when the underlying chip microarchitecture is powered by a Chip Multiprocessor. We use a set three graph benchmarks each with three different input problems varying in size and connectivity to characterize the importance of how we partition the problem space among cores and how that partitioning can happen at multiple levels of the cache leading to better performance because of good utilization of the caches at the lowest level and because of the increased sharing of data items

that can be boosted at the shared cache level (L2 in our case) which can effectively be a prefetching effect among different compute cores.

The thesis has two thrusts. The first is exploring the design space represented by different parallelization schemes (we devise some tweaks on top of existing techniques) and different graph partitionings (a locality optimization techniques suited for graph problems). The combination of the parallelization strategy and graph partitioning provides a large and complex space that we characterize using detailed simulation results to see how much gain we can obtain over a baseline legacy parallelization technique with a partition sized to fit in the L1 cache. We show that the legacy parallelization is not the best alternative in most of the cases and other parallelization techniques perform better. Also, we show that there is a search problem to determine the partitioning size and in most of the cases the best partitioning size is smaller than the baseline partition size.

The second thrust of the thesis is exploring how we can predict the best combination of parallelization and partitioning that performs the best for any given benchmark under any given input data set. We use a PIN based reuse distance profile computation tool to build an execution time prediction model that can rank order the different combinations of parallelization strategies and partitioning sizes. We report the amount of gain that we can capture using the PIN prediction relative to what detailed simulation results deem the best under a given benchmark and input size. In some cases the prediction is 100% accurate and in some other cases the prediction projects worse performance than the baseline case. We report the difference between the simulation best performing combination and the PIN predicted

ones as well as other statistics to evaluate how good the predictions are. We show that the PIN prediction method performs very well in predicting the partition size compared to predicting the parallelization strategy. In this case, the accuracy of the overall scheme can be highly improved if we only use the partitioning size predicted by the PIN prediction scheme and then we use a search strategy to find the best parallelization strategy for that partition size.

In this thesis, we use a detailed performance model to scan a large solution space for the best parameters for locality optimization of a set of graph problems. Using the $M5$ performance simulation we show gains of up to 20% vs. a naively picked baseline case. Our prediction scheme can achieve up to 100% of the best performance gains obtained using a search method on real hardware or performance simulation without running at all on the target hardware and up to 48% on average across all of our benchmarks and input sizes.

There are several interesting aspects to this work. We are the first to devise and verify a performance model against detailed simulation results. We suggest and quantify that locality optimization and problem partitioning can increase sharing synergistically to achieve better performance overall. We have shown a new utilization for coherent reuse distance profiles as a helping tool for program developers and compilers to a optimize program's performance.

Locality Transformations and Prediction Techniques for Optimizing Multicore Memory Performance

by

Abdel-Hameed A. Badawy

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2013

Advisory Committee:
Professor Donald Yeung, Chair/Advisor
Professor Eyad H. Abed
Professor Manoj Franklin
Professor Gang Qu
Professor Amr M. Baz, Dean's Representative

© Copyright by
Abdel-Hameed A. Badawy
2013

Dedication

To my father and my new born baby girl, Fatima.

Acknowledgments

Peace be unto you all!

I owe deep gratitude first and foremost to God (Allah) the All-Mighty for helping me finish this endeavor. I owe also deep gratitude to many people with whom I interacted during my PhD journey at UMD.

First and foremost is my advisor Prof. Donald Yeung. I learnt a lot from Don. He has been to me more than an advisor. He stood for me and has shown support for me in many incidents far beyond what he has ever done for any of his students. I believe I am very lucky to have such an advisor. Don is a real researcher who is always looking for the next big thing in his research, always systematic in his approach to research. His involvement with his advisees is phenomenal. He trains his students well. I am really honored to have worked with him. I wish Don and his family the best for the future.

I would like to thank all the members of my committee (Prof. Amr Baz, Prof. Eyad Abed, Prof. Manoj Franklin, and Prof. Gang Qu) for accepting to serve. I am really indebted to them for their professionalism and willingness to serve — some of them on very short notice to replace other members who could not be present for the defense due to reasons beyond their control.

I have been entangled with the staff of the Center for Teaching Excellence (CTE) for some time. They are a wonderful group of people. I owe huge gratitude to Prof. Spencer Benson for being a mentor and for introducing me to the broad area of scholarship of teaching and learning. The work that I have done with CTE in

my mind amounts to another PhD even though I will not officially earn one for that work. I really appreciate the commitment and mentorship the people at CTE have shown over the years. Dr. Sabrina Kramer will always be a good friend who offers help editorial and otherwise when called upon. Cynthia Shaw is an indispensable help and resource at CTE. All of the Graduate Assistants of CTE were wonderful in their willingness to help, namely, Henrike, Alexis, and Emily. I have also to thank the CTE Graduate Lilly Fellows 2012 for being such a wonderful research team, working with such an elite group was a great and fulfilling honor that also produced several interesting publications.

I would like to thank all of my group, office, and lab mates, namely, Choi, Dongkeun, Gautham, Deepak, Sumit, Meng-Ju, Wanli, Xuanhua, Xu, Inseok, Mike, Jeff, Minshu, Aneesh, Aamer, Zahran, Kursad, Anashua, Brenda, Sadagopan, Khaled and all the rest of the members of the SCAL lab. They were a wonderful group of folks to have around in the lab to discuss things, talk to, and collaborate with.

I have a lot of gratitude for Meng-Ju since I used a lot of his tools and we have engaged in more discussions than with anyone else in the group. I really appreciate your help Meng-Ju and I wish you the best of luck in your life at Mathworks. I want to also acknowledge the author of the latex thesis template.

I have to acknowledge all of the graduate student body with whom I interacted with at UMD. Specifically, I would want to mention the Muslim, Arab, and Egyptian student body specifically in both ECE, CS and UMD at large. They have made life at UMD more socially appropriate and gave me and my family a sense of having family at UMD. It is not feasible to name all of you but I am sure each knows how

much I do appreciate having you guys as friends. Each knows that I have a place for them in my heart.

I have to mention a specific couple by name here. They are Hajj Ahmed Bakry and his wife Paula Ottinger. Ahmed has been like a father to me. He was even tough on me sometimes but fathers are like that sometimes. I am indebted to Paula for her always sincere and genuine advice and for being a good and dear friend.

I owe my parents and my extended family a lot of gratitude and I will never be able to pay them back no matter what I would do for them. They never doubted me. They always stood behind me. They always prayed for me and they always showed genuine concern for my prosperity and success. Once again I cannot mention everyone by name but they know who they are.

Last but not least, I have to mention my gratitude for my wife, Dr. Mona Elshinawy, for always standing by me, encouraging and supporting me at every step of the way. I owe you so much.

I owe my children Mohammad and Zaynab a lot of apologies for not being there sometimes. But I know for sure they will be very happy that their dad finally finished his PhD.

It is definitely impossible to name everybody but I pray that God always guides you all and I wish all those whom I owe morally and otherwise the best of luck in their lives and their careers and their afterlives.

And my final thanks has to go again to Allah. Thanks is fully and only to God for all the blessings that he has bestowed upon me.

Peace be unto you all!

Table of Contents

List of Tables viii

List of Figures xi

List of Abbreviations xiv

1 Introduction 1
- 1.1 Motivation . . . 1
  - 1.1.1 Single Chip Multiprocessor . . . 1
  - 1.1.2 Chip Multiprocessor Challenges . . . 3
  - 1.1.3 Chip Multiprocessor Limits . . . 5
  - 1.1.4 Chip Multiprocessor Memory Hierarchy . . . 7
  - 1.1.5 This Thesis . . . 9
- 1.2 Contributions . . . 11
- 1.3 Roadmap . . . 13

2 Related Work 15

3 Background and Methodology 27
- 3.1 Methodology . . . 27
  - 3.1.1 M5 Architecture Simulation Infrastructure . . . 27
  - 3.1.2 Simulation Parameters . . . 31
  - 3.1.3 Benchmarks . . . 33
- 3.2 Reuse Distance . . . 35
- 3.3 Multicore Reuse Distance . . . 36
  - 3.3.1 Concurrent Reuse Distance . . . 38
  - 3.3.2 Private-stack Reuse Distance . . . 38
  - 3.3.3 Cache Miss Count (CMC) . . . 39
  - 3.3.4 Pin-based Profiling Tool . . . 40

4 Techniques for Graph Problems:
Locality Optimizations & Parallelizations 43
- 4.1 Locality Optimizations . . . 43
- 4.2 Reordering for Indexed Accesses . . . 44
  - 4.2.1 Using Metis . . . 48
- 4.3 Parallelization Strategies . . . 49
  - 4.3.1 SMP Legacy Parallelization . . . 50
  - 4.3.2 CMP Parallelization . . . 53
  - 4.3.3 Variation on the CMP Parallelization . . . 53
    - 4.3.3.1 $CMP - P$ Parallelization . . . 54
    - 4.3.3.2 $CMP - K$ Parallelization . . . 54
  - 4.3.4 Discussion on the Parallelization Strategies . . . 55

5 Design Space Exploration using detailed M5 Experimental Results 57
5.1 Searching for the best Partition Size using the M5 Simulator 57
5.2 IRREG Results 60
5.2.1 IRREG with 144 input data set 60
5.2.2 IRREG with m14b input data set 61
5.2.3 IRREG with auto input data set 62
5.3 NBF Results 63
5.3.1 NBF with 144 input data set 64
5.3.2 NBF with m14b input data set 64
5.3.3 NBF with auto input data set 65
5.4 MOLDYN Results 67
5.4.1 MOLDYN with 144 input data set 67
5.4.2 MOLDYN with m14b input data set 69
5.4.3 MOLDYN with auto input data set 69
5.5 M5 Results Summary & Conclusions 70

6 PIN Reuse Distance Performance Model Prediction Results 102
6.1 PIN Reuse Distance Performance Model 102
6.1.1 Deriving the Performance Model 104
6.2 Results using the PIN Performance Model 105
6.2.1 IRREG Results 108
6.2.1.1 IRREG with 144 input data set 109
6.2.1.2 IRREG with m14b input data set 110
6.2.1.3 IRREG with auto input data set 111
6.2.2 NBF Results 113
6.2.2.1 NBF with 144 input data set 113
6.2.2.2 NBF with m14b input data set 115
6.2.2.3 NBF with auto input data set 118
6.2.3 MOLDYN Results 119
6.2.3.1 MOLDYN with 144 input data set 120
6.2.3.2 MOLDYN with m14b input data set 121
6.2.3.3 MOLDYN with auto input data set 122
6.3 Summary of the PIN Performance Model Results against M5 124
6.4 The “Three Run” Scheme: Partition Size Prediction Accuracy and Improvements on Parallelization Prediction 129

7 Conclusion and Future Work 163
7.1 Summary 163
7.2 Future Work 165

BIBLIOGRAPHY 168

List of Tables

3.1 Simulator parameters used in the performance experiments. 32
3.2 Total data footprint in MB for each benchmark for each Input size. . 33
3.3 Benchmarks and Input summary. 34

4.1 Partition sizes for the 144, m14b, and auto graphs for the IRREG benchmark. 50
4.2 Partition sizes for the input data sets of IRREG for different L1 sizes. 50
4.3 Partition sizes for the 144, m14b, and auto graphs for the NBF benchmark. 51
4.4 Partition sizes for the input data sets of NBF for different L1 sizes. . 51
4.5 Partition sizes for the 144, m14b, and auto graphs for the MOLDYN benchmark. 52
4.6 Partition sizes for the input data sets of MOLDYN for different L1 sizes. 52

5.1 Summary of the percentage gains relative to the baseline using the M5 simulator for IRREG for the **144** input data set with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches. 61
5.2 Summary of the percentage gains relative to the baseline using the M5 simulator for IRREG for the **m14b** input data set with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches. 62
5.3 Summary of the percentage gains relative to the baseline using the M5 simulator for IRREG for the **auto** input data set with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches. 63
5.4 Summary of the percentage gains relative to the baseline using the M5 simulator for NBF for the **144** input data set with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches. 65
5.5 Summary of the percentage gains relative to the baseline using the M5 simulator for NBF for the **m14b** input data set with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches. 66
5.6 Summary of the percentage gains relative to the baseline using the M5 simulator for NBF for the **auto** input data set with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches. 67
5.7 Summary of the percentage gains relative to the baseline using the M5 simulator for MOLDYN for the **144** input data set with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches. 68
5.8 Summary of the percentage gains relative to the baseline using the M5 simulator for MOLDYN for the **m14b** input data set with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches. 70
5.9 Summary of the percentage gains relative to the baseline using the M5 simulator for MOLDYN for the **auto** input data set with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches. 71

5.10 Summary of the percentage Gains relative to the baseline using the M5 simulator for IRREG for each input data sets with across all L1 cache sizes for L2 cache sizes ($128kB - 4MB$). 72
5.11 Summary of the percentage Gains relative to the baseline using the M5 simulator for NBF for each input data sets with across all L1 cache sizes for L2 cache sizes ($128kB - 4MB$). 73
5.12 Summary of the percentage Gains relative to the baseline using the M5 simulator for MOLDYN for each input data sets with across all L1 cache sizes for L2 cache sizes ($128kB - 4MB$). 73
5.13 Summary of the percentage Gains relative to the baseline using the M5 simulator for all benchmarks for each input data sets with across all L1 cache sizes for L2 cache sizes ($128kB - 4MB$). 74

6.1 A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for IRREG under the **144** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches. 110
6.2 A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for IRREG under the **m14b** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches. 112
6.3 A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for IRREG under the **auto** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches. 114
6.4 A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for NBF under the **144** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches. . 116
6.5 A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for NBF under the **m14b** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches. 117
6.6 A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for NBF under the **auto** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches. 119
6.7 A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for MOLDYN under the **144** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches. 121
6.8 A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for MOLDYN under the **m14b** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches. 123

6.9 A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for MOLDYN under the **auto** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches. 125
6.10 A summary of the percent gains relative to the baseline for M5 and the PIN performance model for IRREG for all input problems averaged over all simulated L1 caches for L2 caches $128kB - 4MB$. Note all numbers are in percent. 127
6.11 A summary of the percent gains relative to the baseline for M5 and the PIN performance model for NBF for all input problems averaged over all simulated L1 caches for L2 caches $128kB - 4MB$. Note all numbers are in percent. 127
6.12 A summary of the percent gains relative to the baseline for M5 and the PIN performance model for MOLDYN for all input problems averaged over all simulated L1 caches for L2 caches $128kB - 4MB$. Note all numbers are in percent. 128
6.13 A summary of the percent gains relative to the baseline for M5 and the PIN performance model averaged across ***all benchmarks***, for all input problems averaged over all simulated L1 caches for L2 caches $128kB - 4MB$. Note all numbers are in percent. 128
6.14 Summary of Partition Size prediction using PIN against the best selected by M5 for the IRREG benchmark with each of the three input data set under all L1 and L2 cache size selections. 131
6.15 Summary of Partition Size prediction using PIN against the best selected by M5 for the NBF benchmark with each of the three input data set under all L1 and L2 cache size selections. 159
6.16 Summary of Partition Size prediction using PIN against the best selected by M5 for the MOLDYN benchmark with each of the three input data set under all L1 and L2 cache size selections. 160
6.17 A summary of the Ranks obtained by PIN's top prediction for each benchmarks. 161
6.18 A summary of the ranks achieved using our "Run Three" optimization to enhance the accuracy of the prediction of the PIN model. The graph partitioning size remains unaltered but the ranking improves significantly as shown below. 162

List of Figures

1.1 Changes in the memory hierarchy from uniprocessor to various CMP designs. Adapted from [133]. 7

3.1 The simulated Tiled CMP architecture. 29
3.2 Chip Multiprocessors with multiple levels of private cache and a shared last level cache 36
3.3 Thread interleaving scheme. 42

4.1 Indexed array access pattern code fragment and data layout illustration. 46
4.2 An example for an index array locality optimization. 56

5.1 M5 results for IRREG under the **144** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 75
5.2 M5 results for IRREG under the **144** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 76
5.3 M5 results for IRREG under the **144** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 77
5.4 M5 results for IRREG under the **m14b** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 78
5.5 M5 results for IRREG under the **m14b** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 79
5.6 M5 results for IRREG under the **m14b** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 80
5.7 M5 results for IRREG under the **auto** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 81
5.8 M5 results for IRREG under the **auto** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 82
5.9 M5 results for IRREG under the **auto** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 83
5.10 M5 results for NBF under the **144** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 84
5.11 M5 results for NBF under the **144** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 85
5.12 M5 results for NBF under the **144** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 86
5.13 M5 results for NBF under the **m14b** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 87
5.14 M5 results for NBF under the **m14b** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 88
5.15 M5 results for NBF under the **m14b** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 89

5.16 M5 results for NBF under the **auto** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 90
5.17 M5 results for NBF under the **auto** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 91
5.18 M5 results for NBF under the **auto** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. . . . 92
5.19 M5 results for MOLDYN under the **144** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. . . . 93
5.20 M5 results for MOLDYN under the **144** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. . . . 94
5.21 M5 results for MOLDYN under the **144** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. . . . 95
5.22 M5 results for MOLDYN under the **m14b** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. . . . 96
5.23 M5 results for MOLDYN under the **m14b** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. . . . 97
5.24 M5 results for MOLDYN under the **m14b** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. 98
5.25 M5 results for MOLDYN under the **auto** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. . . . 99
5.26 M5 results for MOLDYN under the **auto** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. . . . 100
5.27 M5 results for MOLDYN under the **auto** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline. . . . 101

6.1 Ordered PIN Performance Model results for IRREG under the **144** problem with $4kB$ L1 caches and $128kB - 4MB$ L2 caches. 132
6.2 Ordered PIN Performance Model results for IRREG under the **144** problem with $8kB$ L1 caches and $128kB - 4MB$ L2 caches. 133
6.3 Ordered PIN Performance Model results for IRREG under the **144** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches. 134
6.4 Ordered PIN Performance Model results for IRREG under the **m14b** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches. 135
6.5 Ordered PIN Performance Model results for IRREG under the **m14b** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches. 136
6.6 Ordered PIN Performance Model results for IRREG under the **m14b** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches. 137
6.7 Ordered PIN Performance Model results for IRREG under the **auto** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches. 138
6.8 Ordered PIN Performance Model results for IRREG under the **auto** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches. 139
6.9 Ordered PIN Performance Model results for IRREG under the **auto** problem with $32kB$ L1 cache and $128kB - 4MB$ L2 caches. 140
6.10 Ordered PIN Performance Model results for NBF under the **144** problem with $4kB$ L1 caches and $128kB - 4MB$ L2 caches. 141

6.11 Ordered PIN Performance Model results for NBF under the **144** problem with $8kB$ L1 caches and $128kB - 4MB$ L2 caches. 142
6.12 Ordered PIN Performance Model results for NBF under the **144** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches. 143
6.13 Ordered PIN Performance Model results for NBF under the **m14b** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches. 144
6.14 Ordered PIN Performance Model results for NBF under the **m14b** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches. 145
6.15 Ordered PIN Performance Model results for NBF under the **m14b** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches. 146
6.16 Ordered PIN Performance Model results for NBF under the **auto** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches. 147
6.17 Ordered PIN Performance Model results for NBF under the **auto** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches. 148
6.18 Ordered PIN Performance Model results for NBF under the **auto** problem with $32kB$ L1 cache and $128kB - 4MB$ L2 caches. 149
6.19 Ordered PIN Performance Model results for MOLDYN under the **144** problem with $4kB$ L1 caches and $128kB - 4MB$ L2 caches. 150
6.20 Ordered PIN Performance Model results for MOLDYN under the **144** problem with $8kB$ L1 caches and $128kB - 4MB$ L2 caches. 151
6.21 Ordered PIN Performance Model results for MOLDYN under the **144** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches. 152
6.22 Ordered PIN Performance Model results for MOLDYN under the **m14b** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches. 153
6.23 Ordered PIN Performance Model results for MOLDYN under the **m14b** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches. 154
6.24 Ordered PIN Performance Model results for MOLDYN under the **m14b** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches. 155
6.25 Ordered PIN Performance Model results for MOLDYN under the **auto** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches. 156
6.26 Ordered PIN Performance Model results for MOLDYN under the **auto** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches. 157
6.27 Ordered PIN Performance Model results for MOLDYN under the **auto** problem with $32kB$ L1 cache and $128kB - 4MB$ L2 caches. 158

List of Abbreviations

CMP	Chip Multi-Processor
CMPs	Chip Multi-Processors
SMP	Symmetric multiprocessor
SMPs	Symmetric multiprocessors
RD	Reuse Distance
MRD	Multicore Reuse Distance
CRD	Concurrent Reuse Distance
PRD	Private Reuse Distance
sPRD	Scaled Private Reuse Distance
CMC	Cache Miss Count
SIMD	Single Instruction, Multiple Data
VLIW	Very Long Instruction Word
DSP	Digital Signal Processor
DRAM	Dynamic Random Access Memory
P	Number of Cores in the System
C	Cache Size
CMT	Chip MultiThreading
LRU	Least Recently Used
LLC	Last-Level cache
CPI	Cycles Per Instruction
IPC	Instructions Per Cycle
BW	Bandwidth
PDE	Partial Differential Equation
CHARMM	Chemistry at HARvard Macromolecular Mechanics
GROMOS	GROningen MOlecular Simulation
NBF	Non Bonded Force Kernel
ETH	The Swiss Federal Institute of Technology, Zurich, Switzerland
Inf	Infinity
METIS	Graph Partitioning Library
MPKI	Misses Per Kilo Instruction
GPART	Graph Partitioning Techniques
RCB	Recursive Coordinate Bisection

Chapter 1

Introduction

In this chapter, we introduce and motivate the research problem addressed in this thesis. We first discuss the prevalence of CMPs. Then, we discuss the challenges facing CMPs. We also briefly discuss CMP memory hierarchies. Then, we introduce the research problem addressed in this thesis. We end this chapter by articulating the contributions of the thesis. Finally, we outline a roadmap for the rest of this thesis.

1.1 Motivation

1.1.1 Single Chip Multiprocessor

A multi-core processor (also known as a Multiprocessor Microprocessor or a single-chip multiprocessor) [16, 72, 128, 134–137] is a single computing component with at least two independent actual central processing units ("cores"), which are the units that execute a program's instructions. Manufacturers integrate the cores onto a single die (a chip multiprocessor or "CMP"), or onto multiple dies packaged together in a single package. The hydra project is one of the pioneers in this idea [72, 128, 134–137].

Chip Multiprocessors, or CMPs, have been advocated since the early 90s by the Hydra group [72, 127, 128, 134–137]and others [16]. Chip-multiprocessing is

clearly "the solution" to the unavoidable hit of diminishing returns [5, 6, 71, 151] on superscalar uniprocessors.

Processors were originally developed with only one core. Cores of a CMP might share caches, and inter-core communication can happen either using message passing, shared memory, or a hybrid of the two. Network topologies to interconnect cores can be buses, rings, two-dimensional meshes, crossbars, *etc.* Multi-core systems may include identical cores (homogenous CMP), or non-identical (non-homogenous CMP). Cores in a multi-core system can implement any architecture. Some are superscalars, VLIWs, vector processing, SIMD, multithreading, and often times a combination of these. CMPs are widely used across a variety of application domains including, but not limited to, general-purpose, embedded, network, digital signal processing (DSP), graphics, medical, mobile, gaming, *etc*.

The industry has also agreed with academia on this particular solution. Intel has a large lineup of Chip Multiprocessor products in all markets from the Server to the desktop to the mobile and embedded markets and their commitment to CMPs in the foreseeable future is undeniable [31, 85, 90]. Other chips from vendors like IBM's Cell Processors [48, 69], Power7+ [131], Sun's Niagra [101] and Sparc64 series [67], Tilera's TILE-Gx72 [2, 87, 171], Qualcomm's Snapdragon processors [143], and ARM's Cortex processors [8] are ubiquitous in real hardware market applications with several other processors in research and development.

The predictions of the Hydra project and others are now in almost all the electronics that we use in our daily lives. This fact highly underscores the importance of research that addresses the problems that are facing or expected to face CMPs

in the near, medium, and distant futures.

1.1.2 Chip Multiprocessor Challenges

Unless the programmer is implementing clever software algorithms and techniques that cover most of the code base allowing it to run in parallel for most of the execution time, no significant gains can be achieved. The fraction of the software that cannot be run in parallel simultaneously on multiple cores can dominate the execution time and thus diminish the utility of a CMP. A best case scenario would be with embarrassingly parallel problems which can achieve gains near the number of cores. Larger than the number of cores gain can be achieved if the problem fits within the collective caches of the cores, avoiding use of main system memory. More commonly however, applications will not achieve such gains without the programmer investing significant amounts of time and effort in re-organizing the whole program to avoid pitfalls and squeeze as much performance as is possible. The successful parallelization of software for performance is a significant ongoing topic of research.

One of the goals of programmers and compilers is to try to optimize programs' use of the caches. This requires understanding the tradeoffs existing in the underlying hardware that will run the program specifically and a general understanding of the underpinnings of existing caches in current CMPs. In many of the contemporary and current CMPs, there is some private cache and some shared caches. The private caches are closer to the execution units, normally at level-1 and sometimes level-2 as well. Below these two levels, there is another level consisting of a much

larger shared cache. There are tradeoffs in designing caches in terms of the number of levels and the size of each cache level. Some other CMPs have private caches in all of the levels but such architectures are not as common as the mixed private and shared CMPs.

There are many questions that need to be answered for CMPs. One of these questions is the type of coherence and consistency protocols for CMPs as well as their scalability. Scalability for CMPs in general is a daunting question for all aspects of CMP microarchitecture. Research on novel techniques for coherence and consistency [170] such as transactions [7, 73, 117, 146], token coherence [113], or memory models [35]. These research projects are solving the coherence and consistency problem among other questions for the new paradigm of processors [78, 192].

Research on power and thermal management of CMPs [46, 79, 88, 106, 107, 123] is also another big area of interest and unanswered questions. Power efficiency is a daunting problem that caused the cancelation of some processors. Thermal and energy efficiency is a significant area of research. In general, research on exploring the design space of CMP [53] and the caches – especially the Last Level cache, its organization and issues – are also important areas with significant on-going research [15, 45, 50, 86, 162, 196].

Many exciting areas of research are arising nowadays due to the issues designers are facing with very small feature sizes and smaller transistors with each technology generation. One major issue is power as indicated above but also soft errors [125, 200] and reliability [14, 63, 138, 164].

1.1.3 Chip Multiprocessor Limits

Academia and industry [17, 31, 101, 133] expect in the next decade or so as many as tens or even hundreds of cores per chip. For example, Tilera has announced TILE-Gx72 with 72 advanced 64-bit cores [171]. For these large multicore CPUs, a significant portion of the chip die-area will be devoted to the memory hierarchy. Managing this on-chip memory will be extremely important to achieving high performance and power efficiency.

CMPs have more limits compared to uniprocessor. The power wall is a major hurdle that can prevent scaling. The memory wall for CMPs is not far away either. It takes hundreds of cycles to access off-chip memory. With more cores on chip, the rate of requests for cache blocks will go up as the number of cores increases. This will lead to even more contention at the off-chip buses and pins. Currently, the average access latency for off-chip DRAMs is hitting 200–300 cycles per access. Minimizing the number of off-chip accesses is crucial in order to reduce the demand on pin-bandwidth [62, 65, 165, 191, 199]. This means that careful optimization of off-chip bandwidth usage is very important to get the most out of the scarce resource for the system which is the off-chip bandwidth.

Even though performance is an important issue that we address directly in this thesis but for CMPs power efficiency is a crucial issue that might be even more important than performance. It is beyond this thesis to go into details about the power efficiency problems and their solutions even though there is a myriad of significant research on the topic. But we indirectly affect energy consumption and

usage of hardware which affects power efficiency. Locality optimizations work first and foremost on enhancing performance of the benchmarks running on the specific hardware but they also improve the utilization of the hardware itself and can enhance power efficiency. They enhancement of the power efficiency stem from the fact that the main premise of graph partitioning – which is the locality optimization technique we are using this thesis – is that it makes all the uses of the graph nodes and edges happen close in time by the cores accessing it. Furthermore, we utilize techniques that enhance sharing and reduce the working set of the running benchmark by making the cores work on a smaller part of the problem all together. The overhead that is introduced by locality optimizations when spread over a large number of iterations would be negligible compared to the gain achieved. Also, the better usage of the data to the caches makes the dynamic energy consumed at the caches smaller but it would not decrease the leakage component. Thus, Locality optimization work on both performance and conserving energy unlike other techniques such as prefetching which would increase energy expenditure due to the extra instructions in case of software prefetching or extra hardware in case of hardware prefetching *i.e.* extra silicon area for the hardware prefetcher. We have to note also that not all prefetches would be useful or arrive on time. On the contrary, locality optimizations data are rearranged and are computed upon after it has been adjusted to improve temporal and spatial locality. Furthermore, the techniques we adopt in this thesis to increase sharing would also improve both inter-processor locality and intra-processor locality.

1.1.4 Chip Multiprocessor Memory Hierarchy

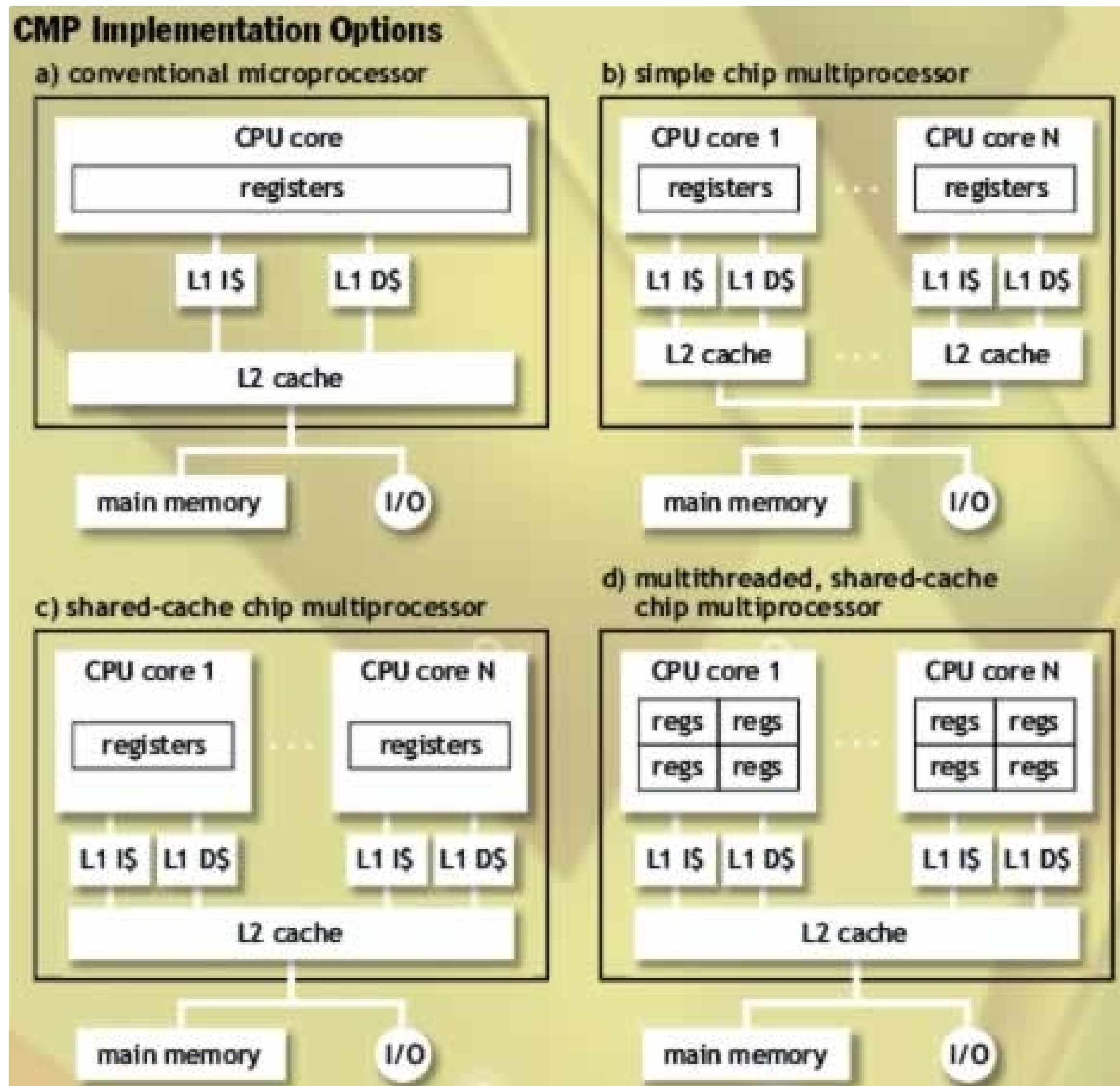


Figure 1.1: Changes in the memory hierarchy from uniprocessor to various CMP designs. Adapted from [133].

Figure 1.1 shows the changes to the memory hierarchy of different CMP designs compared to a uniprocessor memory hierarchy. In a uniprocessor, the CPU's caches are dedicated to a single processing core. In a CMP, depending on the design used, each core might see a different image of the memory hierarchy from what it would see in a uniprocessor chip. If each core in the CMP chip has its own L2 cache as

in Figure 1.1(b), this is called a simple CMP. This is in some sense taking several conventional microprocessors and putting them together on one single chip without any sharing of resources. Figure 1.1(c) is an implementation where the L2 caches of all the cores in Figure 1.1(b) are merged together to form a unified shared L2 cache. This is a fundamental change to the memory hierarchy compared to Figure 1.1(b). Now, the sharing point – defined as the point where the cores are connected together and data can flow from a core to another – has moved to become on chip which gives new opportunities for memory performance optimization. Communication costs for shared data among cores is cheaper. Bandwidth on chip is very high compared to the pin-bandwidth limitations. Figure 1.1(d) shows another alternative to CMP implementation. This implementation employs SMT cores instead of simple cores as in Figure 1.1(c) and (b), and is similar to SUN's Niagara [101] (SUN coined the term CMT "Chip MultiThreading" [163] for this implementation). In this thesis, implementation (c) is the implementation of choice for its simplicity and representation of a good base case. Yet, the work in this thesis can be applied with no to minimal change to the rest of the CMP implementations in Figure 1.1.

Private caches across all of the hierarchy allow the same data to be replicated in multiple caches, whereas a shared cache keeps a single copy of data in cache. Private versus shared caches provide two extreme cases of the design spectrum. If cache is shared at all levels, then a data item will have only one instance (copy) in each cache level of the shared cache. This makes for the best utilization of the cache space since there is no replication whatsoever in such a cache. However, shared caches exhibit higher access latency compared to private caches since they cannot

keep all the shared data close to every core. On the other hand, if we look at private caches at all cache levels, access latency will be better, whereas replication of data that is contemporaneously used by different cores will make cache space utilization not as good as the shared cache case. As we explained already, these are the two extremes of the spectrum. Contemporary CMPs exploit the benefits of both by using a hybrid system. The system has private caches at levels closer to the cores and shared caches at levels closer to the external DRAM system as is shown in Figure 1.1. This hybrid system which results in better performance and power efficiency makes optimizing locality for the caches a more complex problem.

1.1.5 This Thesis

Due to the importance of the memory hierarchy to CMP architecture, and the increasing scarcity of off-chip bandwidth, proper management of on-chip resources in CMPs will become an important research area over the next several years. Specifically, optimizing locality of access for on-chip memories in CMPs will become critical.

Although locality has been heavily researched in the past, it needs to be revisited at a fundamental level since the underlying architecture of CMPs compared to both uniprocessors and traditional multiprocessor machines is significantly different.

We have published on locality optimizations and software prefetching in the context of uniprocessors [12, 13]. One major conclusion of that work is that locality optimizations are a fundamental optimization that cannot be ignored, especially

given growing memory latencies and the scarcity of the off-chip bandwidth.

We believe that CMPs will benefit from locality optimizations even more than uniprocessors. Therefore, we have began looking at the kernels and benchmarks studied in our previous work in the context of CMPs. One fundamental difference that we think needs to be reconsidered in the context of locality optimizations for CMPs is the fact that sharing, which was normally looked at as having a negative effect on performance in the context of traditional multiprocessors can actually have a positive impact in CMPs due to the low cost of communication among cores on-chip.

The idea of promoting more sharing on-chip in CMPs has been suggested in many research works [45, 82, 91]. The whole premise is that data that can be accessed or computed upon are brought in the caches by one core and then used within a short time period will have a prefetching effect that will cut the latency of getting the data to the cores that access the data after its first access. The trade-off here is in bringing data all the way from memory to the L1 cache as opposed to bringing data from a shared cache or from another core's private cache depending on the protocol and the access type of the data.

In the proposal that preceded this thesis, we proposed a sharing scheme to boost sharing. The idea was simply to divide the problem space to P parts (where P is the number of cores in the processor) and then each part of the P parts is to be worked upon using all P cores. This idea was published by Shen and his students at The College of William and Mary [193–195].

We came to a conclusion after some experimental runs that for the benchmarks

that locality optimizations have a large leverage in achieving better performance. The above idea was easily beaten in performance by utilizing the proper cache locality optimizations techniques. We took a step further and combined both techniques. We did locality optimizations and then did the P partitions where all cores work on one of these while there is locality optimizations on top of it.

1.2 Contributions

Our work is focused on graph problems where the locality optimizations are done through graph partitioning. We use Metis [97] as the partitioning tool. We thus show better results than what is suggested by Shen and his students by using locality optimization (graph partitioning). We use a detailed M5 simulator to find the best optimization by searching different number of partitions for different machines where we acheive gains of up to 20%.

We use multicore Reuse distance profiles as in [181, 182, 184–187] to analyze the graph benchmarks we study in this thesis. We collect profiles using a tool based on Intel PIN [10, 111, 141]. We develop a performance model similar to the model developed in [184]. We use the model to predict the performance of machines upon which profiling was not performed. We are able to predict perfect performance gains that match the M5 performance or very close to the M5 performance numbers. On average, we were able to get 48% of the maximum performance gain across all the machines simulated which are machines with L1 caches with sizes of *4kB, 16kB, and 32kB* and L2 caches with sizes of *128kB, 256kB, 512kB, 1MB, 2MB, and 4MB*.

This is a total of 18 *machines*. Each machine is simulated with a workload of three benchmarks each with three different inputs. This accounts for *nine* simulations per machine. The total simulations verified is 162 *simulations* per parallelization strategy. We have four different parallelization strategies. This makes the total number of simulations reach 648 *simulations*. This number should be multiplied by the number of partitions we try for each input size which is around 12 different sizes, we will explain how to choose these sizes in a later chapter. Thus, the total number of simulation is 7776 *simulations*. The PIN performance model is verified against the M5 simulations results. The model also predicts with good accuracy the best or close to the best partitioning for each input data set for each benchmark.

Our work is unique in its use of a detailed performance simulation in combination with a reuse distance based performance model that is validated against the detailed simulation results. We use graph problems as a case study yet our methodology should be applicable to any optimizations that requires the compiler or the programmer to search for a specific value for the optimization among a large set of values.

We verified that sharing among cores can be beneficial if used properly and locality optimization in contrast to sharing optimizations alone is the clear winner. The best solution is to use both sharing optimizations with locality optimizations when possible. The graph problems are unique; these problem are not analyzable at compile time and there has to be analysis done after the input is known.

We have shown that using a rule of thumb computation for a baseline optimization according to the L1 cache size only in many cases does not score the

highest marks in terms of performance. We have shown that we can see gains above this baseline by up to 20% in some instances.

Using several modeling iterations based on the performance numbers we get from the performance simulation, we were able to come up with a model that can predict execution time using only information from the Cache Miss Count (CMC) of the CRD and PRD profiles and the instruction count of the program running.

Our reuse distance performance model is unique in how it is derived since we have used the simulation results to guide its creation. This makes the model a bit better than the model(s) in the literature.

We can predict the best locality optimization, parallelization method, and their combination. The accuracy for predicting partition sizes which is the locality optimization. The accuracy of choosing the parallelization is not quite as accurate though. This accuracy can be pushed higher if the programmer can try two to three choices suggested by the model which would raise the accuracy.

Our work, to our knowledge, is the first work that verifies against performance simulations the utility of multicore reuse distance performance models for deriving optimizations like how reuse distance profiles were used in the past.

1.3 Roadmap

In this Chapter, we have motivated the importance of CMPs, the challenges and the limitations that CMPs face, and the memory hierarchy of CMPs. We also articulated the contributions of the thesis.

The rest of this thesis is organized as follows: Chapter 2 discusses the related work in detail. Chapter 3 discusses the necessary background and methodology used for the work such as multicore reuse distance, the simulation infrastructure for both M5 and PIN. We also discuss the simulation parameters used in our experiments and we end the chapter with an introduction to the set of benchmarks and the input data sets used. Chapter 4 discusses the techniques used in the thesis. We also discuss the parallelization techniques for the benchmarks as well as the locality optimization techniques for them as well. In Chapter 5, we present the design space exploration results conducted using the M5 simulator. In Chapter 6, we discuss the multicore reuse distance performance modeling results. We will show the performance model we have developed on top of the multicore reuse distance profiles and we compare its results in choosing the parallelization technique and the locality optimization. In Chapter 7, we summarize and conclude the thesis.

Chapter 2

Related Work

In this chapter, we discuss the related work. Our work spans several areas therefore, we cover several areas of related work. We will discuss locality optimizations in the context of multiprocessors and uniprocessors as well as locality optimizations for different access patterns and for graph (indexed array) benchmarks. We also discuss works that promote sharing in the context of CMPs. We discuss compile time and run time techniques introduced to capture and foster more sharing among cores. We also cover related work that use multicore reuse distance profiles. And we end with a discussion of work that introduced models for performance prediction. Finally, we show what our model builds upon and outline the drawbacks of the RD profile based models.

The literature is split into two categories of locality optimizations. First, there is the legacy multiprocessor locality optimizations and the uniprocessor locality work. The major multiprocessor locality work is Monica Lam's work in the early $90s$ [4, 56, 119, 121]. Although that work was ground breaking, the underlying architectural assumptions makes it different compared to our work. The most significant difference is the role of sharing. In Lam's work, sharing was avoided due to the high cost of inter-processor communication in conventional multiprocessors. In contrast, we seek to increase sharing if it can reduce the number of off-chip accesses

in a CMP.

Second, there is the uniprocessor locality optimizations research. Researchers have attacked the memory bottleneck problem by improving data locality. Computation reordering transformations such as loop permutation and tiling are the primary optimization techniques [180]. Loop fission (distribution) and loop fusion have also been found to be helpful [118].

Data layout optimizations such as padding and transposing have been shown to be useful in eliminating conflict misses and improving spatial locality [148]. Several cache miss estimation techniques have been proposed to help guide data locality optimizations [68, 180]. Tiling has been proven useful for linear algebra codes [52, 104, 180] and multiple loop nests across time-step loops [161]. Rivera *et al.* in [149, 150] studied existing tiling techniques and extended them to tile 3D scientific computations. We have used Rivera's technique for tiling 3D stencil codes in our previous work [11–13]. We have studied how locality optimization and software prefetching compare to each other and how they can be combined for maximum performance gain rather than just naively and blindly combining them.

Researchers have examined irregular computations mostly in the context of parallel computing, using run-time [54, 74, 75] and compiler [110] support to support accesses on message-passing multiprocessors. Many have also looked at techniques for improving locality [4, 56, 119, 121].

Metrics such as reference affinity have been used to guide algorithms for splitting data structures and regrouping arrays to improve spatial locality [203]. Strout *et al.* have examined models for determining which combination of run-time data

and iteration reordering heuristics will result in the best locality for a given data set [166, 167]. Once again, our take on this issue is different in the sense that the technology point is different, the underlying architectural assumptions are also different, and most importantly, communication costs and the shared on-chip L2 cache we are assuming here would make the optimizations different. A partitioner like Metis [97] or GPART [77] can be modified to take into account the fact that sharing is not harmful anymore like it was once before in conventional legacy multiprocessors.

There has been research efforts trying to parallelize and optimize graph partitioners. Karypis *et al.*, the authors of Metis, have an MPI implementation of Metis called ParMetis [98] which has shown non-linear speed up over Metis's partitioning time but the quality ParMetis partitions is below those of the serial Metis. Karypis *et al.* have developed algorithms that can address the partitioning requirements of scientific computations, and can correctly model the architectural characteristics of emerging hardware platforms [124].

Chan *et al.* [43] propose a hierarchical framework for resource-aware graph partitioning on heterogeneous multi-core clusters. Their evaluation demonstrates the potential of the framework and motivates directions for incorporating application requirements into graph partitioning. Very recently, Karypis *et al.* have published a new OpenMP based implementation of the Metis algorithms and compared it against MPI based partitioners on three different multi-core architectures. The multi-threaded implementation achieves greater than a factor of two speedup over the other partitioners and also uses significantly less memory [105].

Our work benefits from the optimized, parallelized, and fine tuned graph par-

titioner discussed above. Parallel partitioners provide speedup over the existing serial or parallel MPI partitioners with less memory requirements. MPI ParMetis provides lower quality partitions compared to the regular serial Metis. We are yet to compare the quality of the partitions of the OpenMP partitions compared with MPI partitions or serial partitions. Effectively this will make the overhead cost of using the partitioners smaller, thus the number of iterations needed to acheive break-even will be much smaller than that of regular serial Metis.

Some researchers have investigated data layout transformations for pointer-based data structures [172, 198]. Chilimbi *et al.* investigated allocation time and run time techniques to improve locality for linked lists and trees [49]. Further extensions were proposed in [11–13].

Recent dynamic memory allocators [66] have been devised to improve dynamically allocated data locality at both the cache level and the page level more than Linux and FreeBSD allocators do.

Calder *et al.* use profiling to guide layout of global and stack variables to avoid conflicts [36]. Carlisle *et al.* investigate parallel performance of pointer-based codes in the Olden [41].

Carter *et al.* have studied hierarchical tiling [122]. They proposed a hierarchical tiling algorithm spanning multiple levels of the memory system. We are influenced by the main idea, and we are suggesting transformations that would make use of the locality that can be exploited at the shared L2 cache and at the L1 caches of each core. Carter *et al.* implemented a compiler framework for the techniques. We are performing optimizations by hand but a compiler framework to automate

the process given that the input data is known is planned for future work.

We are proposing two techniques based on the idea of hierarchical locality optimizations. Our proposition for changes in the order the computation is happening is most related to Kandemir*et al.* in [47, 93, 95] and in their compile time technique that boasts sharing among cores by a search technique.

Kandemir *et al.* [93] propose and evaluate a compiler-based solution to optimize inter-processor data locality on embedded chip multiprocessors. This work is primarily focused on data shared across processors on the same chip. This is where our proposed work intersects with theirs. They re-organize loop iterations assigned to processors in a coordinated fashion so that the reuse distance to shared data is minimized. We are optimizing the shared data in the shared L2 cache and also the L1 cache as opposed to their work. We are also looking at non-stencil codes whereas their work was concerned with stencil codes only. We are using graph partitioners to maximize reuse for graph problems whereas they do not optimize for the maximum reuse at the L1 caches nor the L2 caches. They have achieved a gain of 15% for the case of a CMP with four cores.

In [94, 95], they propose a compiler algorithm that solely rearranges the iterations and assigns them to cores according to how much sharing can be sustained among the cores while doing the iterations close by in time. One major difference between our work and theirs is that our benchmarks cannot be analyzed at compile time. The analysis needs to happen at run-time. Our work and their will introduce overhead. Our overhead lies in the analysis time and the cost of partitioning and the instructions added, while their overhead is in significant increase in instruction

count and normally compile-time cost is considered a drawback among compiler experts. Both of us justify the overhead with the gain running the benchmarks long enough to amortize the overhead on many iterations.

There is related work on the idea of adapting software to shared resources. The first work that we know of that tried this idea is the SMT compiler optimizations [108, 109] work. SMT processors [173–175] have some shared resources on the processor core, while some other resources are replicated *i.e.* are private. The Register file and the program counter are replicated while the functional units, the fetch hardware, the reorder buffer and cache ports are all shared. Normally, the L1 cache is also shared.

The SMT-Compiler [108, 109] assumed a shared L1 cache and up to an 8-way SMT. They looked at the sharing of a single tile of a tiled affine-array benchmark that was chosen to fit in the L1 cache on the SMT processor. The different SMT contexts would cooperate to compute that tile of computation together. The paper did not outline the algorithm chosen to pick the tile sizes although they varied the tile sizes and they concluded that SMT is tile size insensitive.

We are looking at a relatively similar problem but the underlying architecture is different; we have a CMP not an SMT. Also, the sharing point is at the L2 cache level and not the L1 cache as in the SMT case. The problem is different due to the cache sizes. Moreover, we are looking at a different access pattern and program types than just only affine arrays where only tiling is applicable.

There is other work [129, 130] that looked in particular at an Intel Xeon® hyperthreaded multiprocessor machine running Linux. This work tried to customize

tiling optimizations at the L1-cache level only in a way that would be aware of the sharing or even detect sharing at run time. Sharing was detected at run-time using either OS status bits to check if two processes are co-scheduled on the same physical processor *i.e.* each on one way of a Xeon SMT. They also discussed the optimizations while there is sharing by having conditional code depending on whether there is multiple threads assigned to the same core or not. If there is sharing, *i.e.* multiple threads scheduled on the same core, then they tile for the L1 Cache Size divided by the number of SMT ways per core ((L1 cache size)/(No. of SMT ways per core)) to avoid conflicts induced by the fact that each context assumes it has exclusive ownership of the entire L1 cache. In our work, we search for the best partition size and we consider partitionings that would fit in a smaller cache size than the L1 cache and we did find significant performance improvement (in some cases up to 20%) than the baseline partition that is sized to fit in the L1 cache.

Mattson *et al.* [115] were the first to introduce reuse distance analysis. Researchers have applied RD analysis to study and predict cache performance, modeling program locality, providing hints during code generation, and helping in structuring code and data to improve locality for uniprocessors [21–26, 59, 201–203].

In recent years, there has been a lot of interest in multicore reuse distance. It has become a very interesting tool to study program performance and predict performance of future systems and to guide architecture exploration efforts [181–184, 186, 187].

Suh *et al.* [168] have developed a model capturing the reuse distance dilation effect *i.e.* the effect that the reuse distance of a memory reference is increased by

memory reference accesses from other threads.

Chandra *et al.* [44] have looked at thread co-scheduling and the impact of cache sharing on it. They calculated the relative execution speed between threads and computed how many unique memory blocks from the second thread should be inserted into the reuse distance profile of the co-running thread. The benchmarks used were multi-programmed workloads.

Ding and Chilimbi [58] extended Chandra's techniques and present techniques to construct CRD profiles for multi-threaded programs. CRD profiles are constructed using memory traces to extract statistics on per-thread locality and data sharing. Their approach can handle non-symmetric threads unlike the PINtool we are using in this work. The algorithm incurs exponential time due to investigating combinatorial interleaving possibilities of the memory references.

Ding and Chilimbi [57] developed a measurement of the footprint of concurrently executing applications. Xiang *et al.* [189, 190] followed in the footsteps of Ding and Chilimbi and proposed an efficient model using the footprint of each program in order to predict its cache interference with other programs. The work is limited to multi-programmed workloads only.

Shi *et al.* [160] used a simulated stack to study the performance of multiple cache organizations in one run. They used a shared stack among all cores and a private per core stack to compute the reuse distance profiles. They organize shared and private stacks as groups similar to [100], and these groups can represent various cache sizes.

Schuff *et al.* [157] investigated the accuracy of RD analysis for multicore

processors. They predicted the miss-rate of shared and private caches. They used multicore RD analysis and communication analysis to model memory systems [103]. Both Shi *et al.* [160] and Schuff *et al.* [103, 157] predict cache performance at different cache sizes.

Jiang *et al.* [92] proposed a probabilistic model for deriving CRD profiles using traces of individual threads. The model requires knowing traces of each individual thread.

Our work uses the framework developed by Wu *et al.* [181–184, 186, 187]. Their framework can analyze any parallel program provided that it is a loop-based parallel program. They have studied design space exploration, cache hierarchy design, and both strong and weak scaling. Their work lacks cross validation of the relative performance among benchmarks like the what we do in the performance model we use in this thesis. They cannot do such a validation since their design space is huge and some of it cannot be simulate in a reasonable time frame. One cannot easily simulate significant workloads running on a simulator of a 1000 or more cores. The advantage of their approach is that they can explore design spaces too large to simulate. They fully validated the profiles themselves by cross checking profiles obtained from the M5 simulator and the PIN tool. They also cross validated MPKIs obtained from M5 and the PIN tool. In [186], they did some validation against M5 simulations.

Hill and Marty [84] built a CMP cost model using Amdahl's law [5, 6, 71, 151]. The cost model was then utilized to study the speedups for symmetric, asymmetric, and dynamic – cores that change the architecture depending on the benchmark

running – CMPs. They did not consider the impact of cache organizations.

Rogers *et al.* [152] developed an analytical model in order to study the memory bandwidth wall problem for CMP systems. They found that the memory bandwidth wall problem can severely limit core scaling.

Ho *et al.* [132] studied the trade-offs between cache organization, capacity, and core counts. They employed the square root rule to model the cache miss rate. They estimated the trade-offs among private, shared, and hybrid cache organizations. They found that different cache organizations have different optimal core counts and cache capacities. They predicted that shared caches can sustain more cores than private caches at its peak performance point.

Wentzlaff *et al.* [179] developed an IPC model at the system level. They used it to evaluate a large range of core counts and cache configurations. They assumed a workload running independent SPEC Int 2000 programs on each core in an effort to model cloud computing applications. They did not consider sharing among threads. They found that the increased area provided by technology advances is better used for cache due to the off-chip bandwidth constraints. They suggested that embedded DRAM should be used as L2 caches.

Esmaeilzadeh *et al.* [64] modeled multicore scaling and showed that multicore systems are doomed by power first and foremost. They predicted that regardless of the microarchitecture, the cache organization, or any other design parameter, multicore scaling will be limited by the power budget. They estimated that 21% of the die area of a fixed-size chip must be powered off at 22nm, and this number grows to more than 50% at 8nm.

Sun *et al.* [169] developed a model to optimize, under power constraints, the cache hierarchy. They applied the 2-to-$\sqrt{2}$ rule [83] which suggests that if the cache size is doubled, the miss rate drops by a factor of $\sqrt{2}$.

Krishna *et al.* [102] extended the model in [179] taking into account sharing for multithreaded benchmarks. They found that data sharing can improve system throughput significantly compared to applications with no data sharing but the benefits from data sharing is diminished with constraints on the off-chip bandwidth.

Multicore Reuse Distance profiles both shared and private (CRD and PRD) provide detailed memory behavior capturing effects due to interference and data sharing at varying cache capacities using one single run of the binary as long as the input did not change given the same assumed number of cores.

The performance model we use only needs the CRD, the sPRD, and the number of executed RISC instructions. The PINtool lacks the translation model for the CSIC instruction of the x86 architecture to x86 micro ops (the equivalent of RISC instruction). Therefore, we use the M5 Alpha instruction count since it is readily available and we have past experience that suggests that the number of micro ops follows the alpha instruction count.

Our model extended the model developed by Wu and Yeung in [181, 184]. The model developed there was not verified against a cycle accurate simulator, did not compute execution cycles for the program, and was used mainly for making architectural decisions for the cache hierarchy design in terms of private vs. shared and, given a fixed total die area budget for the entire cache, the breakdown of that area among the different cache levels.

The major drawback for models built using RD profiles is that the CRD and sPRD profiles are idealistic. They do not account for associativity, timing, and conflicts. Qasem and Kennedy have a cache conflict model [142]. We do not expect much changes to the results if we included it in the computation of the CRD and PRD profiles. After all, it cannot capture pathological conflicts.

Chapter 3

Background and Methodology

This chapter describes the necessary concepts, background, and simulation infrastructure used in this thesis. We will also discuss reuse distance (RD) analysis and the methodology used to acquire multicore reuse distance profiles. Section 3.1 introduces the graph benchmarks and their inputs. We present the M5 simulator that we use to collect the performance results. Section 3.2 introduces reuse distance including multicore reuse distance. Finally, we introduce the Intel Pin tool which we use to profile our set of benchmarks and we build our performance model using these collected profiles.

3.1 Methodology

In this section, we will provide a detailed description of the tools, simulators, and benchmarks that we use in this thesis. We introduce the M5 cycle-accurate performance simulator, and we end with a detailed description of the benchmarks and their different problem sizes.

3.1.1 M5 Architecture Simulation Infrastructure

In recent years, many simulator infrastructures have been developed for studying microprocessors. One of the most popular simulators is SimpleScalar [9, 34]

which modeled sequential machines. Unfortunately, there was no follow up multiprocessor simulator that was built on top of SimpleScalar. A large ensemble of multiprocessor simulators exist. RSIM is one of those simulators [89]. RSIM was not meant to be distributed to others and hence was not maintained and heavily used. But this has changed quite substantially in the past few years.

Many multiprocessor and Chip-Multiprocessor simulators are being developed, such as RASE [55], SimFlex [80, 178], GEMS [114], DRAMsim [177], M5 [29, 30], SESC [147], Graphite [120], and Gem5 [27]. GEMS is no longer under active development. The Gem5 simulator system combines both the M5 and Gems platforms. It is a fully open-source software. Graphite is another simulator that is gaining a lot of acceptance since one of its goals is running simulations across multiple machines and utilizing multiple cores that exist in contemporary processors.

Many of the simulators that are available publicly that have been used in the recent past were based on SIMICS [112] which is a commercial product that simulates many aspects of computer systems in great detail such that it can boot the full Solaris operating system.

M5 is from the University of Michigan [29, 30]. M5 is a full system simulator capable of booting Linux. Now, it has become integrated into Gem5. Our research group at the University of Maryland have modified and used the M5 simulator before the merger with Gems.

Originally, M5 has only supported a cache coherence protocol which was a bus-based snoop protocol. Lately, a point-to-point protocol became available. Our group was able to scale the number of processors simulated in the stand-alone (non-

full system mode) simulator beyond 128 processors. The point-to-point protocol allowed scaling beyond the bus based protocol limits.

The instruction set that is simulated is the alpha instruction set. Our research group has built an x86-Linux-based cross-compiler to build alpha binaries on a Linux based x86 box to run on the stand-alone simulator. Our group have extensively utilized M5 to run lots of benchmarks on it.

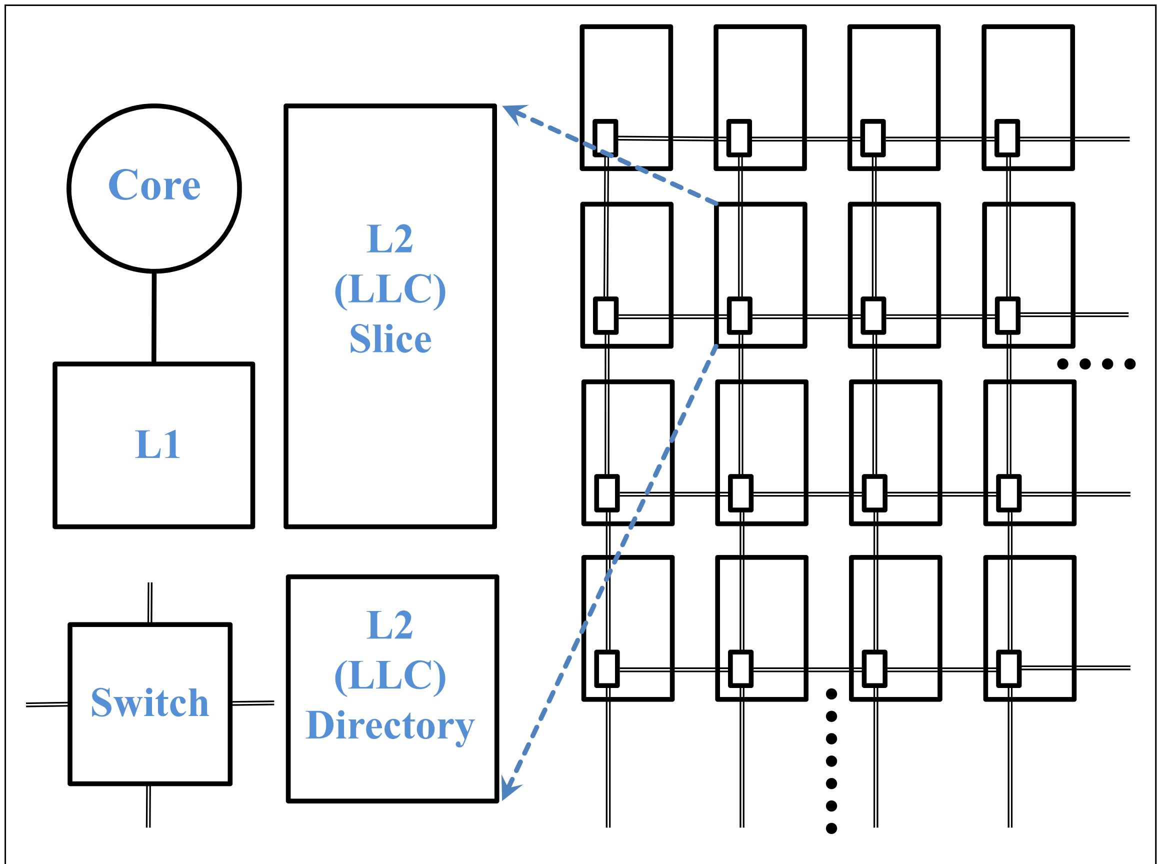


Figure 3.1: The simulated Tiled CMP architecture.

Many of our modifications to M5 enable simulation of tiled CMPs. A tiled CMP, illustrated in Figure 3.1, consists of several identical replicated tiles. A tile consists of a processor core, a multi-level cache hierarchy, and a switch for a 2D mesh network. The switch connects to its four neighbors (North, South, East and West).

When a node sends a request to a distant non-neighboring node, the request goes through multiple hops. Different configurations can be achieved easily by replicating a variable number of tiles. As a result, tiled CMPs are regarded as a scalable CMP organization [81, 197]. Tiled CMPs are an extensible, easily replicated and extended design that has been used in many architecture research and we think it is a representative architecture.

We use the M5 simulator [28] to model a tiled CMPs. Our simulator simulates in-order, single issue cores executing one instruction per cycle in the absence of memory stalls. Each core has its own private caches, and the last level of cache (LLC) is shared. We allow data to be replicated in each tile's private caches. Coherence across local caches is maintained via a directory-based MESI cache coherence protocol. In a directory-based system, data that is being shared is placed in a common directory that maintains coherence between caches. The directory acts as a filter through which the processor must ask permission to load an entry from the primary memory to its cache. When an entry is modified, the directory either updates or invalidates the other caches with that entry. We utilize the MESI protocol to maintain cache coherence. Scalable cache coherence protocols are being researched heavily in academia [61, 144, 145].

The last-level shared cache we are simulating is partitioned among tiles. Each tile has its own partition, called a slice. No migration or replication is allowed among LLC slices. Each cache block resides in a fixed LLC slice which is known as the cache block's home tile. Each cache block's directory entry is also co-located with its associated data. Cache block homes are page-interleaved using 8kB page

sizes across the LLC slices according to their physical address.

We assume each cache block's directory entry is co-located with its associated data, *i.e.* on the cache block's home tile. We also assume full-map directories, with each slice of the directory containing an entry for every cache block in its associated LLC slice. We also assume full-map directories. This can lead to large directories. Scalable directory schemes can reduce the directory size.

Old [3, 42, 70, 126] and new [1, 139, 153, 154] research on scalable directory schemes can reduce the directory size and make its implementation more scalable and feasible for large systems. Although the directory design is an important processor design decision investigating the directory design is beyond the scope of our research.

3.1.2 Simulation Parameters

Our simulator's core model is very simple, with each core executing one instruction per cycle (with no memory stalls) in program order. The simulator assumes a machine with an ideal CPI. Contrary to the processor model, the memory system model is very detailed. A detailed directory-based MESI cache coherence protocol and a two dimensional point-to-point mesh network are implemented.

The M5 simulator was also modified to support multiple DRAM channels, each connected to memory controllers on special memory tiles. The simulator accurately models cache access, hops through the network, and DRAM access. It also models queuing at the on-chip network and the memory controller(s).

All of our experiments simulate application code only without any operating

system code. The OS effects are emulated and it is not a full system simulation infrastructure. The simulator can record the directly-measured whole-program CRD and PRD profiles for the pre-L1 memory reference stream across all cores. CRD and PRD are computed at the granularity of 64 bytes, the block size for the caches.

Table 3.1: Simulator parameters used in the performance experiments.

Cores	4 core, Alpha ISA, CPI = 1, clock speed = 2GHz
IL1/DL1	4/8/32KB, 64B blocks, 4-way, 16 MSHRs, 1 CPU cycle
L2 Cache Size	128kB, 256kB, 512kB, 1MB, 2MB, 4MB
L2 Slice size	32kB, 64kB, 128kB, 256kB, 512kB, 1MB
L2 parameters	64B blocks, 8-way, 512 MSHRs, 16 Write Buffers, 10 CPU cycles
2-D Mesh	3 CPU cycles per-hop, bi-directional channels, 500Gbps, 2GHz
Memory	4GB, 200 cycles latency, 16GB/sec BW

Table 3.1 lists the parameters used in the M5 simulations. As Table 3.1 shows, we use a two-level cache hierarchy. Each core has its own private split L1 instruction and data caches. Also, the parameters and sizes of a physically distributed and logically unified L2 cache are also given. We simulate processors with four cores, 4kB – 32kB IL1/DL1 private cache and 128kB – 4MB Shared L2 cache.

The machine combinations we run include 18 different machine configurations for each optimization (12 optimizations) for each input data set (three data sets) for each benchmark (three benchmarks) for each parallelization strategy (four parallelizations strategies). The overall number of M5 simulations is $7,776$. This same number of simulation runs are accompanied by the same number of PIN tool runs to compute the CRD and PRD profiles for each benchmark, data set, parallelization, and optimization scheme. The overall number of simulations for both M5 and PIN is $15,552$ simulations.

3.1.3 Benchmarks

Table 3.2 lists the benchmarks and their input data sets that we simulate in this work. The three applications employ perform indexed array accesses and data access can be viewed as a graph and can be optimized as such.

Table 3.2: Total data footprint in MB for each benchmark for each Input size.

Benchmark	144.graph	m14b.graph	auto.graph	Benchmark Type
IRREG	10.40MB	16.09MB	32.1MB	iterative PDE solver
NBF	10.96MB	16.91MB	33.85MB	molecular dynamics kernel
MOLDYN	18.13MB	27.56MB	56.1MB	non-bonded force kernel

IRREG is an iterative Partial Differential Equation (PDE) solver for an irregular mesh. MOLDYN is abstracted from the non-bonded force calculation in CHARMM [32, 33]. CHARMM is an acronym for "Chemistry at HARvard Macromolecular Mechanics" computer program package. CHARMM is a key and highly flexible program which uses empirical energy functions to model macromolecular systems. CHARMM is used at the NIH to model macromolecular systems.

NBF (Non Bonded Force kernel) is a molecular dynamics simulation. NBF is taken from the GROMOS software package [51, 176]. GROMOS^{TM} is an acronym for "GROningen MOlecular Simulation" computer program package, which has been developed since 1978 for the dynamic modeling of (bio)molecules, until 1990 at the University of Groningen, The Netherlands, and since then at the ETH, the Swiss Federal Institute of Technology, in Zurich, Switzerland.

The three benchmarks can run similar data sets. Only MOLDYN needs an extra data file containing the cartesian coordinates for nodes in the graph data files.

Table 3.3: Benchmarks and Input summary.

Benchmark	144.graph	m14b.graph	auto.graph
IRREG	144649 nodes 1074393 edges	214765 nodes 1679018 edges	448695 nodes 3314611 edges
NBF	144649 nodes 1074393 edges	214765 nodes 1679018 edges	448695 nodes 3314611 edges
MOLDYN	144649 nodes 1074393 edges xyz coordinates for each node	214765 nodes 1679018 edges xyz coordinates for each node	448695 nodes 3314611 edges xyz coordinates for each node

Tables 3.3 and 3.2 show the three data input files, named 144.graph, m14b.graph and auto.graph, respectively. Table 3.3 shows the total data size footprint of the graphs in each input data set for all three benchmarks. The footprint sizes of the input data sets are between $10MB - 57MB$. Even though the data set sizes are not extremely large, they are consistent with the size of the simulated systems' caches. Thus, even though the input data is limited in size, the results we obtain in this thesis are representative of the results one would obtain for larger inputs running on CPUs with larger caches.

In the next Chapter, we will discuss the different parallelization techniques for the benchmarks. We will also discuss the locality optimizations possible for the applications as well. We apply the locality optimizations by hand. The locality optimizations for the parallel version of the benchmarks will draw from the optimizations for the serial versions of the benchmarks we have already studied in [11–13].

3.2 Reuse Distance

More than 40 years ago, Mattson *et al.* [115] introduced reuse distance analysis (also called stack distance analysis) as a method for characterizing application memory access behavior without being tied to a specific memory hierarchy. Later, researchers applied RD analysis for studying, validating, and predicting cache performance, modeling program locality, providing hints during code generation, and helping in structuring code and data to improve locality for uniprocessor [21–26, 59, 201–203]. And in recent years, there has been a lot of interest in multicore reuse distance. We will discuss this in the next section.

We can measure Reuse Distance (RD) by counting the number of unique data blocks referenced between two references to the same data block in a Least Recently Used (LRU) stack. If a never before seen data block is accessed by the running program, this data block gets pushed on the top of the aforementioned LRU stack. When a data block that was accessed previously is accessed again, we search the LRU stack for its position in the stack. The reuse distance is represented by the number of entries in the LRU stack between the referenced data block and the top of the stack.

A reuse distance profile for a certain program is simply the histogram of the RD values for all references in the program. Given a cache that implements an LRU replacement policy with total capacity C, we can estimate the total number of cache misses as the sum of all reference counts with reuse distance equal to or larger than C in the RD profile.

Since sequential uniprocessor benchmarks only have a single non-varying memory reference stream, RD profiles for sequential uniprocessor benchmarks are both *architecture independent* and *cache size independent*. We can compute RD profiles on one particular machine, and later use these profiles for whatever purpose on a different cache size and different machine as long as the memory reference stream remains the same.

3.3 Multicore Reuse Distance

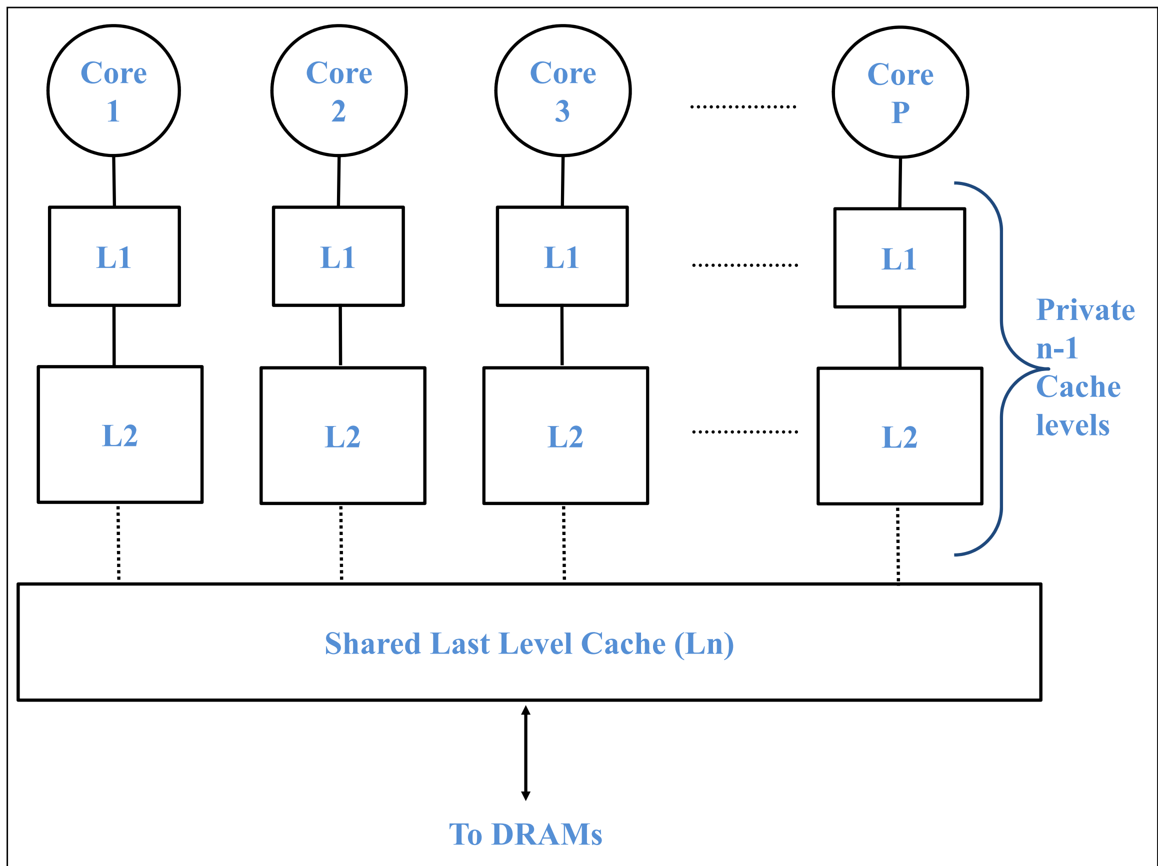


Figure 3.2: Chip Multiprocessors with multiple levels of private cache and a shared last level cache

With the prevalence of CMPs, it is highly desirable to extend reuse distance to

handle parallel programs. Researchers have devised ways to compute reuse distance profiles for parallel programs on multiprocessors. We use the reuse distance profiling techniques introduced by Wu and Yeung in [182, 186, 187].

As we have explained earlier in Chapter 1, most contemporary multicore processors contain some levels of private caches, normally a level or two and a shared level of cache. Figure 3.2 shows a generalized multicore processor with P cores, $n-1$ levels of private caches and backed by a single shared Last-Level cache (LLC) L_n.

Threads interact very differently given a private or a shared cache. For each type of cache, there has to be a different locality profile representing the particular cache type. Shared caches react differently to shared data. Data sharing reduces the size of the aggregate working set of the running program. This makes the cache effectively larger since the pressure on the cache is smaller. On the contrary for private caches, shared data causes replication of data in multiple cores' private caches and causes communication among cores to send data across the private caches. Replication reduces the effective cache size due to having multiple copies of data in the caches.

Normally as Figure 3.2 suggests, there are some levels of private cache and some shared cache. Therefore, we have to have profiles that account for the private caches. To model shared caches and private caches, we use *Private-stack Reuse Distance (PRD)* for private caches and *Concurrent Reuse Distance (CRD)* for shared cache. Intra-thread data locality and inter-thread data locality dictate the memory behaviour of multicore cache systems. CRD and PRD profiles capture the different

thread interactions in shared and private caches.

3.3.1 Concurrent Reuse Distance

Initially, reuse distance analysis was extended for shared caches. This version of reuse distance is computed from the stream of interleaved memory references from all the system cores using a single LRU stack. The profile that we obtain from this is called the concurrent reuse distance (CRD) profile [58, 92, 156, 157, 160].

There are several effects due to memory interleaving that are captured by CRD profiles. One of these effects is dilation. Dilation happens because references from separate cores interleave and disrupt each other's locality. This causes CRD values to be larger than RD values collected for single core profiles because of the interleaved private references from each core.

Data sharing on the other hand reduces dilation. If multiple cores reference the same item, the reuse distance experienced by each core can be reduced because of the reuse of the shared data by the different cores. This tends to offset the impact of dilation, and can even result in a smaller CRD value compared to the single core RD value.

3.3.2 Private-stack Reuse Distance

While Concurrent Reuse Distance (CRD) profiles are computed using only a single LRU stack, Private-stack Reuse Distance (PRD) profiles are computed by having as many LRU stacks as there are cores in the system. The references of each

core is fed into its own LRU stack. This happens while coherence among the cores is observed across the per-thread LRU stacks [156, 157, 160]. Maintaining coherence is done by invalidating data that is write shared. Read sharing causes replication of data among the caches or the LRU stacks of cores that access the same data block. These replications make the effective cache size smaller in the case of private caches. Cache misses to cache blocks that were invalidated because of write sharing are quantified as cache coherence misses.

3.3.3 Cache Miss Count (CMC)

Using the Concurrent Reuse Distance (CRD) and Private-stack Reuse Distance (PRD) profiles, we can estimate the cache miss count (CMC) for a cache of capacity i by using a simple summation formula. CMC is defined as the number of cache misses incurred at capacity i in a CRD or an sPRD profile as given in Equations 3.1 and 3.2 respectively, where N is the total number of reference count bins in each profile. CRD[Inf] and sPRD[Inf] are the compulsory misses a cache must suffer no matter how big the cache is.

$$CRD_CMC[i] = \sum_{j=i}^{N-1} CRD[j] + CRD[Inf] \tag{3.1}$$

$$sPRD_CMC[i] = \sum_{j=i}^{N-1} sPRD[j] + sPRD[Inf] \tag{3.2}$$

Using CRD_CMC and $sPRD_CMC$, we can compute the number of misses in a shared cache of a certain size and a private cache of a certain size, respectively.

We will use these equations to build a performance model predicting execution time in Section 6.1

3.3.4 Pin-based Profiling Tool

In this section, we introduce the profiling tool that is developed in-house in our group on top of the Intel Pin toolkit that can acquire CRD and PRD profiles. This profiling tool is used to profile and analyze three graph benchmarks running three different problem sizes on different L1 and L2 cache sizes on four cores.

Intel's Pin is a dynamic binary instrumentation framework for the IA-32 and x86-64 instruction-set architectures enabling the creation of dynamic program analysis tools [20, 99, 141]. The tools created using Pin, called "PINtools", can be used to perform program analysis on user space applications on both Linux and Windows. As a dynamic binary instrumentation tool, instrumentation is performed at run time on the compiled binary files. Thus, it requires no recompiling of source code and can support instrumenting programs that dynamically generate code.

Pin provides a rich API that abstracts away the underlying instruction-set and allows context information such as register contents to be passed to the injected code as parameters. Pin automatically saves and restores the registers that are overwritten by the injected code so the application continues to work. The instrumented binary runs natively on the hardware, so it provides much higher performance and compatibility than simulators. We use a PINtool developed in our group. The tool has been used in several research studies. Wu *et al.* [181, 182, 186, 187] explain the

framework the tool offers in much deeper and greater detail. Our main goal from using the PINtool is to acquire the CRD and PRD profiles for the benchmarks, input data sets, parallelization techniques, and locality optimizations we are investigating in this thesis.

The tool implements a controlled context switch in the OS scheduler to simulate simultaneous thread execution, faithfully mimicking how a CMP would execute the threads. The fine-grain context switch method proposed by McCurdy and Fischer [116], as illustrated in Figure 3.3, is what is used in the tool. In this scheme, a centralized master scheduler control passes an active signal to the thread that should be activated in a round-robin fashion, thus only one thread is active at any given time. The rest of the threads wait for their turn to run after the active signal from the master scheduler is sent signaling their turn. Since the active signal is passed in a round-robin fashion, the memory accesses are interleaved in a known consistent order across all threads. One last important point is that the scheduler simulates the synchronization mechanisms of Pthreads.

Several assumptions are made when acquiring the CRD and PRD profiles. The first is that the Pin tool like its toolkit performs only functional execution, context switching between threads every memory reference. Hence, the memory references from each thread are interleaved uniformly in time with a known order. In the tool memory interleaving model, there is no particular cache hierarchy or CMP architecture built-in the developed tool on top of PIN. So, there are no timing interactions or considered in the CRD and PRD profiles. The assumption of uniform memory interleaving is accurate enough to acquire profiles for loop-based parallel

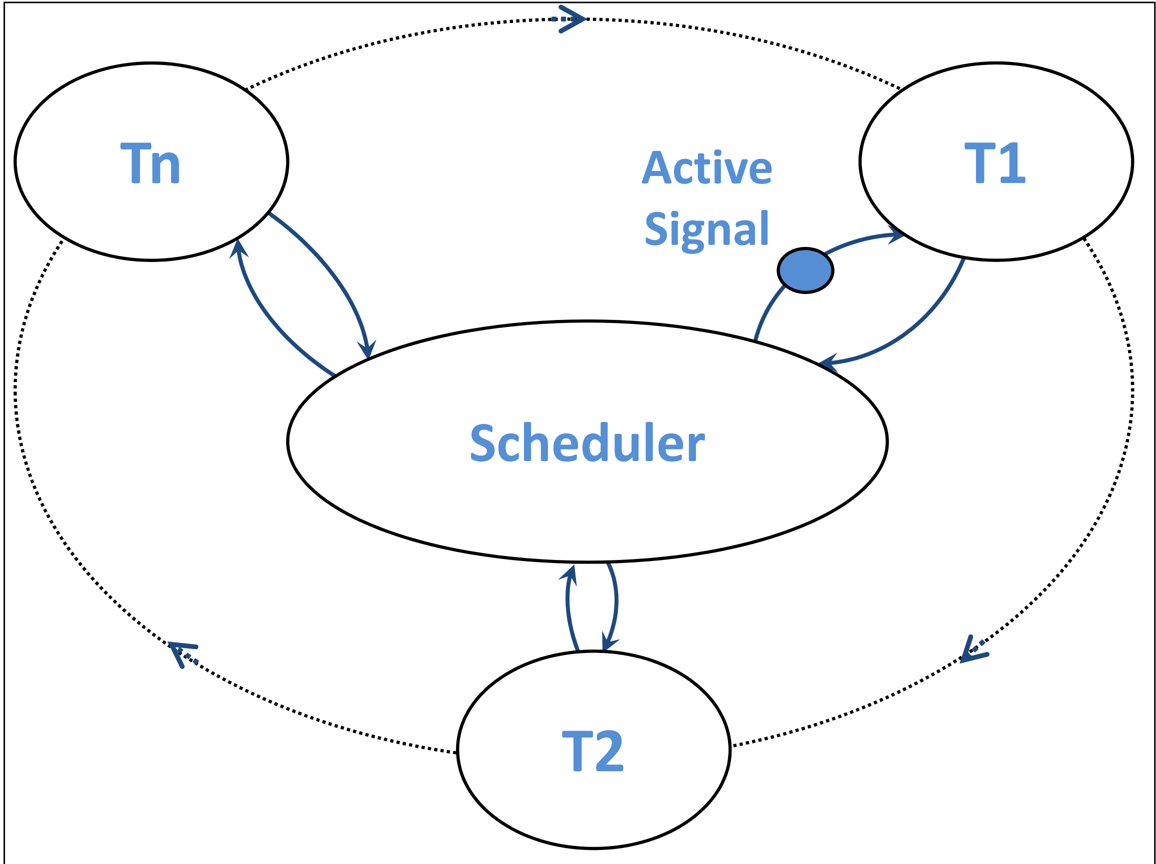


Figure 3.3: Thread interleaving scheme.

programs as Wu and Yeung have shown in their work [181, 182, 186, 187]. No interrupts are thrown by the OS to the threads. The benchmarks run on the tool unopposed by any other processes.

Chapter 4

Techniques for Graph Problems: Locality Optimizations & Parallelizations

In this chapter, we discuss the techniques that are central to the work in this thesis. First, we will discuss locality optimizations for graph problems. Second, we will introduce the different parallelization choices. Finally, we show how locality optimizations and parallelizations techniques can be combined.

4.1 Locality Optimizations

Locality optimizations are a set of techniques used to achieve better performance through optimizing cache performance. These techniques are orthogonal to prefetching [37–40]. Prefetching aims at hiding the latency of references by issuing loads early for those references that are expected to miss in the cache. On the other hand, locality optimizations change data layout, computations' order of the programs, or both so as to improve the application's *data locality* [180]. Data locality optimizations improve the application's data reuse. Changing the computation order or data layout of the program at compile or run time exposes data reuse.

There are two form of data reuse temporal reuse and spatial reuse. Temporal reuse happens when references to data go to the same location and repeats at different times. We can exploit temporal reuse by reordering the computations to

execute all the computations on a data item before moving on to other data.

Spatial reuse happens when references to data destined to nearby locations in memory are performed together. We can exploit spatial reuse by reordering the computations to perform computations on data that are close in space before moving on to the next computation.

The compiler or run-time system of the application examines what type of data locality can be exploited to improve the application's cache performance. There are different locality optimizations for different access patterns [36, 49, 52, 104, 140, 148–150, 180].

4.2 Reordering for Indexed Accesses

Borrowing from our work in [11–13], locality optimizations for indexed array benchmarks reorder computations using graph partitioning techniques to improve reuse, locality, and memory performance.

Many graph partitioners exist. One of the most famous is the metis library of partitioning programs [96–98]. In the literature, many algorithms, graph partitioning techniques, and graph partitioners have been used for indexed array benchmarks. We call programs that exhibit this access pattern graph problems or indexed array problems interchangeably.

We have applied the recursive coordinate bisection RCB computation reordering algorithm for the benchmarks used in this work in our previous work [11–13]. But we are not restricted to it. Researchers have studied a lot of techniques for

the benchmarks we are using here in [76, 77]. They have studied and compared the different techniques to parallelize and improve locality for single threaded and parallel versions of this set of benchmarks. They have noted that the quality of the partitions produced by the metis library is the best but the overhead of using the library to partition is the highest among all the investigated techniques. We will opt to use metis for its better quality partitions and we assume that the program runs for a number of iterations that would justify the overhead cost. In this thesis, we will not study how many iterations are needed to ameliorate the overhead. Other works have performed that study in detail and it is beyond the scope of this thesis to re-perform those studies.

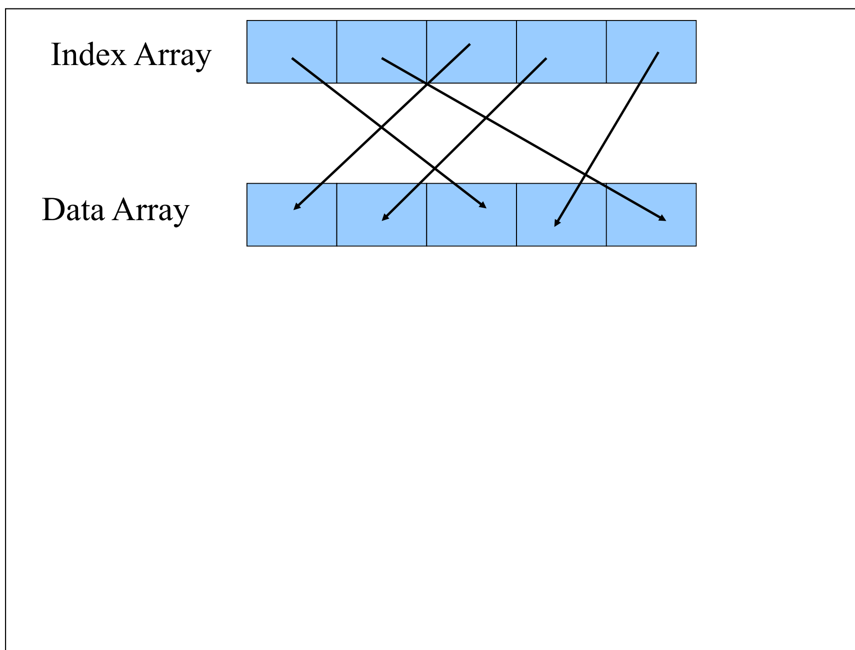
Indexed array accesses are difficult to optimize at compile time. Since the analysis at compile time cannot determine the access patterns, which are only known at run-time. Instead, indexed array accesses must be optimized at runtime [4, 56, 119, 121] or before compiling only if the input data set is known ahead of time.

Saltz *et al.* [54] designed a compiler which generates calls to an inspector to process memory access patterns at run-time. This approach is called an inspector-executor approach. This approach can be used with any benchmarks that cannot be analyzed at compile time. An example of these programs is any computation that computes on graphs.

Figure 4.1(a) shows an example code fragment for an indexed array computation. Figure 4.1(b) shows the relationship between the index array and the data (indexed) array in terms of data layout and dependency. All optimization techniques try to rearrange both the data layout and the computation order in such a way that

```
// Indexed Array Accesses
// Molecular Dynamics
float X1(M),X2(M); int index(N);
for (t = 1; t == time; t++) {
        for (i = 0; i < N; i++) {
                d = X1(index(i))-X2(index(i));
                force = pow(d, -7) – pow(d, -4);
                X1(index(i)) += force;
                X2(index(i)) += -force;
        }
}
```

(a) Indexed array access pattern example code.



(b) Indexed array access pattern figure representation.

Figure 4.1: Indexed array access pattern code fragment and data layout illustration.

would force the accesses to the data array to be more regular, resulting in better cache performance.

The reordering process is shown in Figure 4.2. This figure shows the partitioning and the reordering of the data and index arrays taking place. Since almost all molecular dynamics and electromagnetic codes interact pairs of data according to their geometric coordinate data. The problem lends itself nicely to a directed graph. Several data and computation locality transformations exist to solve the problem of improving the locality of such graphs. One of these techniques is called Graph Partitioning techniques (GPART). This technique is based on hierarchical clustering. It generates quality partitions quickly. The main advantage of this technique is that it has low overhead since it only considers edges between partitions. GPART closely matches the performance of more sophisticated partitioning algorithms, with one third of the overhead [4, 74, 119]. Yet still, we opt for metis with its higher overhead sine it demonstrates higher partition quality across the board. We might consider GPART for future work or if the overhead cost is critical.

In the example shown in Figure 4.2 circles represent computations (loop iterations), squares represent data (array elements), and arrows represent data accesses. Initially, memory accesses are irregular, but either computation or data may be reordered to improve temporal and spatial locality. Note that each iteration accesses two array elements. Computations can be viewed as edges connecting data nodes, resulting in a graph. Locality optimizations can then be mapped to a graph partitioning problem. Partitioning the graph and putting nodes in a partition close in memory can then improve spatial and temporal locality. Applying lexicographic sorting after partitioning captures even more locality.

The hierarchical structure in GPART is similar to that of recursive coordinate

bisection (RCB), which is a data reordering algorithm used when data is unevenly distributed. RCB is based on geometric coordinate information. RCB recursively splits each dimension into two by finding the median of the data coordinates in that dimension. After partitioning, data items are stored consecutively within each partition. Loop iterations are lexicographically sorted based on the data accessed [56, 74, 121]. RCB has higher overheads than other techniques but is most likely to work well with unevenly distributed data. Figures 4.2(d) and 4.2(e) show the computations after the partitioning technique has been applied and show how the computations will access the array in a different order compared to the original access order. These algorithms are explained in more details in [74, 75].

4.2.1 Using Metis

In this subsection, we will discuss in more details the partitioning methodology we have used for our work here.

For each benchmark, we compute an estimate of the total data that is needed for the computation as we have reported in Table 3.2. We compute several partition sizes for each input data set for each benchmark. These sizes go from a no partitioning case *i.e.* the original graph to the case with as large a partition size we can.

We use the metis METIS_PartGraphKway function to partition all graphs with all benchmarks to the proper number of partitions. The function partitions a graph into k parts using the multilevel k-way partitioning to minimize the total

number of edge cuts among the partitions for each graph for each problem [96–98].

$$No_Partitions = \left\lceil \frac{Input_set_size}{Cache_size} \right\rceil \tag{4.1}$$

We use equation 4.1 to compute the number of partitions for each graph under each benchmark.

Table 4.1 shows the partition sizes computed for all of the three input data sets of IRREG. Table 4.2 shows a summary of the partition sizes for different relevant L1 cache sizes for the input data sets for IRREG.

Tables 4.3 shows the partition sizes computed for all of the three input data sets of NBF. Table 4.4 shows a summary of the partition sizes for the L1 cache sizes simulated for the input data sets for NBF.

Tables 4.5 shows the partition sizes computed for all of the three input data sets of MOLDYN. Table 4.6 shows a summary of the partition sizes for the L1 cache sizes simulated for the input data sets for MOLDYN.

4.3 Parallelization Strategies

In this section, we will discuss the parallelization strategies that we use for the benchmarks in this thesis. We explain the vanilla legacy SMP parallelization and a suggested CMP parallelization and explain its draw backs and advantages of increasing sharing. Furthermore, We show two variations on top of the vanilla CMP parallelization strategy to optimize it to take advantage of the shared cache.

Table 4.1: Partition sizes for the 144, m14b, and auto graphs for the IRREG benchmark.

144		m14b		auto	
# Parts	Partition Size	# Parts	Partition Size	# Parts	Partition Size
1	10.4MB	1	16.087MB	1	32.1MB
4	2.6MB	4	4.02MB	4	8.03MB
11	1MB	17	1MB	33	1MB
42	256kB	65	256kB	65	512kB
84	128KB	129	128KB	129	256KB
333	32kB	515	32kB	257	128kB
666	16kB	1030	16kB	1028	32kB
1332	8kB	2060	8kB	2055	16kB
2664	4kB	4119	4kB	4109	8kB
5327	2kB	8237	2kB	8218	4kB
10654	1kB	16474	1kB	16436	2kB
23170	0.5kB	23170	0.7kB	23170	1.4kB

Table 4.2: Partition sizes for the input data sets of IRREG for different L1 sizes.

Input Graph	L1=4kB	L1=8kB	L1=16kB	L1=32kB
auto	8218	4109	2055	1028
m14b	4119	2060	1030	515
144	2664	1332	666	333

4.3.1 SMP Legacy Parallelization

Legacy parallel multiprocessor machines have each processor in a different board where communication costs are high and normally any developer tries his best to minimize communication among the cores except when it is mandatory and inevitable to happen to cut the cost of going from board to board or to go across chip pins through a board bus to another chip pins which can be prohibitive if it

Table 4.3: Partition sizes for the 144, m14b, and auto graphs for the NBF benchmark.

144		m14b		auto	
# Parts	Partition Size	# Parts	Partition Size	# Parts	Partition Size
1	10.4MB	1	17MB	1	34MB
4	2.6MB	4	4.25MB	4	8.5MB
11	1MB	17	1MB	34	1MB
44	256kB	68	256kB	85	512kB
88	128KB	336	52KB	271	128KB
351	32kB	542	32kB	679	56kB
702	16kB	1083	16kB	1084	32kB
1403	8kB	2165	8kB	2167	16kB
2806	4kB	4329	4kB	4333	8kB
5612	2kB	8658	2kB	8667	4kB
11224	1kB	17316	1kB	17332	2kB
23170	0.5kB	23170	0.75kB	23170	1.5kB

Table 4.4: Partition sizes for the input data sets of NBF for different L1 sizes.

Input Graph	L1=4kB	L1=8kB	L1=16kB	L1=32kB
auto	8667	4333	2167	1084
m14b	4329	2165	1083	542
144	2806	1403	702	351

happens often. Thus, communication minimization is one of the ultimate goals of developing parallel codes for legacy SMP machines.

CMP machines can run out of the box SMP parallelized codes without any modification but such codes over minimize communication. Communication costs in a CMP system are much less than in a SMP system. Normally, for any particular problems the problem space is divided up into P equal parts each is giving to one of the P processors of the multiprocessor system where the communication among

Table 4.5: Partition sizes for the 144, m14b, and auto graphs for the MOLDYN benchmark.

144		m14b		auto	
# Parts	Partition Size	# Parts	Partition Size	# Parts	Partition Size
1	56.1MB	1	28MB	1	18.13MB
4	14.02MB	4	7MB	4	4.6MB
57	1MB	28	1MB	19	1MB
225	256kB	111	256kB	73	256kB
449	128KB	221	128KB	146	128KB
1796	32kB	882	32kB	581	32kB
3591	16kB	1764	16kB	1161	16kB
7181	8kB	3528	8kB	2321	8kB
14362	4kB	7055	4kB	4642	4kB
23170	2.5kB	14109	2kB	9283	2kB
		23170	1.25kB	18565	1kB
				23170	0.8kB

Table 4.6: Partition sizes for the input data sets of MOLDYN for different L1 sizes.

Problem	L1=4kB	L1=8kB	L1=16kB	L1=32kB
auto	14362	7181	3591	1796
m14b	7055	3528	1764	882
144	4642	2321	1161	581

the processors is kept to a minimum while we are dividing the problem.

In the case of the given graph problems, we first run the metis partitioning process to partition the graph into m parts where m is defined as we have shown in previous sections. On top of these partitions, we partition the number of edges of the graph and the number of nodes by the number of processors in the system which is P. We note here that each the P cores works on its own $1/P$ part that consists of multiple smaller parts of the m parts that were partition using metis. Figure **??**

illustrates this parallelization strategy.

4.3.2 CMP Parallelization

What we call CMP parallelization is a technique that partitions the input graph into "m" parts using metis according to the sizes computed an shown in Section 4.2.1. Each of these m partition is computed upon using all the P cores *i.e.* each of the m partitions divided into P further parts each of these parts is computed using one of the P cores thus, at any given time during the execution time all cores are computing one only one of the m parts. This technique makes all the cores share the data of the part they are computing upon. When the number of parts become large enough *i.e.* part size is too small then we are working on a very small amount of data and we have to bring much of the data to processing cores multiple times. This techniques does perform relatively good in the smaller partition sizes when compared to similar partition sizes if compared to SMP parallelization with the same partition sizes but as m increases sharing and communication becomes unreasonable and becomes counterproductive. Figure **??** illustrates this parallelization strategy.

4.3.3 Variation on the CMP Parallelization

The CMP parallelization idea which revolves around sharing of a partition among all the cores is a good idea that we can use or develop further to reap the gain of good L1 cache utilization and sharing benefits in the upper levels of the cache (L2 cache in our case).

4.3.3.1 $CMP - P$ Parallelization

We call this variation of the CMP parallelization "$CMP - P$", where P is the number of core in the multicore system. In this technique, we use metis and partition the problem into m parts as we have explained above in section 4.3.1. After that we divide the entire problem space into P parts and in the computation each of the P is computed upon using all of the P cores in the system. We note that the size of each of these P parts is even with the smallest input data sets larger than the largest L2 cache that we simulate which is 4MB. This means that the L2 cache cannot fit one of these P parts. The idea here is that metis partitioned m parts that would fit in the L1 cache and we boost sharing by having a smaller working set even though it cannot fit into the L2 cache, when all the cores compute on one of the P parts during any given time. Each of the P parts would contain a few of the m parts that metis has partitioned. Thus, we have a hierarchical partitioning at the cost of running metis on the problem only once. We proposed this in the proposal that preceded this thesis in 2006 yet it was published first by Shen *et al.* of the College of William and Mary [193–195]. Our parallelization here adds the partitioning trying to fit in the L1 cache on top of the idea published by shen *et al.*.

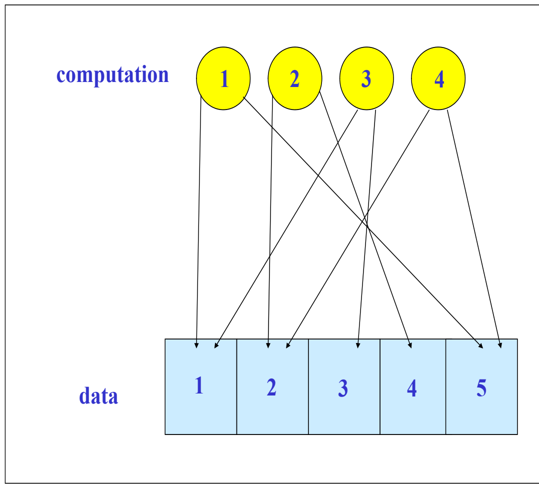
4.3.3.2 $CMP - K$ Parallelization

We call this variation of the CMP parallelization "$CMP - K$", where K can be any size that we can arbitrarily choose for some purpose. The first variation which we call "$CMP - P$" partitions the problem into P parts. For the smallest

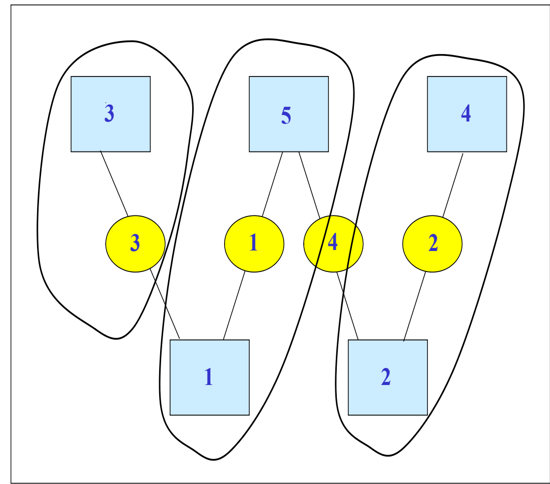
input data set, each partition is larger than 4MB which is the largest L2 cache we simulate. We try to make the partition size smaller than the size of the partition if we partition it P ways. We have chosen to partition the problem into K parts where each part would be equal to or less than 1MB. The interesting finding that we have found when we ran this parallelization is that it did not perform the best among all the parallelizations all the time on the contrary in some instances the $CMP - P$ or the Legacy SMP parallelizations would achieve better performance. We have to also note that each of the K parts is computed upon by all of the P cores exactly like the "$CMP - P$" case. This is another technique that is hierarchical in nature where we try to make the problem to fit in the L1 cache and the L2 cache.

4.3.4 Discussion on the Parallelization Strategies

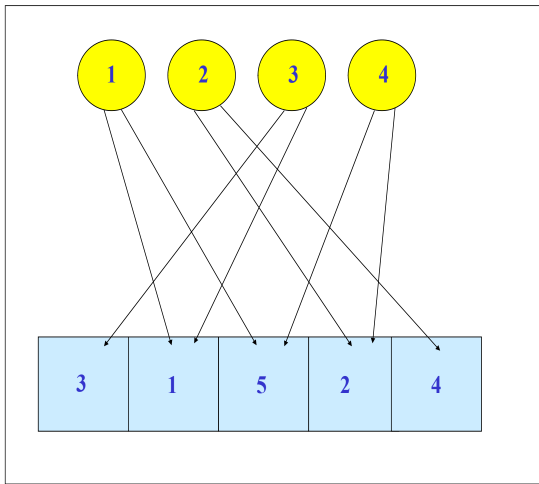
What we have learned as we will show in the results chapters of the thesis is that the design space we have to find the best combination of the parallelization strategy and locality optimization (graph partitioning) is very complex. We learnt from that design space exploration that fitting in the lowest levels of the cache (L1 cache in our case) should be the utmost top priority for developers and compilers then comes any attempts to fit in the upper levels of the cache (L2 cache in our case).



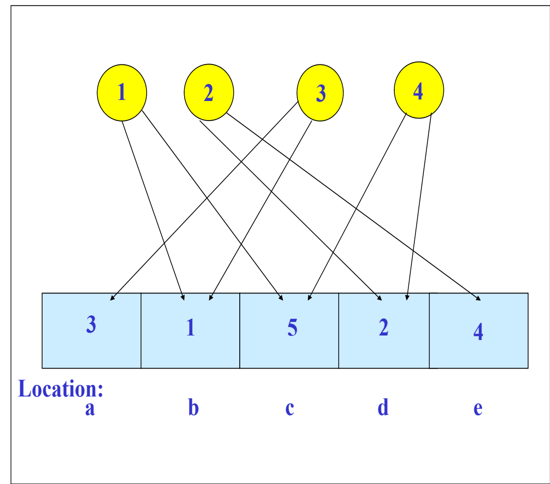
(a) Original access.



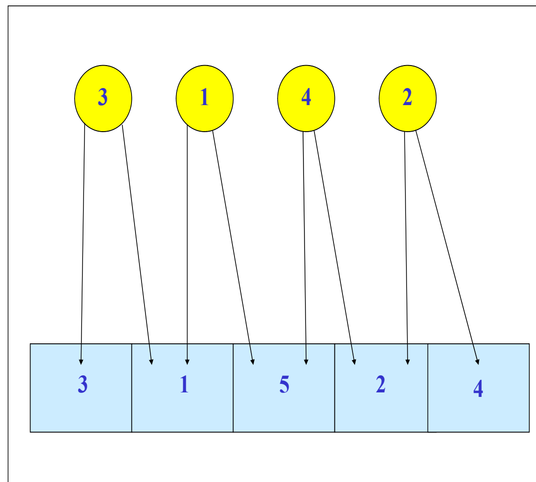
(b) Graph partitioning.



(c) Data reordering.



(d) After data reordering.



(e) Computation reordering.

Figure 4.2: An example for an index array locality optimization.

Chapter 5

Design Space Exploration using detailed M5 Experimental Results

In this chapter, we explore the design space consisting of the combination of the parallelization strategies discussed in Section 4.3 and the partitionings discussed in Sections 4.2.1 and 3.1.1 running our benchmarks and the input data sets, we discussed in Chapter 3. We run the different partition sizes for every input data set for each benchmark. We run the M5 simulator using the parameters presented in Section 3.1.1. We run between $10 - 12$ partition sizes for each machine as described in Section 3.1.1. The results of this chapter will quantify how much performance gain our proposed parallelization techniques and locality optimizations can acheive on top of a baseline case. In particular we will show that we can get gains from a few percent to as much as 20% across the machines, benchmarks, and input data sets simulated.

5.1 Searching for the best Partition Size using the M5 Simulator

The problem we aim to quantify here is a design space exploration problem. We have several dimensions that span the design space. The space is bounded by four parallelizations strategies, $10 - 12$ graph partitionings (Locality optimization dimension), three L1 cache sizes, six L2 cache sizes, three benchmarks each with three input data sets. This search space is comprised of $7,776$ simulations. The

choice of the cache sizes is made in such a way that we span a decent space of cache design.

We run varying partition sizes (Tables 4.1, 4.3, and 4.5) ranging from no partitioning denoted by 1 to a maximum partition size of $23,170$ for each combination of L1 and L2 cache sizes using the M5 simulator with the parameters shown in Section 3.1.1.

Each of the following sections presents the results for each of the benchmarks. Within each section, there are three subsections, one for each input data set. In each subsection, there are three figures each with six sub-figures. Each figure is for the three simulated L1 cache sizes and each of the six sub-figures will present the results for each of the six L2 cache sizes simulated. Each sub-figure shows the results for all four parallelization strategies as shown in Section 4.3 in conjunction with the different locality optimizations (partition sizes).

All results are normalized to a baseline performance point. The baseline performance point in each figure is labeled with a "*b". The baseline is defined as the partition size selected for a partition that is estimated to fit in the L1 cache simulated. Thus, in each of the three figures for each subsection, the baseline is fixed since the L1 cache is fixed. The baselines are shown in Tables 4.2, 4.4, and 4.6. Each sub-figure shows relative performance sorted in ascending order. Data points with relative performance below 1 represent improvements relative to the baseline case. After each set of three figures and before the conclusion of each section there is a table that summarizes the performance in the three figures and an average performance gain number is shown. Each of the parallelization techniques that will

be mentioned below is explained in details in Section 4.3. In each sub-figure, the x-axis can be deciphered according to one of the four cases as follows:

1. "$SMP.4kB.32kB.10654$" means the parallelization is SMP, the L1 cache simulated is 4kB, the L2 cache slice per core for each core is 32kB, and the number of partitions is $10,654$.

2. "$CMP.4kB.32kB.10654$" means the parallelization is CMP, the L1 cache simulated is 4kB, the L2 cache slice per core for each core is 32kB, and the number of partitions is $10,654$.

3. "$CMP.4kB.32kB.10654-4$" means the parallelization is CMP-4, the L1 cache simulated is 4kB, the L2 cache slice per core for each core is 32kB, and the number of partitions is $10,654$. Note that four here is the number of cores.

4. "$CMP.4kB.32kB.10654 - 11$" means the parallelization is CMP-11, the L1 cache simulated is 4kB, the L2 cache slice per core for each core is 32kB, and the number of partitions is $10,654$. Note that 11 here represents the maximum partition size that can fit in a 1MB cache. This changes according to each input data set.

The results of each benchmark with an input data set is summarized in a Table similar to Table 5.1. The reported gains "%Gain M5" are relative to the baseline case which is the legacy SMP parallelization strategy with a partition size that is estimated to fit into the simulated L1 cache and are in percentage. For each benchmark there is a table similar to Table 5.10 which summarizes the results

across all input data sets for each particular benchmark. The reported numbers are gains in percentage from the baseline cases.

5.2 Irreg Results

In this section, we present the results of the M5 simulator for the Irreg benchmark with all of the input data sets; **144, m14b, and auto**. We simulate the cross product of machines with 4kB, 8kB, and 32kB L1 caches and 128kB, 256kB, 512kB, 1MB, 2MB, and 4MB L2 caches. All the results are with 4 cores.

5.2.1 Irreg with 144 input data set

In this section, we present the results of the M5 simulator for the Irreg benchmark with the **144** input data set. Figures 5.1, 5.2, and 5.3 show in detail the results for the three L1 caches. We clearly see that there is a large spectrum for the values relative to the baseline. We see that in some cases the unmodified vanilla benchmark can sometime be larger than the baseline by a factor of from 5.4 to 3.4. This clearly shows that there is a large gain to be captured if we compare to the unmodified benchmark. The gains relative to the baseline are summarized in Table 5.1. The range of gains for the 4kB simulated machines ranges from 1.34% to 2.36% and the average is 1.88%. The range of gains for the 8kB simulated machines ranges from 1.38% to 4.36% and the average is 3.56%. The range of gains for the 32kB simulated machines ranges from 2.09% to 11.20% and the average is 4.96%. The overall average for the 144 input data set for all L1 caches is 3.47%.

Table 5.1: Summary of the percentage gains relative to the baseline using the M5 simulator for IRREG for the **144** input data set with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.34	2.19	2.36	1.85	2.04	1.49	1.88
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	3.53	4.36	3.87	4.12	4.09	1.38	3.56
L1 size	32 kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	11.20	5.65	4.05	3.57	3.21	2.09	4.96
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. 144
%Gain M5	5.35	4.07	3.43	3.18	3.11	1.66	3.47

5.2.2 IRREG with m14b input data set

In this section, we present the results for the M5 simulator for IRREG benchmark with the **m14b** input data set. Figures 5.4, 5.5, and 5.6 show in detail the results for the three L1 caches. We clearly see that there is a large spectrum for the values relative to the baseline. We see that in some cases the unmodified benchmark can sometimes be larger than the baseline by a factor from 5.8 to 3.0. This clearly shows that there is a large gain to be captured if we compare to the unmodified benchmark. The gains relative to the baseline are summarized in Table 5.2. The range of gains for the 4kB simulated machines are from 0.58% to 1.33% and the average is 0.95%. The range of gains for the 8kB simulated machines are from 1.37% to 2.38% and the average is 1.86%. The range of gains for the 32kB

Table 5.2: Summary of the percentage gains relative to the baseline using the M5 simulator for IRREG for the **m14b** input data set with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	0.85	1.33	1.13	1.03	0.79	0.58	0.95
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	2.38	2.01	1.75	2.07	1.55	1.37	1.86
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	6.68	3.73	3.32	2.29	1.59	1.32	3.16
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. m14b
%Gain M5	3.30	2.36	2.07	1.80	1.31	1.09	1.99

simulated machines are from 1.32% to 6.68% and the average is 3.16%. The overall average for the m14b input data set for all L1 caches is 1.99%.

5.2.3 IRREG with auto input data set

In this section, we will show the results of the M5 simulator for IRREG benchmark with the **auto** input data set. Figures 5.7, 5.8, and 5.9 show in detail the results for the three L1 caches. We clearly see that there is a large spectrum for the values relative to the baseline. We see that in some cases the unmodified benchmark can sometime be larger than the baseline by a factor of from 5.8 to 2.4. This clearly shows that there is a large gain to be captured if we compare to the unmodified benchmark. The gains relative to the baseline is summarized in Table

Table 5.3: Summary of the percentage gains relative to the baseline using the M5 simulator for IRREG for the **auto** input data set with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	4.73	4.39	4.48	3.49	3.33	3.63	4.01
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	10.00	9.08	8.73	7.65	7.01	7.21	8.28
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	19.69	11.01	8.58	6.69	6.68	4.65	9.55
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. auto
%Gain M5	11.48	8.16	7.26	5.94	5.67	5.17	7.28

5.3. The range of gains for the 4kB simulated machines are from 3.33% to 4.73% and the average is 4.01%. The range of gains for the 8kB simulated machines are from 7.01% to 10.00% and the average is 8.28%. The range of gains for the 32kB simulated machines are from 4.65% to 19.69% and the average is 9.55%. The overall average for the auto input data set for all L1 caches is 7.28%.

5.3 NBF Results

In this section, we present the results of the M5 simulator for NBF benchmark with all of the input data sets; **144, m14b, and auto**. We simulate the cross product of machines with 4kB, 8kB, and 32kB L1 caches and and 128kB, 256kB, 512kB, 1MB, 2MB, and 4MB L2 caches. All the results are with 4 cores.

5.3.1 NBF with 144 input data set

In this section, we will show the results of the M5 simulator for NBF benchmark with the **144** input data set. Figures 5.10, 5.11, and 5.12 show in detail the results for the three L1 caches. We clearly see that there is a large spectrum for the values relative to the baseline. We see that in some cases the unmodified benchmark can sometime be larger than the baseline by a factor of from 5.6 to 3.8. This clearly shows that there is a large gain to be captured if we compare to the unmodified benchmark. The gains relative to the baseline is summarized in Table 5.4. The range of gains for the 4kB simulated machines are from 0% to 1.30% and the average is 0.39%. The range of gains for the 8kB simulated machines are from 0.71% to 5.22% and the average is 3.41%. The range of gains for the 32kB simulated machines are from 3.34% to 11.97% and the average is 6.44%. The overall average for the 144 input data set for all L1 caches is 3.41%.

5.3.2 NBF with m14b input data set

In this section, we will show the results of the M5 simulator for NBF benchmark with the **144** input data set. Figures 5.13, 5.14, and 5.15 show in detail the results for the three L1 caches. We clearly see that there is a large spectrum for the values relative to the baseline. We see that in some cases the unmodified benchmark can sometime be larger than the baseline by a factor of from 6.2 to 3.6. This clearly shows that there is a large gain to be captured if we compare to the unmodified benchmark. The gains relative to the baseline is summarized in Table

Table 5.4: Summary of the percentage gains relative to the baseline using the M5 simulator for NBF for the **144** input data set with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.30	0.07	0.00	0.88	0.12	0.00	0.39
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	5.22	4.41	4.00	4.04	2.05	0.71	3.41
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	11.97	8.62	6.26	4.89	3.54	3.34	6.44
Average across all L1 Sizes							
L2 size	128 kB	256 kB	512 kB	1 MB	2 MB	4 MB	Avg. 144
%Gain M5	6.16	4.37	3.42	3.27	1.90	1.35	3.41

5.5. The range of gains for the 4kB simulated machines are from 0% to 1.24% and the average is 0.60%. The range of gains for the 8kB simulated machines are from 0.43% to 1.83% and the average is 1.39%. The range of gains for the 32kB simulated machines are from 0.35% to 6.16% and the average is 3.21%. The overall average for the m14b input data set for all L1 caches is 1.74%.

5.3.3 NBF with auto input data set

In this section, we will show the results of the M5 simulator for NBF benchmark with the **auto** input data set. Figures 5.16, 5.17, and 5.18 show in detail the results for the three L1 caches. We clearly see that there is a large spectrum for the values relative to the baseline. We see that in some cases the unmodified

Table 5.5: Summary of the percentage gains relative to the baseline using the M5 simulator for NBF for the **m14b** input data set with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	0.00	0.59	0.84	0.94	1.24	0.00	0.60
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.37	1.80	1.78	1.83	1.15	0.43	1.39
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	6.16	3.28	3.55	3.03	2.91	0.35	3.21
Average across all L1 Sizes							
L2 size	128 kB	256 kB	512 kB	1 MB	2 MB	4 MB	Avg. m14b
%Gain M5	2.51	1.89	2.06	1.94	1.77	0.26	1.74

benchmark can sometime be larger than the baseline by a factor of from 6.2 to 2.7. This clearly shows that there is a large gain to be captured if we compare to the vanilla benchmark. The gains relative to the baseline is summarized in Table 5.6. The range of gains for the 4kB simulated machines are from 3.10% to 4.50% and the average is 3.57%. The range of gains for the 8kB simulated machines are from 5.78% to 8.64% and the average is 6.74%. The range of gains for the 32kB simulated machines are from 6.33% to 16.18% and the average is 9.81%. The overall average for the auto input data set for all L1 caches is 6.70%.

Table 5.6: Summary of the percentage gains relative to the baseline using the M5 simulator for NBF for the **auto** input data set with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	3.60	3.10	4.50	3.30	3.30	3.60	3.57
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	8.64	6.30	7.23	6.26	5.78	6.22	6.74
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	16.18	11.60	9.74	8.00	7.01	6.33	9.81
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. auto
%Gain M5	9.47	7.00	7.16	5.85	5.36	5.38	6.70

5.4 Moldyn Results

In this section, we will show the results of the M5 simulator for Moldyn benchmark with all of the input data sets; **144, m14b, and auto**. We simulate the cross product of machines with 4kB, 8kB, and 32kB L1 caches and 128kB, 256kB, 512kB, 1MB, 2MB, and 4MB L2 caches. All the results are with 4 cores.

5.4.1 Moldyn with 144 input data set

In this section, we will show the results of the M5 simulator for Moldyn benchmark with the **144** input data set. Figures 5.19, 5.20, and 5.21 show in detail the results for the three L1 caches. We clearly see that there is a large spectrum for the values relative to the baseline. We see that in some cases the

Table 5.7: Summary of the percentage gains relative to the baseline using the M5 simulator for MOLDYN for the **144** input data set with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	2.43	2.11	2.32	2.55	2.28	2.32	2.34
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	3.28	2.91	2.84	2.86	2.87	2.94	2.95
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	8.42	5.97	4.35	4.08	3.97	3.77	5.09
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. 144
%Gain M5	4.71	3.66	3.17	3.16	3.04	3.01	3.46

unmodified benchmark can sometime be larger than the baseline by a factor of from 2.1 to 1.4. This clearly shows that there is a large gain to be captured if we compare to the unmodified benchmark. The gains relative to the baseline is summarized in Table 5.7. The range of gains for the 4kB simulated machines are from 2.11% to 2.55% and the average is 2.34%. The range of gains for the 8kB simulated machines are from 2.84% to 3.28% and the average is 2.95%. The range of gains for the 32kB simulated machines are from 3.77% to 8.42% and the average is 5.09%. The overall average for the 144 input data set for all L1 caches is 3.46%.

5.4.2 Moldyn with m14b input data set

In this section, we will show the results of the M5 simulator for Moldyn benchmark with the **m14b** input data set. Figures 5.22, 5.23, and 5.24 show in detail the results for the three L1 caches. We clearly see that there is a large spectrum for the values relative to the baseline. We see that in some cases the unmodified benchmark can sometime be larger than the baseline by a factor of from 2.1 to 1.15. This clearly shows that there is a large gain to be captured if we compare to the unmodified benchmark. The gains relative to the baseline is summarized in Table 5.8. The range of gains for the 4kB simulated machines are from 0.15% to 1.14% and the average is 0.41%. The range of gains for the 8kB simulated machines are from 0.88% to 1.18% and the average is 0.99%. The range of gains for the 32kB simulated machines are from 1.96% to 3.58% and the average is 2.29%. The overall average for the m14b input data set for all L1 caches is 1.23%.

5.4.3 Moldyn with auto input data set

In this section, we will show the results of the M5 simulator for Moldyn benchmark with the **auto** input data set. Figures 5.25, 5.26, and 5.27 show in detail the results for the three L1 caches. We clearly see that there is a large spectrum for the values relative to the baseline. We see that in some cases the unmodified benchmark can sometime be larger than the baseline by a factor of from 2.2 to 1.7. This clearly shows that there is a large gain to be captured if we compare to the unmodified benchmark. The gains relative to the baseline is summarized in

Table 5.8: Summary of the percentage gains relative to the baseline using the M5 simulator for MOLDYN for the **m14b** input data set with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.14	0.29	0.15	0.15	0.33	0.37	0.41
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.18	1.02	0.93	0.88	0.93	0.99	0.99
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	3.58	2.20	1.99	1.97	2.04	1.96	2.29
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. m14b
%Gain M5	1.97	1.17	1.03	1.00	1.10	1.11	1.23

Table 5.9. The range of gains for the 4kB simulated machines are from 0.00% to 2.23% and the average is 0.47%. The range of gains for the 8kB simulated machines are from 1.08% to 1.76% and the average is 1.34%. The range of gains for the 32kB simulated machines are from 2.39% to 8.68% and the average is 4.32%. The overall average for the auto input data set for all L1 caches is 2.04%.

5.5 M5 Results Summary & Conclusions

In this chapter, we have explored the design space of different locality optimizations (graph partitioning sizes) combined with different parallelization strategies under different machines with different L1 and L2 cache sizes for each benchmark for each input data sets. This design space exploration shows that the space is very

Table 5.9: Summary of the percentage gains relative to the baseline using the M5 simulator for MOLDYN for the **auto** input data set with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
% Gain M5	0.12	0.00	2.23	0.30	0.17	0.01	0.47
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
% Gain M5	1.19	1.76	1.08	1.26	1.33	1.42	1.34
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
% Gain M5	8.68	5.70	3.55	3.00	2.59	2.39	4.32
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. auto
% Gain M5	3.33	2.48	2.29	1.52	1.36	1.28	2.04

complex and there is gains from a few percentage points to double digit gains in some cases. Clearly, this is a complex search problem.

The problematic issue with capturing these gains is that there is no rule of thumb for picking the best performing combination of graph partitioning size and parallelization given a certain machine cache structure and size. For example, it is not the case that using the largest number of partitions results in the best performance. In fact, sometimes the baseline is the best case. In some cases, the gains are as large as 19.69%, which happens in one of the cache sizes and input data sets running IRREG. We notice that as the L2 cache size increases, the gains get lower. This suggests that if there is enough cache to hold most or a large portion of the working set of the benchmark input data set, then locality optimizations become

Table 5.10: Summary of the percentage Gains relative to the baseline using the M5 simulator for IRREG for each input data sets with across all L1 cache sizes for L2 cache sizes ($128kB - 4MB$).

Graph	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
144	5.35	4.07	3.43	3.18	3.11	1.66	3.47
m14b	3.30	2.36	2.07	1.80	1.31	1.09	1.99
auto	11.48	8.16	7.26	5.94	5.67	5.17	7.28
Avg. IRREG	6.71	4.86	4.25	3.64	3.37	2.64	4.24

less beneficial. We notice that as the L1 cache size increases given a fixed L2 cache size, the gains increase. This observation is true most of the time. This observation is not surprising since the increased L1 cache size should , in general, translate to better performance. The larger L1 cache allows us to search a bigger space of partitioning sizes that can be accommodated in the larger L1 cache and thus would have better performance. We also notice that the most important cache fitting is to fit in the L1 cache first and then fitting in the L2 cache or having a smaller working set comes next. Fitting in the private caches supersedes increasing sharing in the L2 shared caches.

Tables 5.10, 5.11, and 5.12 are a summary of the results for each benchmark for each input data set. Also, the tables show an overall average for each benchmark.

Table 5.10 summarizes the results for IRREG. For IRREG with the 144 data set, on average, there is a gain of 3.47%. For the m14b data set, on average, there is a gain of 1.99%. For the auto data set, on average, there is a gain of 7.28%. IRREG, on average, has a performance gain of 4.24%.

Table 5.11 summarizes the results for NBF. For NBF with the 144 data set, on average, there is a gain of 3.41%. For the m14b data set, on average, there is a

Table 5.11: Summary of the percentage Gains relative to the baseline using the M5 simulator for NBF for each input data sets with across all L1 cache sizes for L2 cache sizes ($128kB - 4MB$).

Graph	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
144	6.16	4.37	3.42	3.27	1.90	1.35	3.41
m14b	2.51	1.89	2.06	1.94	1.77	0.26	1.74
auto	9.47	7.00	7.13	4.97	5.37	5.37	6.55
Avg. NBF	6.05	4.42	4.20	3.39	3.01	2.32	3.90

gain of 1.74%. For the auto data set, on average, there is a gain of 6.55%. NBF, on average, has a performance gain of 3.90%.

Table 5.12 summarizes the results for MOLDYN. For MOLDYNwith the 144 data set, on average, there is a gain of 3.46%. For the m14b data set, on average, there is a gain of 1.23%. For the auto data set, on average, there is a gain of 2.04%. MOLDYN, on average, has a performance gain of 2.24%.

Table 5.13 summarizes the results for all problems IRREG, NBF, and MOLDYN. For the 144 data set, on average, there is a gain of 3.45%. For the m14b data set, on average, there is a gain of 1.65%. For the auto data set, on average, there is a gain of 5.29%. Over all benchmarks and all input data sets, on average, there is a performance gain of 3.46%. There is no correlation between problem size and gain.

Table 5.12: Summary of the percentage Gains relative to the baseline using the M5 simulator for MOLDYN for each input data sets with across all L1 cache sizes for L2 cache sizes ($128kB - 4MB$).

Graph	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
144	4.71	3.66	3.17	3.16	3.04	3.01	3.46
m14b	1.97	1.17	1.03	1.00	1.10	1.11	1.23
auto	3.33	2.48	2.29	1.52	1.36	1.28	2.04
Avg. MOLDYN	3.33	2.44	2.16	1.89	1.84	1.80	2.24

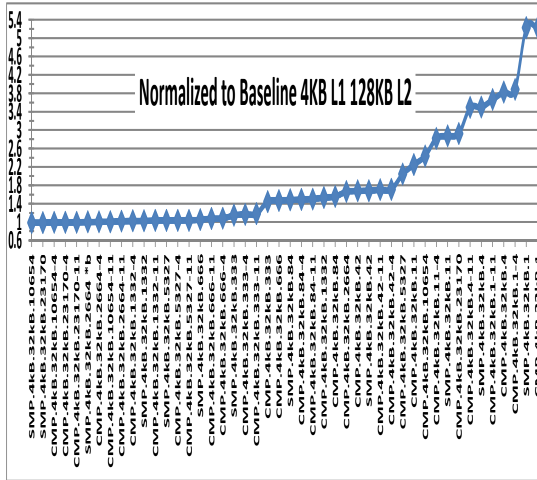
Table 5.13: Summary of the percentage Gains relative to the baseline using the M5 simulator for all benchmarks for each input data sets with across all L1 cache sizes for L2 cache sizes ($128kB - 4MB$).

Graph	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
Avg. 144	5.41	4.03	3.34	3.20	2.69	2.01	3.45
Avg. m14b	2.59	1.81	1.72	1.58	1.39	0.82	1.65
Avg. auto	8.09	5.88	5.56	4.14	4.13	3.94	5.29
Avg. all	5.36	3.91	3.54	2.97	2.74	2.25	3.46

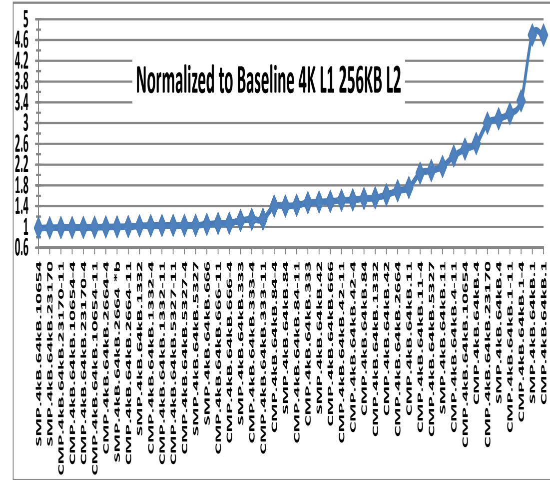
144, the smallest input data set in terms of size, has the 2nd largest gain. Auto, the largest input data set in terms of size, has the largest gain. m14b, the middle sized problem, has the least performance gain. The conclusion here is that the gains are input data set dependent and the connectivity of the graph, the numbers of edges, the number of nodes, and the computation that happens in each benchmark control the gains we get which explains the gains we get under the different data sets.

The main point here is that this is a search problem for the best performance and unless the machine that will run the benchmark exists and the developer and the compiler have access to it, we cannot chose a partition size unless we somehow have a model or can somehow predict the performance given that we do not have access to the real machine.

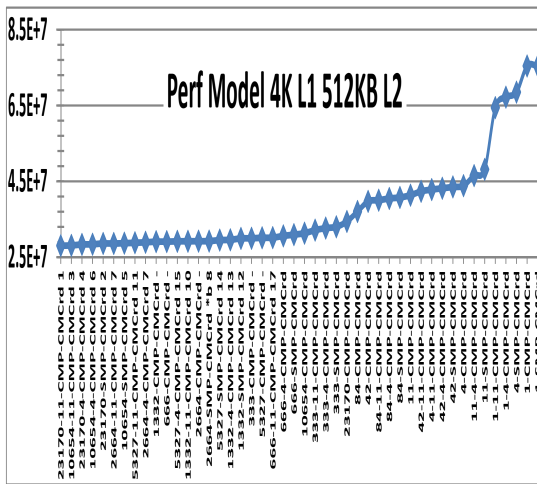
In Chapter 6, we will show a reuse distance based performance model that predicts performance given the structure and size of the caches.



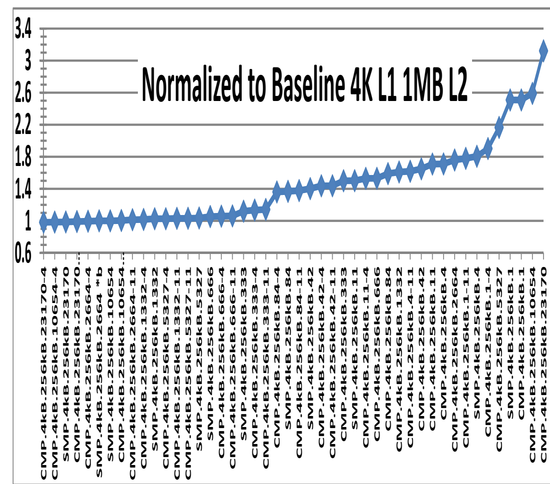
(a) 128kB L2.



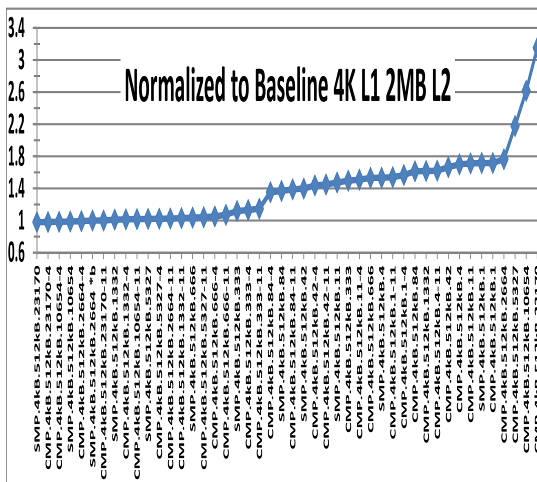
(b) 256kB L2.



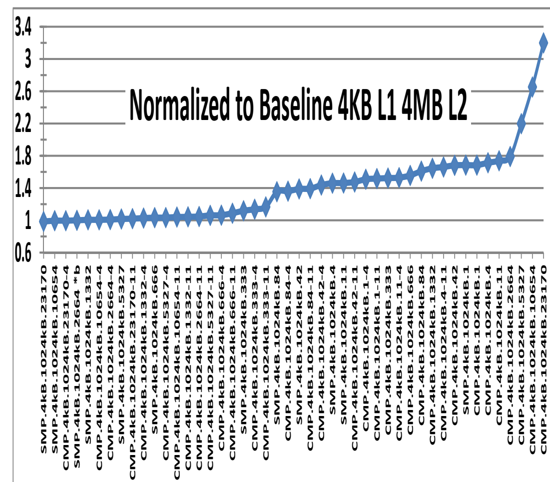
(c) 512kB L2.



(d) 1MB L2.

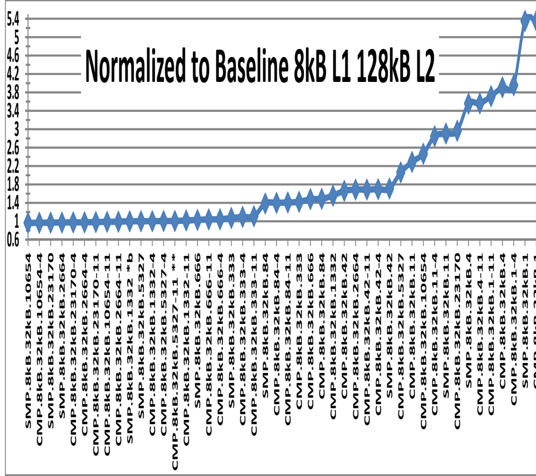


(e) 2MB L2.

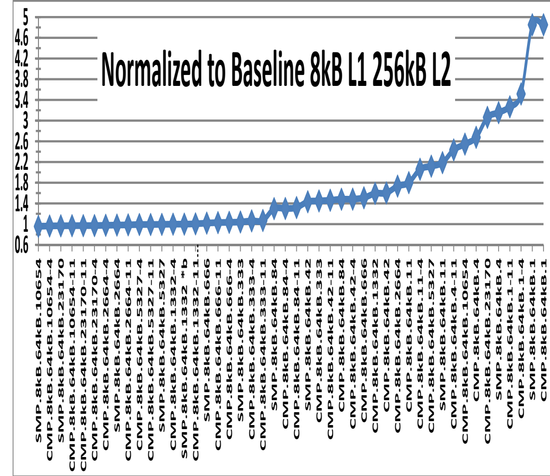


(f) 4MB L2.

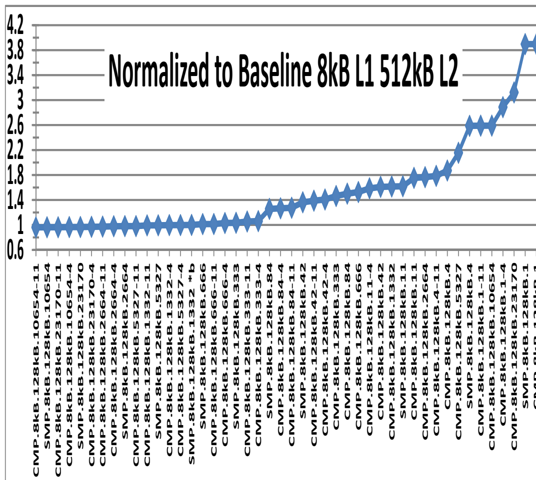
Figure 5.1: M5 results for IRREG under the **144** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



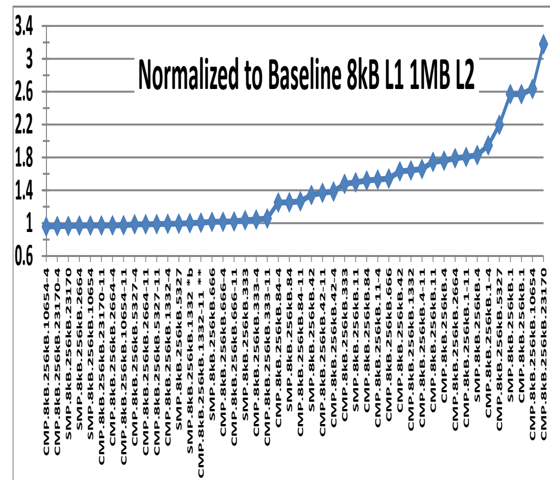
(a) 128kB L2.



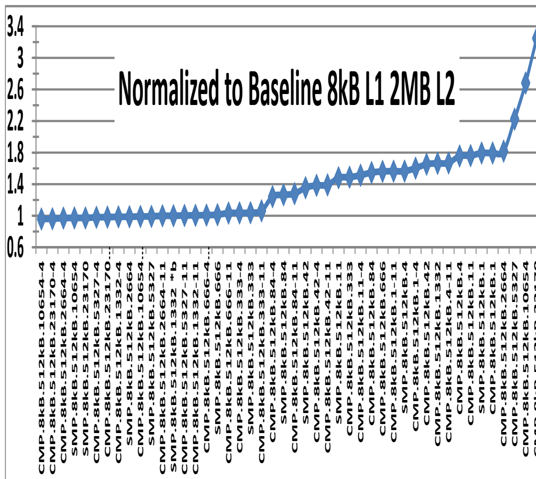
(b) 256kB L2.



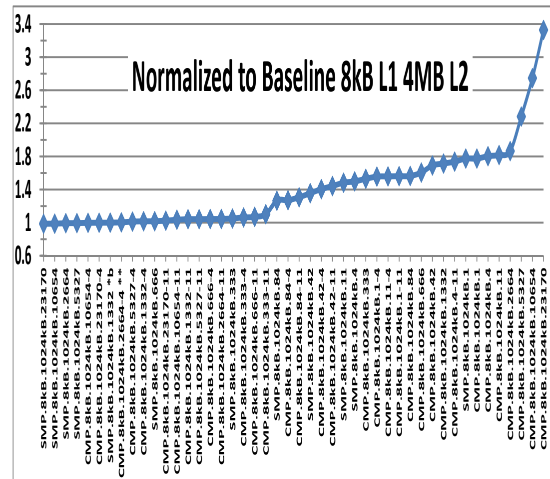
(c) 512kB L2.



(d) 1MB L2.

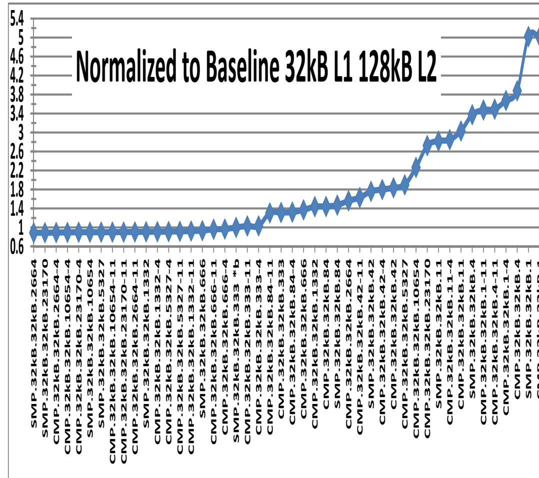


(e) 2MB L2.

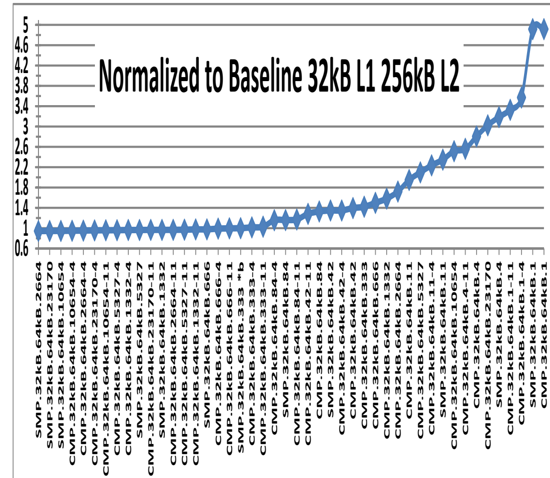


(f) 4MB L2.

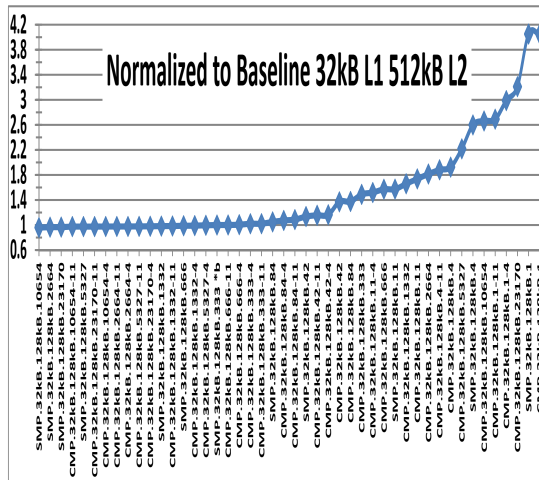
Figure 5.2: M5 results for IRREG under the **144** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



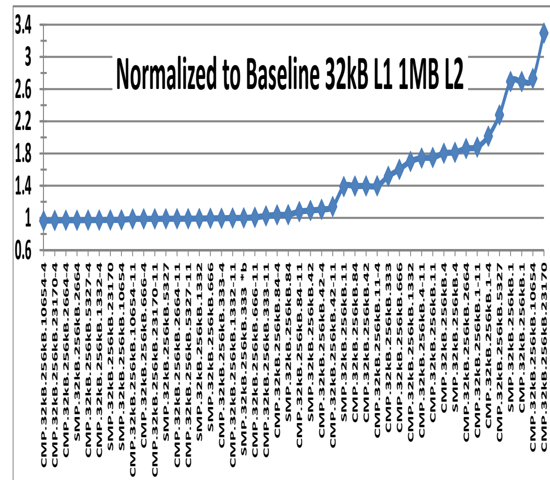
(a) 128kB L2.



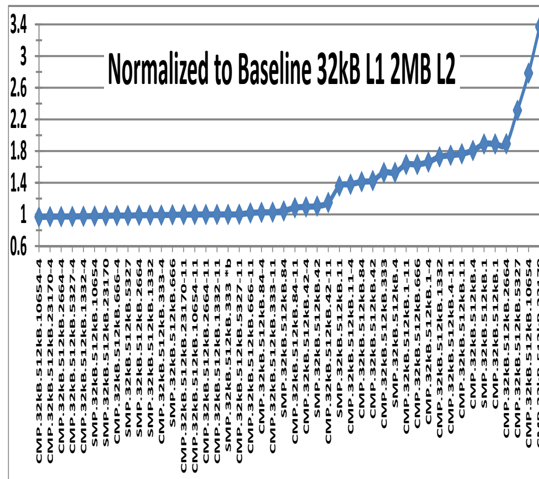
(b) 256kB L2.



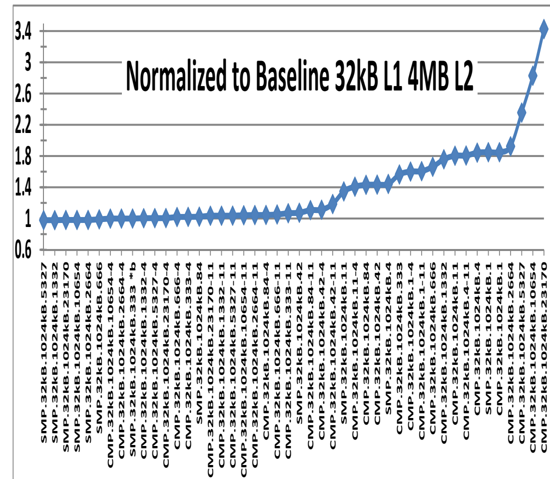
(c) 512kB L2.



(d) 1MB L2.

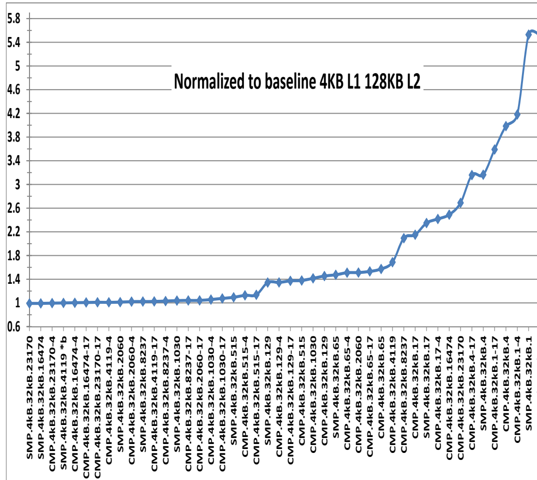


(e) 2MB L2.

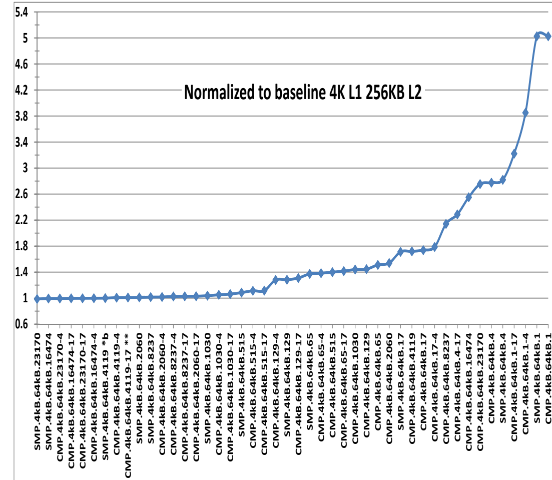


(f) 4MB L2.

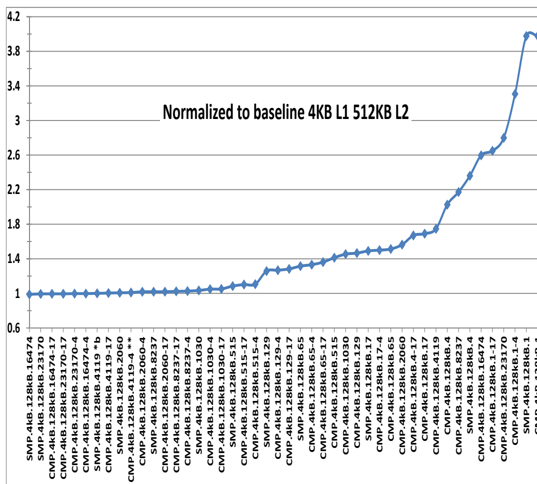
Figure 5.3: M5 results for IRREG under the **144** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



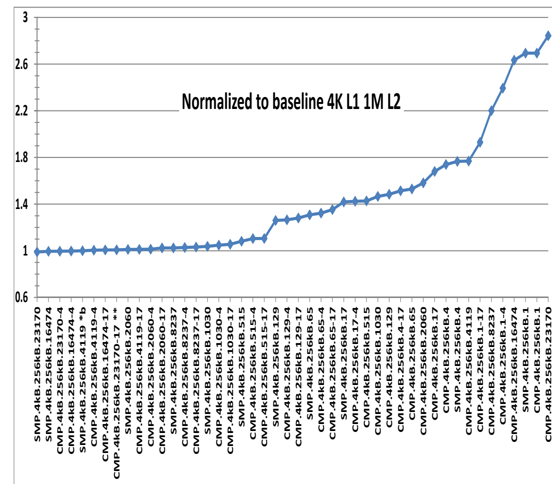
(a) 128kB L2.



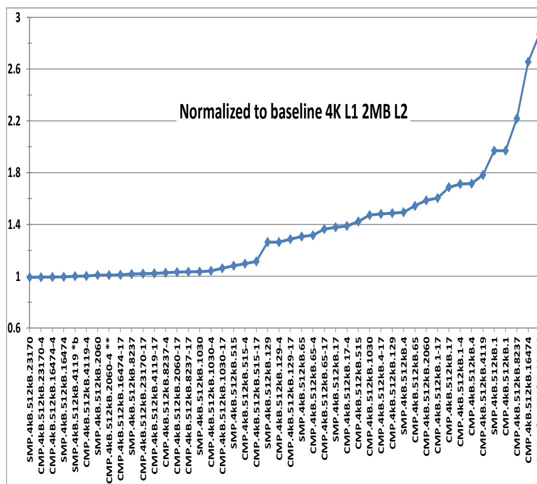
(b) 256kB L2.



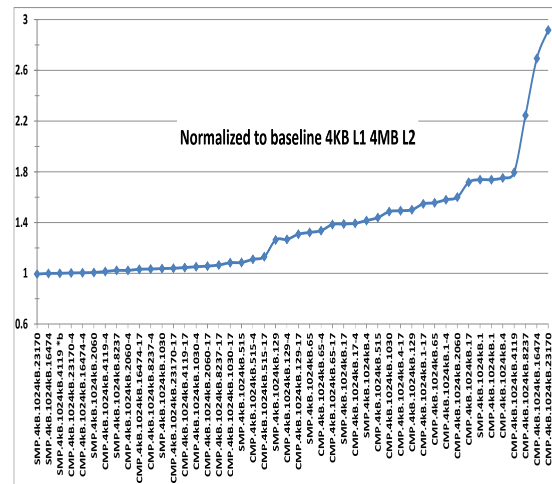
(c) 512kB L2.



(d) 1MB L2.

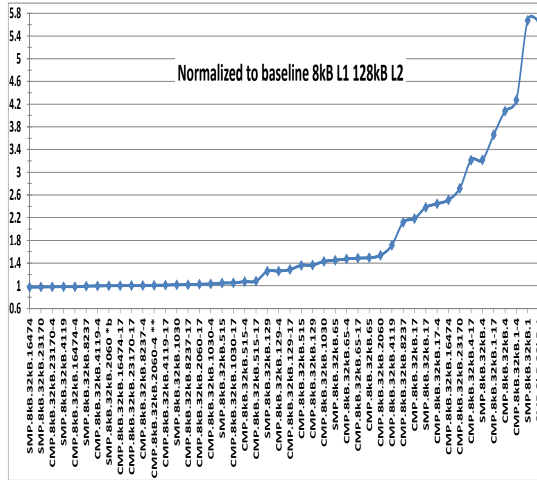


(e) 2MB L2.

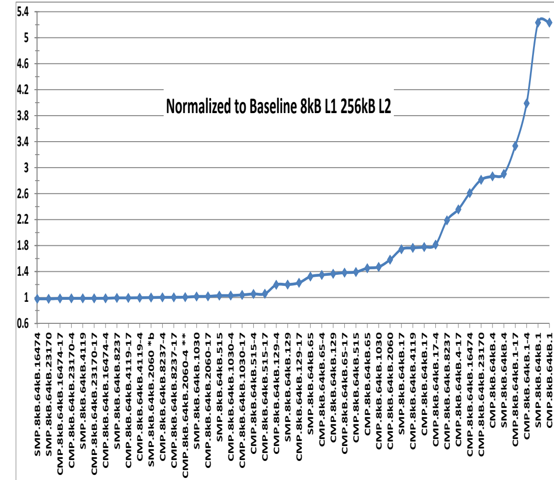


(f) 4MB L2.

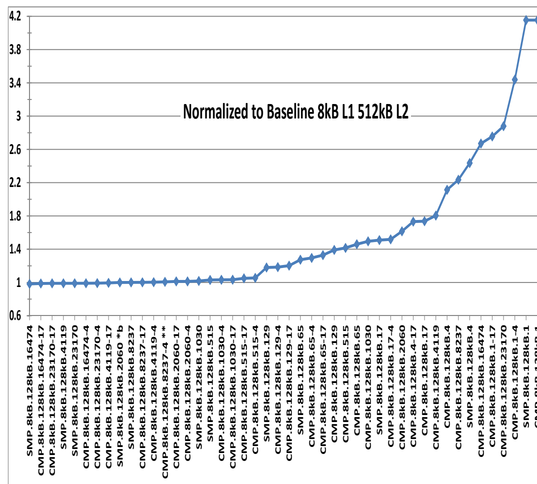
Figure 5.4: M5 results for IRREG under the **m14b** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



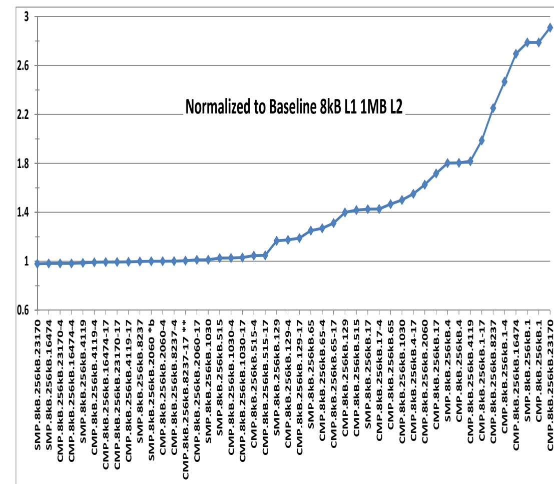
(a) 128kB L2.



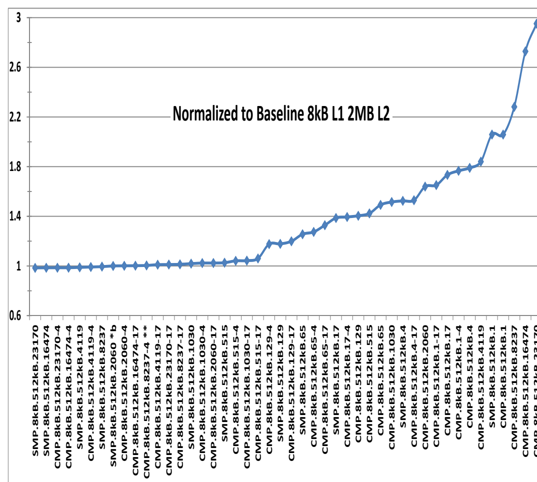
(b) 256kB L2.



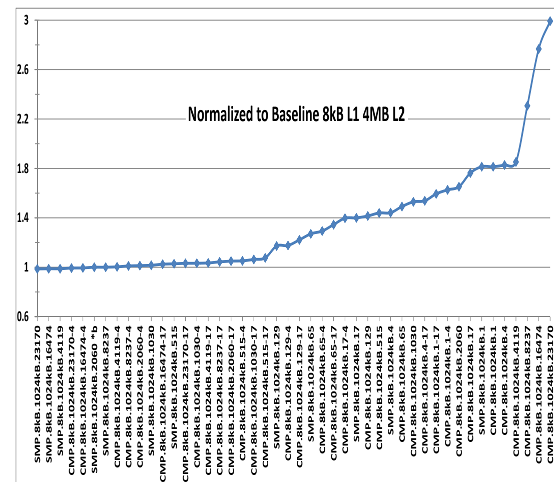
(c) 512kB L2.



(d) 1MB L2.

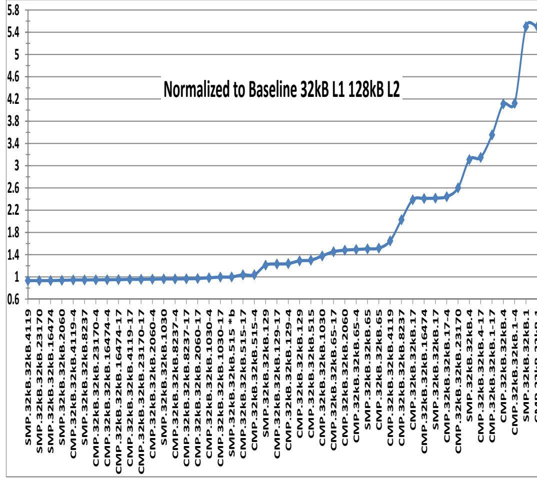


(e) 2MB L2.

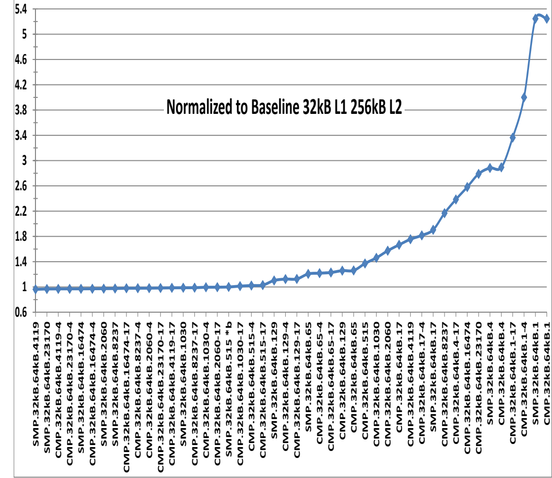


(f) 4MB L2.

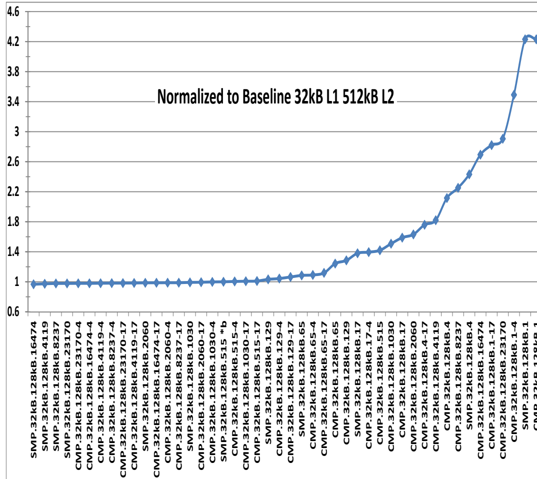
Figure 5.5: M5 results for IRREG under the **m14b** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



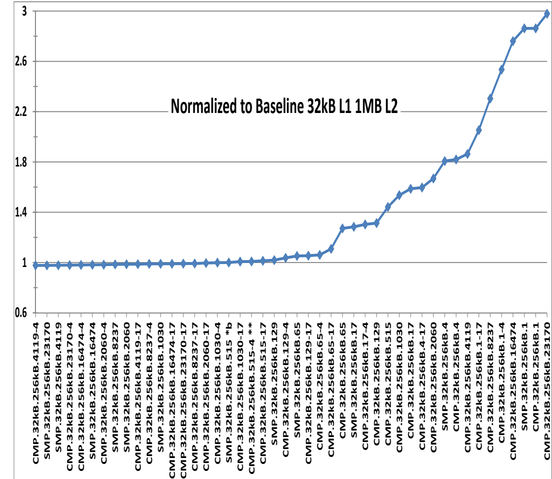
(a) 128kB L2.



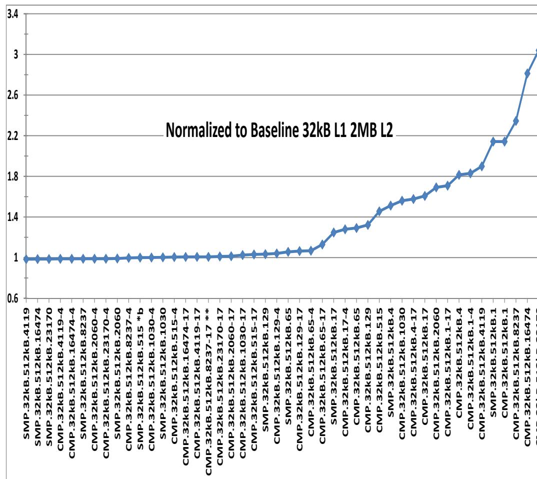
(b) 256kB L2.



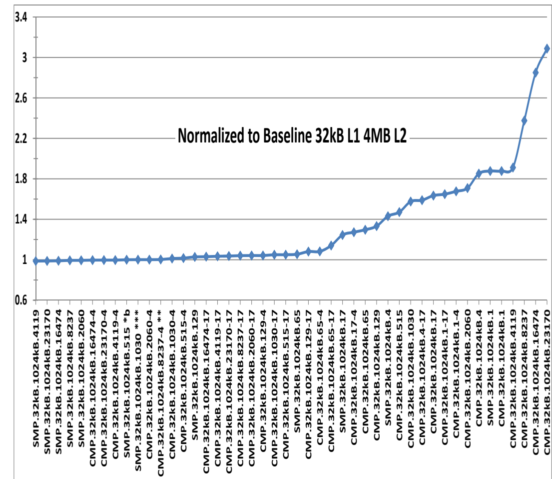
(c) 512kB L2.



(d) 1MB L2.

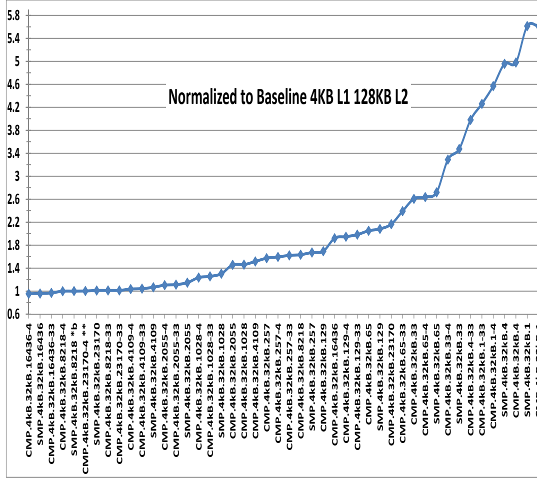


(e) 2MB L2.

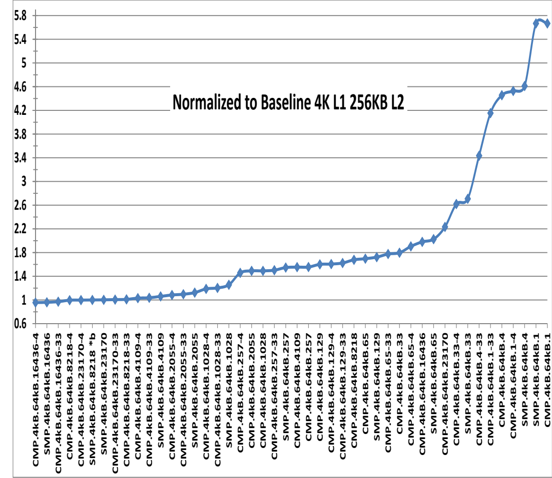


(f) 4MB L2.

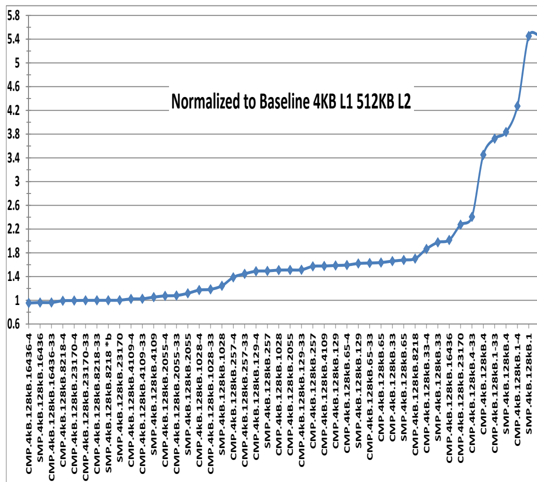
Figure 5.6: M5 results for Irreg under the **m14b** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



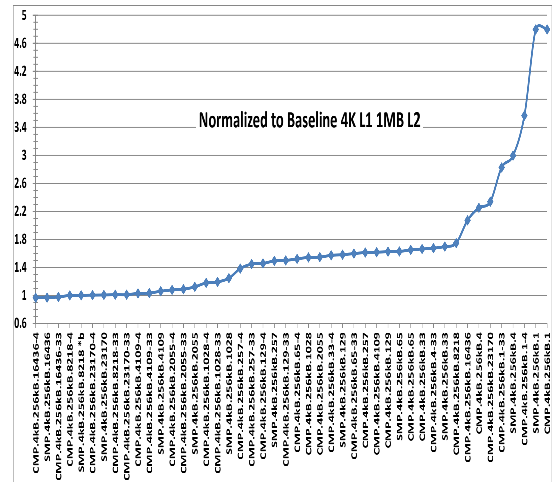
(a) 128kB L2.



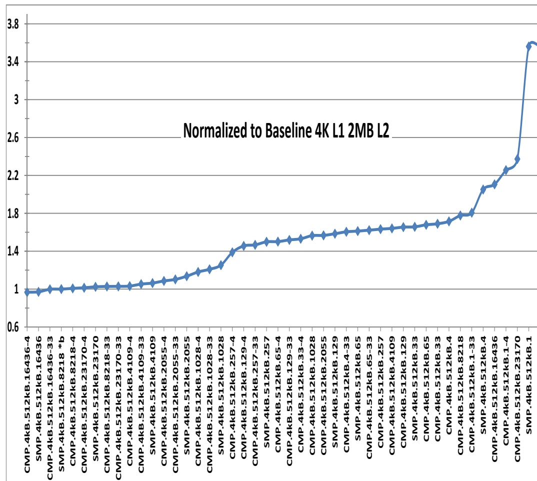
(b) 256kB L2.



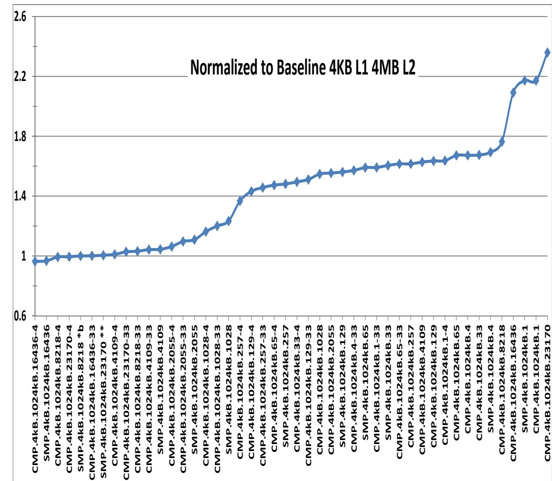
(c) 512kB L2.



(d) 1MB L2.

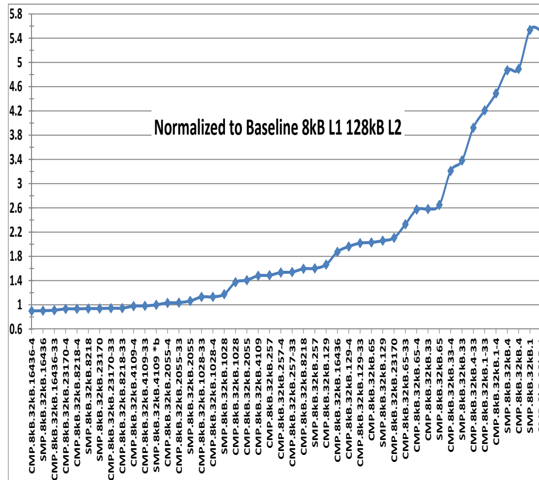


(e) 2MB L2.

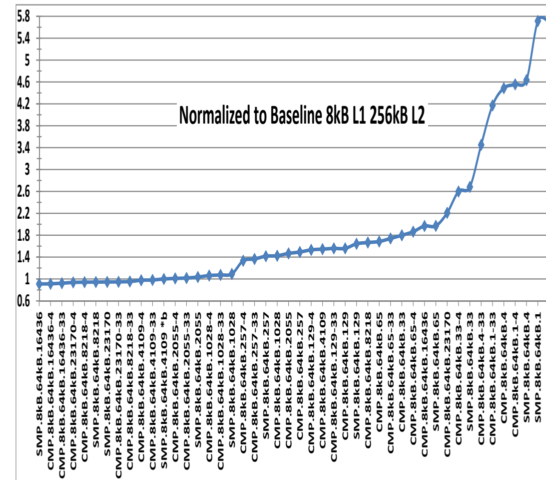


(f) 4MB L2.

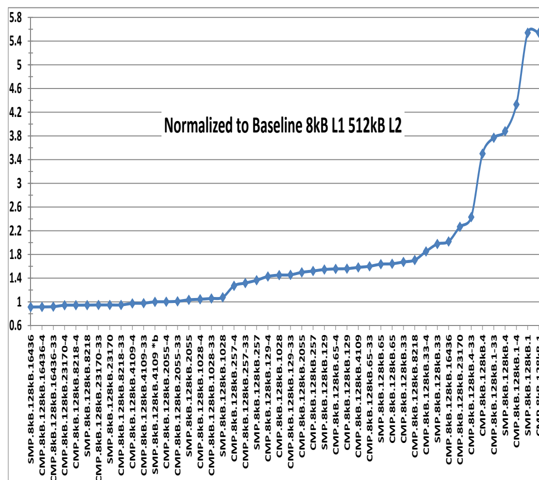
Figure 5.7: M5 results for IRREG under the **auto** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



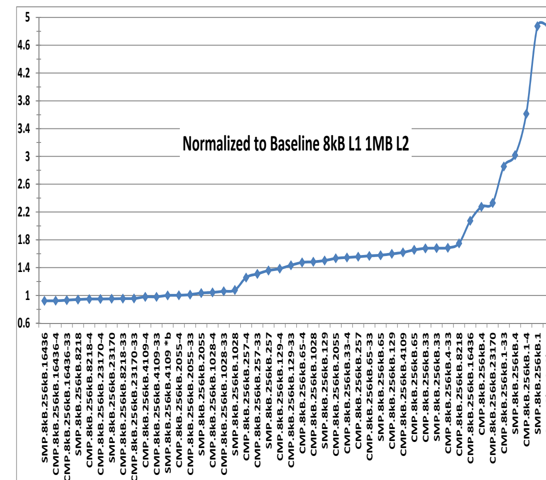
(a) 128kB L2.



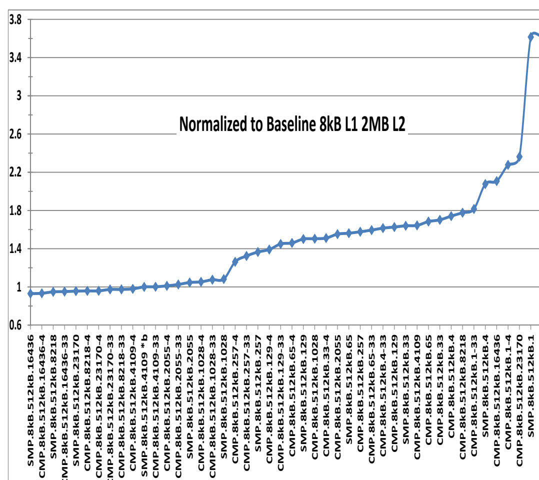
(b) 256kB L2.



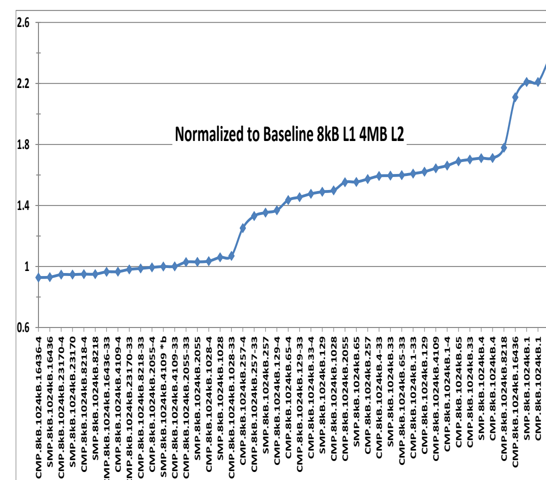
(c) 512kB L2.



(d) 1MB L2.

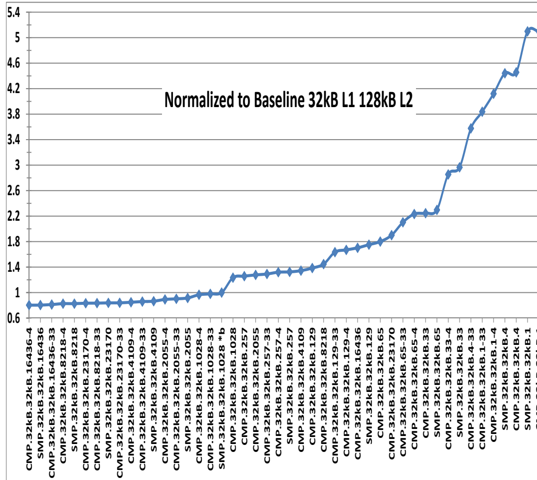


(e) 2MB L2.

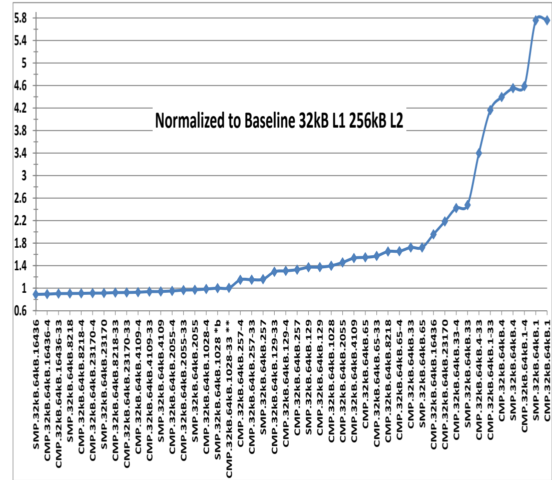


(f) 4MB L2.

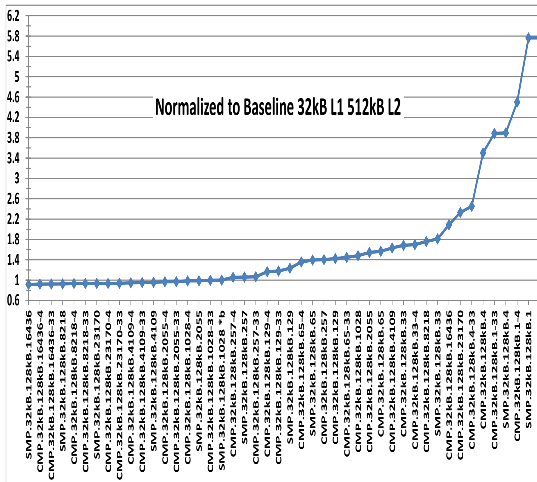
Figure 5.8: M5 results for IRREG under the **auto** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



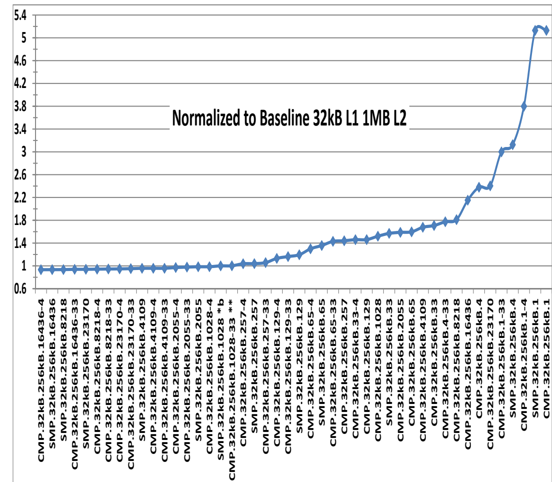
(a) 128kB L2.



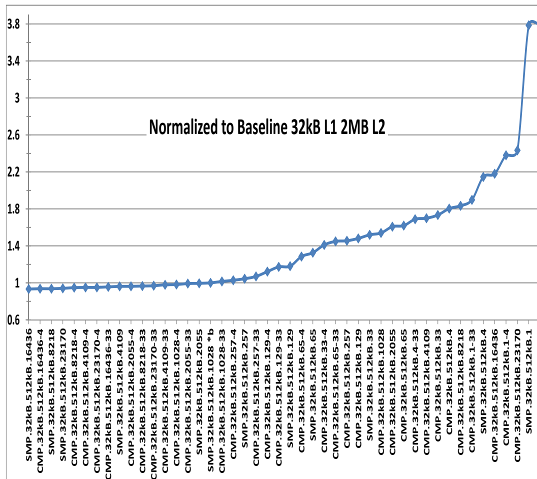
(b) 256kB L2.



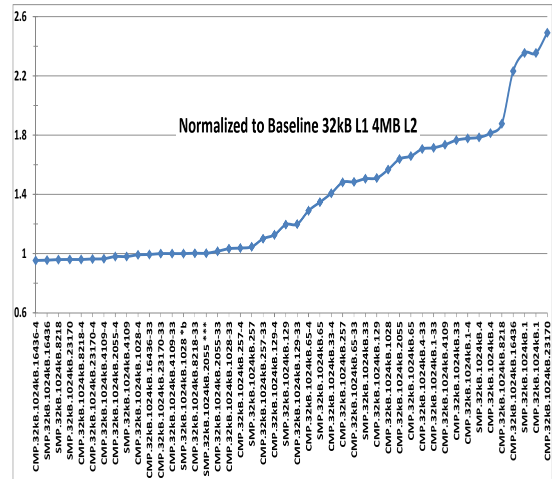
(c) 512kB L2.



(d) 1MB L2.

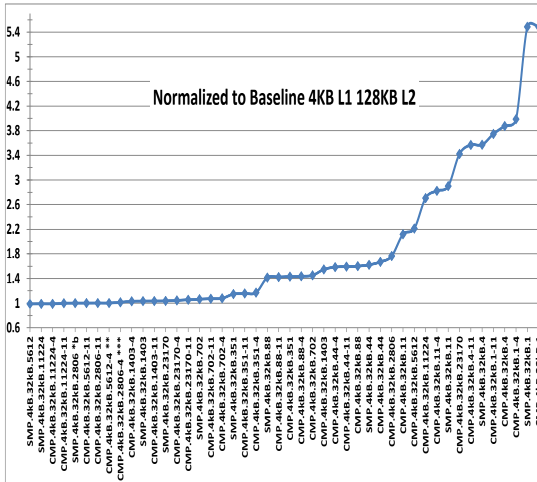


(e) 2MB L2.

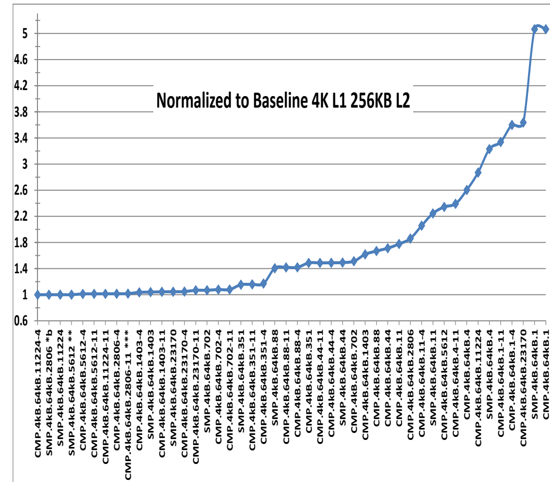


(f) 4MB L2.

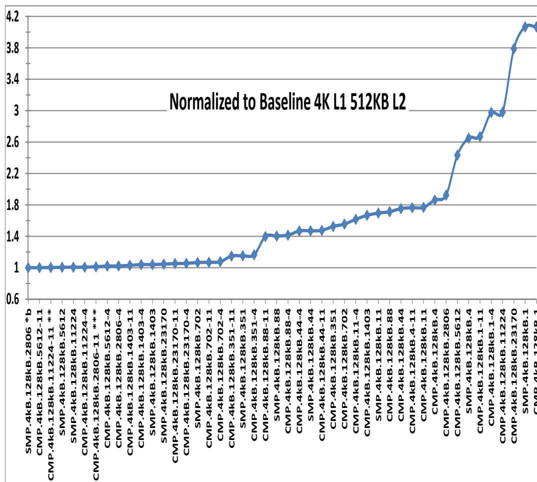
Figure 5.9: M5 results for IRREG under the **auto** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



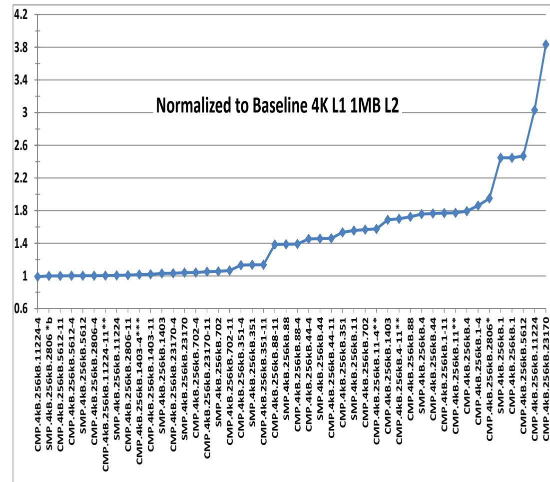
(a) 128kB L2.



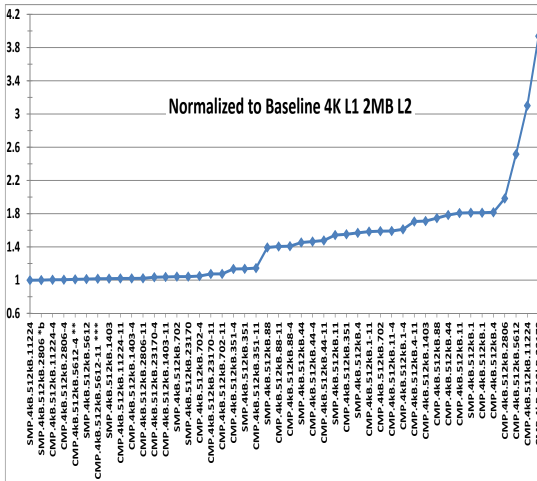
(b) 256kB L2.



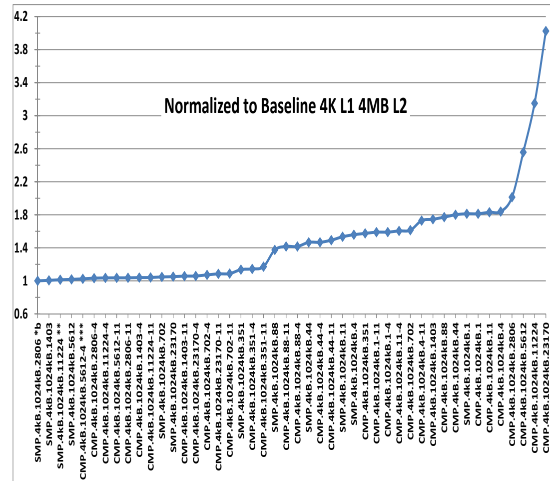
(c) 512kB L2.



(d) 1MB L2.

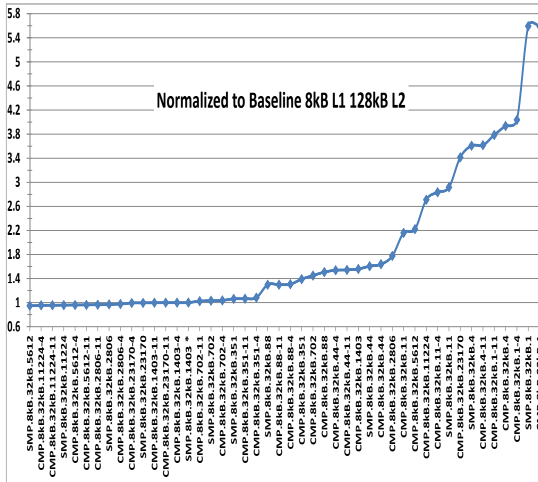


(e) 2MB L2.

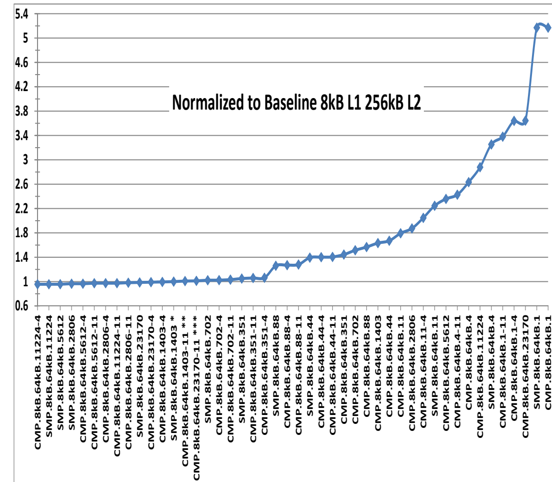


(f) 4MB L2.

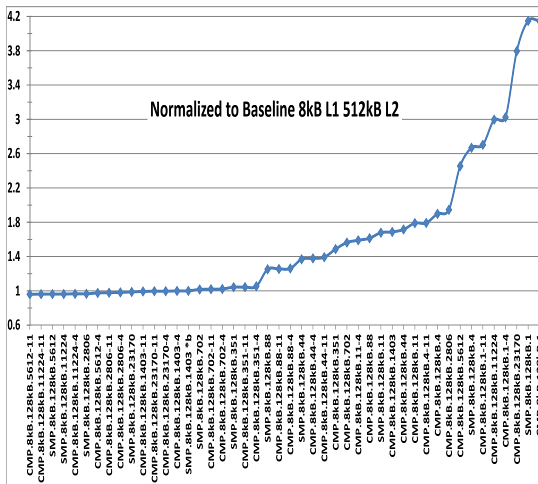
Figure 5.10: M5 results for NBF under the **144** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



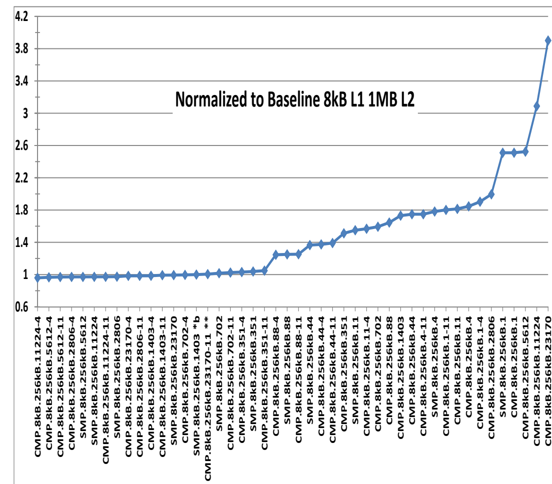
(a) 128kB L2.



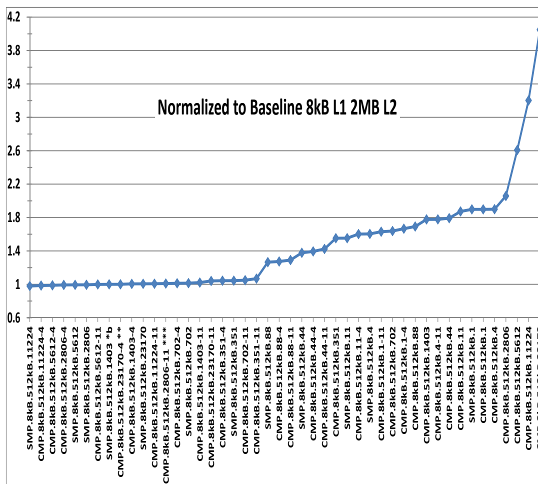
(b) 256kB L2.



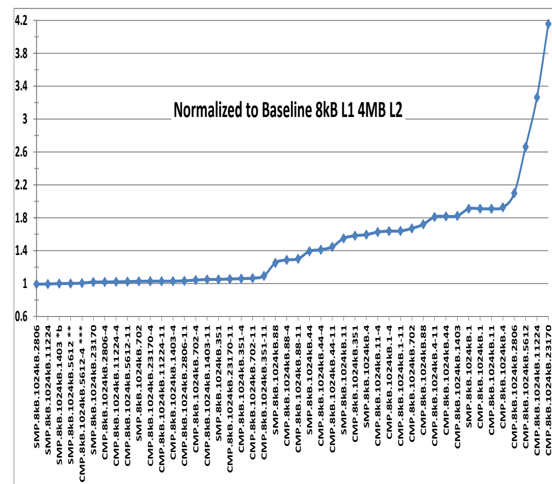
(c) 512kB L2.



(d) 1MB L2.

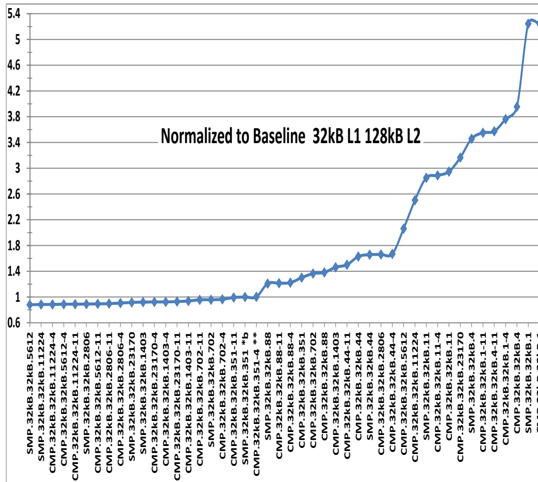


(e) 2MB L2.

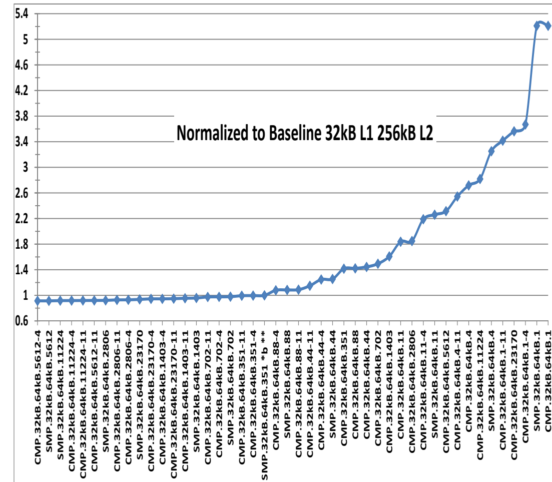


(f) 4MB L2.

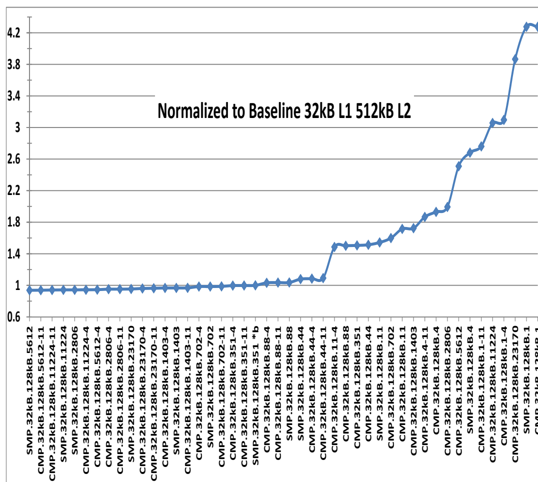
Figure 5.11: M5 results for NBF under the **144** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



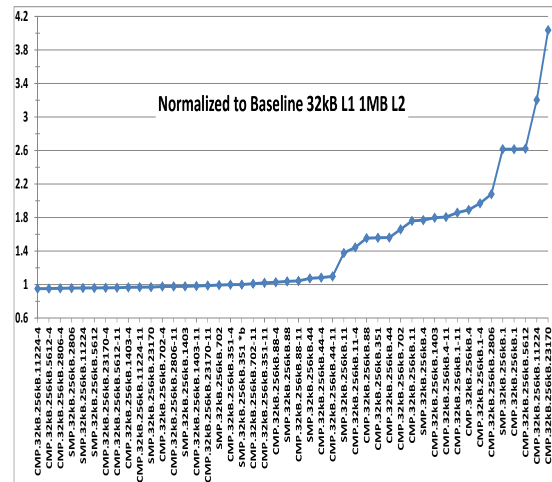
(a) 128kB L2.



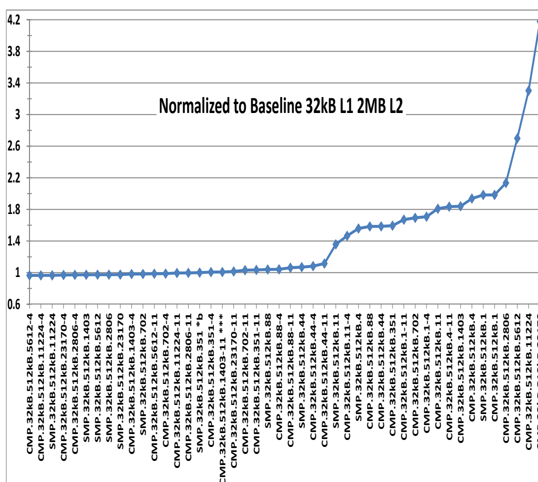
(b) 256kB L2.



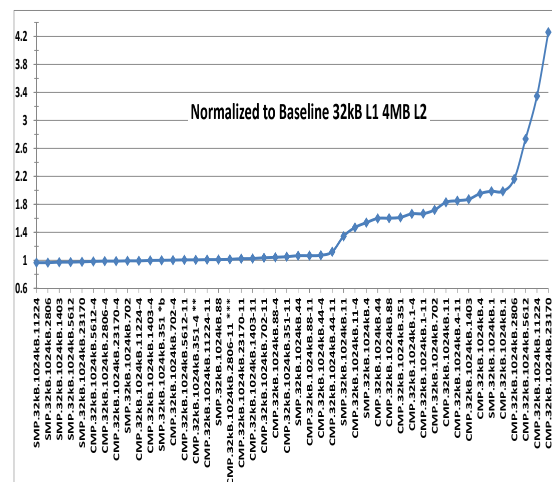
(c) 512kB L2.



(d) 1MB L2.

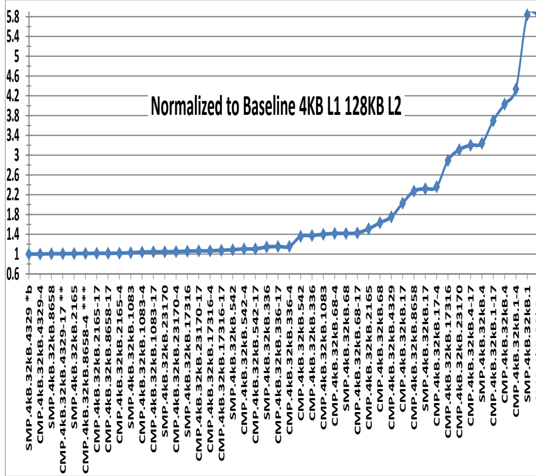


(e) 2MB L2.

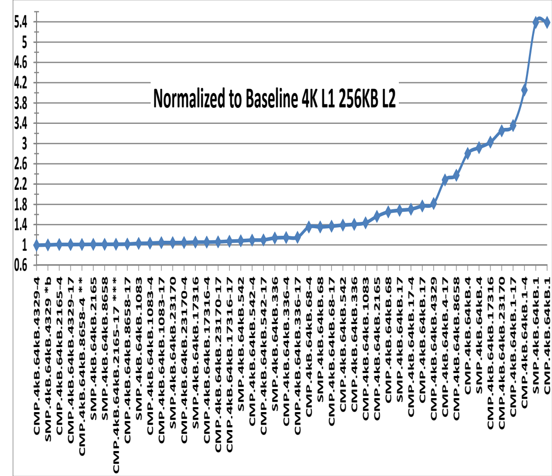


(f) 4MB L2.

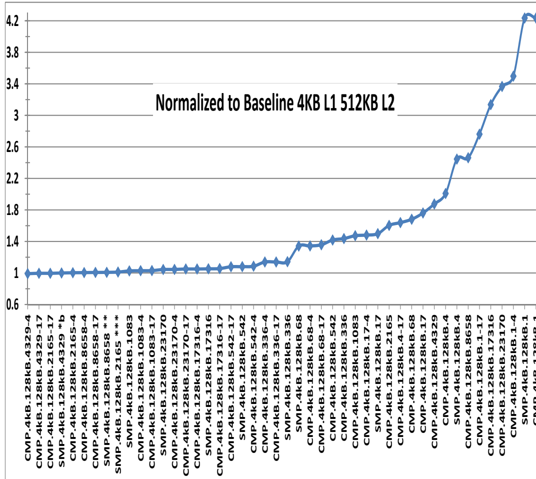
Figure 5.12: M5 results for NBF under the **144** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



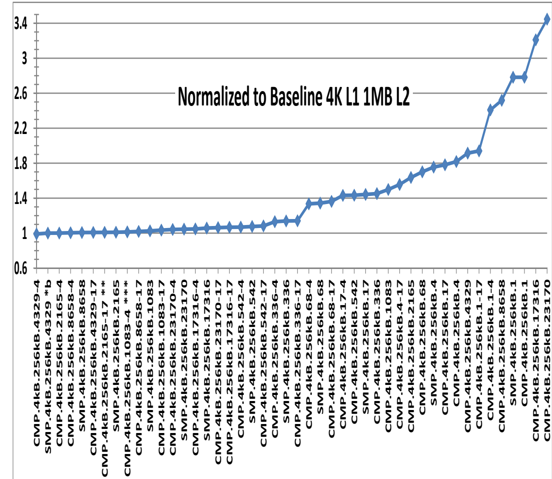
(a) 128kB L2.



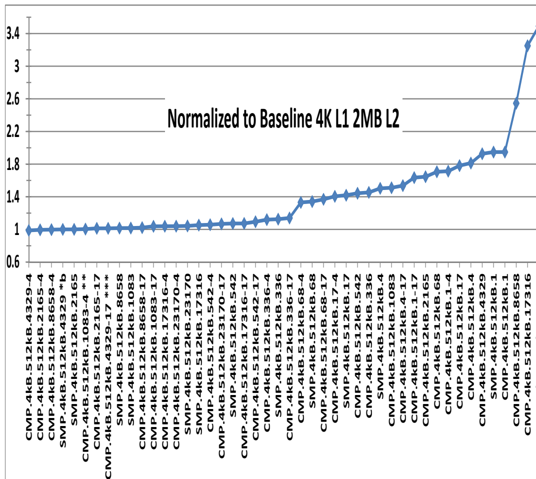
(b) 256kB L2.



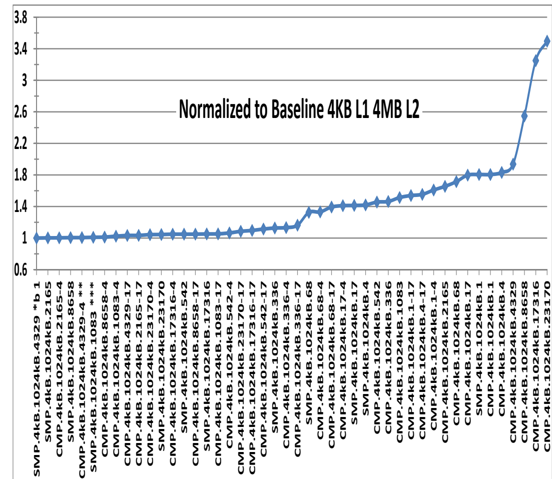
(c) 512kB L2.



(d) 1MB L2.

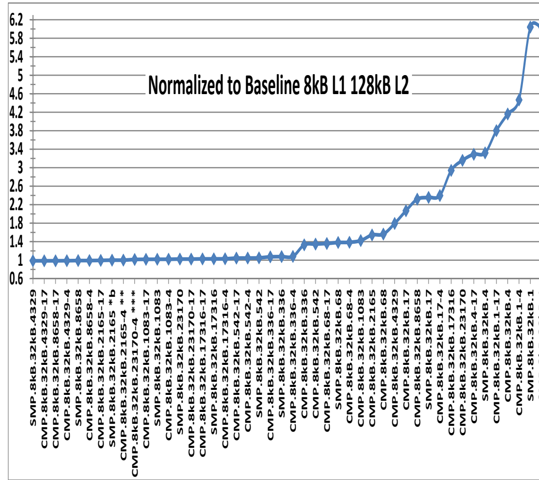


(e) 2MB L2.

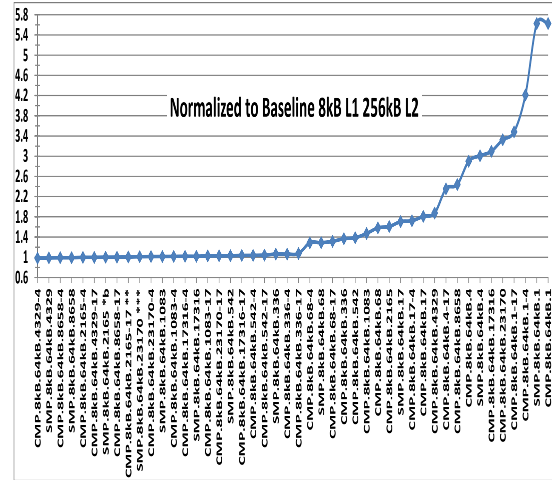


(f) 4MB L2.

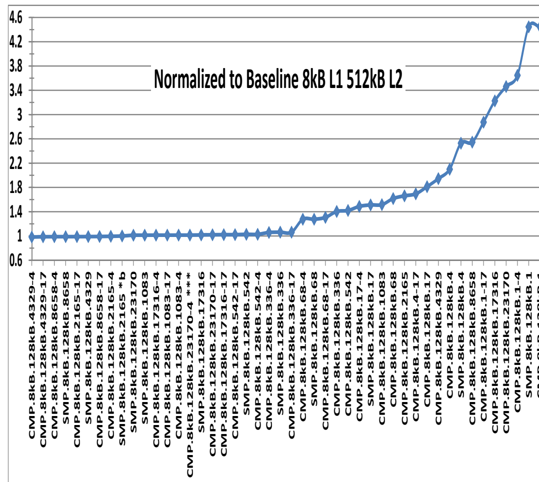
Figure 5.13: M5 results for NBF under the **m14b** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



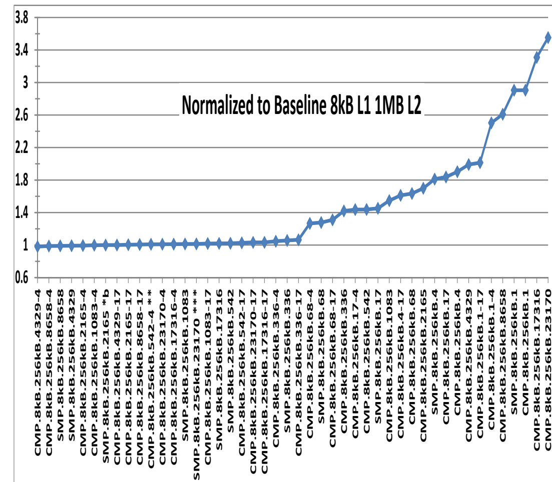
(a) 128kB L2.



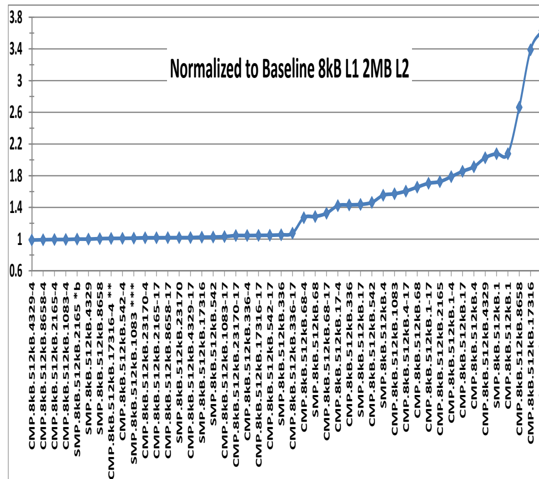
(b) 256kB L2.



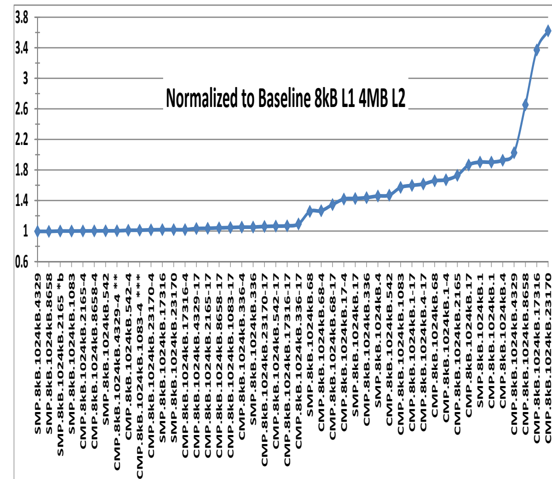
(c) 512kB L2.



(d) 1MB L2.

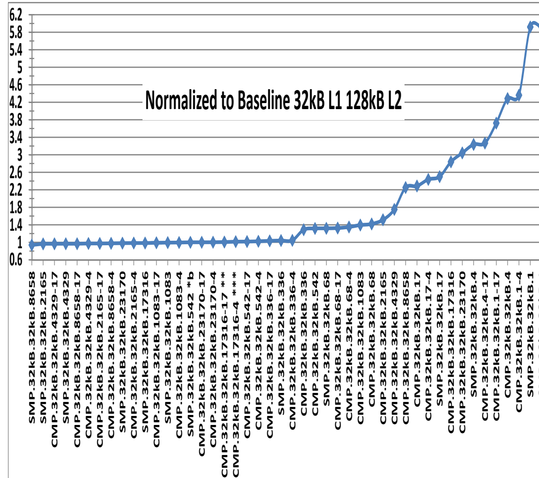


(e) 2MB L2.

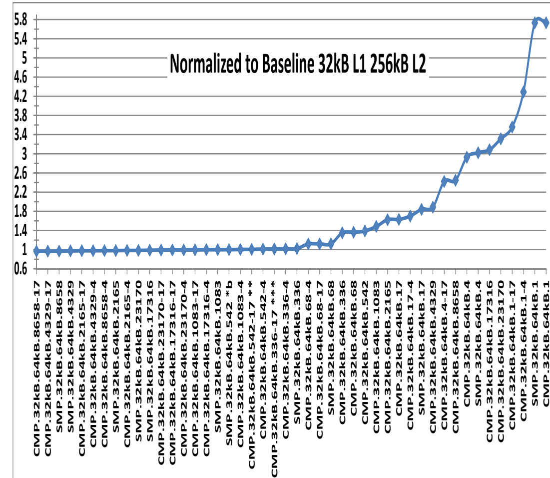


(f) 4MB L2.

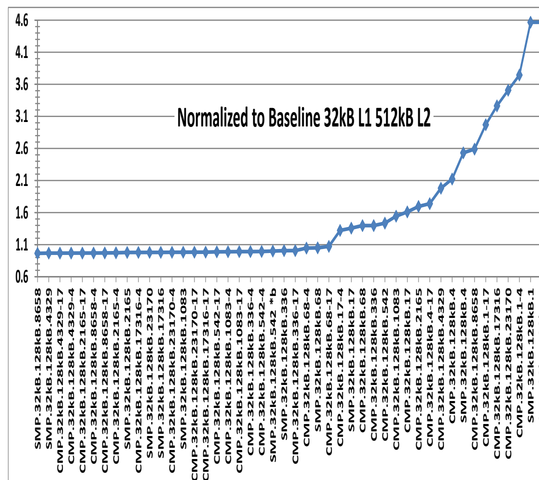
Figure 5.14: M5 results for NBF under the **m14b** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



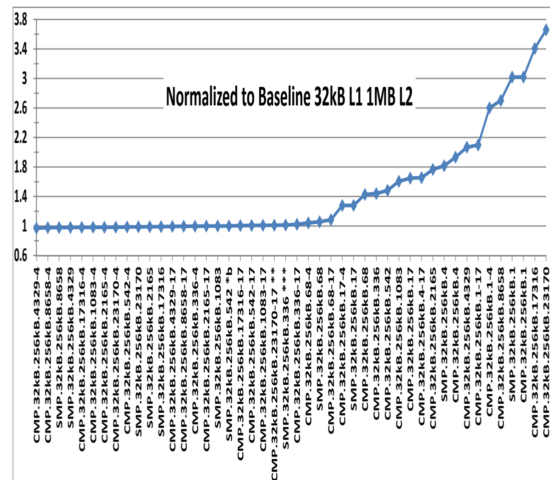
(a) 128kB L2.



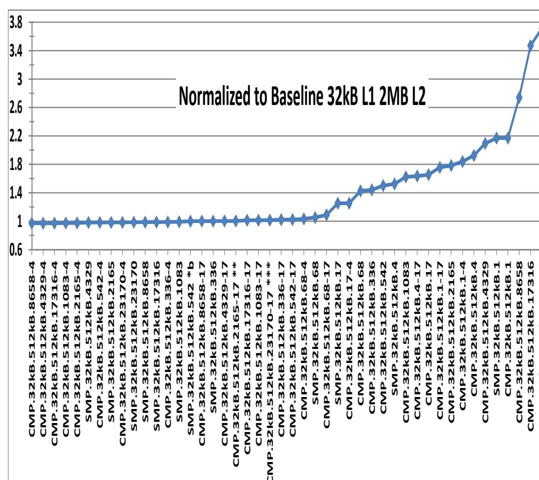
(b) 256kB L2.



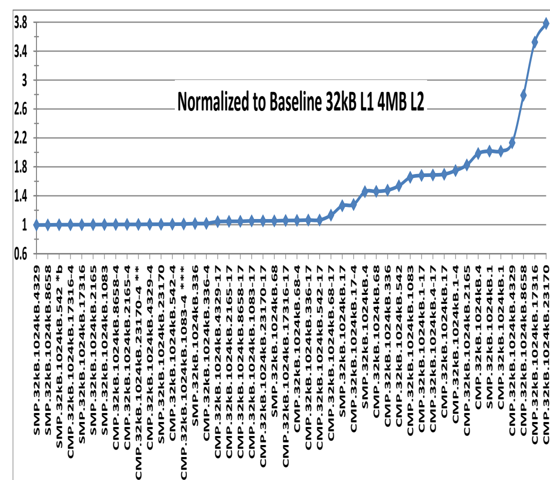
(c) 512kB L2.



(d) 1MB L2.

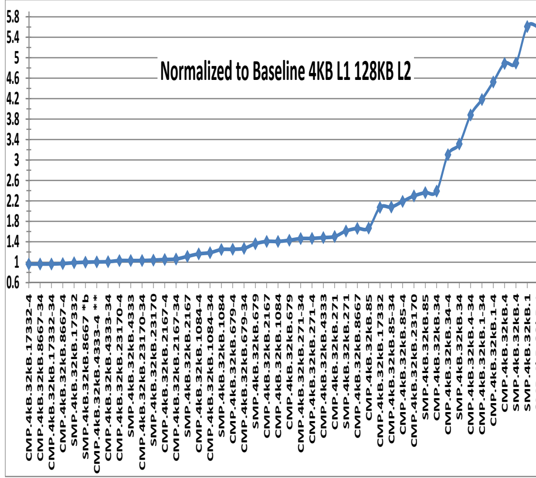


(e) 2MB L2.

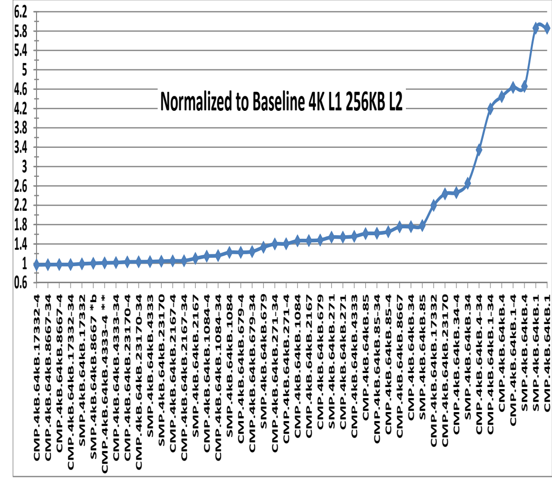


(f) 4MB L2.

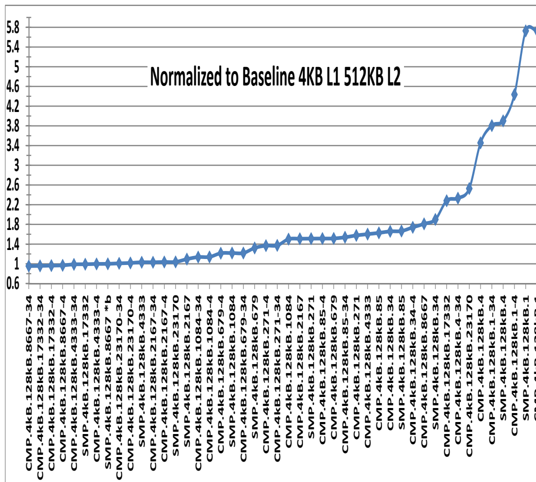
Figure 5.15: M5 results for NBF under the **m14b** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



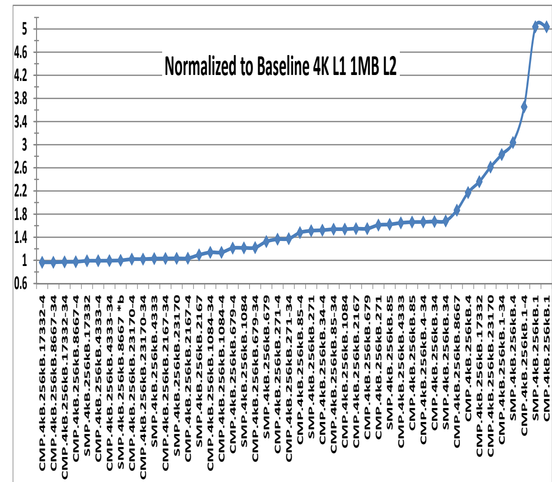
(a) 128kB L2.



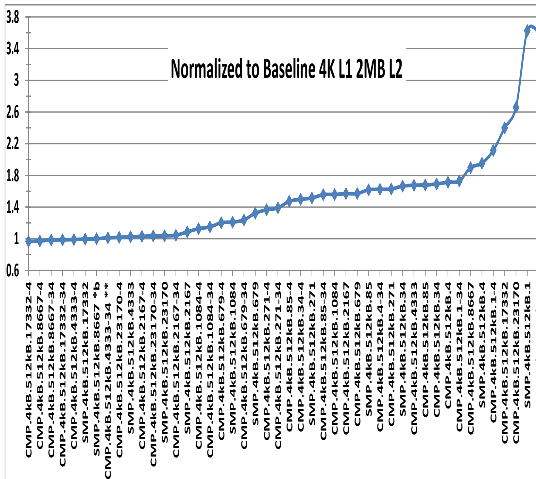
(b) 256kB L2.



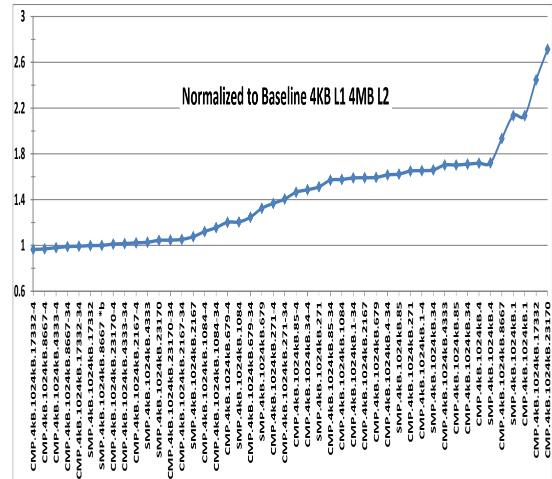
(c) 512kB L2.



(d) 1MB L2.

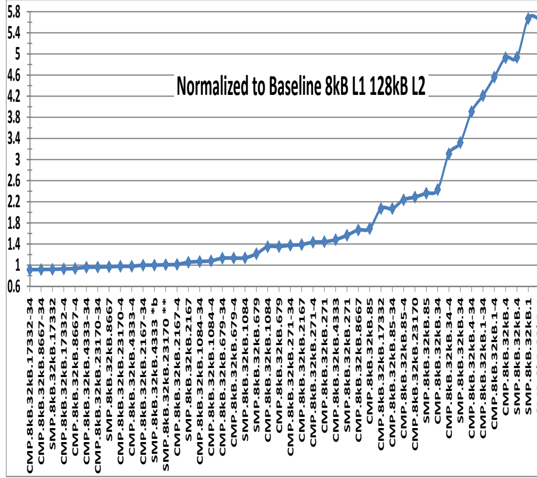


(e) 2MB L2.

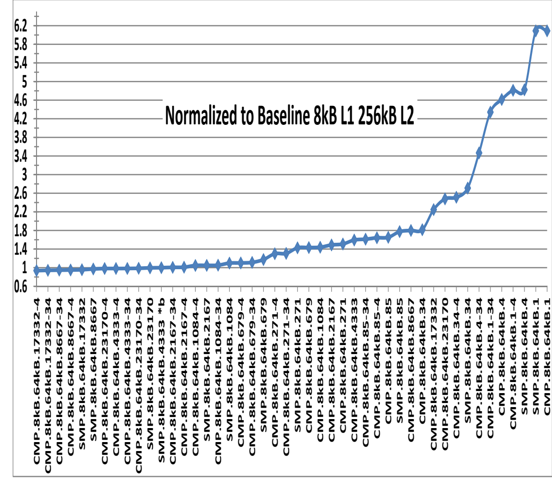


(f) 4MB L2.

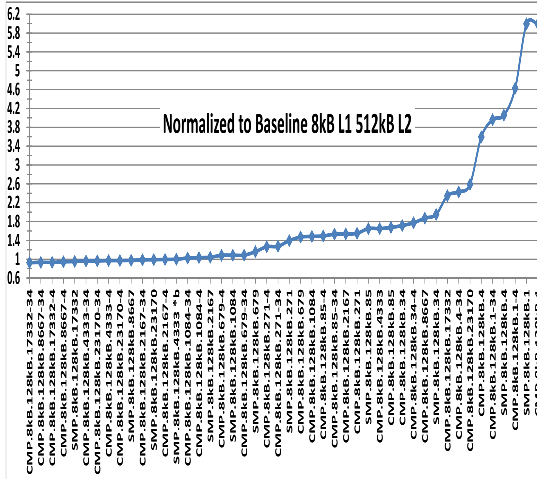
Figure 5.16: M5 results for NBF under the **auto** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



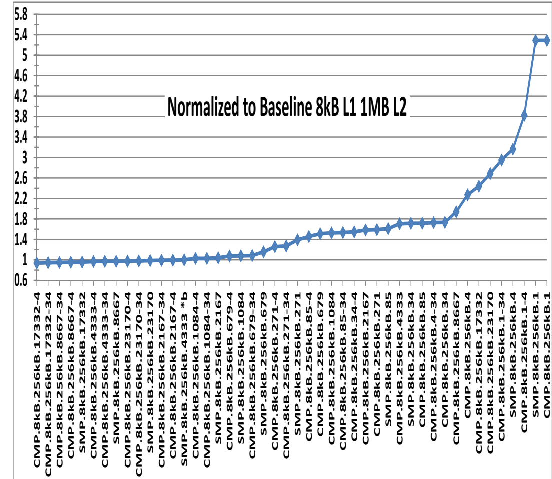
(a) 128kB L2.



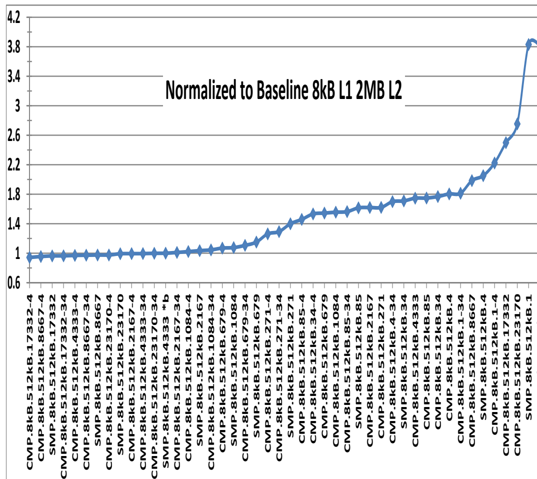
(b) 256kB L2.



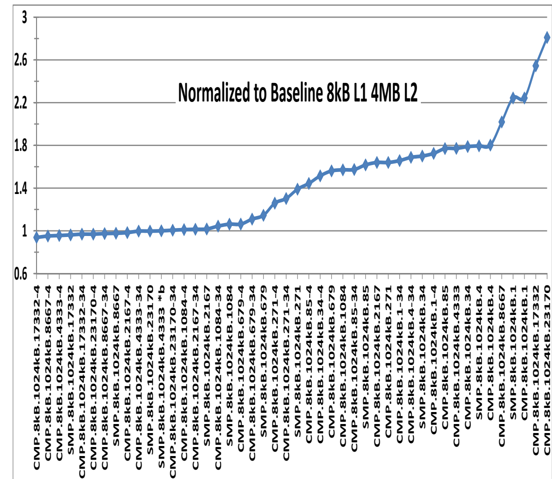
(c) 512kB L2.



(d) 1MB L2.

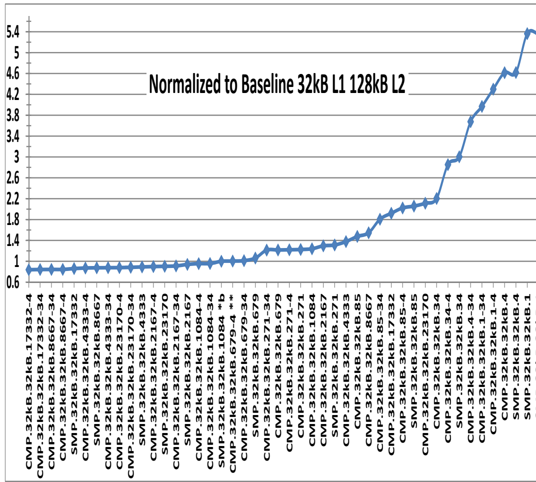


(e) 2MB L2.

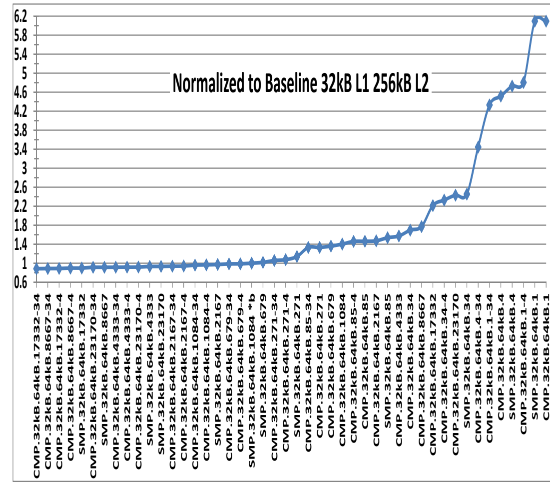


(f) 4MB L2.

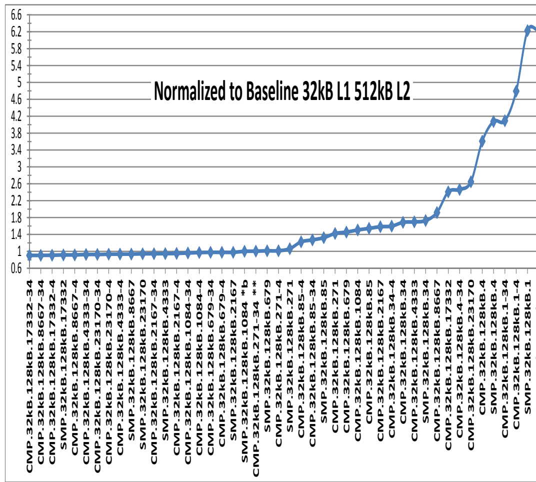
Figure 5.17: M5 results for NBF under the **auto** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



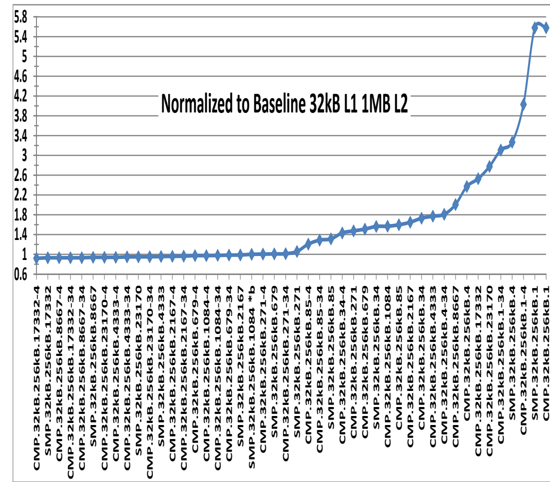
(a) 128kB L2.



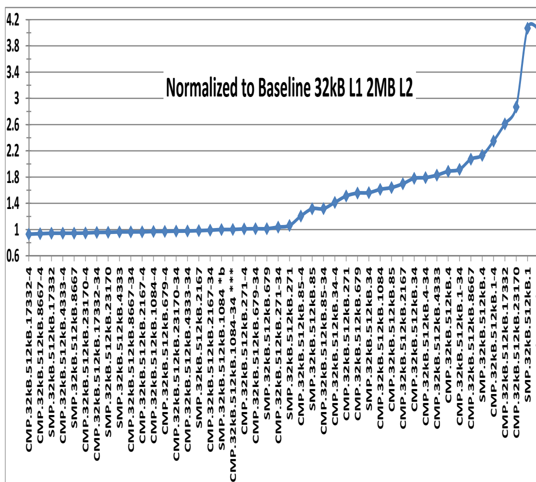
(b) 256kB L2.



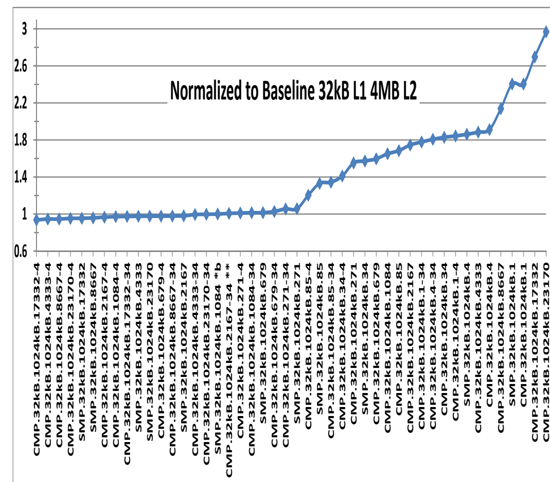
(c) 512kB L2.



(d) 1MB L2.

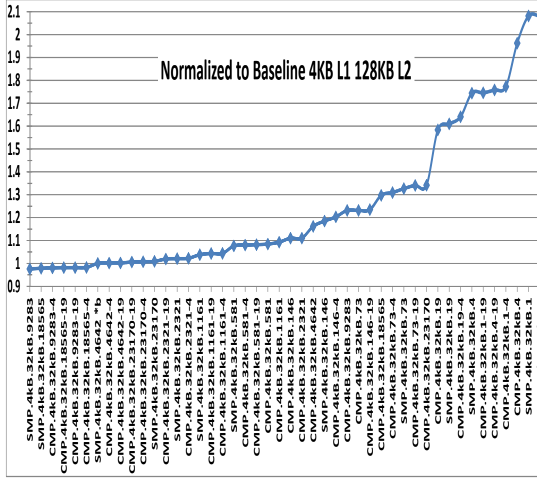


(e) 2MB L2.

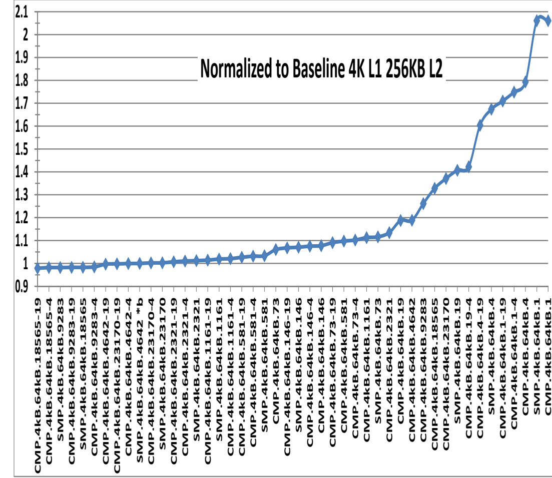


(f) 4MB L2.

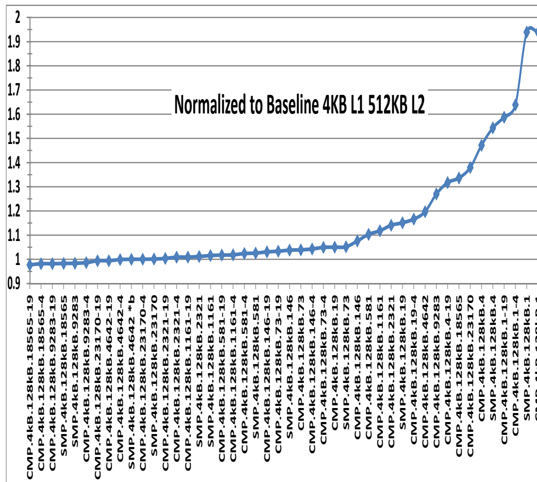
Figure 5.18: M5 results for NBF under the **auto** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



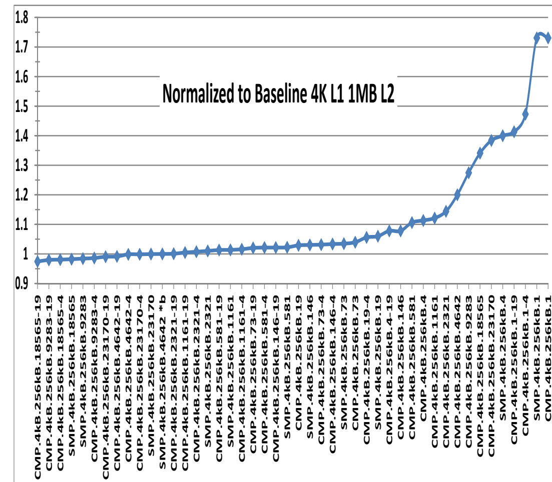
(a) 128kB L2.



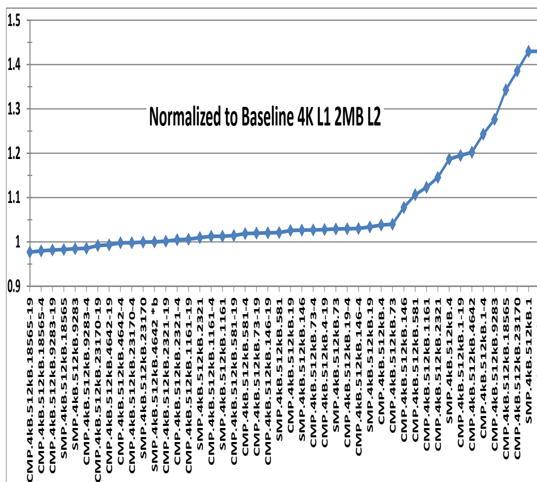
(b) 256kB L2.



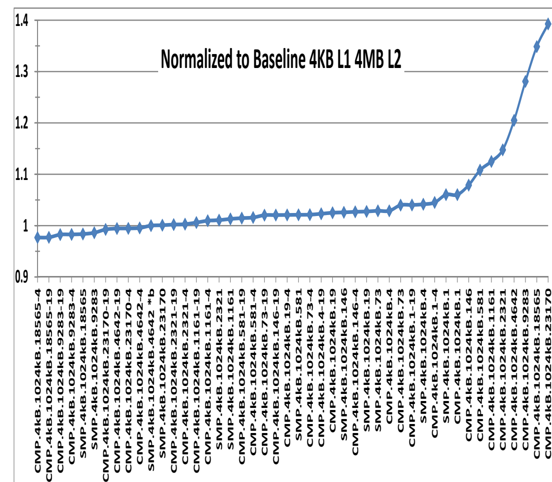
(c) 512kB L2.



(d) 1MB L2.

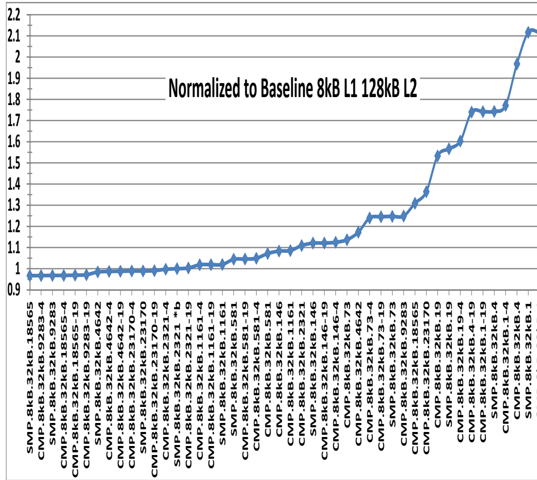


(e) 2MB L2.

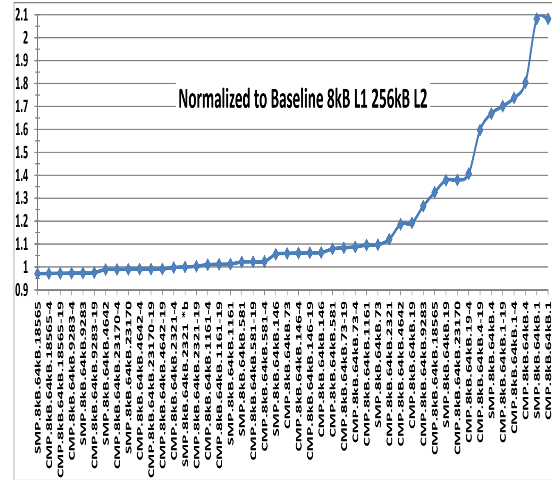


(f) 4MB L2.

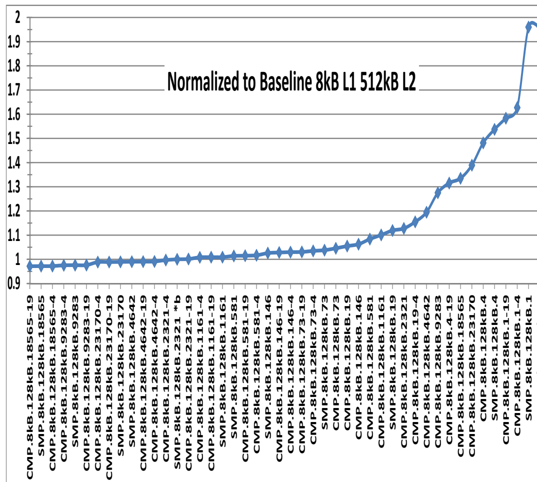
Figure 5.19: M5 results for MOLDYN under the **144** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



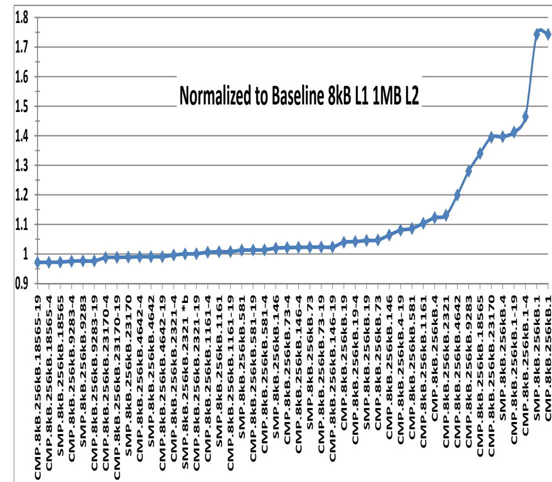
(a) 128kB L2.



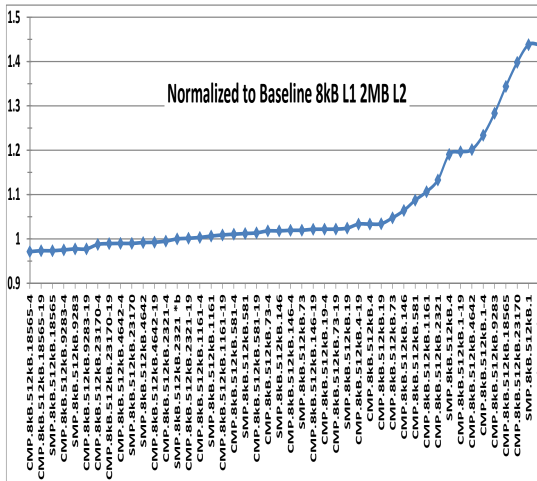
(b) 256kB L2.



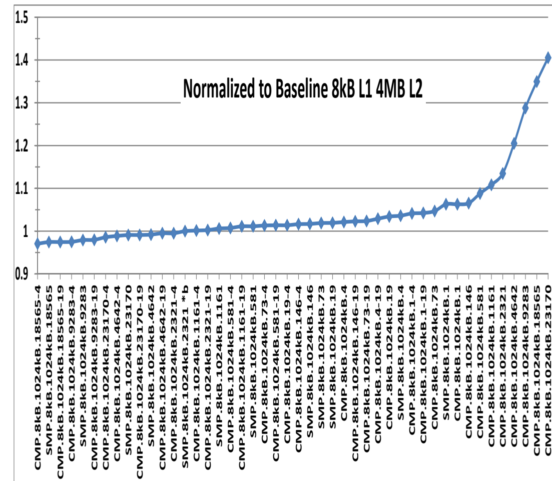
(c) 512kB L2.



(d) 1MB L2.

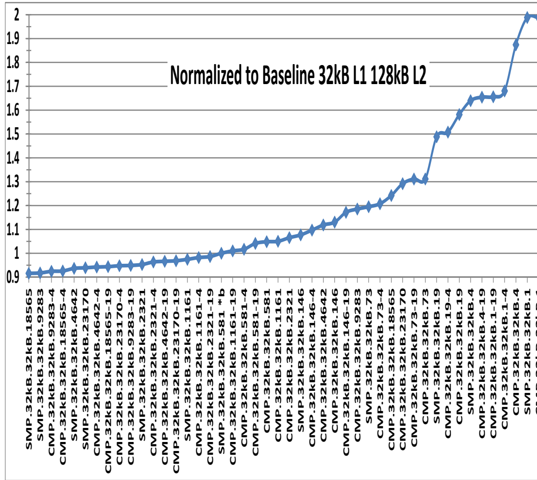


(e) 2MB L2.

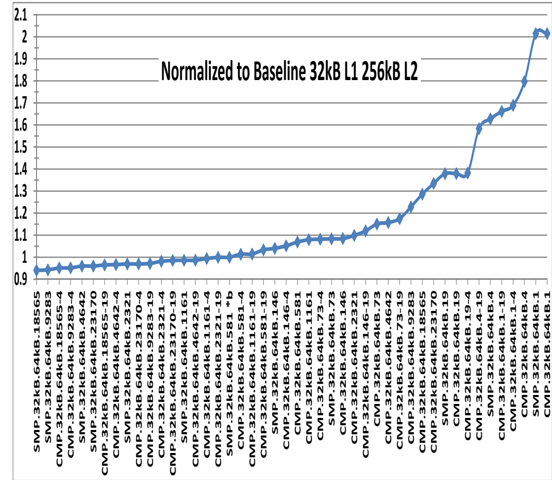


(f) 4MB L2.

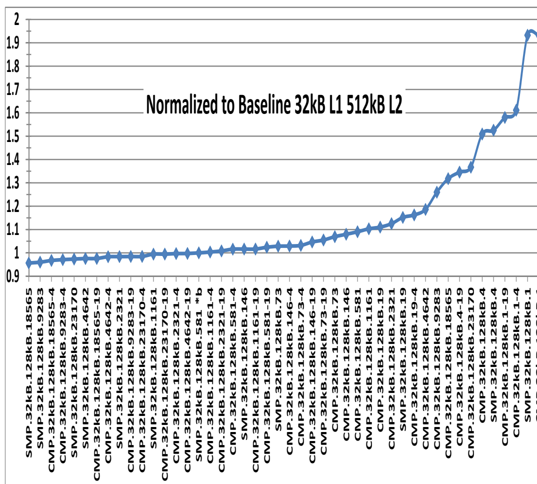
Figure 5.20: M5 results for MOLDYN under the **144** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



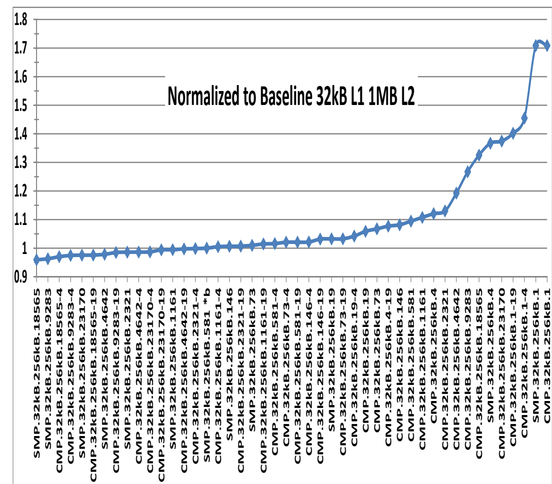
(a) 128kB L2.



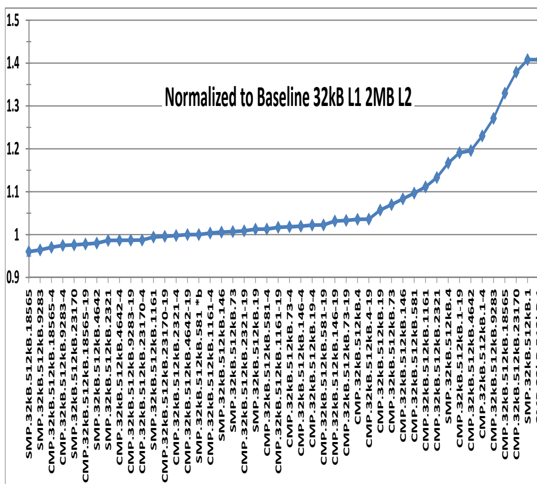
(b) 256kB L2.



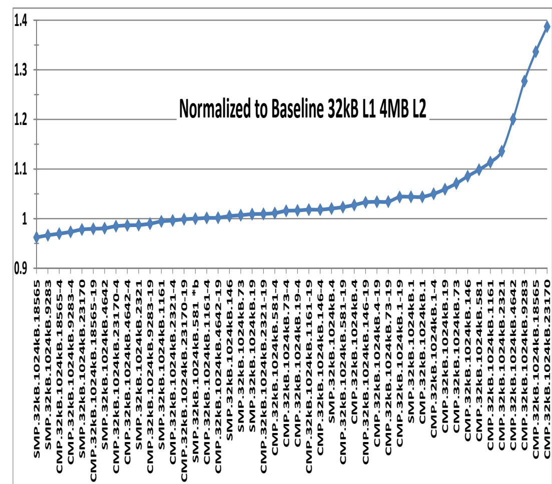
(c) 512kB L2.



(d) 1MB L2.



(e) 2MB L2.



(f) 4MB L2.

Figure 5.21: M5 results for MOLDYN under the **144** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.

(a) 128kB L2.

(b) 256kB L2.

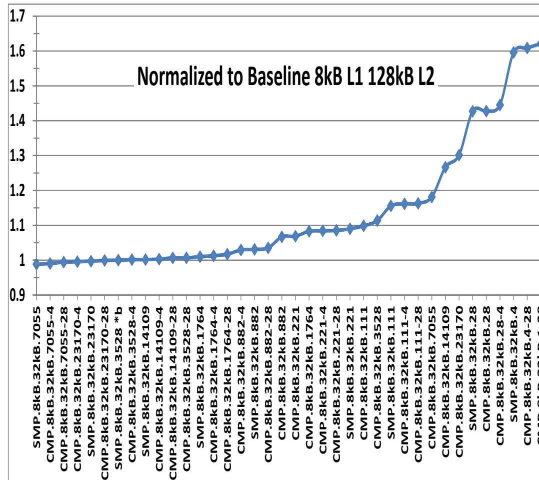
(c) 512kB L2.

(d) 1MB L2.

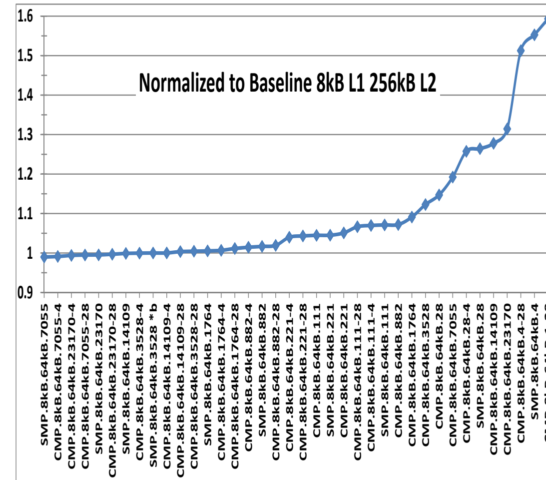
(e) 2MB L2.

(f) 4MB L2.

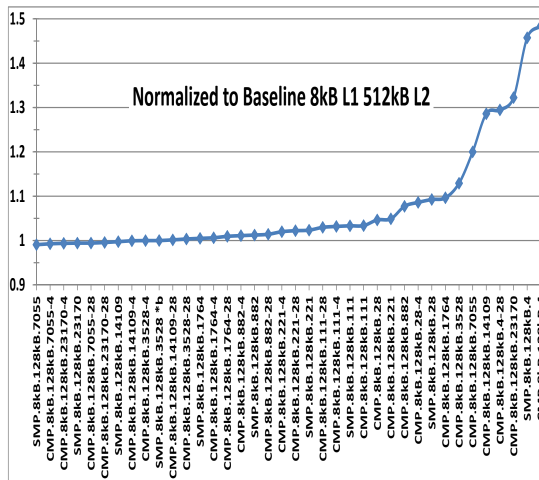
Figure 5.22: M5 results for MOLDYN under the **m14b** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



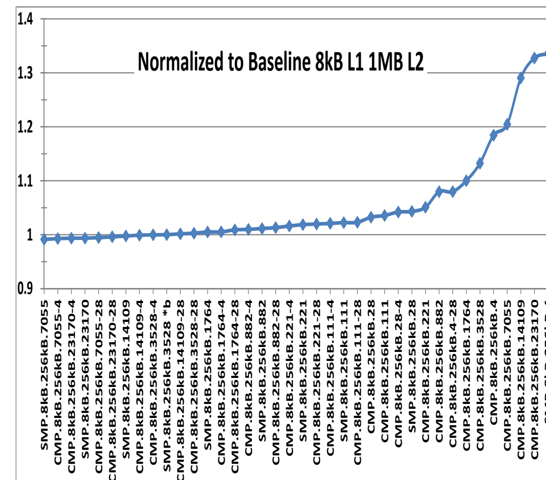
(a) 128kB L2.



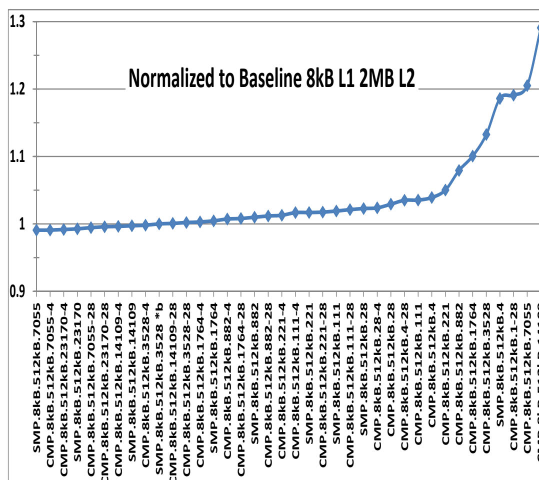
(b) 256kB L2.



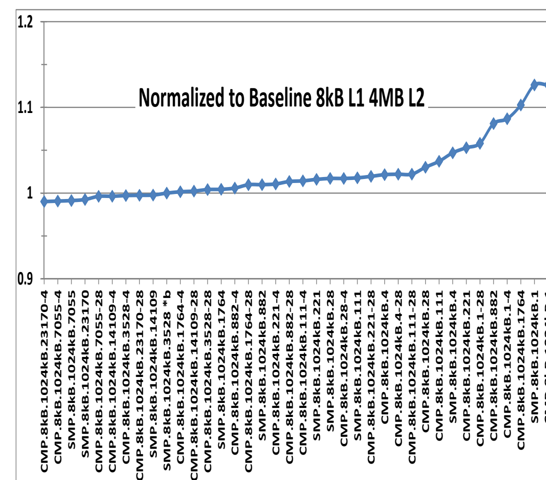
(c) 512kB L2.



(d) 1MB L2.

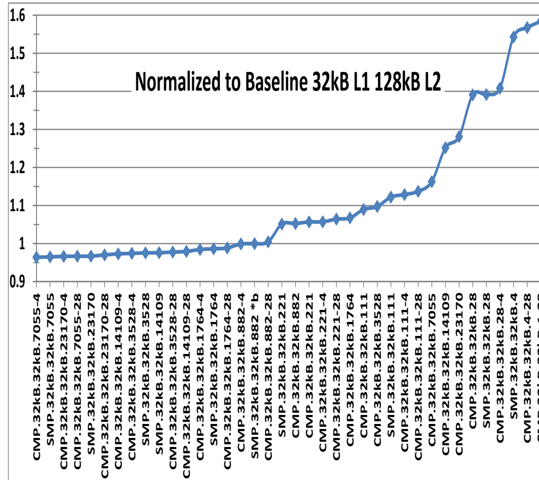


(e) 2MB L2.

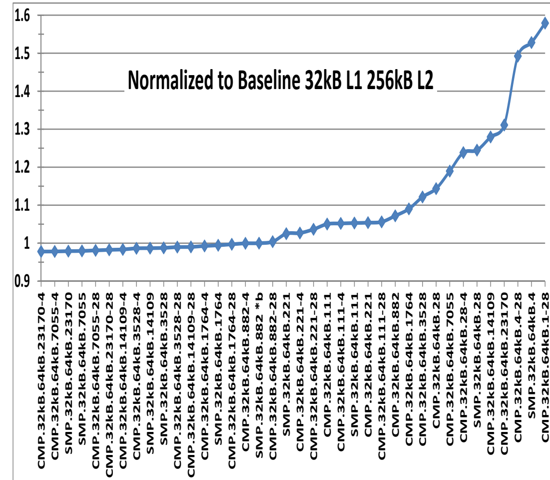


(f) 4MB L2.

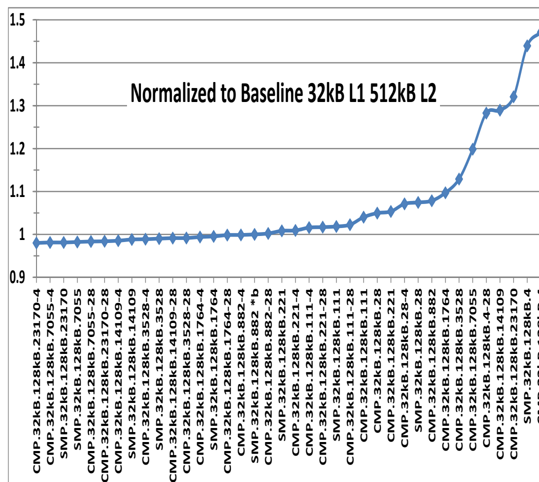
Figure 5.23: M5 results for MOLDYN under the **m14b** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



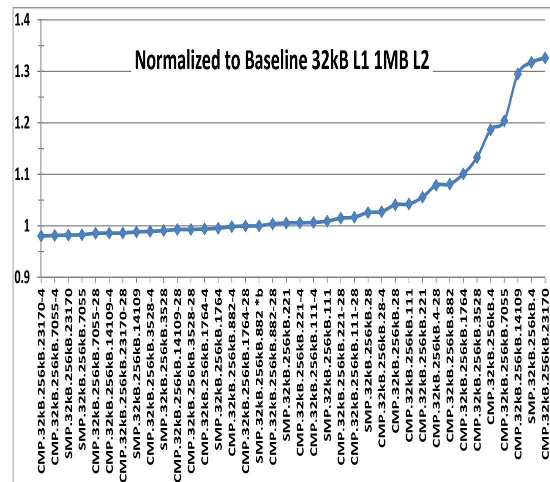
(a) 128kB L2.



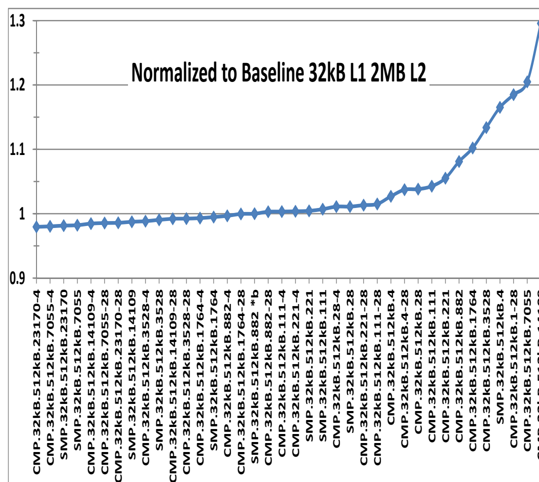
(b) 256kB L2.



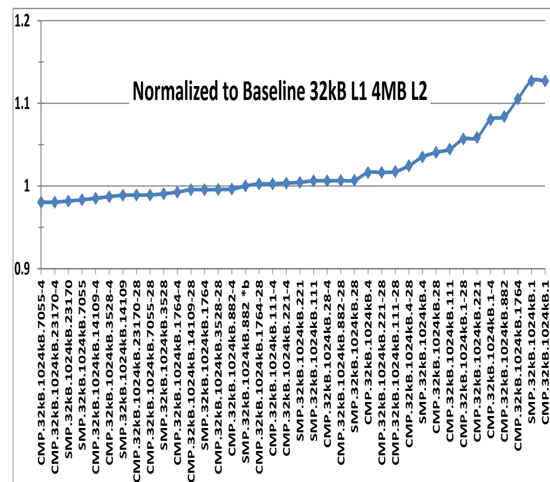
(c) 512kB L2.



(d) 1MB L2.

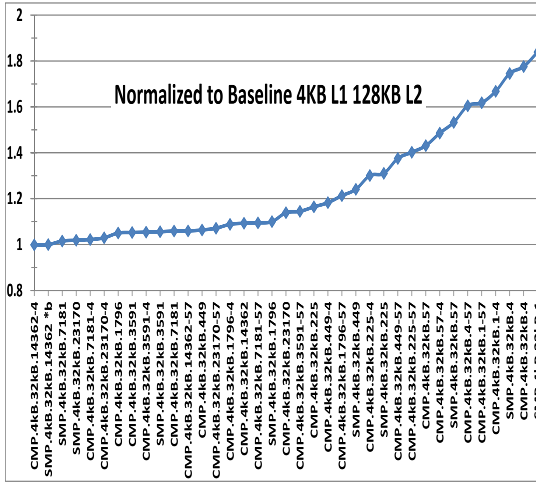


(e) 2MB L2.

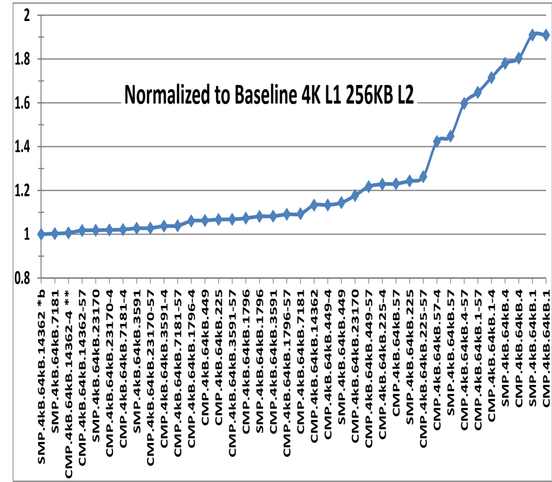


(f) 4MB L2.

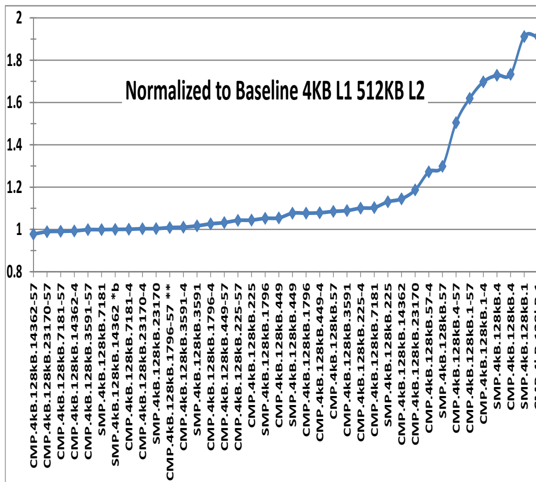
Figure 5.24: M5 results for MOLDYN under the **m14b** problem with $32kB$ **L1** cache and $128kB-4MB$ L2 caches all results normalized to baseline.



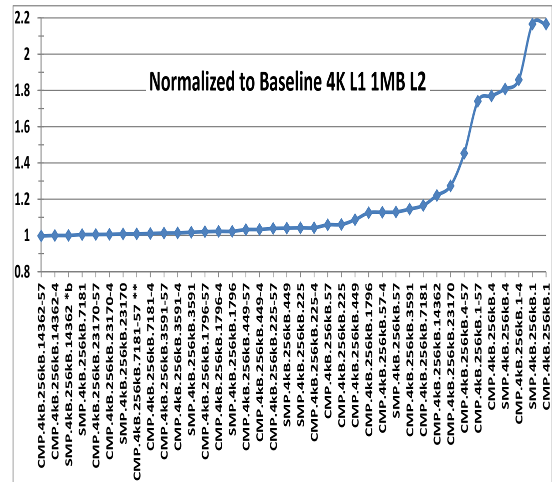
(a) 128kB L2.



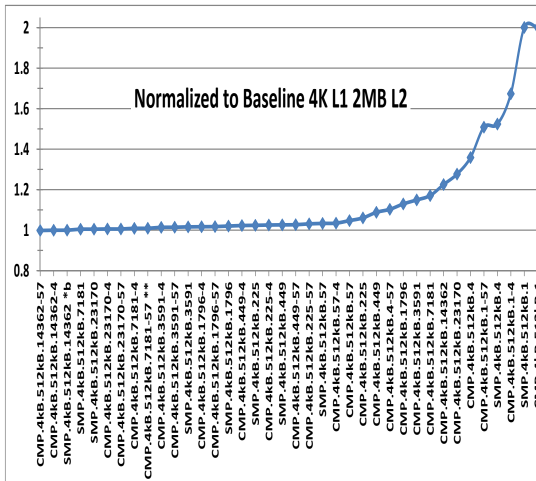
(b) 256kB L2.



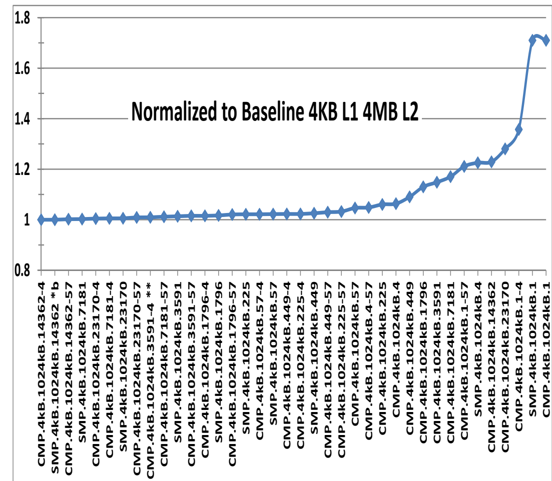
(c) 512kB L2.



(d) 1MB L2.

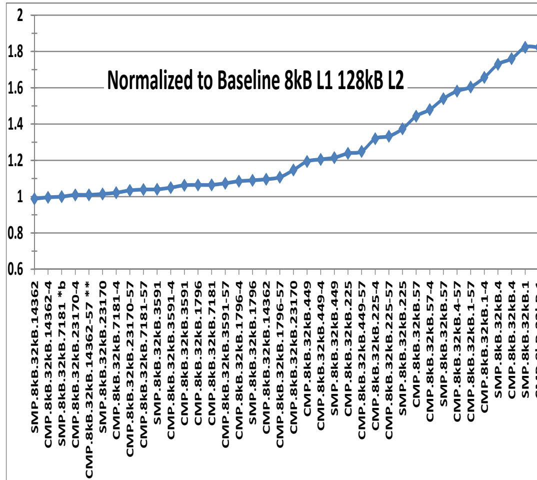


(e) 2MB L2.

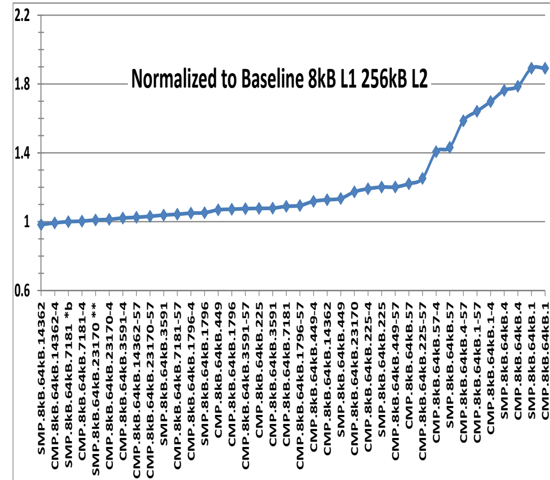


(f) 4MB L2.

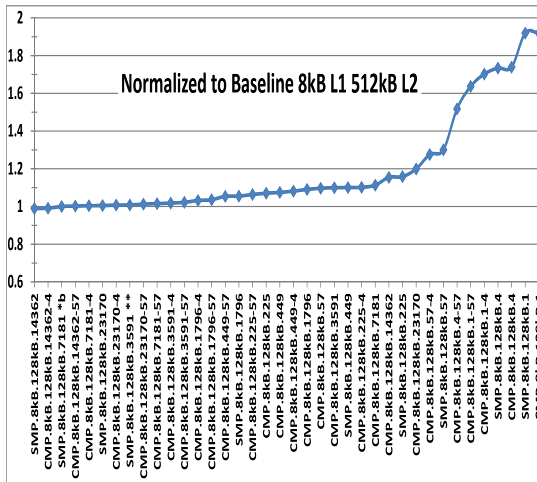
Figure 5.25: M5 results for MOLDYN under the **auto** problem with $4kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.



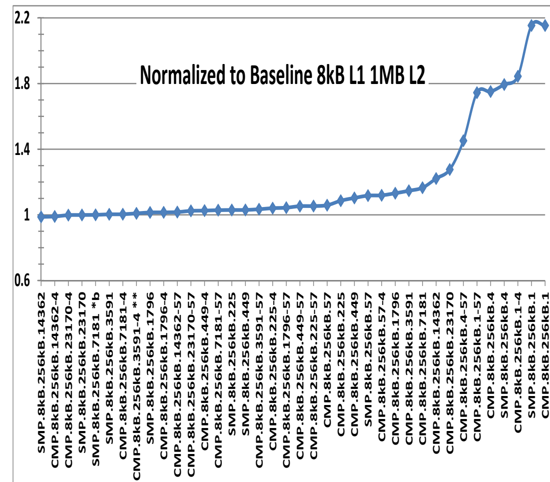
(a) 128kB L2.



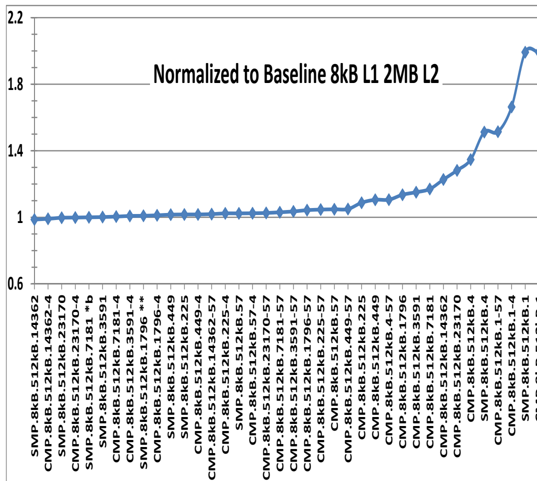
(b) 256kB L2.



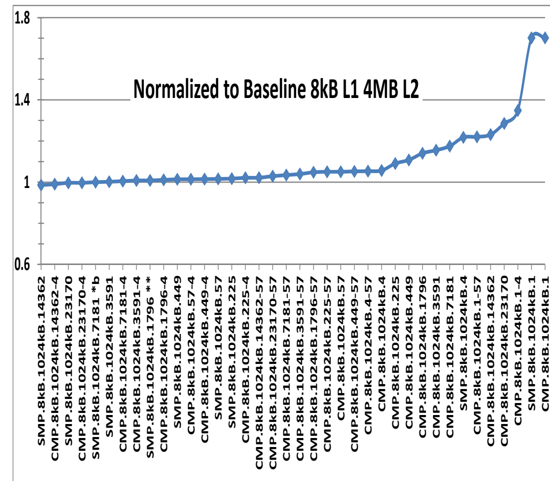
(c) 512kB L2.



(d) 1MB L2.



(e) 2MB L2.



(f) 4MB L2.

Figure 5.26: M5 results for MOLDYN under the **auto** problem with $8kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.

(a) 128kB L2.

(b) 256kB L2.

(c) 512kB L2.

(d) 1MB L2.

(e) 2MB L2.

(f) 4MB L2.

Figure 5.27: M5 results for MOLDYN under the **auto** problem with $32kB$ **L1** cache and $128kB - 4MB$ L2 caches all results normalized to baseline.

Chapter 6

PIN Reuse Distance Performance Model Prediction Results

In this chapter, we use the PIN Reuse Distance Profiles that we have collected for all of our benchmarks under all input data sets to predict the performance under the same set of machines we have simulated in Chapter 5. Recall, there are machines consisting of 4 core with the cross product of 4kB, 8kB, and 32kB L1 caches and 128kB, 256kB, 512kB, 1MB, 2MB, and 4MB L2 caches. We have the CMC of the CRD and PRD profiles for each parallelization under each partition size. Using these profiles, we developed a performance prediction model to predict execution time in cycles. We verify this model against the M5 simulations and show the difference in the predicted gains using the PIN model and M5 in two metrics that we will discuss in details. We show the absolute difference between the best performing partitioning, as well as the percentage of the gain achieved through the M5 simulation results picks that the model prediction can achieve.

6.1 PIN Reuse Distance Performance Model

In this section, we will derive the PIN reuse distance model and show how it was influenced by the M5 statistics and will also show how we came up with it.

The goal of our model is to compute the expected execution time in cycles of the benchmarks profiled. We base our model on the model developed by by Wu

et al. in [184]. A major difference is that Wu did not compute execution time but computed average memory access time. In order to verify our model initially, we extracted information from the M5 simulation results such as the alpha instruction count, L1 cache miss count, the L2 cache miss count, and L2 access whether they are regular misses or coherence misses *etc.*. We have computed using these M5 simulation statistics the execution time and we graphed the results for one on-chip memory size (4kB L1 and 128kB L2). We also put on the same graph the performance model as well as the M5 simulation. The goal was to compare the model to a mock model given by the statistics of the M5 simulator to compare them.

The PIN tool did not spit out for us the number of μops (micro ops) which are the RISC translation of the CISC x86 instructions. In some previous research we have done, we have verified that the alpha instruction count is similar or very close in value to the μops of the Intel architecture. Therefore, we have decided to use the M5 alpha instruction count in our model. Through private communication with Dr. Aamer Jaleel of Intel's VASSAD, he kindly informed us that the average CISC instruction gets translated to 1.3 RISC instructions. We have tried this number to model the instruction count but it did skew some of the orders and therefore we have decided to use the M5 alpha instruction count since there is μop translators that the PIN tool can use to count the RISC instruction count. One of such translators exists already in the x86 branch of the M5 simulator. The PIN model cannot predict exact execution time nor this is its goal. Its goal is to rank order different benchmarks to tell us in the ball park which benchmarks would run faster than others.

6.1.1 Deriving the Performance Model

We model a tiled architecture as is depicted in Figure 3.1. There is two levels of cache. L1 is private and L2 is shared. In such a system, the total cache misses are $sPRD_CMC[T \times L1_{size}]$, where the $sPRD_CMC$ is the profile representing the cache miss count of the private reuse distance profile. $L1_{size}$ is the per core private L1 cache sizes, T is the number of tiles, P is the number of cores, and $L1_{lat}$ is the latency for accessing data in the private L1 cache. Note that P equals T in our model.

We model a shared L2 cache. LLC_{size} is the size of the shared LLC for all cores, which is the L2 cache in our case. The LLC accesses are given by $sPRD_CMC[T \times L1_{size}]$ with access latency of $(LLC_{lat} + 2 \times HOP_{lat})$ cycles, where (LLC_{lat} is the latency of accessing the LLC. We assume data blocks are distributed uniformly on the LLC slices. Thus, network messages incur $\sqrt{T} + 1$ hops on average [155]. In the shared LLC, the data always resides on the home tile, so there is two-way communication. We use a factor of 2.5 that was suggested by some of the parameters of M5 and by our matching of the models. This factor accounts for writebacks and other L2 accesses that take place.

The off-chip accesses are $CRD_CMC[LLC_{size}]$, where the CRD_CMC is the profile representing the cache miss count of the (shared) concurrent reuse distance profile. The access latency for the off-chip data is given by $(DRAM_{lat} + 2 \times HOP_{lat})$ which contains two-way data communication to the DRAM controller and DRAM device total latency. The total hop latency is given by:

HOP_{lat} = (Latency per Hop) × (The Average number of Hops).

Or, HOP_{lat} = (Latency per Hop) $\times(\sqrt{T} + 1)$.

The average memory access time ($AMAT_{sharedL2}$) for a tiled CMP with shared L2 caches as the LLC can be modeled using Equation 6.1, where $sPRD_CMC[0]$ is the number of total memory references.

$$\begin{aligned} AMAT_{sharedL2} = (1/sPRD_CMC[0]) \times (L1_{lat} \times sPRD_CMC[0]\ + \\ 2.5 \times (LLC_{lat} + 2 \times HOP_{lat}) \times sPRD_CMC[T \times L1_{size}]\ + \\ (DRAM_{lat} + 2 \times HOP_{lat}) \times CRD_CMC[LLC_{size}]) \end{aligned} \quad (6.1)$$

The total execution time in cycles ($ExecTime$) for a tiled processors with shared L2 cache can be modeled using Equation 6.2, where $Inst_count$ is the number of μop instruction executed. We use the M5 instruction count.

$$ExecTime = (1/P) \times (Inst_count + sPRD_CMC[0] \times AMAT_{sharedL2}) \quad (6.2)$$

The performance model does not consider queuing in the on-chip nor off-chip, coherence traffic, cache conflicts, and associativity effects, nor load imbalance are not modeled as well.

6.2 Results using the PIN Performance Model

We have to recall here for explaining the figures that will presented in this section what we have explained in Section 5.1. We will summarize this here for completeness.

Figures 6.1 — 6.27 show the PIN predicted performance in cycles. All subfigures show an ordered execution time from the smallest execution time in cycles to the largest. Each subfigure shows the results for a particular L1 cache size and a particular L2 cache size. Each of the following subsections shows the results for a single benchmark. For each benchmark, there are three subsections that present results for each of the three input data sets. In each subsection there are three figures each with six sub-figures. These sub-figures present the three simulated L1 cache sizes and the six simulated L2 cache sizes. Each sub-figure shows the results for the cross product of the four parallelization strategies as shown in Section 4.3 with all the different locality optimizations (partition sizes).

The first few combinations of parallelizations and partition sizes are labeled on the x-axis with a number denoting their rank in the corresponding M5 simulation and figure and subfigure in Chapter 5. The baseline case is identified in each subfigure with the label "*b". The x-axis in each of these subfigures can be interpreted in a similar fashion to Section 5.1 as follows:

1. "$10654-SMP-CMCrd\,1$" means the parallelization is SMP and the number of partitions is $10,654$. The number that appears after each of these is the order of that particular data point in M5 obtained from the corresponding figure in Chapter 5. In this particular example, "1" means that this is the best performing partition size and parallelization strategy for the 4kB L1 cache, 128kB L2 cache for IRREG under the 144 input data set.

2. "$10654-CMP-CMCrd$" means the parallelization is CMP and the number

of partitions is $10,654$.

3. "$10654 - 4 - CMP - CMCrd$ 3" means the parallelization is CMP-4 and the number of partitions is $10,654$. Note that four here is the number of cores. Note that 3 means the ranking order for this parallelization and partition size when run on M5.

4. "$10654 - 11 - CMP - CMCrd$ 8" means the parallelization is CMP-11 and the number of partitions is $10,654$. Note that 11 here represents the maximum partition size that can fit in a 1MB cache. This changes according to each input data set data size. Also, 8 means the ranking order for this parallelization and partition size when run on M5.

The y-axis in Figures 6.1 — 6.27 will show the predicted execution time in cycles that has been computed using the PIN performance model we derived in the previous section. The reported %Gain PIN is computed as the gain the partitioning and the parallelization predicted to achieve the best performance by PIN if we ran it on M5 relative to the baseline as was shown in Chapter 5. We show the difference between the maximum gain obtained from the M5 runs and that predicted by PIN. Also, we report the percentage PIN captured gain which is the percentage of the maximum M5 gain that the PIN prediction would achieve.

After each set of graphs for any benchmark with a given input data set there is a table that summarizes the results in the graphs such as Table 6.1. In such a table there is four performance metrics presented. The first is the "%Gain M5". This is defined in Section 5.1 as the relative performance gain of the best performing

parallelization and partition size in percentage points given an input data set and benchmark given a specific machine to the baseline performance. The second is the “%Gain PIN”. This is defined as the gain that we would get when we run the PIN predicted best pick if we run on M5 relative to the baseline case as we explained it earlier.

There are two characterization metrics that show how good the PIN prediction is relative to the best M5 pick. These metrics are “Diff(%M5 - %PIN)” and “%PIN Captured Gain”. These two metrics quantify how good the PIN prediction is compared to the M5 simulation result for a given machine under a specific problem and input data set. “Diff(%M5 - %PIN)” is defined as the difference between the “%Gain M5” and “%Gain PIN”. This measures the error between the M5 choice and the PIN prediction. “%PIN Captured Gain” is the second metric and is defined as the percentage of the M5 gain that the PIN prediction was able to capture. If PIN picks the exact same pick that M5 comes up with then that value would be 100%.

6.2.1 Irreg Results

In this section, we will show the results of the PIN performance model for the Irreg benchmark with all of the input data sets; **144, m14b, and auto**. We predict performance for the cross product of machines with 4kB, 8kB, and 32kB L1 caches and 128kB, 256kB, 512kB, 1MB, 2MB, and 4MB L2 caches. All the results are with 4 cores.

6.2.1.1 Irreg with 144 input data set

In this section, we will show the results of the PIN performance model for Irreg benchmark with the **144** input data set. The PIN predicted gains are summarized in Table 6.1. The gains for the 4kB simulated machines range from -2.06% to 2.36%, the average PIN predicted gain is 0.19% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.69%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 10.19%. The gains for the 8kB simulated machines range from -3.53% to 2.66%, the average PIN predicted gain is 0.54% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 3.02%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 15.04%. The gains for the 32kB simulated machines range from -4.17% to 9.57%, the average PIN predicted gain is 2.19% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 2.78%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 44.04%. The overall average PIN predicted gain for Irregwith the 144 input data set for all L1 caches is 0.97%, the overall average gain difference between the M5 maximum gain and the PIN predicted gain is 2.50%, and the overall average percentage of the maximum M5 gain captured by the PIN prediction is 28%. Figures 6.1, 6.2, and 6.3 show in detail the results for the three L1 caches.

Table 6.1: A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for IRREG under the **144** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.34	2.19	2.36	1.85	2.04	1.49	1.88
%Gain PIN	0.62	0.65	2.36	-0.42	0.00	-2.06	0.19
Diff(%M5 - %PIN)	0.72	1.53	0.00	2.27	2.05	3.55	1.69
%PIN Captured Gain	46.24	29.87	100.00	-22.67	-0.20	-137.70	10.19
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	3.53	4.36	3.87	4.12	4.09	1.38	3.56
%Gain PIN	1.26	1.40	2.66	1.37	0.05	-3.53	0.54
Diff(%M5-%PIN)	2.27	2.96	1.21	2.76	4.04	4.91	3.02
%PIN Captured Gain	35.74	32.13	68.75	33.15	1.20	-255.88	15.04
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	11.20	5.65	4.05	3.57	3.21	2.09	4.96
%Gain PIN	9.57	4.15	2.34	1.07	0.15	-4.17	2.19
Diff(%M5-%PIN)	1.63	1.49	1.71	2.50	3.07	6.26	2.78
%PIN Captured Gain	85.46	73.57	57.72	30.04	4.58	-199.28	44.04
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. 144
%Gain M5	5.35	4.07	3.43	3.18	3.11	1.66	3.47
%Gain PIN	3.82	2.07	2.45	0.67	0.06	-3.25	0.97
Diff(%M5-%PIN)	1.54	2.00	0.97	2.51	3.05	4.91	2.50
% PIN Captured Gain	71.28	50.91	71.58	21.16	2.05	-196.46	28.00

6.2.1.2 IRREG with m14b input data set

In this section, we will show the results of the PIN performance model for IRREG benchmark with the **m14b** input data set. The PIN predicted gains are summarized in Table 6.2. The gains for the 4kB simulated machines range from -3.19% to 0.54%, the average PIN predicted gain is -0.75% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and

the PIN predicted gain is 1.70%, and the average percentage of the maximum M5 gain captured by the PIN prediction is -79.08%. The gains for the 8kB simulated machines range from -3.48% to 0.47%, the average PIN predicted gain is -0.63% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 2.48%, and the average percentage of the maximum M5 gain captured by the PIN prediction is -33.68%. The gains for the 32kB simulated machines range from -0.73% to 5.46%, the average PIN predicted gain is 1.22% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.94%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 38.59%. The overall average PIN predicted gain for Irregwith the m14b input data set for all L1 caches is -0.05%, the overall average gain difference between the M5 maximum gain and the PIN predicted gain is 2.04%, and the overall average percentage of the maximum M5 gain captured by the PIN prediction is -2.67%. Figures 6.4, 6.5, and 6.6 show in detail the results for the three L1 caches.

6.2.1.3 Irreg with auto input data set

In this section, we will show the results of the PIN performance model for Irreg benchmark with the **auto** input data set. The PIN predicted gains are summarized in Table 6.3. The gains for the 4kB simulated machines range from -0.10% to 3.69%, the average PIN predicted gain is 2.02% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the

Table 6.2: A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for IRREG under the **m14b** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	0.85	1.33	1.13	1.03	0.79	0.58	0.95
%Gain PIN	-0.19	0.13	0.54	-0.70	-1.09	-3.19	-0.75
Diff(%M5-%PIN)	1.05	1.20	0.59	1.73	1.88	3.77	1.70
%PIN Captured Gain	-22.88	9.49	47.69	-68.52	-137.47	-551.71	-79.08
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	2.38	2.01	1.75	2.07	1.55	1.37	1.86
%Gain PIN	0.31	0.32	-0.33	0.47	-1.04	-3.48	-0.63
Diff(%M5-%PIN)	2.08	1.70	2.08	1.60	2.59	4.84	2.48
%PIN Captured Gain	12.90	15.67	-18.78	22.89	-67.20	-254.74	-33.68
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	6.68	3.73	3.32	2.29	1.59	1.32	3.16
%Gain PIN	5.46	3.07	1.26	1.30	-0.73	-3.06	1.22
Diff(%M5-%PIN)	1.22	0.66	2.06	0.99	2.32	4.38	1.94
%PIN Captured Gain	81.75	82.34	37.86	56.79	-45.69	-231.08	38.59
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. m14b
%Gain M5	3.30	2.36	2.07	1.80	1.31	1.09	1.99
%Gain PIN	1.86	1.17	0.49	0.36	-0.95	-3.24	-0.05
Diff(%M5-%PIN)	1.45	1.19	1.58	1.44	2.26	4.33	2.04
% PIN Captured Gain	56.22	49.68	23.62	19.89	-72.60	-297.75	-2.67

PIN predicted gain is 1.99%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 47.29%. The gains for the 8kB simulated machines range from 3.49% to 8.86%, the average PIN predicted gain is 6.64% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.64%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 78.83%. The gains for the 32kB simulated machines range from -3.74% to 17.37%, the average PIN predicted gain is 5.54%

relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 4.01%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 58%. The overall average PIN predicted gain for IrregWith the auto input data set for all L1 caches is 4.73%, the overall average gain difference between the M5 maximum gain and the PIN predicted gain is 2.55%, and the overall average percentage of the maximum M5 gain captured by the PIN prediction is 65%. Figures 6.7, 6.8, and 6.9 show in detail the results for the three L1 caches.

6.2.2 NBF Results

In this section, we will show the results of the PIN performance model for NBF benchmark with all of the input data sets; **144, m14b, and auto**. We predict performance for the cross product of machines with 4kB, 8kB, and 32kB L1 caches and 128kB, 256kB, 512kB, 1MB, 2MB, and 4MB L2 caches. All the results are with 4 cores.

6.2.2.1 NBF with 144 input data set

In this section, we will show the results of the PIN performance model for NBF benchmark with the **144** input data set. The PIN predicted gains are summarized in Table 6.4. The gains for the 4kB simulated machines range from -4.06% to 0.18%, the average PIN predicted gain is -1.31% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted

Table 6.3: A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for IRREG under the **auto** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	4.73	4.39	4.48	3.49	3.33	3.63	4.01
%Gain PIN	3.14	2.88	3.69	2.33	0.18	-0.10	2.02
Diff(%M5-%PIN)	1.60	1.51	0.80	1.15	3.15	3.73	1.99
%PIN Captured Gain	66.25	65.56	82.25	66.90	5.54	-2.75	47.29
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	10.00	9.08	8.73	7.65	7.01	7.21	8.28
%Gain PIN	8.86	7.74	8.06	6.80	4.88	3.49	6.64
Diff(%M5-%PIN)	1.14	1.35	0.67	0.85	2.13	3.72	1.64
%PIN Captured Gain	88.61	85.19	92.35	88.85	69.60	48.40	78.83
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	19.69	11.01	8.58	6.69	6.68	4.65	9.55
%Gain PIN	17.37	10.54	7.67	2.90	-1.50	-3.74	5.54
Diff(%M5-%PIN)	2.32	0.47	0.91	3.78	8.19	8.40	4.01
%PIN Captured Gain	88.19	95.74	89.41	43.45	-22.50	-80.44	58.00
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. auto
%Gain M5	11.48	8.16	7.26	5.94	5.67	5.17	7.28
%Gain PIN	9.79	7.05	6.47	4.01	1.19	-0.12	4.73
Diff(%M5-%PIN)	1.69	1.11	0.79	1.93	4.49	5.28	2.55
% PIN Captured Gain	85.30	86.41	89.11	67.52	20.90	-2.27	65.00

gain is 1.71%, and the average percentage of the maximum M5 gain captured by the PIN prediction is -334.09%. When the M5 predicted gain is 0%, it means that the baseline data point is the point with the maximum gain and thus the percentage PIN gain captured cannot be computed since it will result in a division by zero. The gains for the 8kB simulated machines range from -2.83% to 4.43%, the average PIN predicted gain is 1.67% relative to the M5 baseline performance, the average gain

difference between the M5 maximum gain and the PIN predicted gain is 1.74%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 48.96%. The gains for the 32kB simulated machines range from -0.97% to 11.22%, the average PIN predicted gain is 4.62% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.81%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 71.83%. The overall average PIN predicted gain for NBFwith the 144 input data set for all L1 caches is 1.66%, the overall average gain difference between the M5 maximum gain and the PIN predicted gain is 1.74%, and the overall average percentage of the maximum M5 gain captured by the PIN prediction is 48.61%. Figures 6.10, 6.11, and 6.12 show in detail the results for the three L1 caches.

6.2.2.2 NBF with m14b input data set

In this section, we will show the results of the PIN performance model for NBF benchmark with the **m14b** input data set. The PIN predicted gains are summarized in Table 6.5. The gains for the 4kB simulated machines range from -3.29% to 0.41%, the average PIN predicted gain is -1.09% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.69%, and the average percentage of the maximum M5 gain captured by the PIN prediction is -70.32%. When the M5 predicted gain is 0%, it means that the baseline data point is the point with the maximum gain and thus the percentage PIN gain captured cannot be computed since it will result in a division

Table 6.4: A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for NBF under the **144** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.30	0.07	0.00	0.88	0.12	0.00	0.39
% Gain PIN	0.18	-1.32	-0.24	-0.43	-2.01	-4.06	-1.31
Diff(%M5-%PIN)	1.12	1.39	0.24	1.31	2.13	4.06	1.71
%PIN Gain Captured	13.51	-1874.98	N/A	-49.24	-1679.84	N/A	-334.09
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	5.22	4.41	4.00	4.04	2.05	0.71	3.41
%Gain PIN	4.43	2.58	3.82	2.72	-0.71	-2.83	1.67
Diff(%M5-%PIN)	0.79	1.83	0.19	1.32	2.77	3.54	1.74
%PIN Gain Captured	84.90	58.57	95.32	67.38	-34.76	-401.66	48.96
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	11.97	8.62	6.26	4.89	3.54	3.34	6.44
%Gain PIN	11.22	8.06	5.90	3.07	0.46	-0.97	4.62
Diff(%M5-%PIN)	0.75	0.56	0.36	1.82	3.08	4.31	1.81
%PIN Gain Captured	93.71	93.47	94.33	62.79	13.02	-29.10	71.83
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. 144
%Gain M5	6.16	4.37	3.42	3.27	1.90	1.35	3.41
%Gain PIN	5.27	3.11	3.16	1.79	-0.75	-2.62	1.66
Diff(%M5-%PIN)	0.89	1.26	0.26	1.48	2.66	3.97	1.75
% PIN Captured Gain	85.60	71.15	92.38	54.67	-39.60	-194.63	48.61

by zero. The gains for the 8kB simulated machines range from -3.27% to 1.36%, the average PIN predicted gain is -0.39% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.78%, and the average percentage of the maximum M5 gain captured by the PIN prediction is -124.37%. The gains for the 32kB simulated machines range from -4.74% to 3.30%, the average PIN predicted gain is 0.89% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the

PIN predicted gain is 2.32%, and the average percentage of the maximum M5 gain captured by the PIN prediction is -186.02%. The overall average PIN predicted gain for NBFwith the m14b input data set for all L1 caches is -0.20%, the overall average gain difference between the M5 maximum gain and the PIN predicted gain is 1.93%, and the overall average percentage of the maximum M5 gain captured by the PIN prediction is -11.30%. Figures 6.13, 6.14, and 6.15 show in detail the results for the three L1 caches.

Table 6.5: A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for NBF under the **m14b** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	0.00	0.59	0.84	0.94	1.24	0.00	0.60
%Gain PIN	-0.74	-0.84	0.41	-0.78	-1.31	-3.29	-1.09
Diff(%M5 - %PIN)	0.74	1.43	0.43	1.72	2.54	3.29	1.69
%PIN Gain Captured	N/A	-141.78	48.89	-82.54	-105.86	N/A	-70.32
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.37	1.80	1.78	1.83	1.15	0.43	1.39
%Gain PIN	1.36	0.26	1.30	-0.04	-1.94	-3.27	-0.39
Diff(%M5-%PIN)	0.01	1.54	0.47	1.87	3.09	3.70	1.78
%PIN Captured Gain	99.13	14.44	73.38	-2.04	-168.30	-762.80	-124.37
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	6.16	3.28	3.55	3.03	2.91	0.35	3.21
%Gain PIN	3.30	3.23	3.19	0.61	-0.25	-4.74	0.89
Diff(%M5-%PIN)	2.86	0.05	0.36	2.42	3.16	5.08	2.32
%PIN Captured Gain	53.56	98.39	89.80	20.07	-8.44	-1373.09	-186.62
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. m14b
%Gain M5	2.51	1.89	2.06	1.94	1.77	0.26	1.74
%Gain PIN	1.30	0.88	1.64	-0.07	-1.16	-3.77	-0.20
Diff(%M5-%PIN)	1.21	1.01	0.42	2.01	2.93	4.03	1.93
% PIN Captured Gain	51.96	46.75	79.49	-3.59	-65.87	-1460.47	-11.30

6.2.2.3 NBF with auto input data set

In this section, we will show the results of the PIN performance model for NBF benchmark with the **auto** input data set. The PIN predicted gains are summarized in Table 6.6. The gains for the 4kB simulated machines range from 0.70% to 4.20%, the average PIN predicted gain is 2.50% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.07%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 69.15%. The gains for the 8kB simulated machines range from 2.58% to 8.27%, the average PIN predicted gain is 5.13% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.61%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 74.07%. The gains for the 32kB simulated machines range from 2.59% to 15.61%, the average PIN predicted gain is 8.37% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.45%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 85.27%. The overall average PIN predicted gain for NBFwith the auto input data set for all L1 caches is 5.33%, the overall average gain difference between the M5 maximum gain and the PIN predicted gain is 1.38%, and the overall average percentage of the maximum M5 gain captured by the PIN prediction is 79.52%. Figures 6.16, 6.17, and 6.18 show in detail the results for the three L1 caches.

Table 6.6: A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for NBF under the **auto** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	3.60	3.10	4.50	3.30	3.30	3.60	3.57
%Gain PIN	3.50	2.50	4.20	2.70	1.40	0.70	2.50
Diff(%M5-%PIN)	0.10	0.60	0.30	0.60	1.90	2.90	1.07
%PIN Captured Gain	97.22	80.65	93.33	81.82	42.42	19.44	69.15
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	8.64	6.30	7.23	6.26	5.78	6.22	6.74
%Gain PIN	8.27	5.30	6.71	5.20	2.72	2.58	5.13
Diff(%M5-%PIN)	0.37	0.99	0.52	1.06	3.05	3.64	1.61
%PIN Captured Gain	95.67	84.24	92.77	83.12	47.13	41.47	74.07
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	16.18	11.60	9.74	8.00	7.01	6.33	9.81
%Gain PIN	15.61	11.23	9.50	6.89	4.36	2.59	8.37
Diff(%M5-%PIN)	0.57	0.38	0.24	1.11	2.65	3.73	1.45
%PIN Captured Gain	96.50	96.76	97.56	86.12	62.24	40.98	85.27
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. auto
%Gain M5	9.47	7.00	7.16	5.85	5.36	5.38	6.70
%Gain PIN	9.13	6.34	6.80	4.93	2.83	1.96	5.33
Diff(%M5-%PIN)	32.32	32.79	32.79	29.26	22.40	15.84	1.38
% PIN Captured Gain	96.34	90.63	95.06	84.24	52.75	36.37	79.52

6.2.3 Moldyn Results

In this section, we will show the results of the PIN performance model for Moldyn benchmark with all of the input data sets; **144, m14b, and auto**. We predict performance for the cross product of machines with 4kB, 8kB, and 32kB L1 caches and 128kB, 256kB, 512kB, 1MB, 2MB, and 4MB L2 caches. All the results are with 4 cores.

6.2.3.1 Moldyn with 144 input data set

In this section, we will show the results of the PIN performance model for Moldyn benchmark with the **144** input data set. The PIN predicted gains are summarized in Table 6.7. The gains for the 4kB simulated machines range from 1.86% to 2.55%, the average PIN predicted gain is 2.24% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 0.09%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 95.95%. The gains for the 8kB simulated machines range from 2.25% to 2.87%, the average PIN predicted gain is 2.46% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 0.49%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 83.50%. The gains for the 32kB simulated machines range from 1.31% to 5.10%, the average PIN predicted gain is 2.50% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 2.60%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 49.02%. The overall average PIN predicted gain for Moldynwith the 144 input data set for all L1 caches is 2.40%, the overall average gain difference between the M5 maximum gain and the PIN predicted gain is 1.06%, and the overall average percentage of the maximum M5 gain captured by the PIN prediction is 76.16%. Figures 6.19, 6.20, and 6.21 show in detail the results for the three L1 caches.

Table 6.7: A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for MOLDYN under the **144** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	2.43	2.11	2.32	2.55	2.28	2.32	2.34
%Gain PIN	1.86	2.11	2.32	2.55	2.28	2.32	2.24
Diff(%M5-%PIN)	0.57	0.00	0.00	0.00	0.00	0.00	0.09
%PIN Captured Gain	76.64	100.00	100.00	100.00	100.00	100.00	95.95
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	3.28	2.91	2.84	2.86	2.87	2.94	2.95
%Gain PIN	2.87	2.44	2.38	2.33	2.25	2.51	2.46
Diff(%M5-%PIN)	0.41	0.47	0.46	0.52	0.61	0.44	0.49
%PIN Captured Gain	87.42	83.84	83.81	81.65	78.58	85.07	83.50
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	8.42	5.97	4.35	4.07	3.97	3.77	5.09
%Gain PIN	5.10	2.82	1.56	1.53	1.31	2.66	2.50
Diff(%M5-%PIN)	3.32	3.15	2.78	2.55	2.67	1.11	2.60
%PIN Captured Gain	60.56	47.26	35.97	37.44	32.89	70.59	49.02
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. 144
%Gain M5	4.71	3.66	3.17	3.16	3.04	3.01	3.46
%Gain PIN	3.28	2.46	2.09	2.14	1.95	2.50	2.40
Diff(%M5-%PIN)	1.43	1.21	1.08	1.02	1.09	0.52	1.06
% PIN Captured Gain	69.56	67.08	65.87	67.59	64.05	82.86	69.38

6.2.3.2 MOLDYN with m14b input data set

In this section, we will show the results of the PIN performance model for MOLDYN benchmark with the **m14b** input data set. The PIN predicted gains are summarized in Table 6.8. The gains for the 4kB simulated machines range from 0.15% to 1.14%, the average PIN predicted gain is 0.41% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the

PIN predicted gain is 0.0%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 100%. We have to note here that this is the only case where a perfect prediction happens for all L2 caches for a certain L1 cache size. The gains for the 8kB simulated machines range from 0.37% to 0.59%, the average PIN predicted gain is 0.53% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 0.46%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 53.26%. The gains for the 32kB simulated machines range from 1.09% to 3.30%, the average PIN predicted gain is 1.81% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 0.49%, and the average percentage of the maximum M5 gain captured by the PIN prediction is 78.75%. The overall average PIN predicted gain for MOLDYNwith the m14b input data set for all L1 caches is 0.91%, the overall average gain difference between the M5 maximum gain and the PIN predicted gain is 0.32%, and the overall average percentage of the maximum M5 gain captured by the PIN prediction is 74.26%. Figures 6.22, 6.23, and 6.24 show in detail the results for the three L1 caches.

6.2.3.3 MOLDYN with auto input data set

In this section, we will show the results of the PIN performance model for MOLDYN benchmark with the **auto** input data set. The PIN predicted gains are summarized in Table 6.9. The gains for the 4kB simulated machines range from -

Table 6.8: A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for MOLDYN under the **m14b** input problem with $4kB-32kB$ L1 caches and $128kB-4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.14	0.29	0.15	0.15	0.33	0.37	0.41
%Gain PIN	1.14	0.29	0.15	0.15	0.33	0.37	0.41
Diff(%M5-%PIN)	0.00	0.00	0.00	0.00	0.00	0.00	0.00
%PIN Captured Gain	100.00	100.00	100.00	100.00	100.00	100.00	100.00
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	1.18	1.02	0.93	0.88	0.93	0.99	0.99
%Gain PIN	0.59	0.51	0.57	0.55	0.55	0.37	0.53
Diff(%M5-%PIN)	0.59	0.50	0.36	0.33	0.37	0.61	0.46
%PIN Captured Gain	50.14	50.47	61.20	62.55	59.73	37.98	53.26
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
%Gain M5	3.58	2.20	1.99	1.97	2.04	1.96	2.29
%Gain PIN	3.30	1.90	1.65	1.44	1.45	1.09	1.81
Diff(%M5-%PIN)	0.28	0.30	0.34	0.53	0.59	0.87	0.49
% PIN Captured Gain	92.11	86.15	82.93	73.18	70.96	55.52	78.75
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. m14b
%Gain M5	1.97	1.17	1.03	1.00	1.10	1.11	1.23
%Gain PIN	1.68	0.90	0.79	0.72	0.78	0.61	0.91
Diff(%M5-%PIN)	0.29	0.27	0.23	0.29	0.32	0.50	0.32
%PIN Captured Gain	85.26	76.97	77.17	71.43	70.71	55.25	74.26

5.94% to 2.23%, the average PIN predicted gain is -0.86% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 1.33%, and the average percentage of the maximum M5 gain captured by the PIN prediction is -181.61%. When the M5 predicted gain is 0%, it means that the baseline data point is the point with the maximum gain and thus the percentage PIN gain captured cannot be computed since it will result in a division by zero. The gains for the 8kB simulated machines range from -2.17% to -0.23%,

the average PIN predicted gain is -1.59% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 2.93%, and the average percentage of the maximum M5 gain captured by the PIN prediction is -118.78%. The gains for the 32kB simulated machines range from -2.51% to 4.59%, the average PIN predicted gain is -0.25% relative to the M5 baseline performance, the average gain difference between the M5 maximum gain and the PIN predicted gain is 4.57%, and the average percentage of the maximum M5 gain captured by the PIN prediction is -5.87%. The overall average PIN predicted gain for MOLDYNwith the auto input data set for all L1 caches is -0.90%, the overall average gain difference between the M5 maximum gain and the PIN predicted gain is 2.94%, and the overall average percentage of the maximum M5 gain captured by the PIN prediction is -44.06%. Figures 6.25, 6.26, and 6.27 show in detail the results for the three L1 caches.

6.3 Summary of the PIN Performance Model Results against M5

In this section, we summarize the PIN performance model results across each benchmark and over all benchmarks. The PIN performance model aimed at ranking and therefore predicting the best combination of the parallelizations strategies and locality optimizations (graph partitionings).

Tables 6.10, 6.11, and 6.12 summarize all of the previous results across all three simulated L1 cache sizes. Each column represents the average across L1 caches for a certain L2 cache size. The bottom of each table summarizes the results across

Table 6.9: A comparative summary of the percent gains relative to the baseline for M5 and the PIN performance model for MOLDYN under the **auto** input problem with $4kB - 32kB$ L1 caches and $128kB - 4MB$ L2 caches.

L1 size	4kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
% Gain M5	0.12	0.00	2.23	0.30	0.17	0.01	0.47
% Gain PIN	-5.94	-1.71	2.23	0.30	0.17	-0.19	-0.86
Diff(%M5-%PIN)	6.06	1.71	0.00	0.00	0.00	0.20	1.33
% PIN Captured Gain	-5059.14	N/A	100.00	100.00	100.00	-1888.81	-181.61
L1 size	8kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
% Gain M5	1.19	1.76	1.08	1.26	1.33	1.42	1.34
% Gain PIN	-0.99	-2.54	-0.23	-1.68	-1.93	-2.17	-1.59
Diff(%M5-%PIN)	2.18	4.30	1.31	2.93	3.26	3.59	2.93
% PIN Captured Gain	-83.31	-144.51	-21.66	-133.60	-145.05	-152.67	-118.78
L1 size	32kB						
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
% Gain M5	8.68	5.70	3.55	3.00	2.59	2.39	4.32
% Gain PIN	4.59	1.08	-1.15	-1.46	-2.08	-2.51	-0.25
Diff(%M5-%PIN)	4.09	4.62	4.70	4.46	4.67	4.90	4.57
% PIN Captured Gain	52.88	18.97	-32.29	-48.67	-80.19	-104.63	-5.87
Average across all L1 Sizes							
L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg. auto
% Gain M5	3.33	2.48	2.29	1.52	1.36	1.28	2.04
% Gain PIN	-0.78	-1.06	0.28	-0.94	-1.28	-1.62	-0.90
Diff(%M5-%PIN)	4.11	3.54	2.00	2.46	2.64	2.90	2.94
%PIN Captured Gain	-23.46	-42.56	12.38	-62.22	-93.80	-127.15	-44.06

all inputs for each benchmark. We also report the same two metrics we used to characterize how PIN predictions measure up to the maximum M5 gains.

We notice that (%Gain Diff(M5-PIN)) the absolute difference between the maximum gains that M5 can obtain and the gains that the PIN model would lead to are relatively small. The values are below 2% for L2 caches smaller than 2MB. As we noted before the larger the L2 cache the smaller the gain get. This is due to the fact that the cache can accommodate the data more and it can fit the working set more

thus the optimization become less relevant. At such a larger cache, the partitioner needs to be more intelligent about how to partition the graph. We have an idea that might help with these cases that we outline in Chapter 7. The prediction accuracy for PIN is good quality for L2 caches smaller than 2MB. For IRREG, PIN can acheive up to 77% of the maximum M5 gains but it records degradation for at 4MB of L2 cache. NBF does relatively better compared to IRREG. PIN achieves the maximum captured gains of up to 92% across all benchmarks. Meanwhile, NBF does not acheive the largest absolute gains on M5 relative to other benchmarks. NBF scores the highest overall average %PIN Captured Gain of %57. Actually, the predictions are above 56% of the maximum M5 gains for L2 caches smaller than 2MB. On the contrary, MOLDYN does the worst among all benchmarks in terms of absolute gains and the PIN predictions with maximum average absolute gain of about 3% and maximum %PIN Captured Gain below 42%. In the next section we can suggest that we use an intelligent search that entails running three runs to pick the best among them for larger L2 caches which are across the board recording performance degradation in the PIN predictions.

Table 6.13 reports an overall summary for all benchmarks. Like all performance reporting tables we have shown in this thesis, the reporting is all in percent and all performance gains are reported relative to the baseline case. The average M5 gain is 3.5% which is not large but we have to note that this is above the baseline case which is already a relatively good data point to compare to. The average gain that PIN predictions score is below 2%. The maximum absolute gain difference between M5 and PIN is below 2%. The overall %PIN Captured Gain is below 48%.

In the next Section, we will show how we can improve by a large margin the choice of the parallelization given the top pick for partition size since PIN does a good job choosing that unlike the parallelization.

Table 6.10: A summary of the percent gains relative to the baseline for M5 and the PIN performance model for IRREG for all input problems averaged over all simulated L1 caches for L2 caches $128kB - 4MB$. Note all numbers are in percent.

Graph	L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
144	%Gain M5	5.35	4.07	3.43	3.18	3.11	1.66	3.47
	%Gain PIN	3.82	2.07	2.45	0.67	0.06	-3.25	0.97
m14b	%Gain M5	3.30	2.36	2.07	1.80	1.31	1.09	1.99
	%Gain PIN	1.86	1.17	0.49	0.36	-0.95	-3.24	-0.05
auto	%Gain M5	11.48	8.16	7.26	5.94	5.67	5.17	7.28
	%Gain PIN	9.79	7.05	6.47	4.01	1.19	-0.12	4.73
Avg.	% Gain M5	6.71	4.86	4.25	3.64	3.37	2.64	4.24
	%Gain PIN	5.15	3.43	3.14	1.68	0.10	-2.20	1.88
	%Gain Diff(M5-PIN)	1.56	1.43	1.11	1.96	3.27	4.84	2.36
	%PIN Captured Gain	76.80	70.58	73.80	46.18	2.94	-83.56	44.37

Table 6.11: A summary of the percent gains relative to the baseline for M5 and the PIN performance model for NBF for all input problems averaged over all simulated L1 caches for L2 caches $128kB - 4MB$. Note all numbers are in percent.

Graph	L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
144	%Gain M5	6.16	4.37	3.42	3.27	1.90	1.35	3.41
	%Gain PIN	5.27	3.11	3.16	1.79	-0.75	-2.62	1.66
m14b	%Gain M5	2.51	1.89	2.06	1.94	1.77	0.26	1.74
	%Gain PIN	1.30	0.88	1.64	-0.07	-1.16	-3.77	-0.20
auto	%Gain M5	9.47	7.00	7.13	4.97	5.37	5.37	6.55
	%Gain PIN	9.13	6.33	6.80	4.03	2.83	1.97	5.18
Avg.	% Gain M5	6.05	4.42	4.20	3.39	3.01	2.32	3.90
	%Gain PIN	5.24	3.44	3.87	1.92	0.31	-1.48	2.22
	%Gain Diff(M5-PIN)	0.81	0.98	0.34	1.47	2.71	3.80	1.68
	%PIN Captured Gain	86.62	77.88	91.94	56.54	10.13	-63.46	56.81

Table 6.12: A summary of the percent gains relative to the baseline for M5 and the PIN performance model for MOLDYN for all input problems averaged over all simulated L1 caches for L2 caches $128kB$ – $4MB$. Note all numbers are in percent.

Graph	L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
144	%Gain M5	4.71	3.66	3.17	3.16	3.04	3.01	3.46
	%Gain PIN	3.28	2.46	2.09	2.14	1.95	2.50	2.40
m14b	%Gain M5	1.97	1.17	1.03	1.00	1.10	1.11	1.23
	%Gain PIN	1.68	0.90	0.79	0.72	0.78	0.61	0.91
auto	%Gain M5	3.33	2.48	2.29	1.52	1.36	1.28	2.04
	%Gain PIN	-0.78	-1.06	0.28	-0.94	-1.28	-1.62	-0.90
Avg.	% Gain M5	3.33	2.44	2.16	1.89	1.84	1.80	2.24
	%Gain PIN	1.39	0.77	1.05	0.64	0.48	0.50	0.80
	%Gain Diff(M5-PIN)	1.94	1.67	1.11	1.26	1.35	1.30	1.44
	%PIN Captured Gain	41.71	31.45	48.78	33.57	26.29	27.55	35.85

Table 6.13: A summary of the percent gains relative to the baseline for M5 and the PIN performance model averaged across ***all benchmarks***, for all input problems averaged over all simulated L1 caches for L2 caches $128kB$ – $4MB$. Note all numbers are in percent.

Graph	L2 size	128kB	256kB	512kB	1MB	2MB	4MB	Avg.
144	%Gain M5	5.41	4.03	3.34	3.20	2.69	2.01	3.45
	%Gain PIN	4.12	2.55	2.57	1.53	0.42	-1.13	1.68
m14b	%Gain M5	2.59	1.81	1.72	1.58	1.39	0.82	1.65
	%Gain PIN	1.61	0.99	0.97	0.33	-0.45	-2.13	0.22
auto	%Gain M5	8.09	5.88	5.56	4.14	4.13	3.94	5.29
	%Gain PIN	6.05	4.11	4.52	2.37	0.91	0.08	3.01
Avg.	% Gain M5	5.36	3.91	3.54	2.97	2.74	2.25	3.46
	%Gain PIN	3.93	2.55	2.69	1.41	0.30	-1.06	1.63
	%Gain Diff(M5-PIN)	1.44	1.36	0.85	1.56	2.44	3.31	1.83
	%PIN Captured Gain	73.22	65.18	75.89	47.44	10.80	-47.09	47.20

6.4 The "Three Run" Scheme: Partition Size Prediction Accuracy and Improvements on Parallelization Prediction

In this section, we show that the PIN prediction model can predict the best partition size with great accuracy but it does not perform similarly when it comes to predicting the best parallelization that goes with the partition size it predicts.

Tables 6.14, 6.15, and 6.16 summarize the partition size prediction using the PIN performance model for Irreg, NBF, and Moldyn respectively. Each entry in the tables reports two ordered pairs of results. The first is the partition size pair which gives the PIN predicted partition size followed by the best partition size as computed by M5. The second is a pair of two ranks. The first rank is the rank of the top pick that the PIN model predicts. The second rank is what we call the "Three Run" rank. The "Run Three" rank is the rank that we would obtain if we run all combinations of parallelizations with the PIN predicted partition size except for the "CMP" parallelization. We clearly have seen that this parallelization when we ran on M5 performs consistently worse than the rest. Furthermore, PIN over-estimates the performance of this parallelization in particular. Therefore, we decided to ignore it for our "Run Three" scheme.

We note that in all the cases where the "Three Run" can claim the best M5 performance with a rank of "1" as the value in the second pair of values the PIN predicted partition size is actually the best partition size that M5 picked as well therefore running all three possible parallelizations guarantees the best performance.

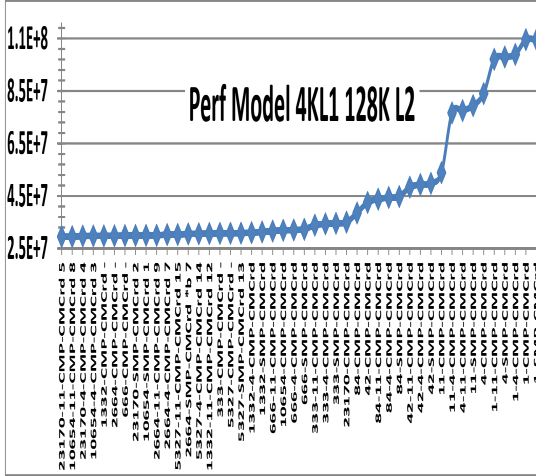
Tables 6.14, 6.15, and 6.16 summarize effectively the results presented in

Figures 6.1 – 6.27, yet we abstract the results in these tables even more. Table 6.17 is a histogram of how many times each rank appeared as the rank that PIN predicts with what we call the "Zero Run" scheme. The "Zero Run" scheme predicts performance using no run on simulators or machines and uses only the for the profiles collected using the PIN tool. We can clearly see that the top rank of "1" appears only 16 times out of a total of 162 or in about 10% of the time only using the "Zero Run" scheme. We also notice that ranks 1 – 19 appear with this scheme. The number 54 in the totals that appear in the table represents the product of the number of L1 sizes simulated, the number of L2 sizes simulated, and the number of inputs we have for each benchmark. The number 162 in the totals that appear in the table represents the previous product multiplied by the number of benchmarks we run.

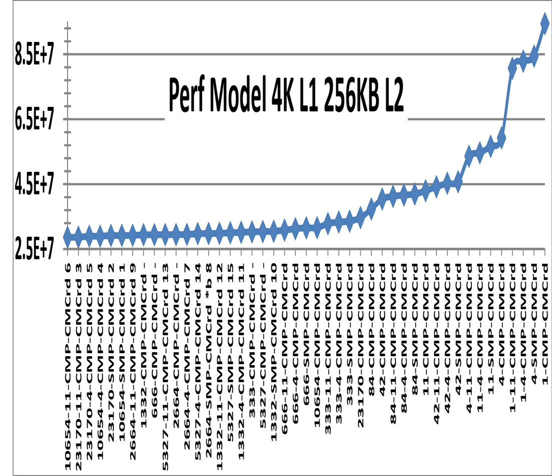
On the other hand, applying the "Three Run" scheme the histogram of the ranks appears as in Table 6.18. The relative sizes of the Table 6.17 and 6.18 summarizes the conclusion. If we are allowed the "Three Run" scheme we can achieve the best M5 performance in 94 out of 162 or 58% of the time. The sum of the percentage of the total of the first three ranks (1, 2, and 3) in the "Three Run" scheme is 90.1%, which means that with the "Three Run" scheme would get one of the top three M5 picks 146 time out of the 162 total.

Table 6.14: Summary of Partition Size prediction using PIN against the best selected by M5 for the IRREG benchmark with each of the three input data set under all L1 and L2 cache size selections.

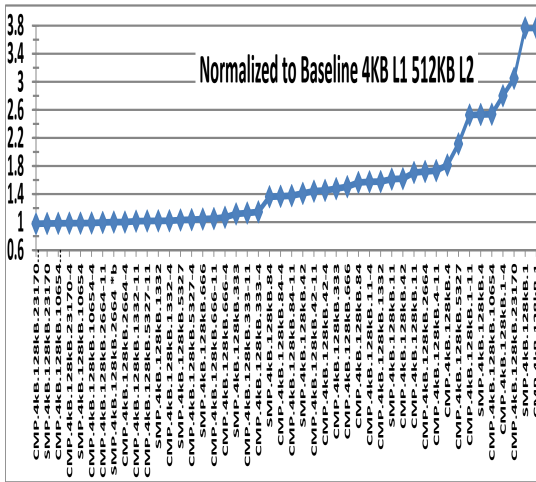
L2/L1	4kB	8kB	32kB
144			
128kB	(23170,10654) (5,2)	(2664,10654) (9,4)	(2664,2664) (10,1)
256kB	(10654,10654) (6,1)	(2664,10654) (9,7)	(10654,2664) (7,3)
512kB	(23170,23170) (1,1)	(2664,10654) (7,7)	(2664,10654) (8,2)
1MB	(10654,23170) (8,2)	(2664,10654) (10,4)	(2664,10654) (13,3)
2MB	(23170,23170) (7,1)	(2664,10654) (12,3)	(10654,10654) (15,1)
4MB	(23170,23170) (9,1)	(10654,23170) (13,2)	(10654,5327) (19,4)
m14b			
128kB	(16474,23170) (5,2)	(4119,16474) (7,4)	(4119,4119) (5,1)
256kB	(16474,23170) (6,2)	(4119,16474) (10,5)	(4119,4119) (3,1)
512kB	(16474,16474) (3,1)	(4119,16474) (12,4)	(2060,16474) (13,11)
1MB	(16474,23170) (7,2)	(4119,23170) (9,5)	(4119,4119) (10,1)
2MB	(16474,23170) (9,3)	(4119,23170) (12,5)	(4119,4119) (16,1)
4MB	(16474,23170) (10,2)	(4119,23170) (16,3)	(164744119) (16,3)
auto			
128kB	(16436,16436) (3,1)	(16436,16436) (3,1)	(16436,16436) (3,1)
256kB	(16436,16436) (3,1)	(16436,16436) (3,1)	(16436,16436) (3,1)
512kB	(16436,16436) (3,1)	(16436,16436) (3,1)	(16436,16436) (3,1)
1MB	(16436,16436) (3,1)	(16436,16436) (3,1)	(16436,16436) (4,1)
2MB	(16436,16436) (3,1)	(16436,16436) (4,1)	(16436,16436) (8,1)
4MB	(16436,16436) (6,1)	(16436,16436) (7,1)	(16436,16436) (11,1)



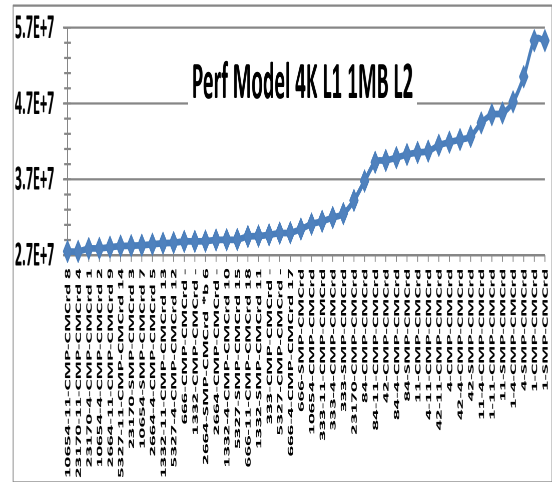
(a) 128kB L2.



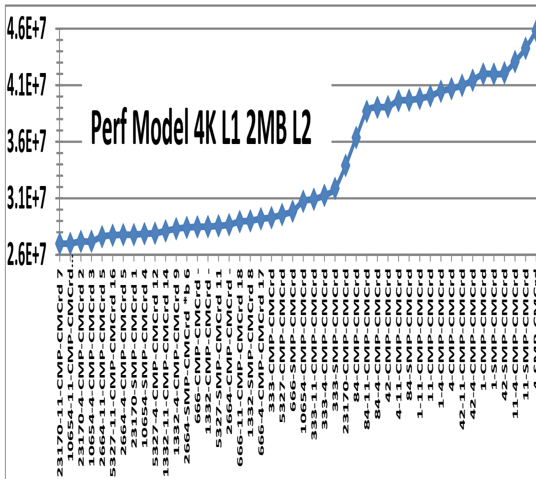
(b) 256kB L2.



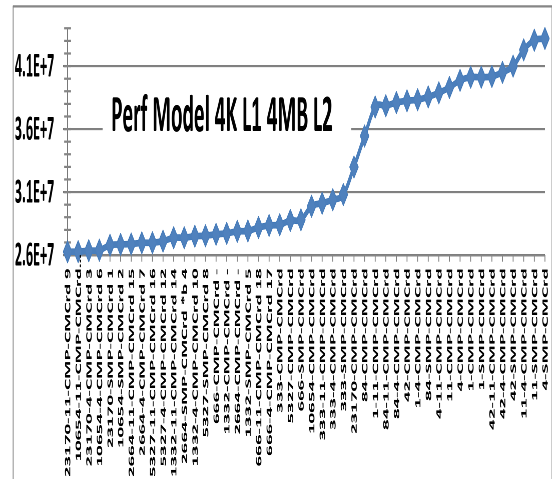
(c) 512kB L2.



(d) 1MB L2.

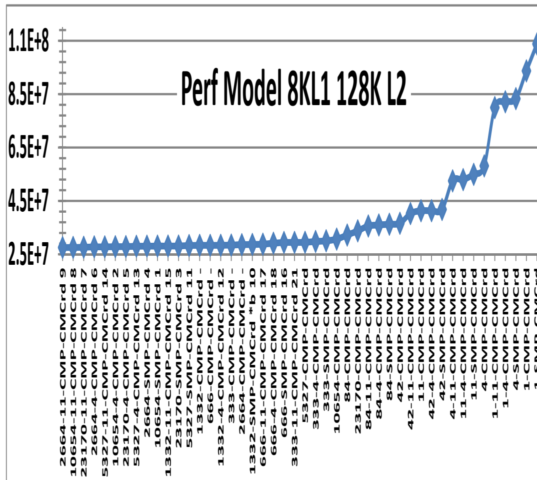


(e) 2MB L2.

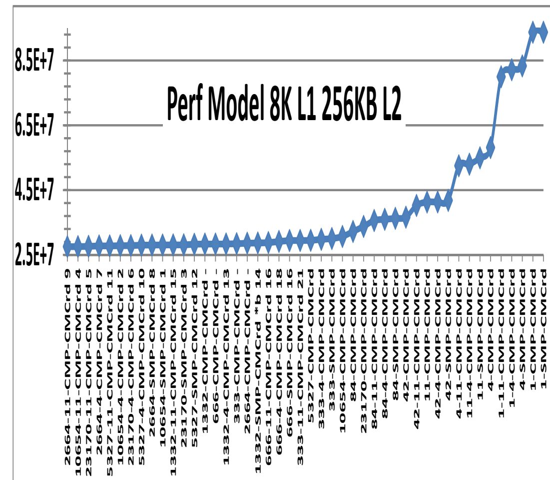


(f) 4MB L2.

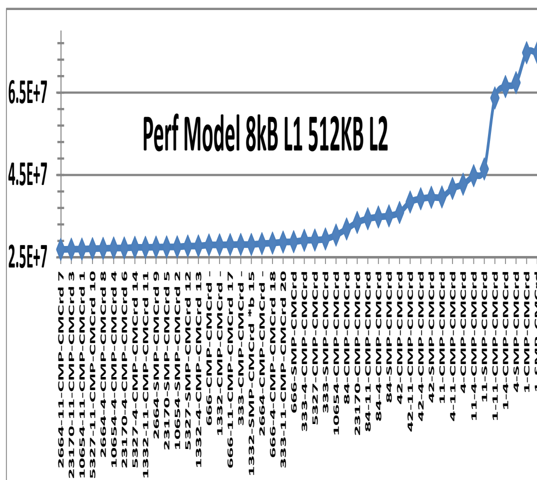
Figure 6.1: Ordered PIN Performance Model results for IRREG under the **144** problem with $4kB$ L1 caches and $128kB - 4MB$ L2 caches.



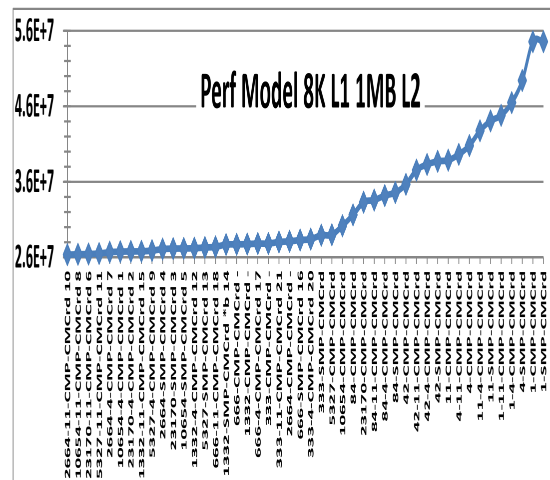
(a) 128kB L2.



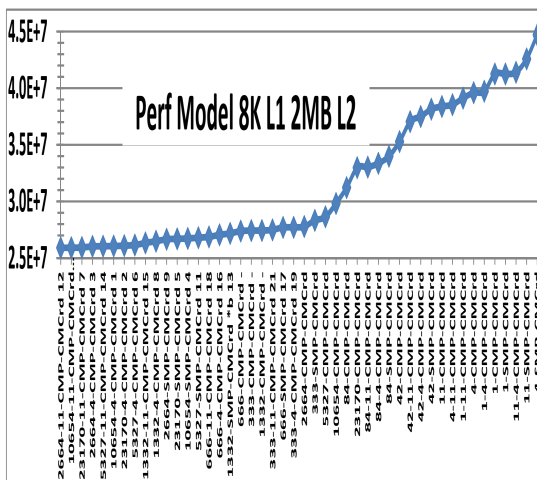
(b) 256kB L2.



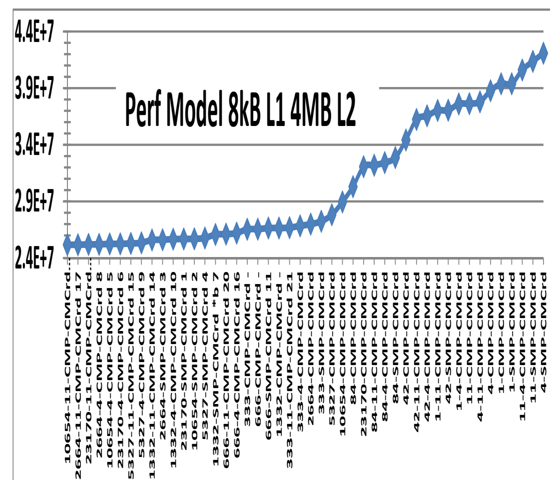
(c) 512kB L2.



(d) 1MB L2.

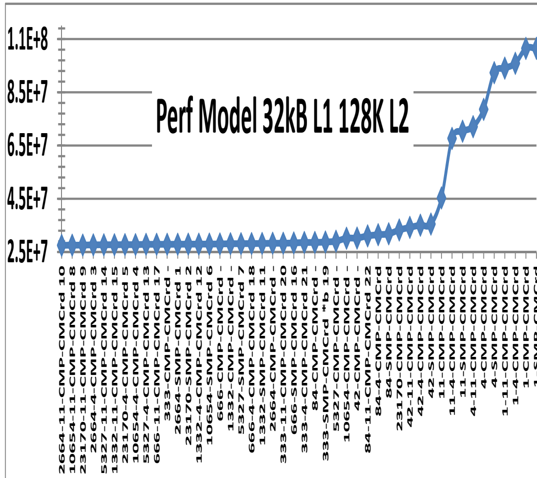


(e) 2MB L2.

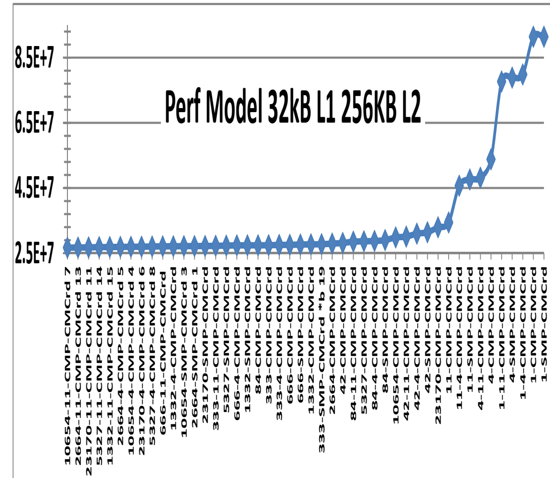


(f) 4MB L2.

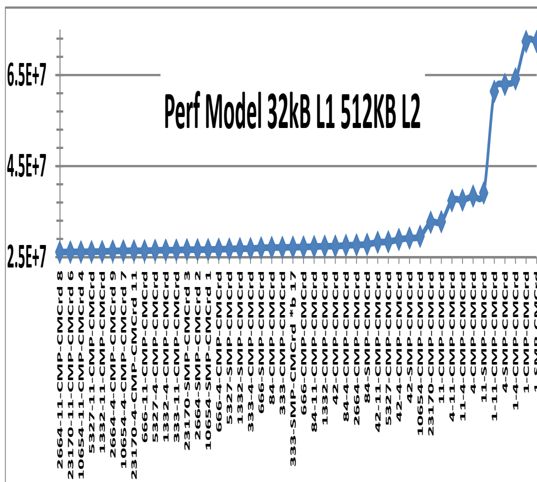
Figure 6.2: Ordered PIN Performance Model results for IRREG under the **144** problem with $8kB$ L1 caches and $128kB - 4MB$ L2 caches.



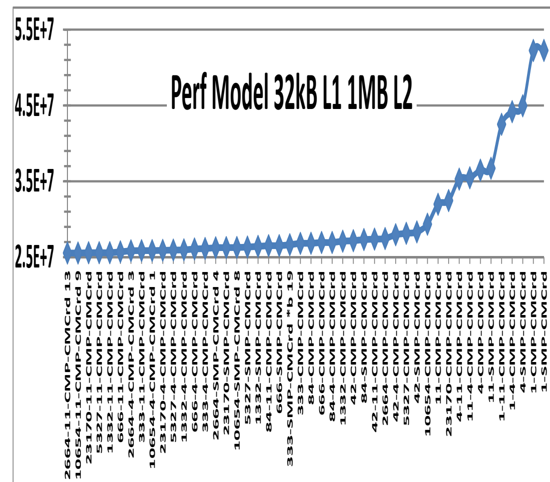
(a) 128kB L2.



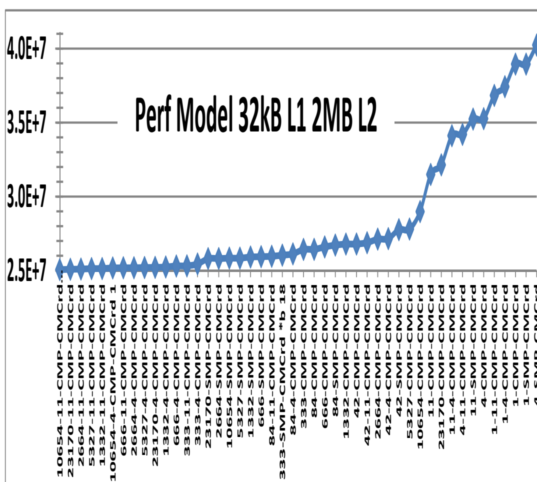
(b) 256kB L2.



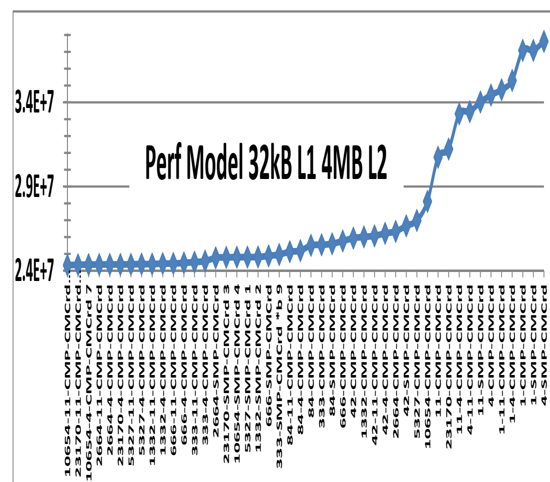
(c) 512kB L2.



(d) 1MB L2.

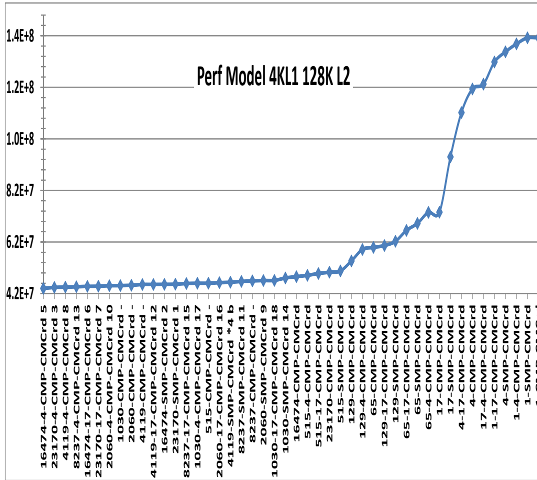


(e) 2MB L2.

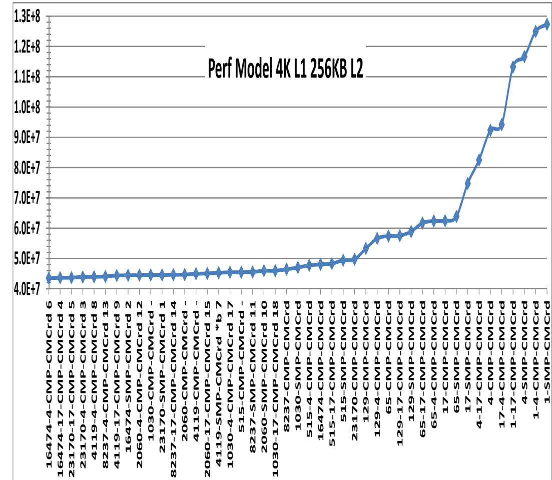


(f) 4MB L2.

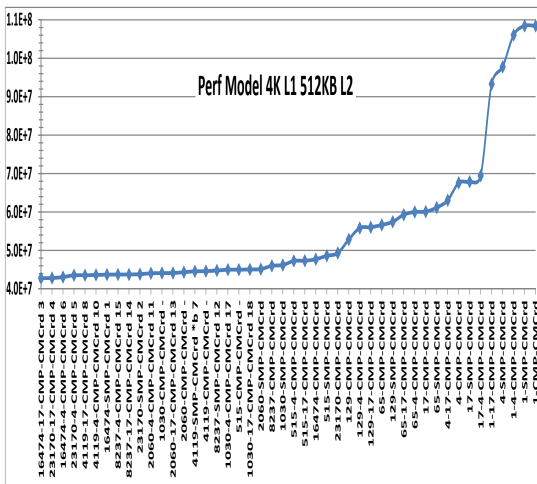
Figure 6.3: Ordered PIN Performance Model results for IRREG under the **144** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches.



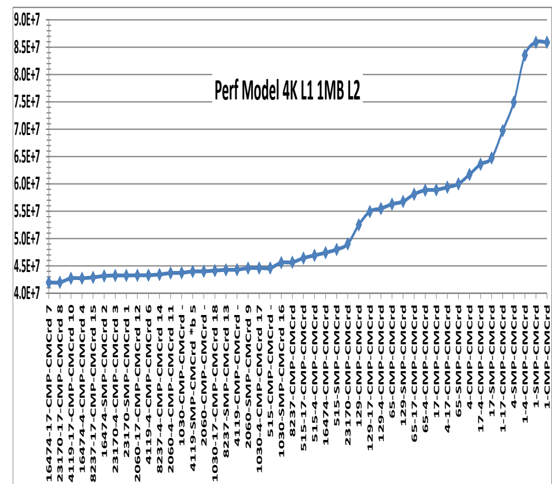
(a) 128kB L2.



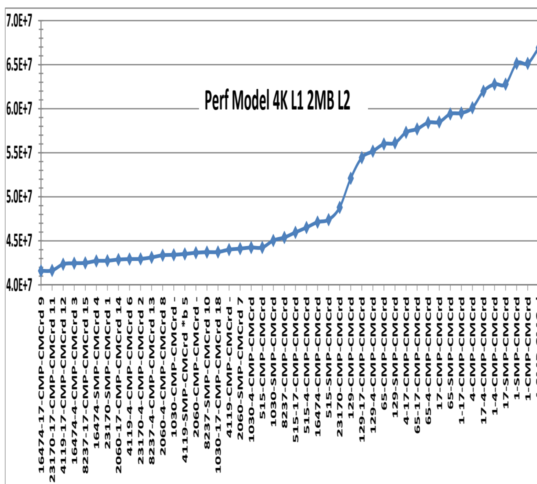
(b) 256kB L2.



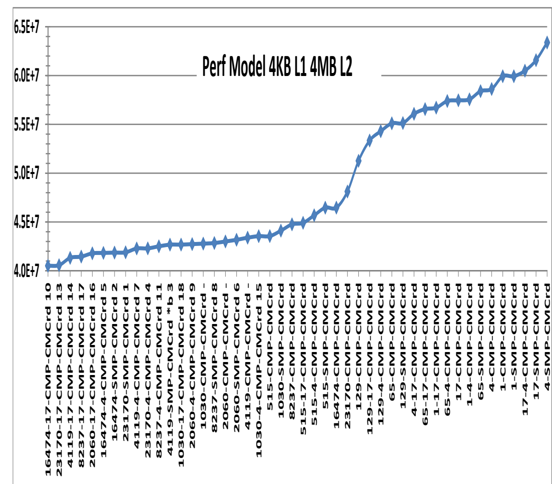
(c) 512kB L2.



(d) 1MB L2.

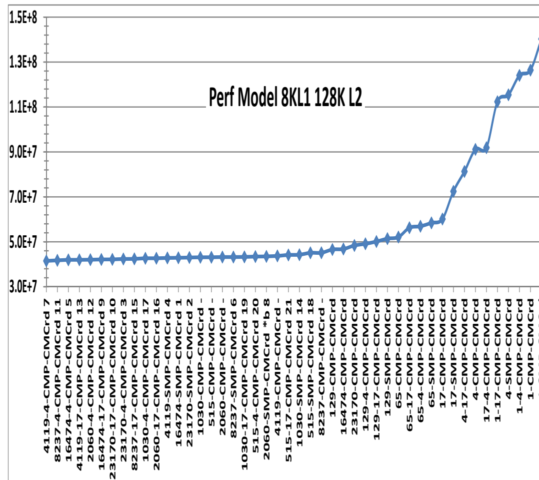


(e) 2MB L2.

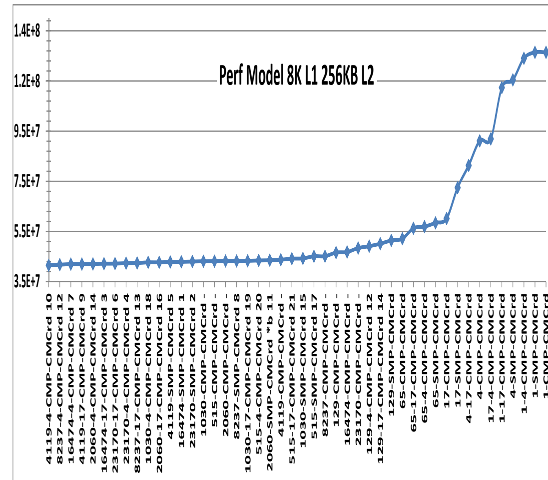


(f) 4MB L2.

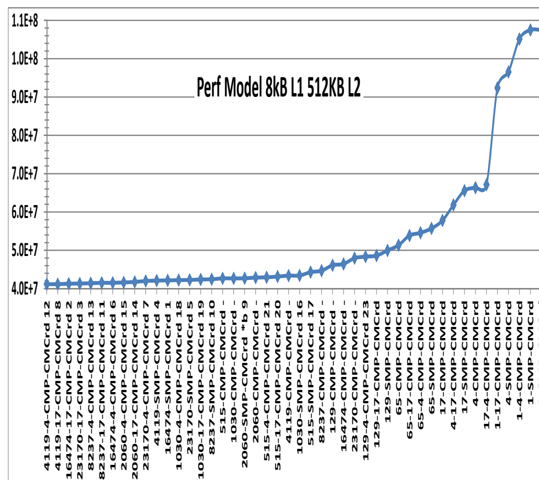
Figure 6.4: Ordered PIN Performance Model results for IRREG under the **m14b** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches.



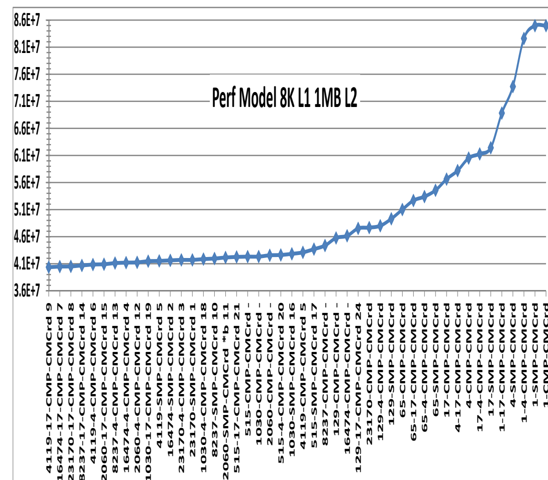
(a) 128kB L2.



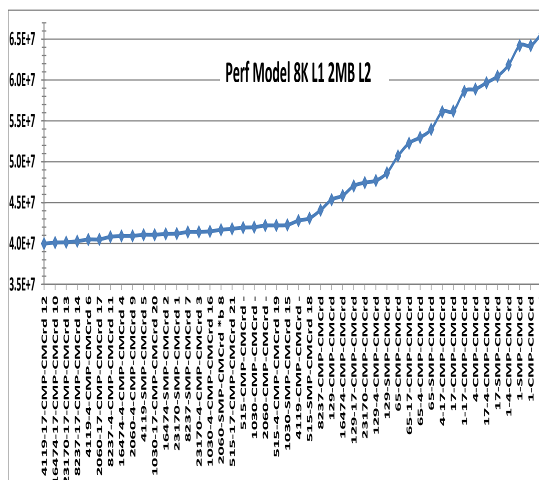
(b) 256kB L2.



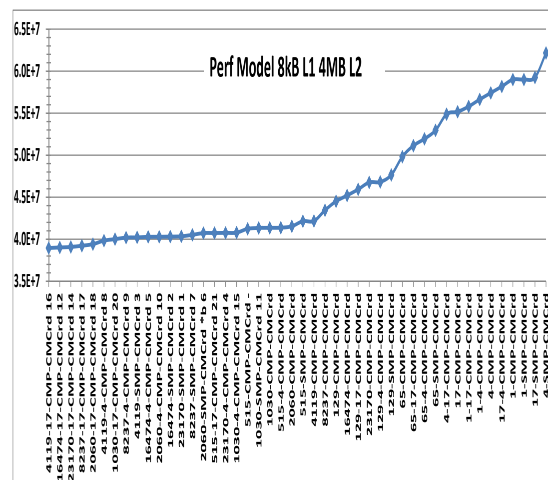
(c) 512kB L2.



(d) 1MB L2.

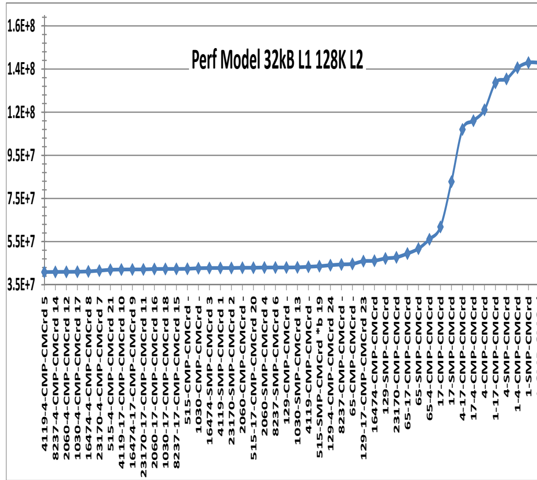


(e) 2MB L2.

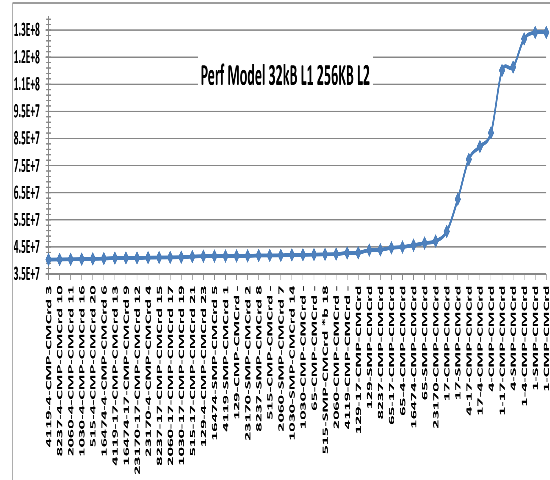


(f) 4MB L2.

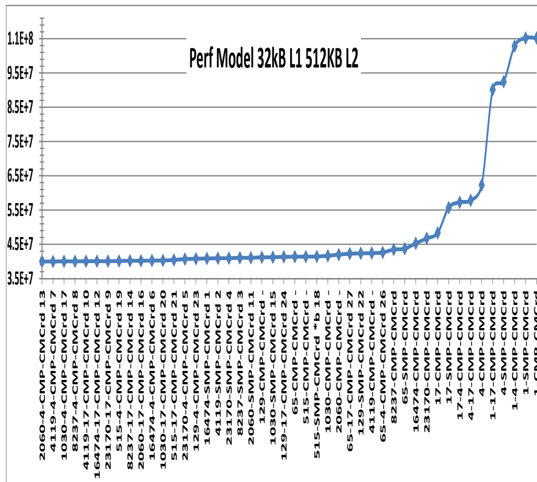
Figure 6.5: Ordered PIN Performance Model results for IRREG under the **m14b** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches.



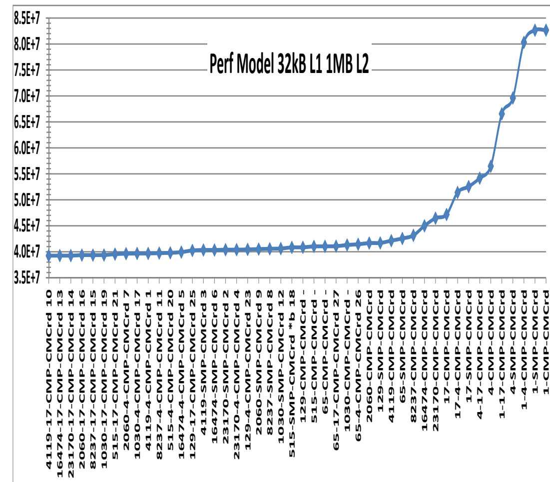
(a) 128kB L2.



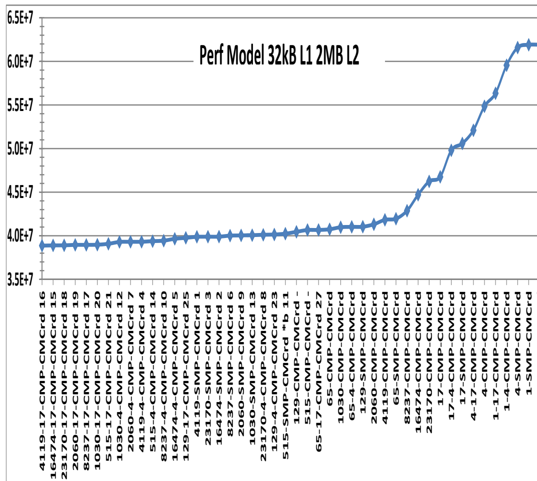
(b) 256kB L2.



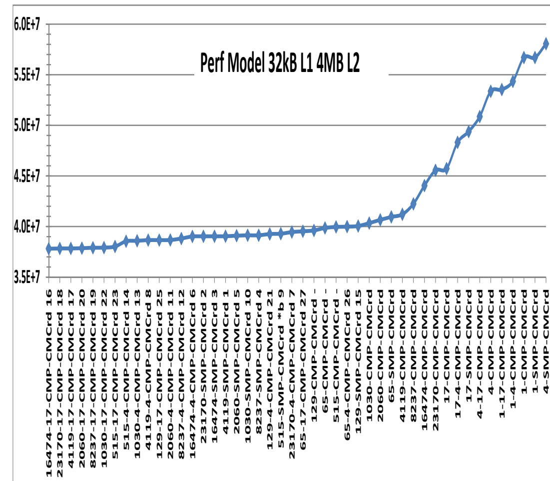
(c) 512kB L2.



(d) 1MB L2.

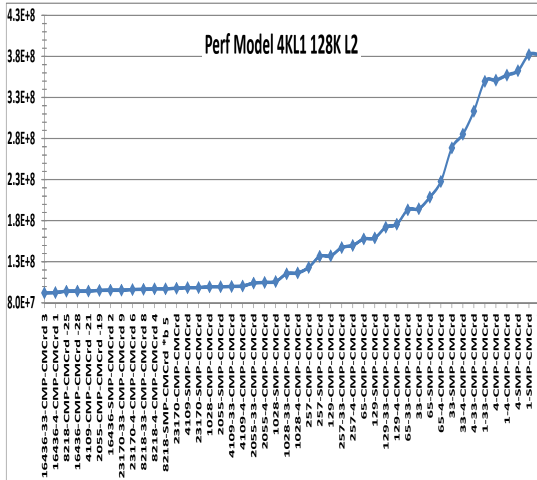


(e) 2MB L2.

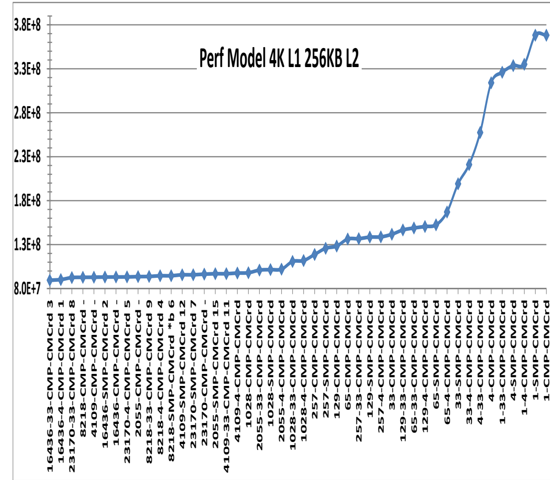


(f) 4MB L2.

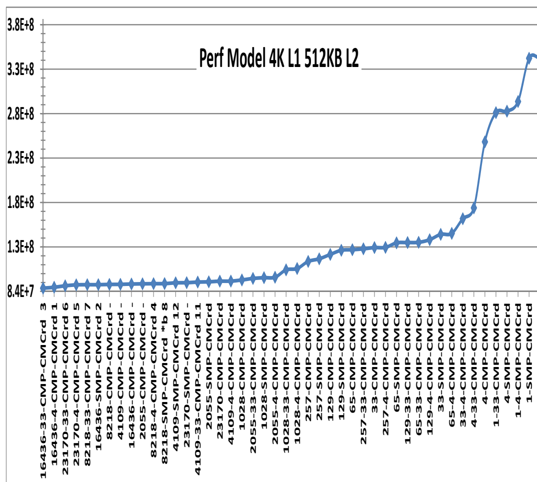
Figure 6.6: Ordered PIN Performance Model results for IRREG under the **m14b** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches.



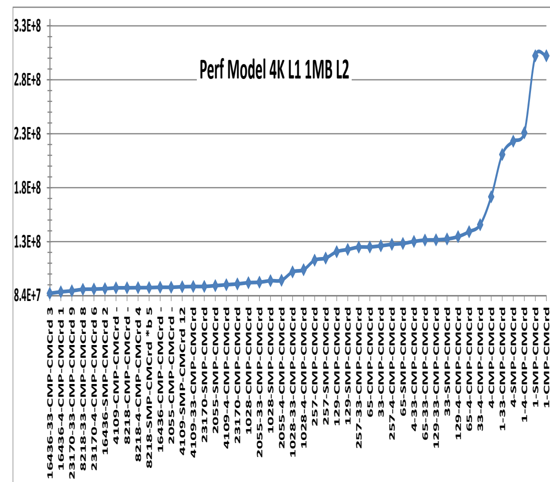
(a) 128kB L2.



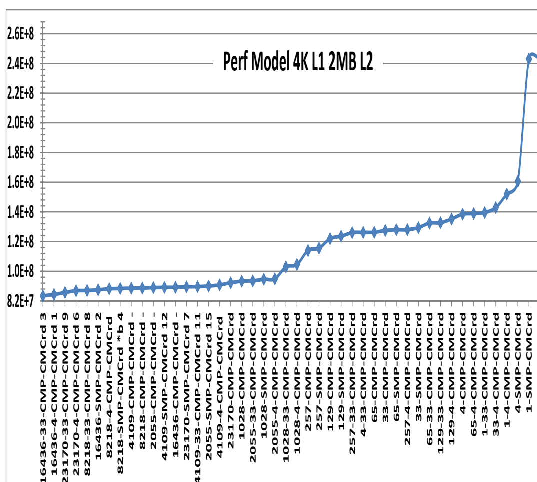
(b) 256kB L2.



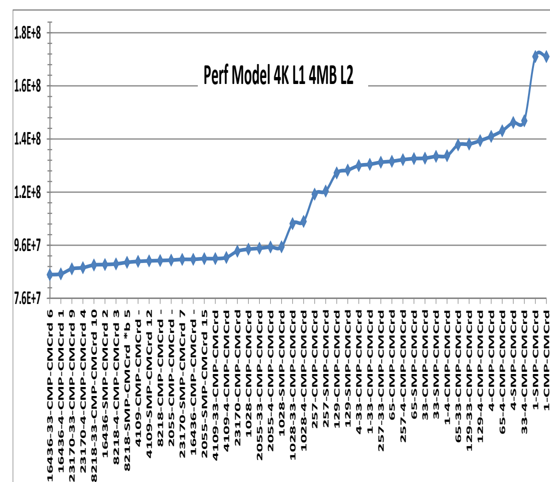
(c) 512kB L2.



(d) 1MB L2.

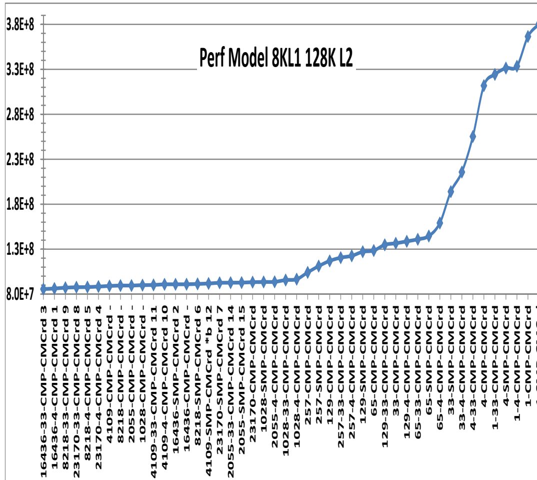


(e) 2MB L2.

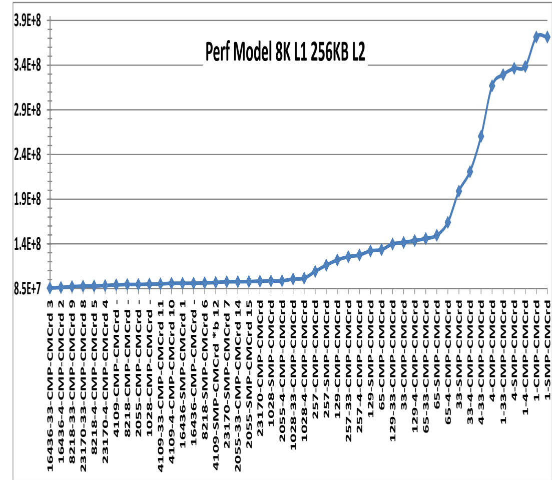


(f) 4MB L2.

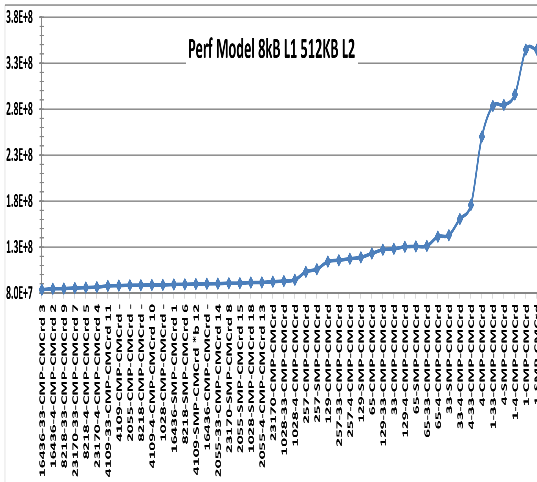
Figure 6.7: Ordered PIN Performance Model results for IRREG under the **auto** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches.



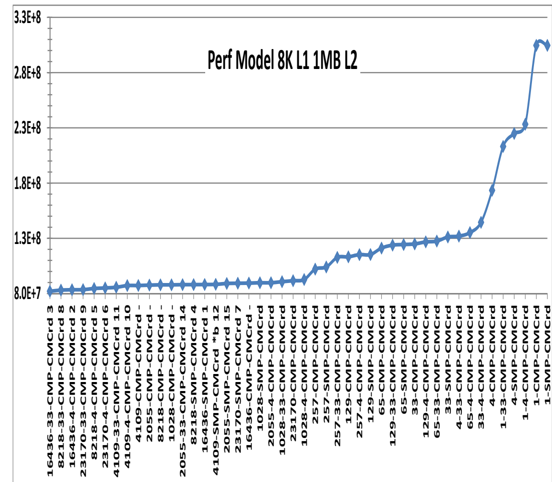
(a) 128kB L2.



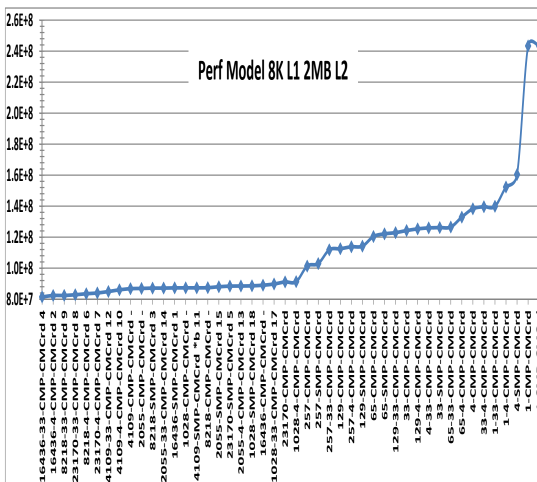
(b) 256kB L2.



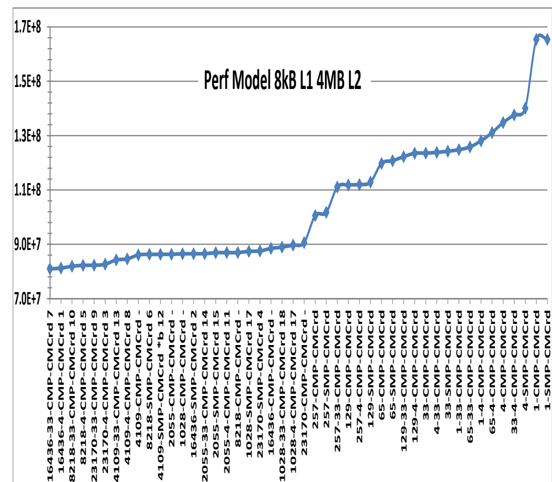
(c) 512kB L2.



(d) 1MB L2.

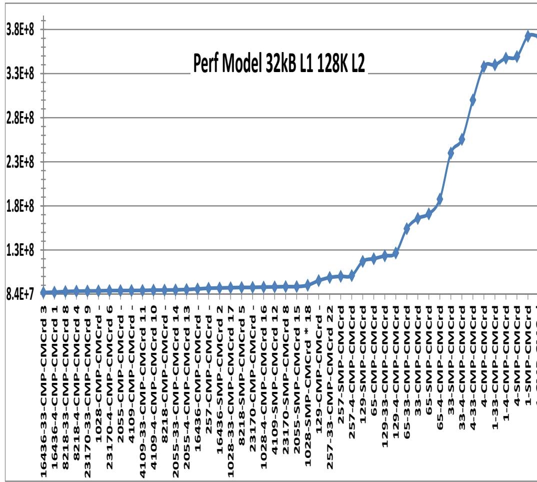


(e) 2MB L2.

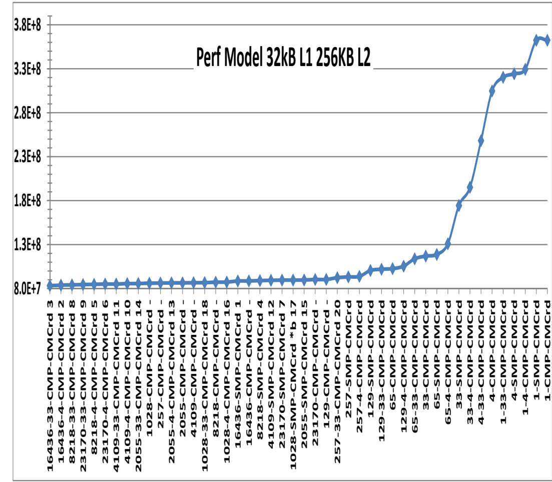


(f) 4MB L2.

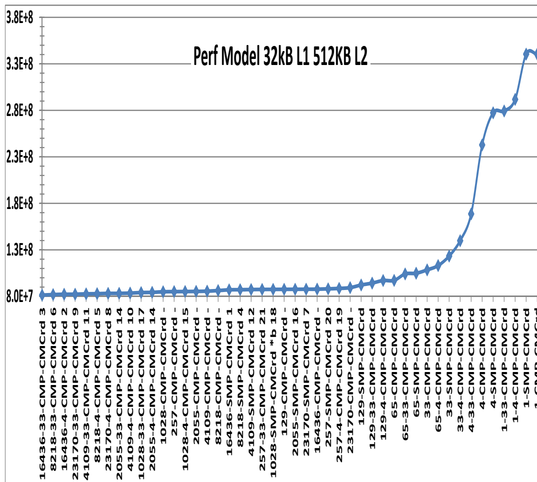
Figure 6.8: Ordered PIN Performance Model results for IRREG under the **auto** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches.



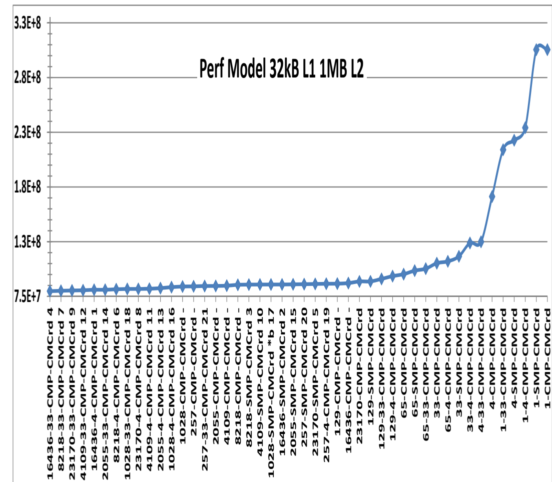
(a) 128kB L2.



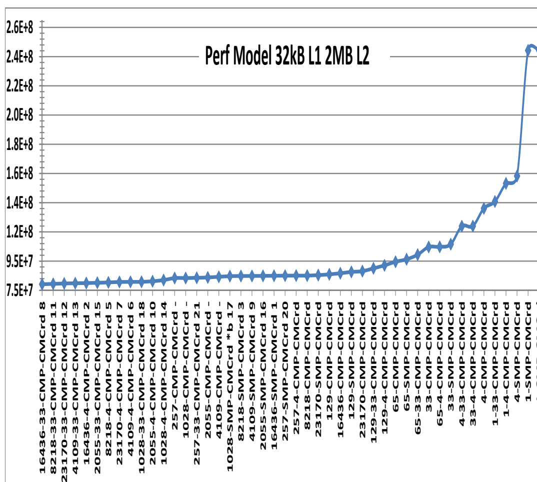
(b) 256kB L2.



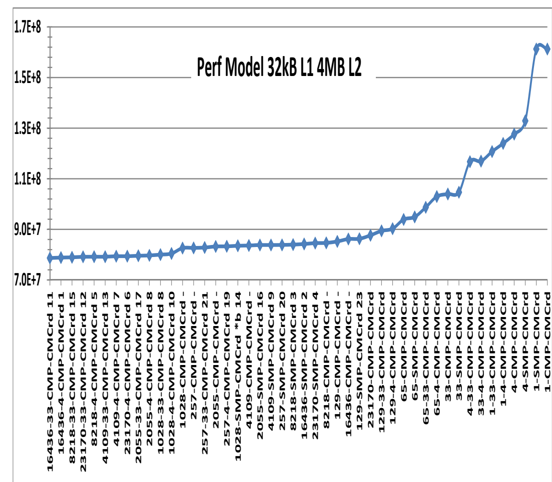
(c) 512kB L2.



(d) 1MB L2.

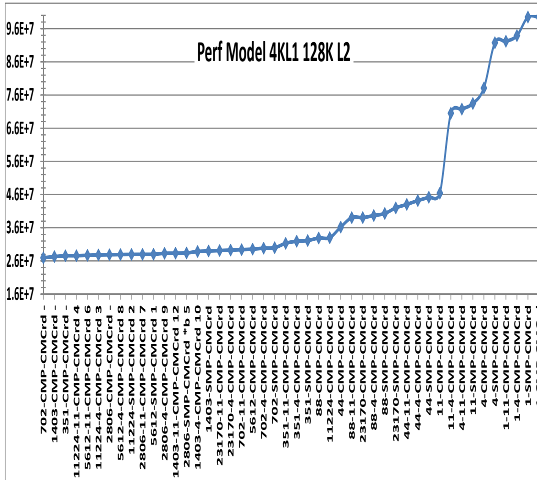


(e) 2MB L2.

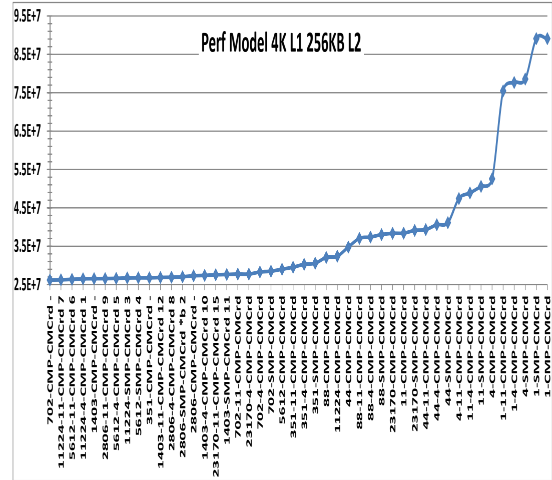


(f) 4MB L2.

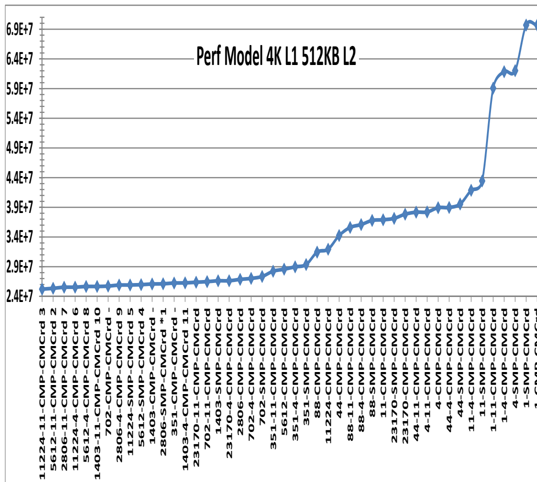
Figure 6.9: Ordered PIN Performance Model results for Irreg under the **auto** problem with $32kB$ L1 cache and $128kB - 4MB$ L2 caches.



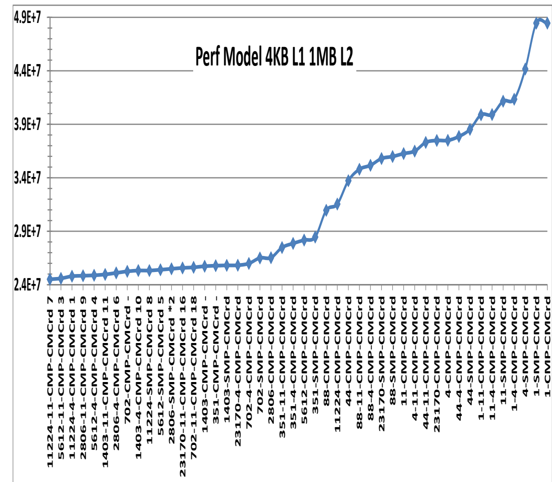
(a) 128kB L2.



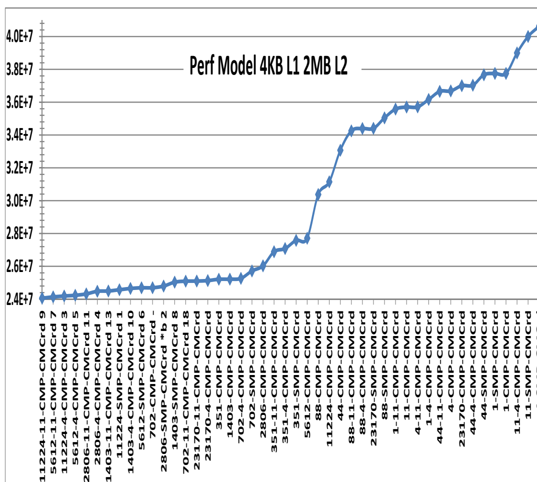
(b) 256kB L2.



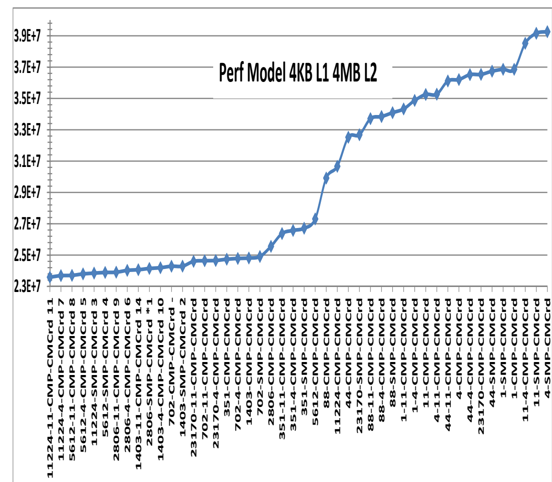
(c) 512kB L2.



(d) 1MB L2.

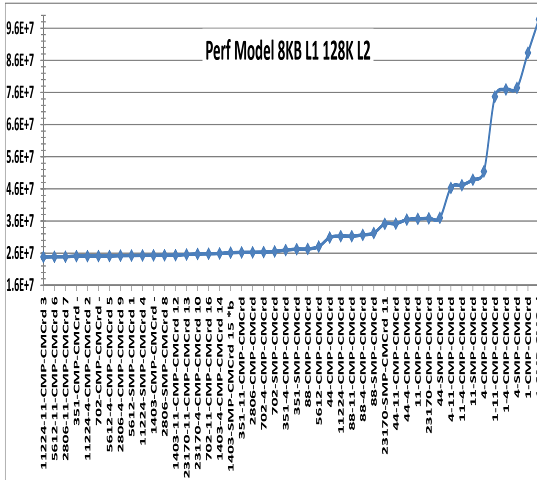


(e) 2MB L2.

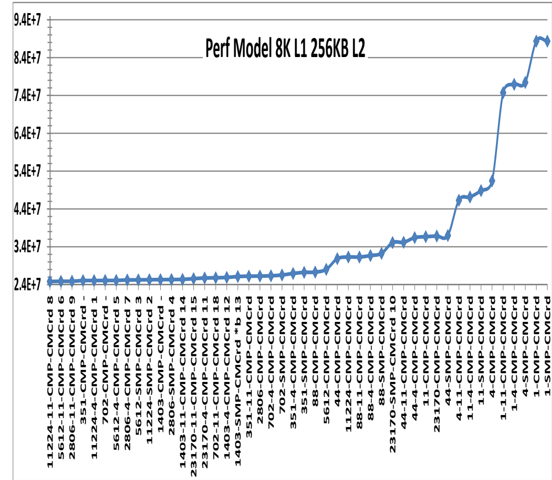


(f) 4MB L2.

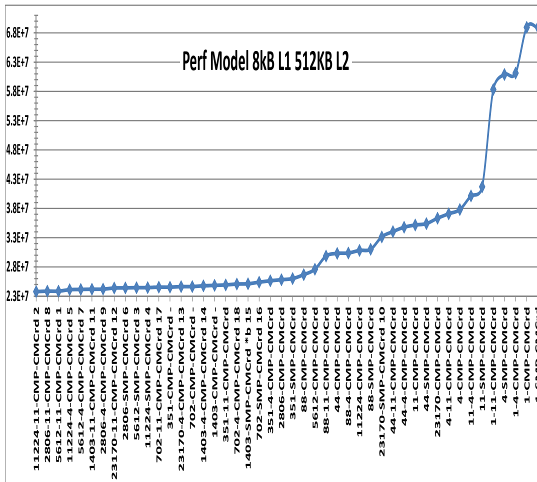
Figure 6.10: Ordered PIN Performance Model results for NBF under the **144** problem with $4kB$ L1 caches and $128kB - 4MB$ L2 caches.



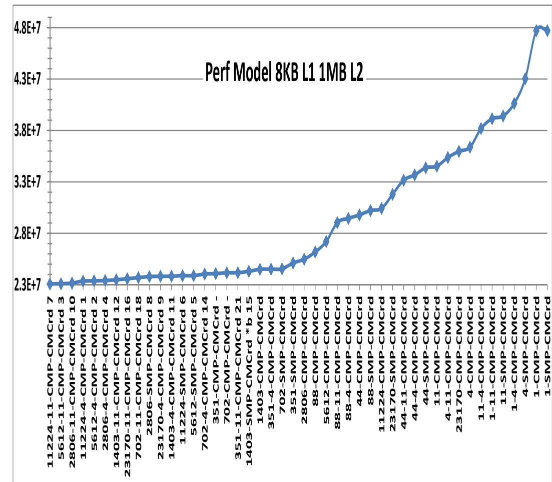
(a) 128kB L2.



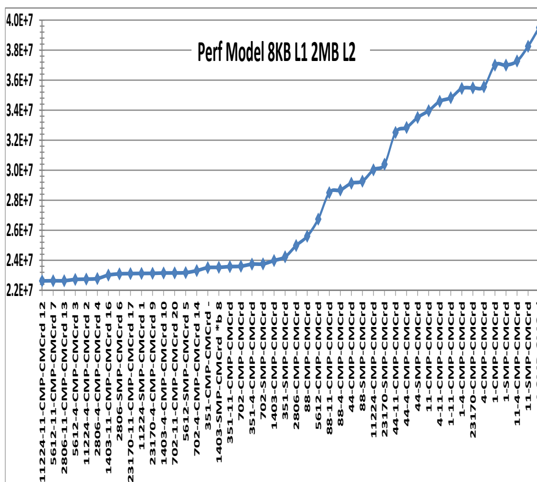
(b) 256kB L2.



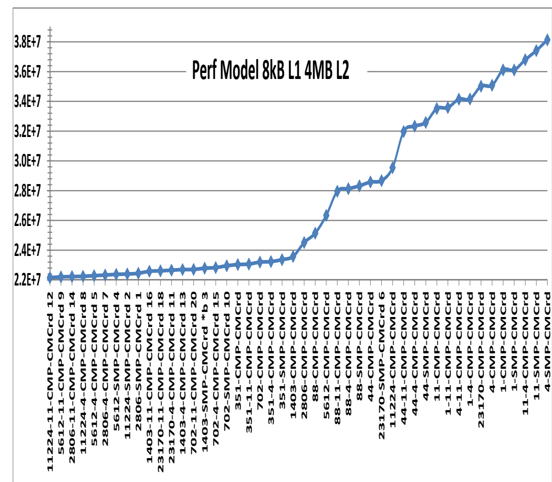
(c) 512kB L2.



(d) 1MB L2.

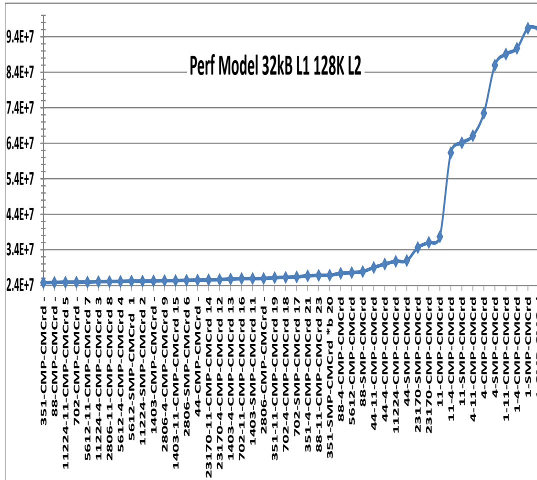


(e) 2MB L2.

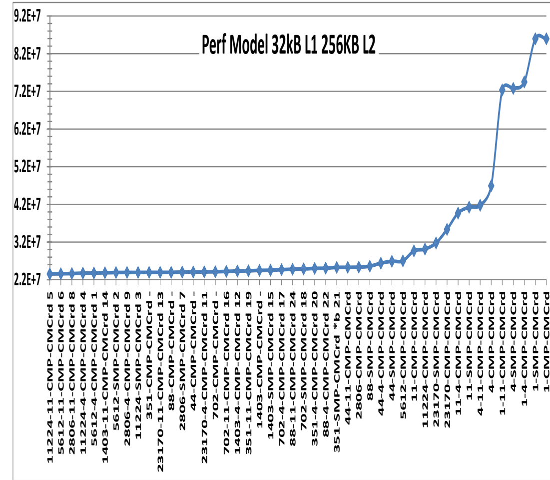


(f) 4MB L2.

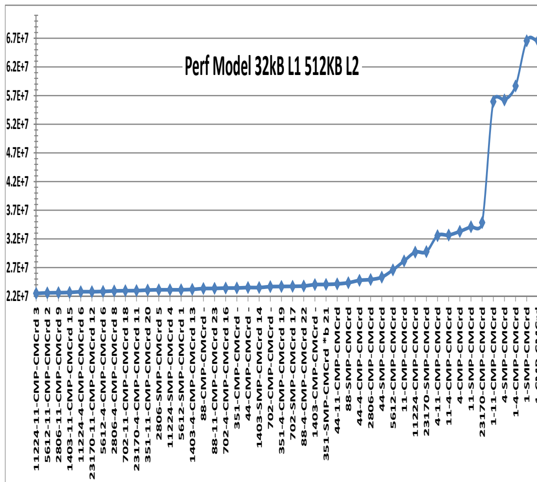
Figure 6.11: Ordered PIN Performance Model results for NBF under the **144** problem with $8kB$ L1 caches and $128kB - 4MB$ L2 caches.



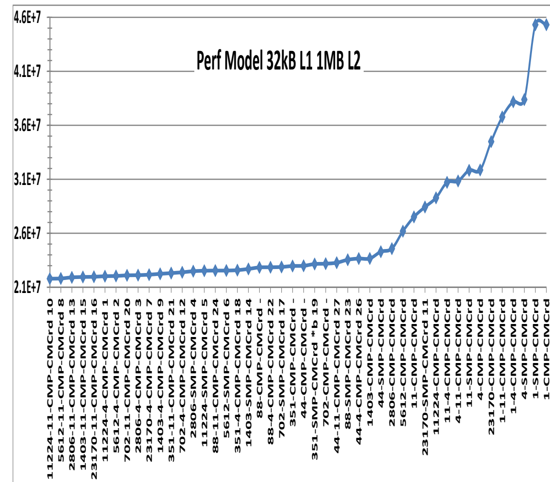
(a) 128kB L2.



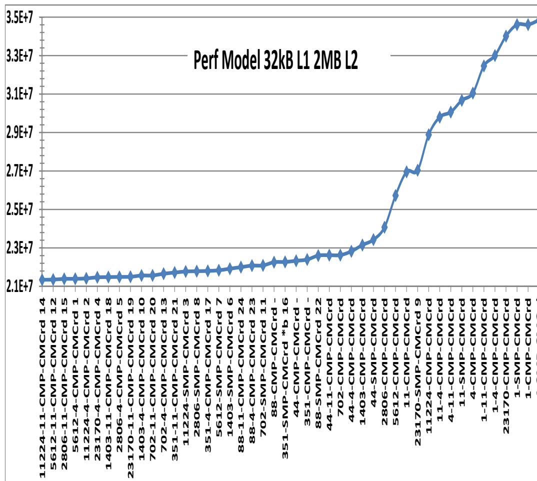
(b) 256kB L2.



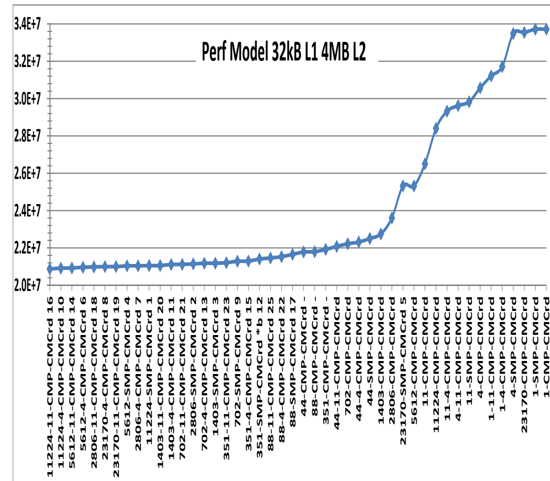
(c) 512kB L2.



(d) 1MB L2.

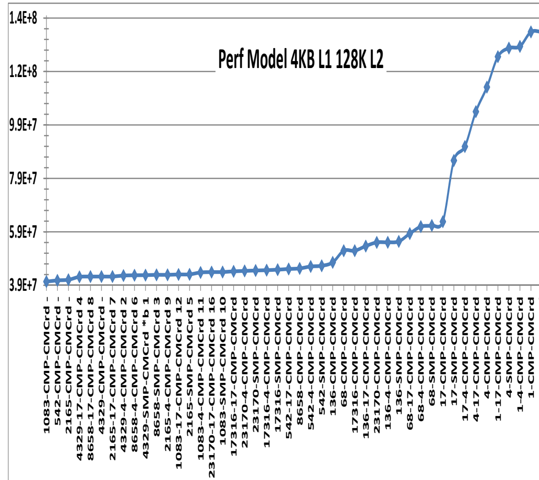


(e) 2MB L2.

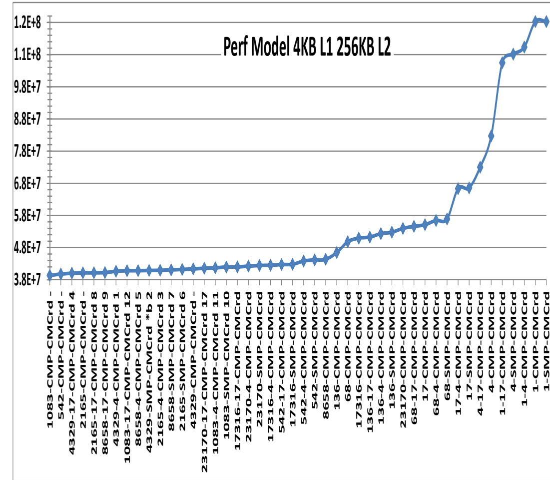


(f) 4MB L2.

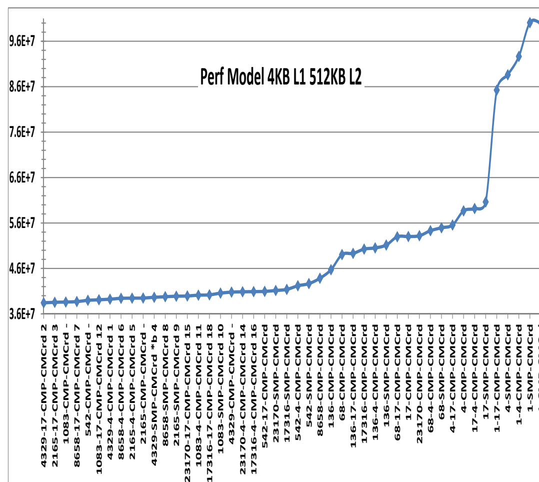
Figure 6.12: Ordered PIN Performance Model results for NBF under the **144** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches.



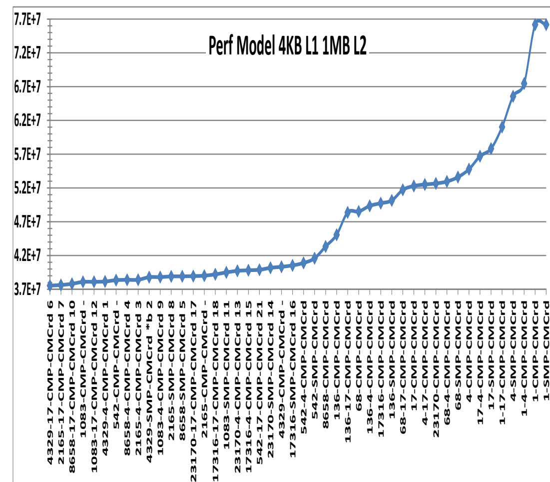
(a) 128kB L2.



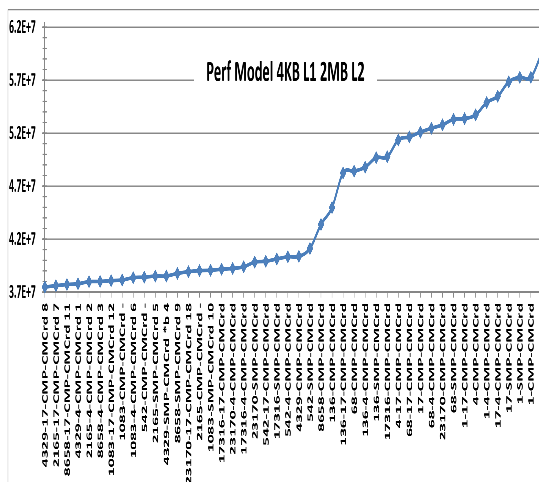
(b) 256kB L2.



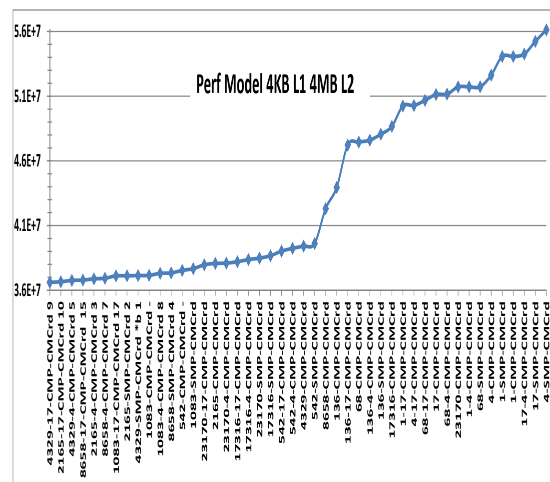
(c) 512kB L2.



(d) 1MB L2.

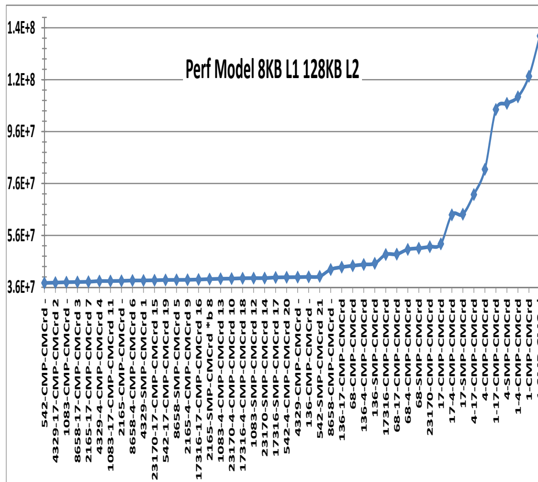


(e) 2MB L2.

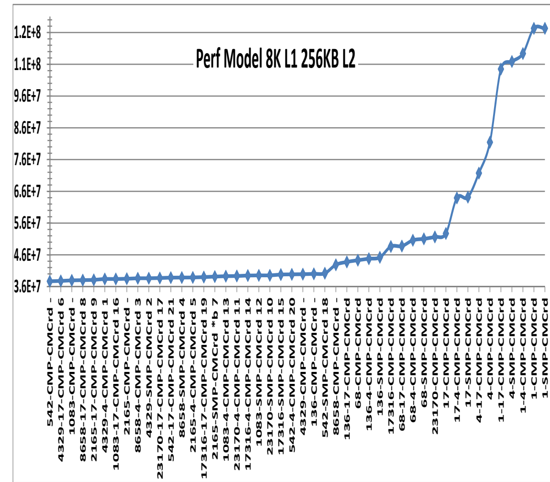


(f) 4MB L2.

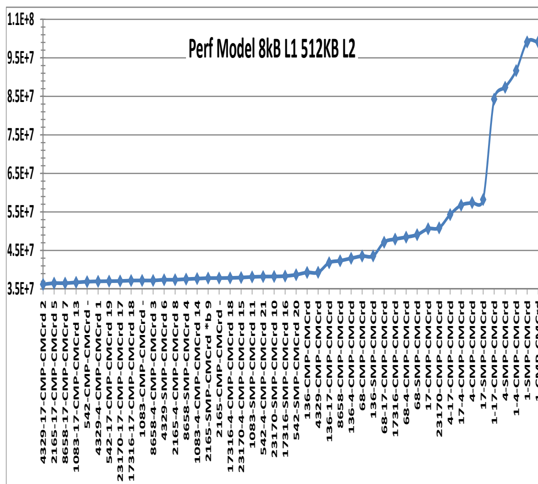
Figure 6.13: Ordered PIN Performance Model results for NBF under the **m14b** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches.



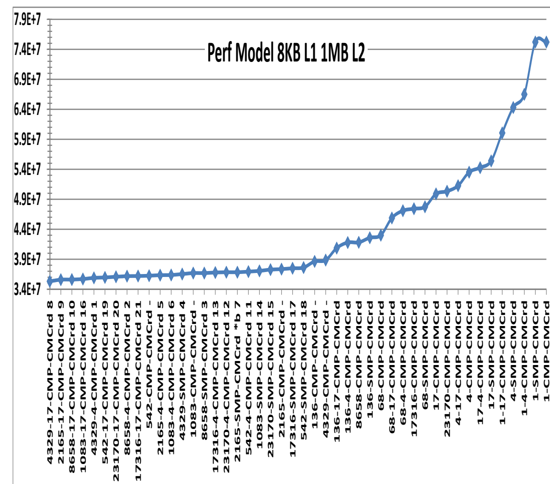
(a) 128kB L2.



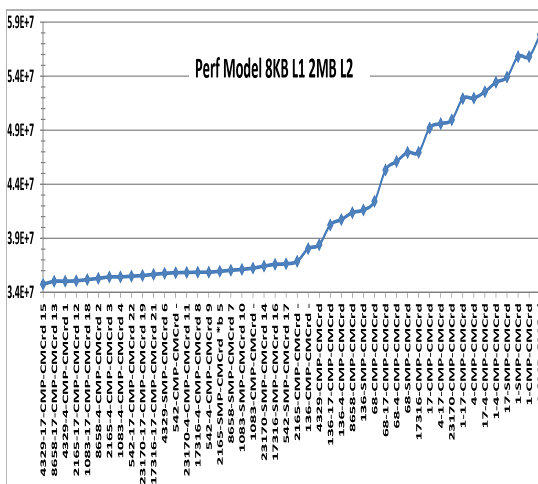
(b) 256kB L2.



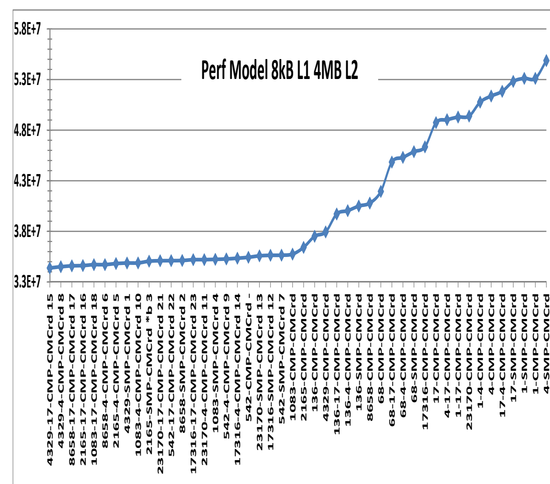
(c) 512kB L2.



(d) 1MB L2.

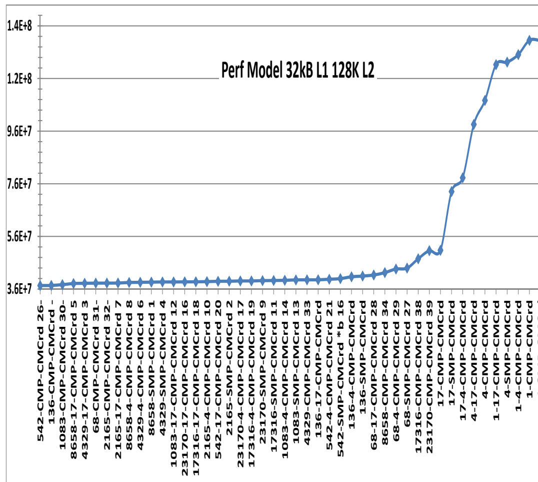


(e) 2MB L2.

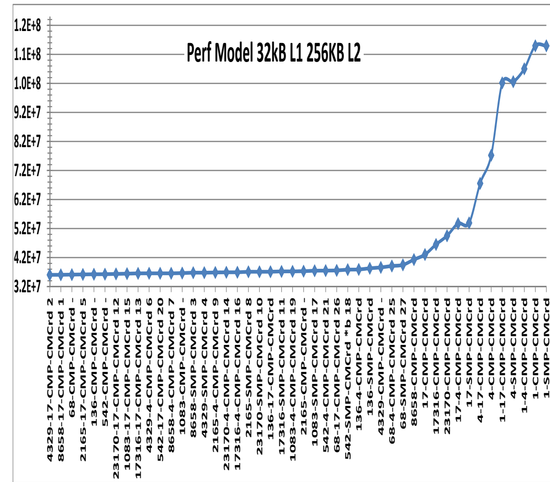


(f) 4MB L2.

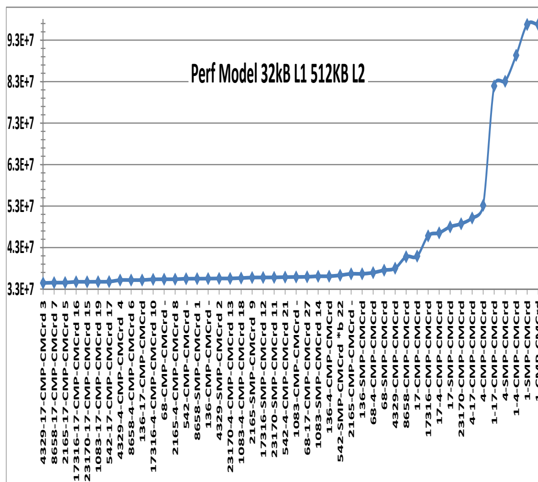
Figure 6.14: Ordered PIN Performance Model results for NBF under the **m14b** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches.



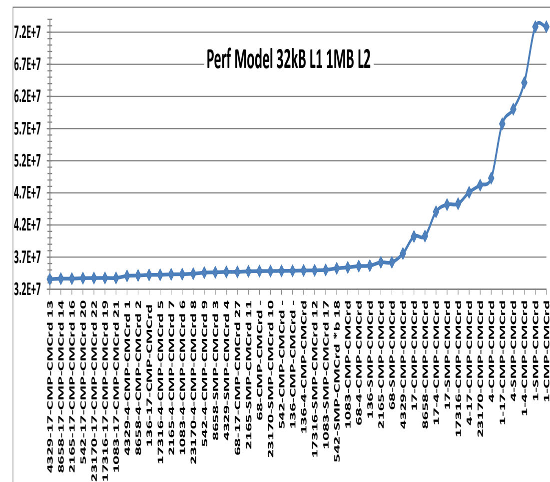
(a) 128kB L2.



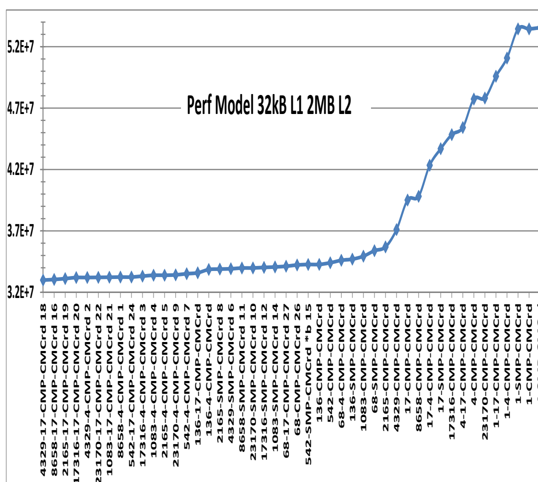
(b) 256kB L2.



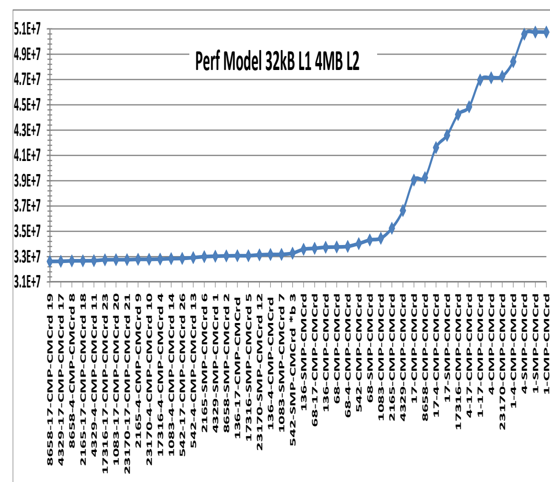
(c) 512kB L2.



(d) 1MB L2.

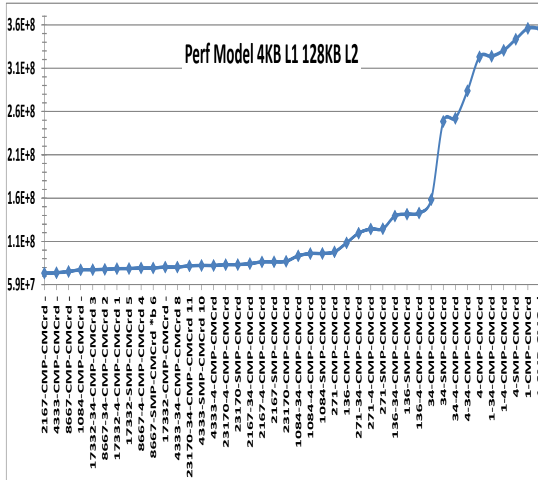


(e) 2MB L2.

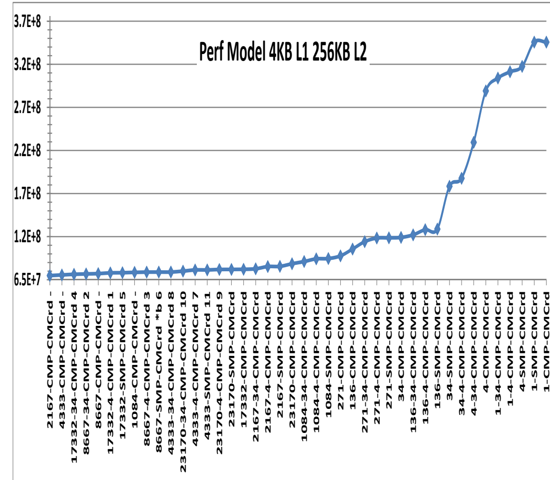


(f) 4MB L2.

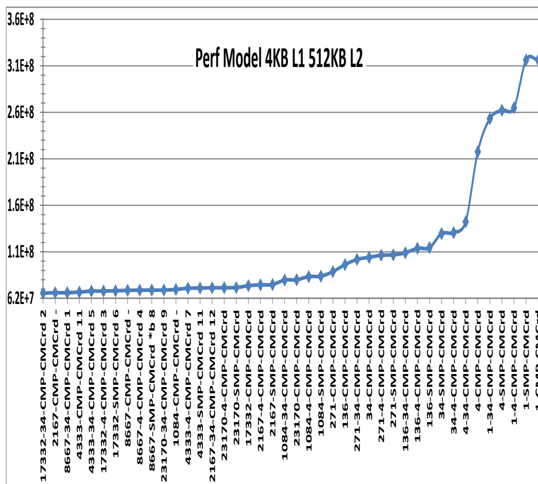
Figure 6.15: Ordered PIN Performance Model results for NBF under the **m14b** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches.



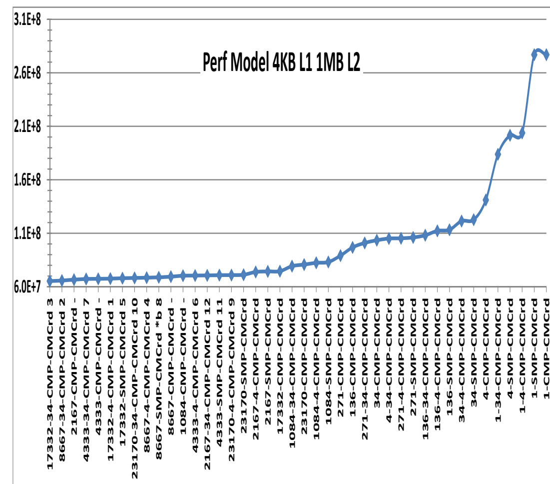
(a) 128kB L2.



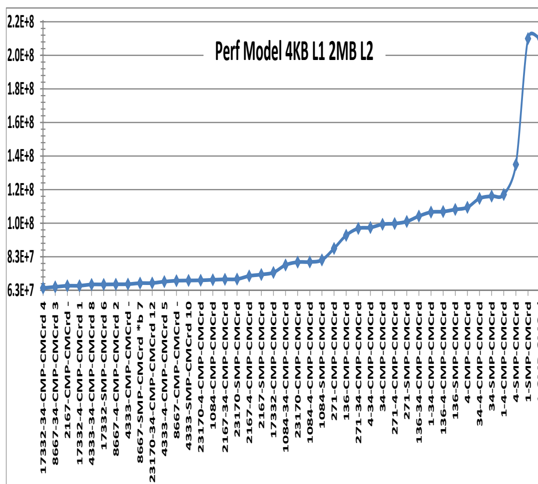
(b) 256kB L2.



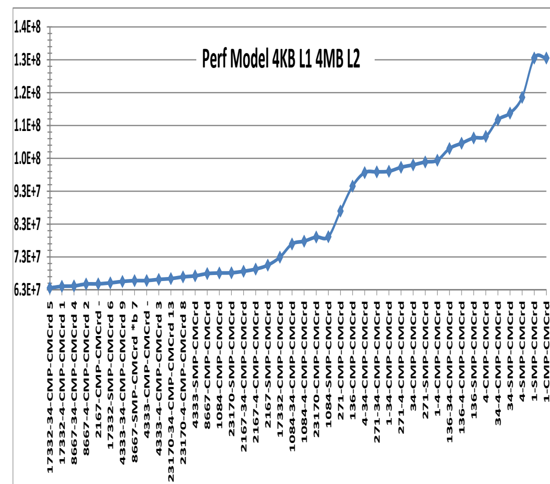
(c) 512kB L2.



(d) 1MB L2.

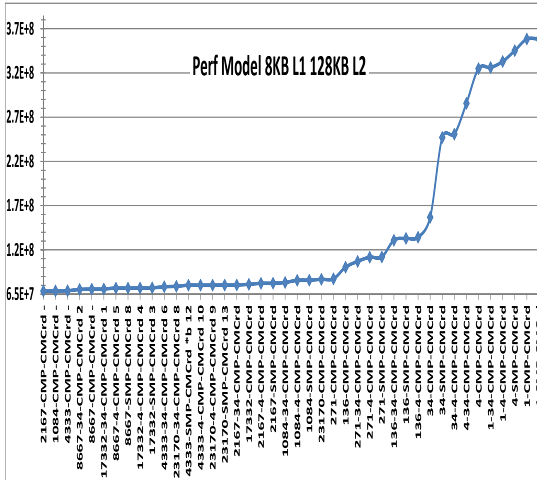


(e) 2MB L2.

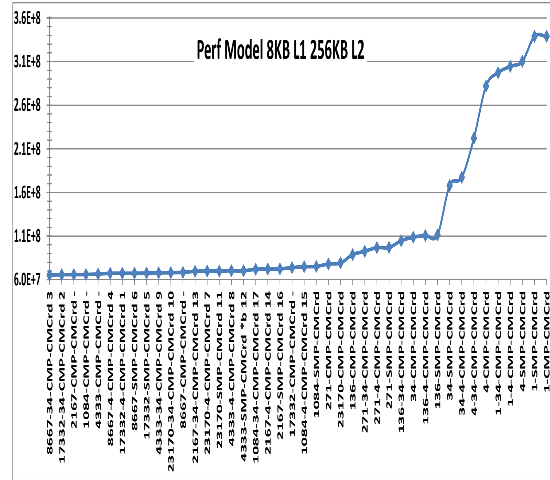


(f) 4MB L2.

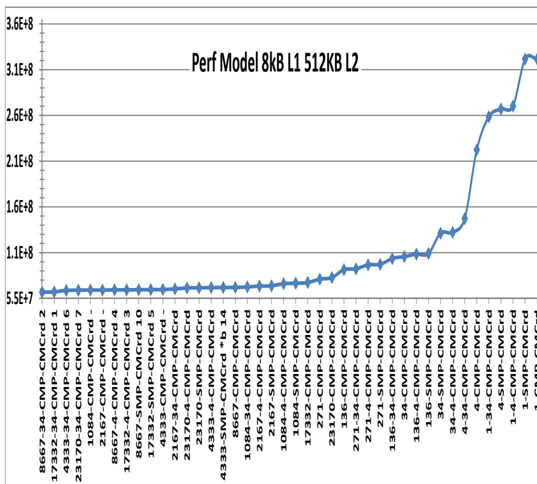
Figure 6.16: Ordered PIN Performance Model results for NBF under the **auto** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches.



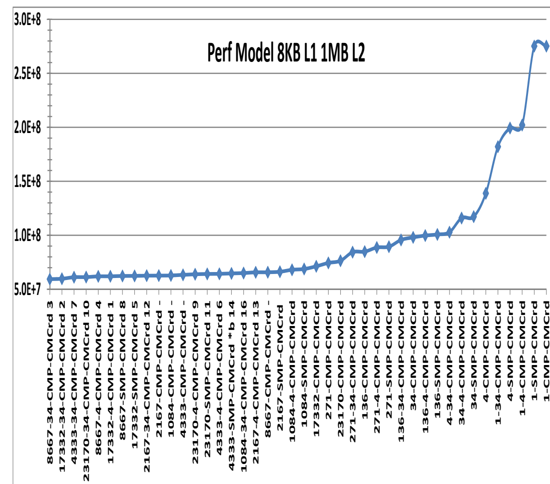
(a) 128kB L2.



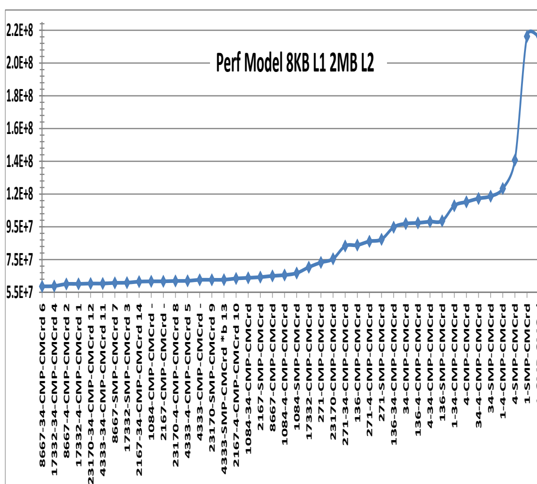
(b) 256kB L2.



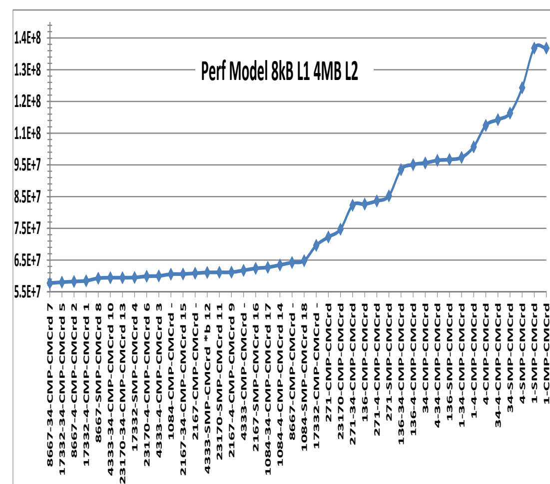
(c) 512kB L2.



(d) 1MB L2.

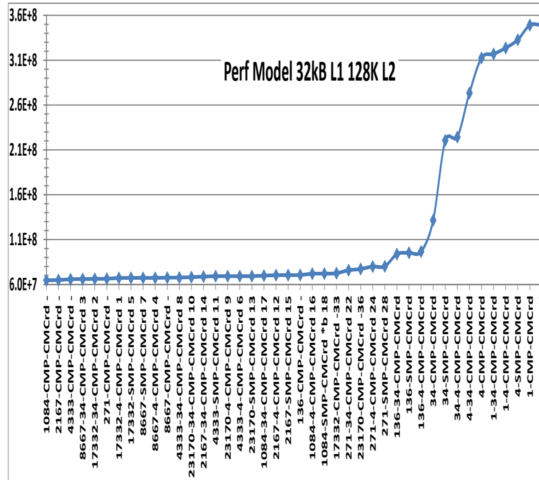


(e) 2MB L2.

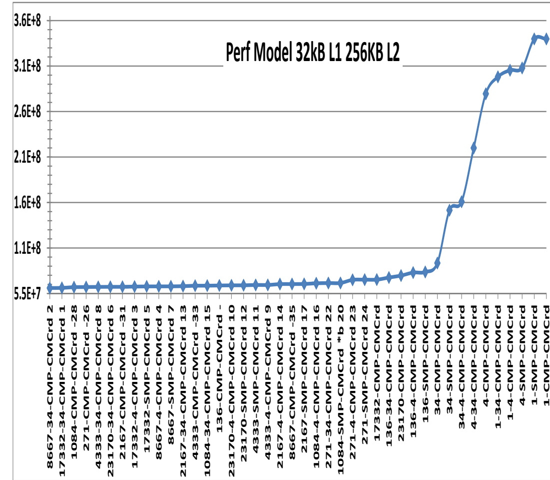


(f) 4MB L2.

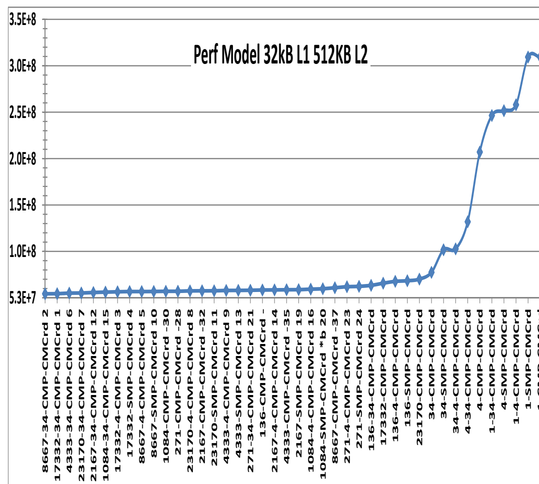
Figure 6.17: Ordered PIN Performance Model results for NBF under the **auto** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches.



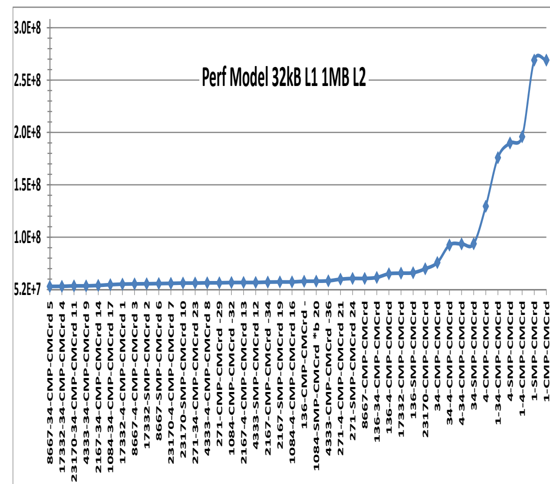
(a) 128kB L2.



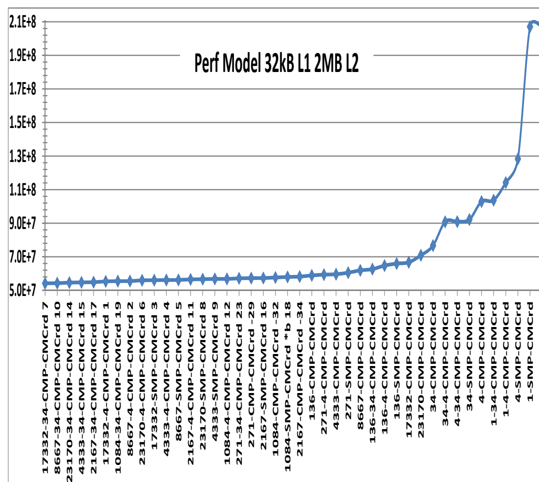
(b) 256kB L2.



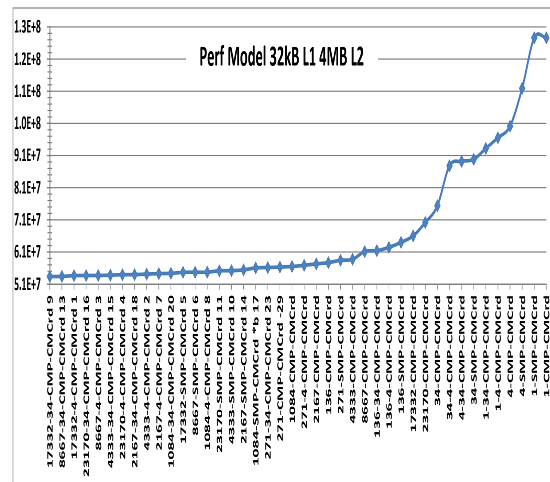
(c) 512kB L2.



(d) 1MB L2.

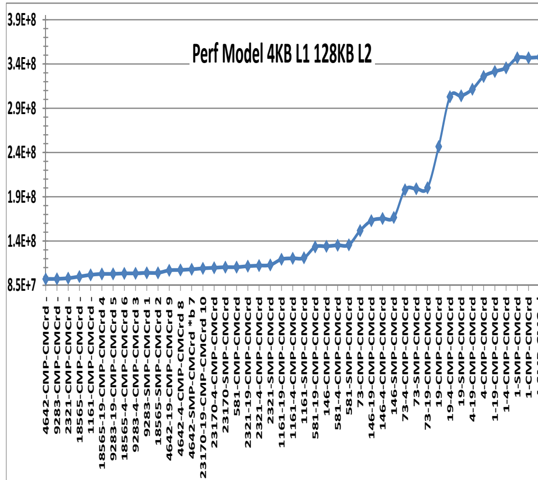


(e) 2MB L2.

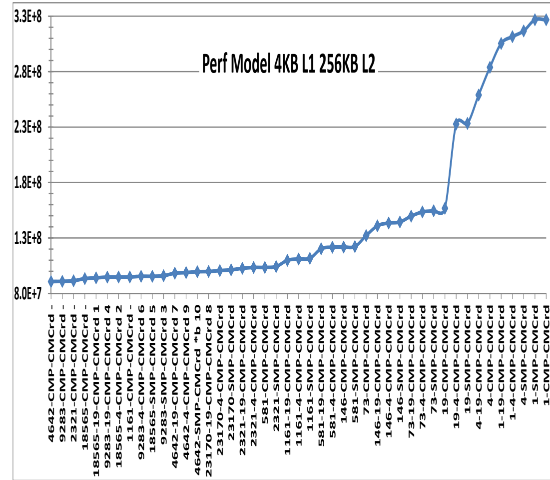


(f) 4MB L2.

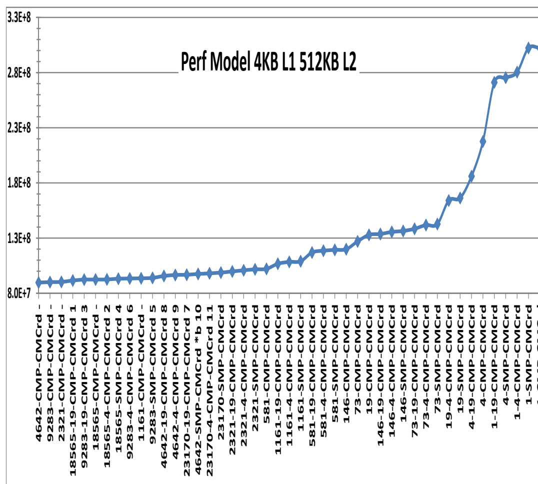
Figure 6.18: Ordered PIN Performance Model results for NBF under the **auto** problem with $32kB$ L1 cache and $128kB - 4MB$ L2 caches.



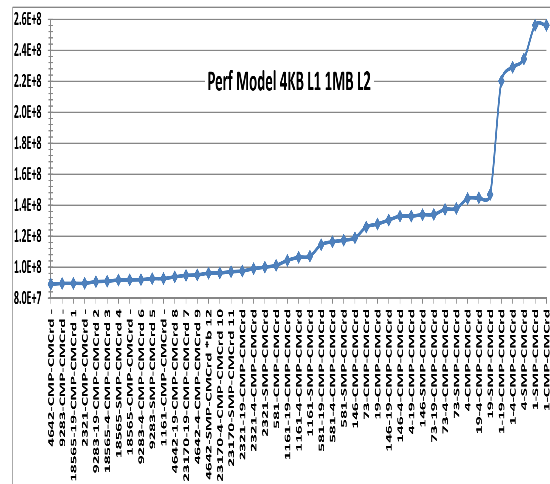
(a) 128kB L2.



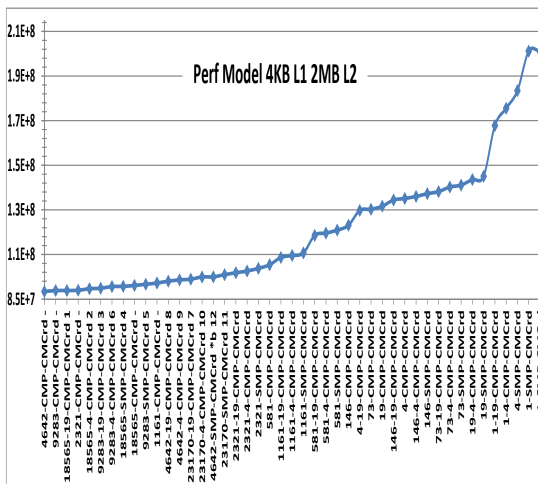
(b) 256kB L2.



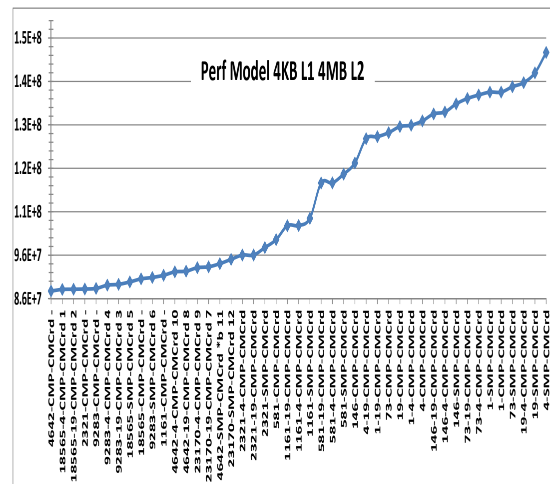
(c) 512kB L2.



(d) 1MB L2.

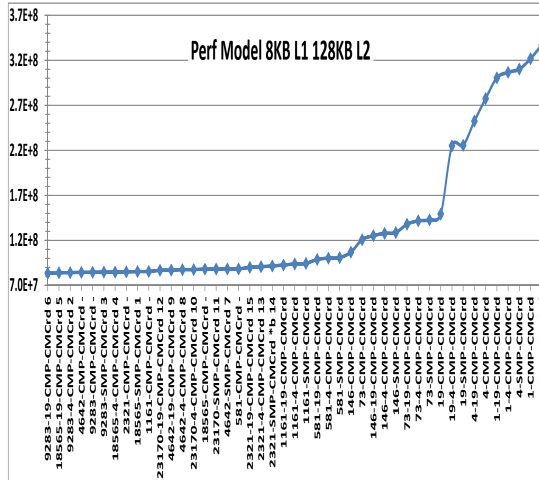


(e) 2MB L2.

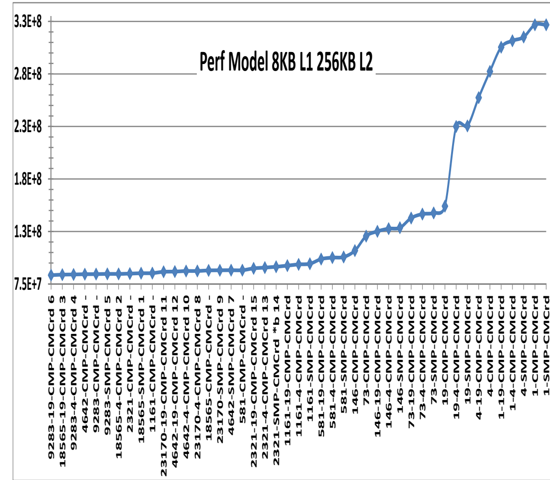


(f) 4MB L2.

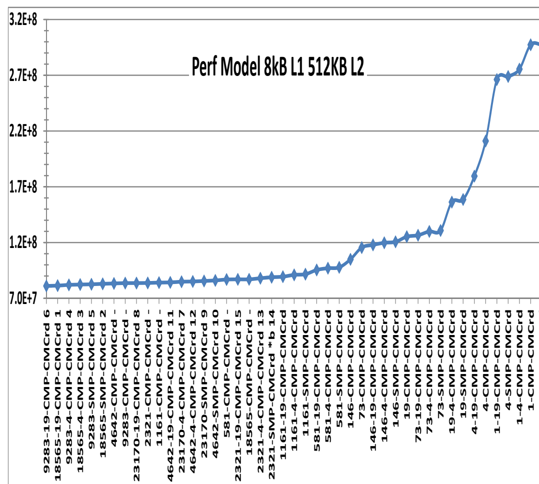
Figure 6.19: Ordered PIN Performance Model results for MOLDYN under the **144** problem with $4kB$ L1 caches and $128kB - 4MB$ L2 caches.



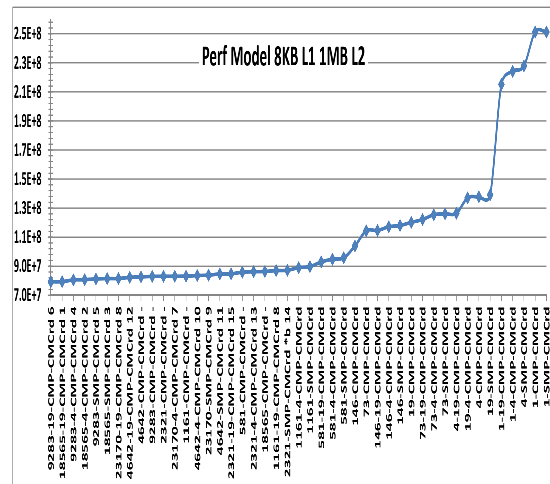
(a) 128kB L2.



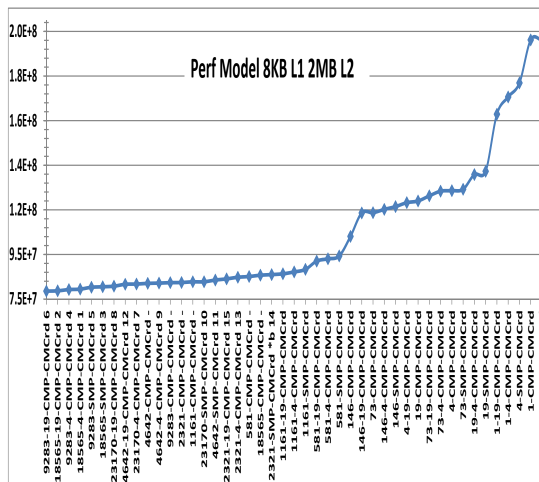
(b) 256kB L2.



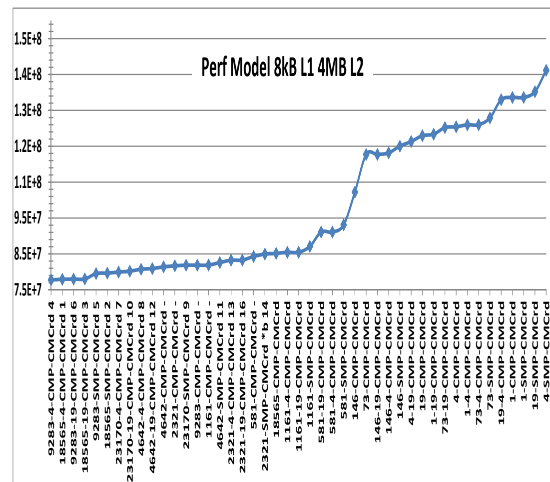
(c) 512kB L2.



(d) 1MB L2.

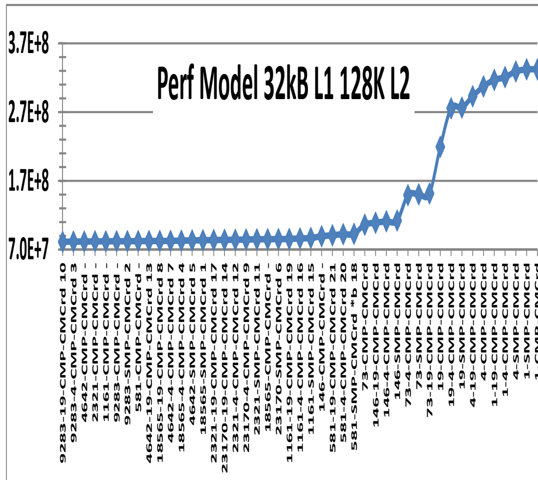


(e) 2MB L2.

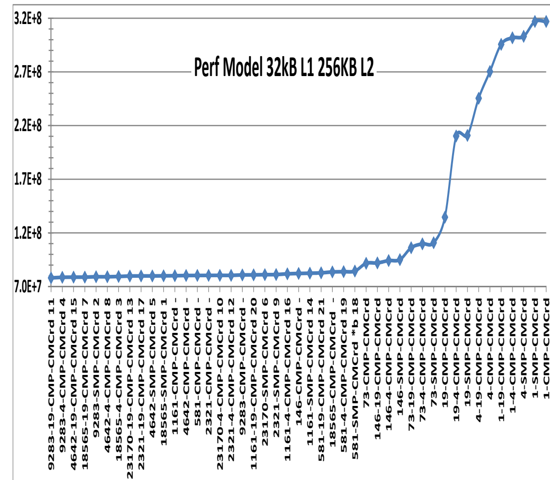


(f) 4MB L2.

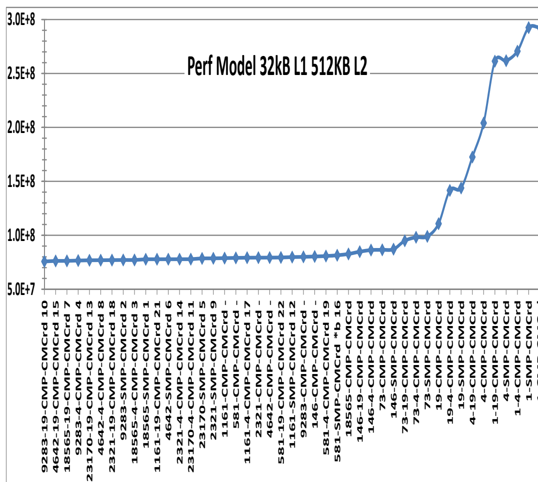
Figure 6.20: Ordered PIN Performance Model results for Moldyn under the **144** problem with $8kB$ L1 caches and $128kB - 4MB$ L2 caches.



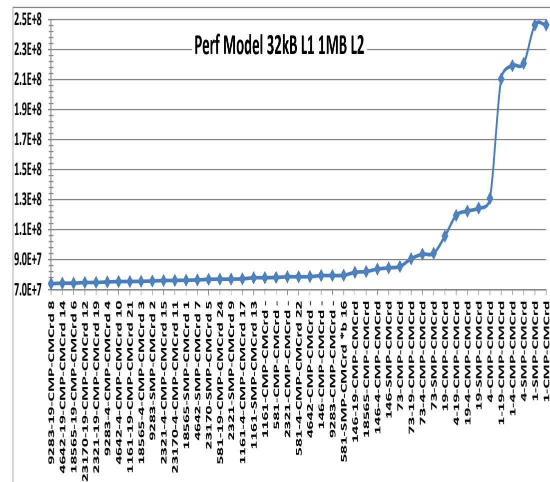
(a) 128kB L2.



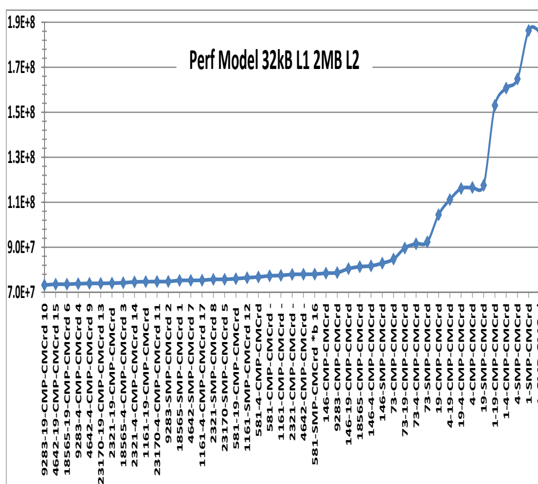
(b) 256kB L2.



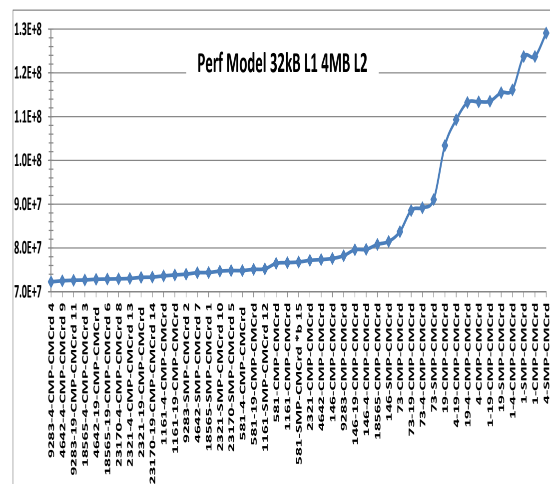
(c) 512kB L2.



(d) 1MB L2.

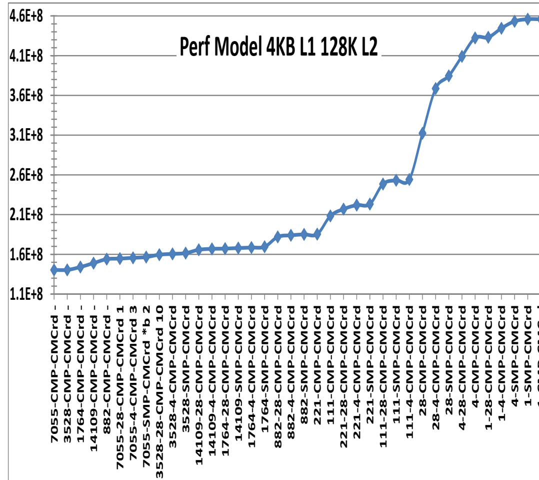


(e) 2MB L2.

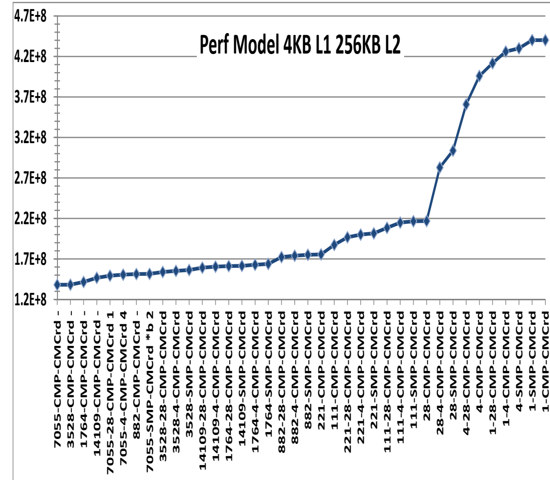


(f) 4MB L2.

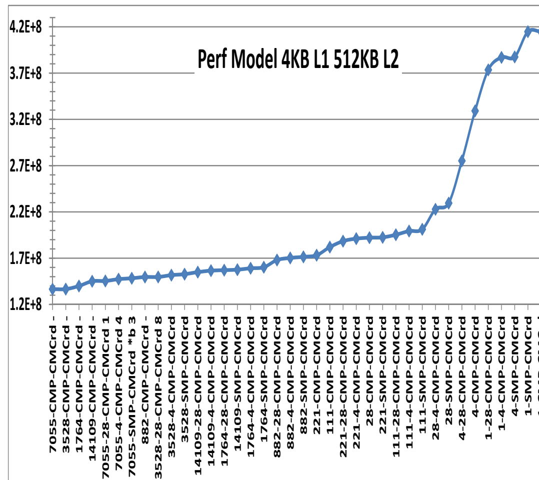
Figure 6.21: Ordered PIN Performance Model results for MOLDYN under the **144** problem with $32kB$ L1 caches and $128kB - 4MB$ L2 caches.



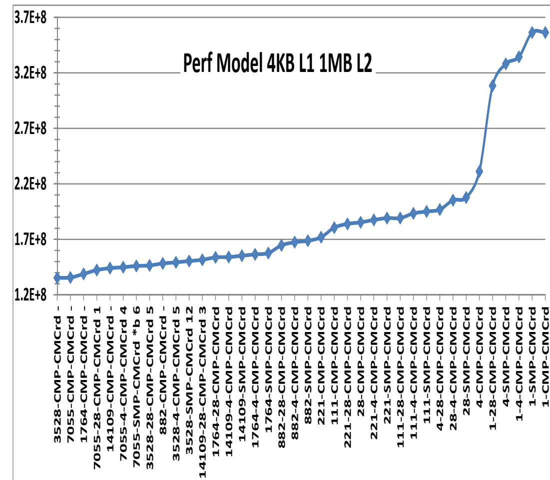
(a) 128kB L2.



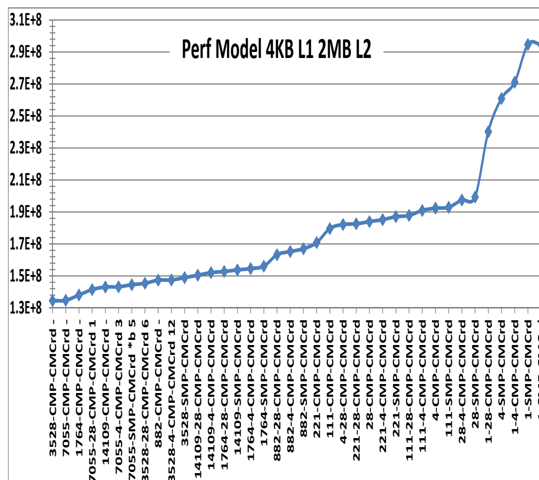
(b) 256kB L2.



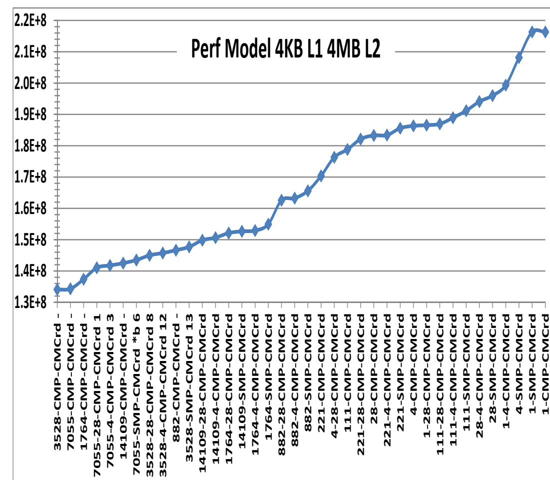
(c) 512kB L2.



(d) 1MB L2.

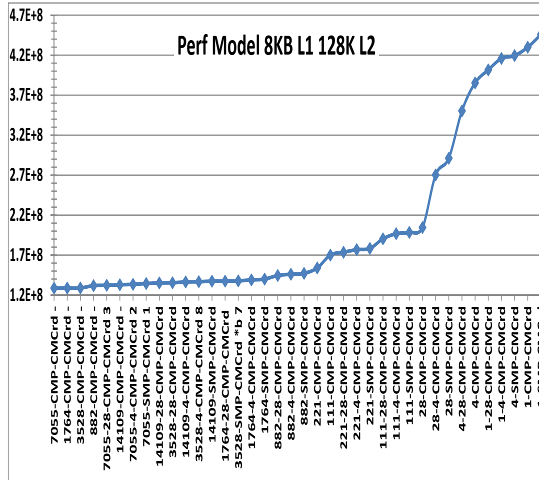


(e) 2MB L2.

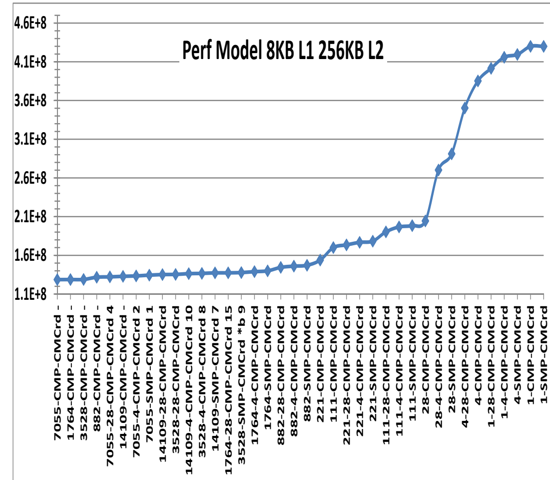


(f) 4MB L2.

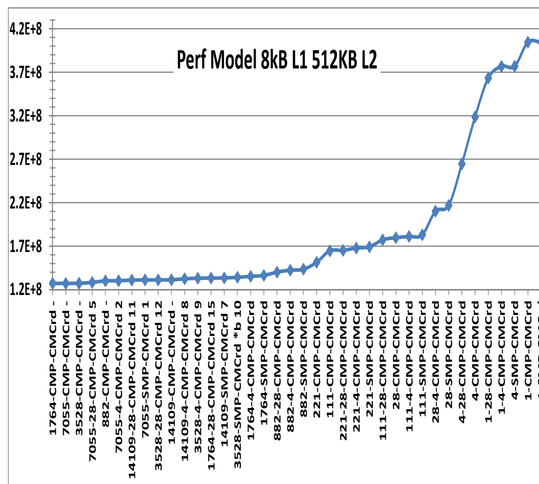
Figure 6.22: Ordered PIN Performance Model results for MOLDYN under the **m14b** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches.



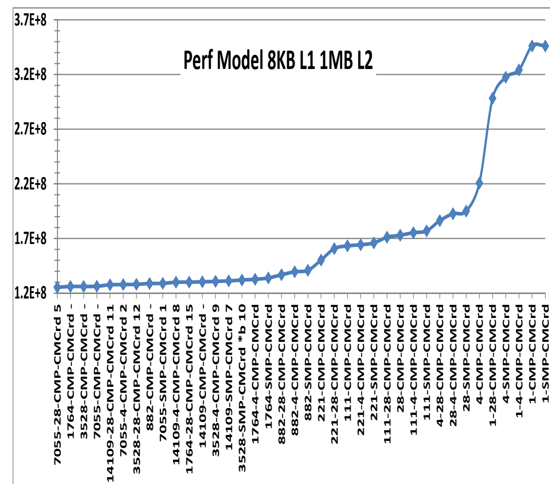
(a) 128kB L2.



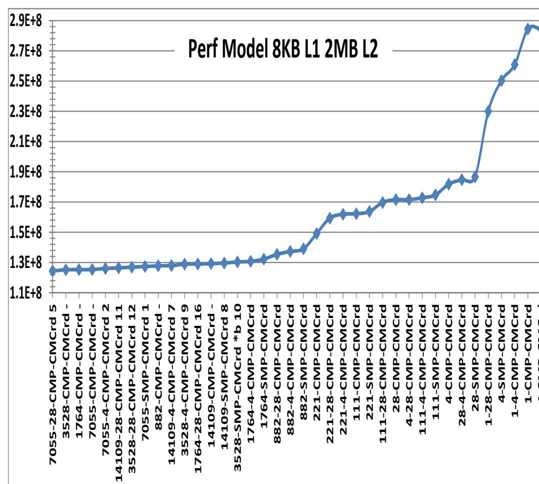
(b) 256kB L2.



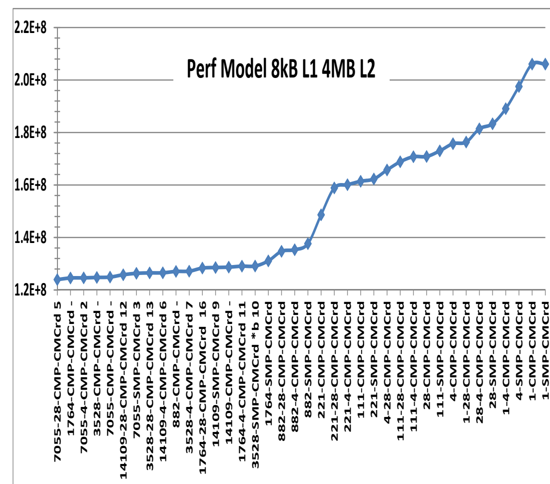
(c) 512kB L2.



(d) 1MB L2.

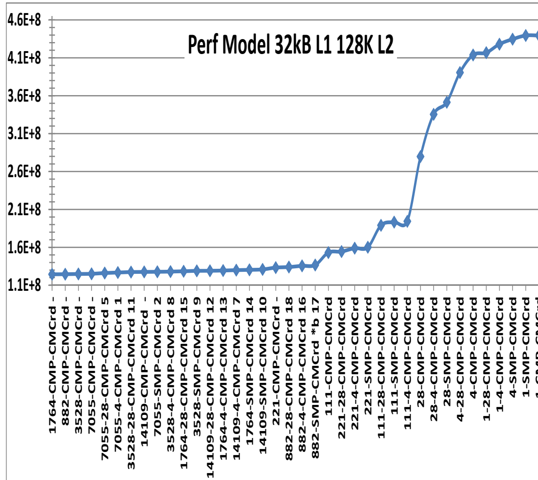


(e) 2MB L2.

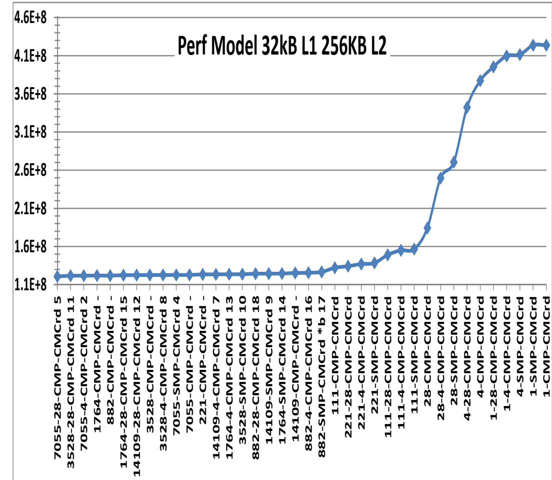


(f) 4MB L2.

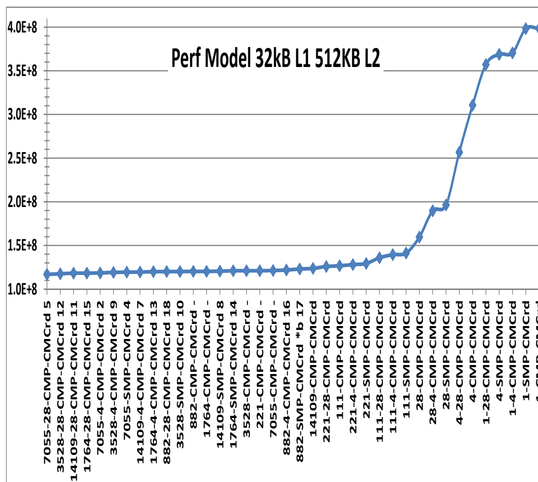
Figure 6.23: Ordered PIN Performance Model results for MOLDYN under the **m14b** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches.



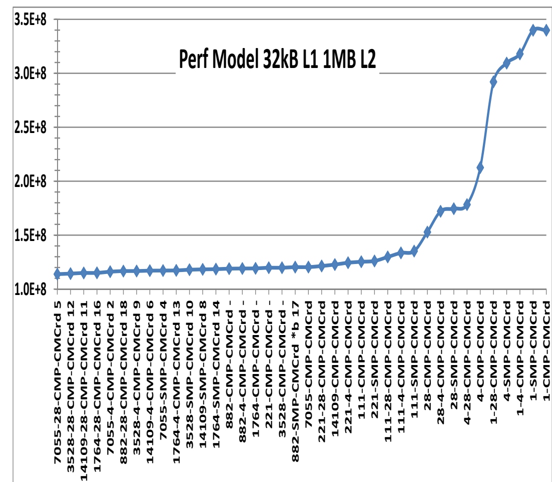
(a) 128kB L2.



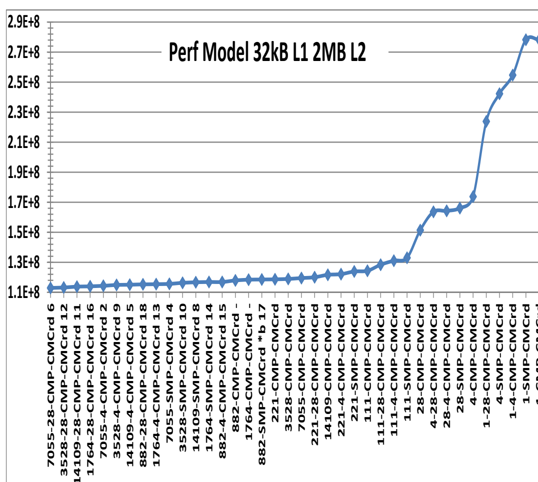
(b) 256kB L2.



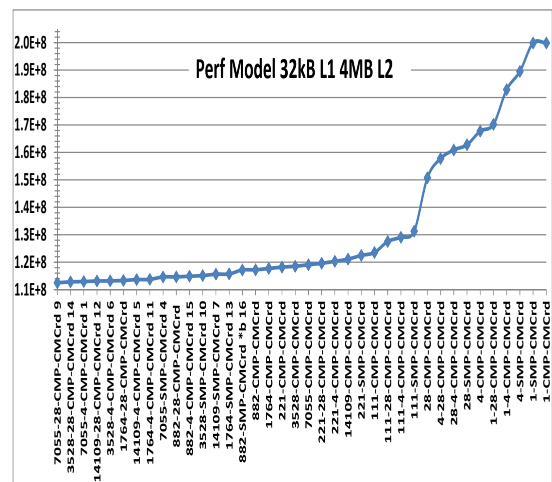
(c) 512kB L2.



(d) 1MB L2.

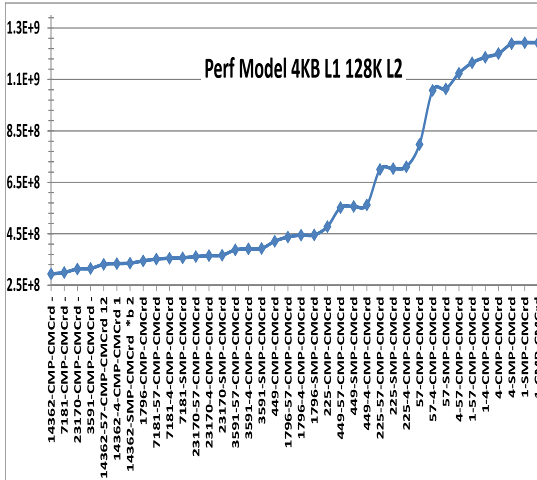


(e) 2MB L2.

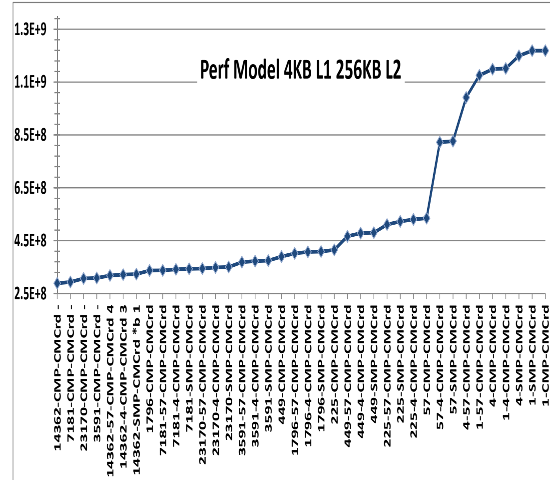


(f) 4MB L2.

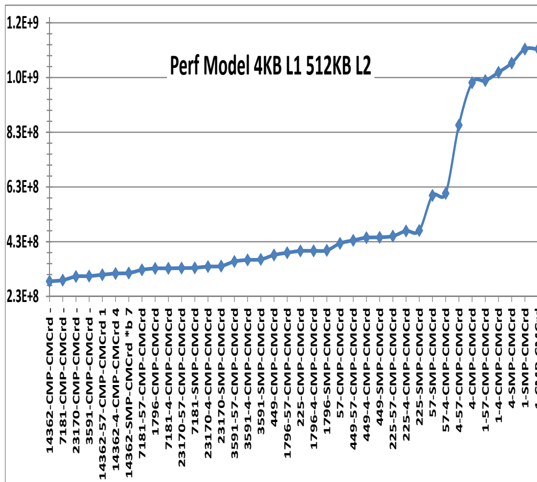
Figure 6.24: Ordered PIN Performance Model results for Moldyn under the **m14b** problem with 32kB L1 caches and $128kB - 4MB$ L2 caches.



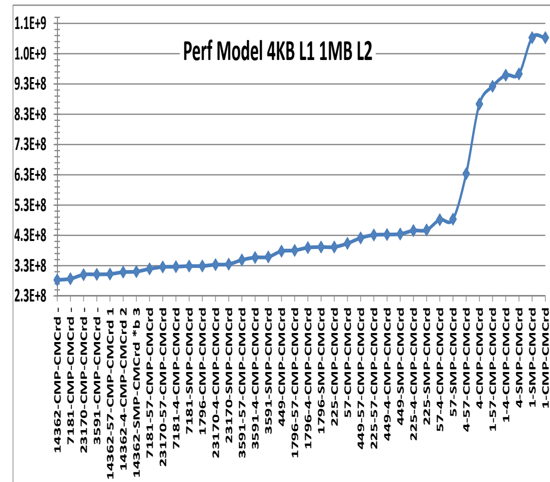
(a) 128kB L2.



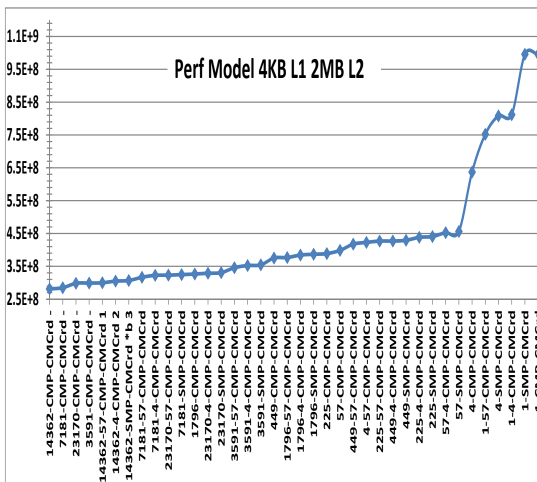
(b) 256kB L2.



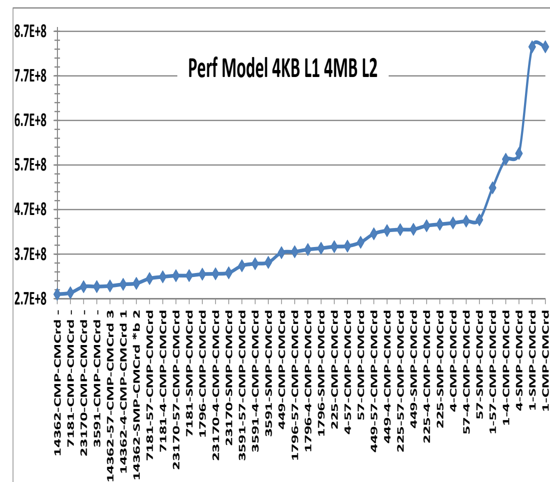
(c) 512kB L2.



(d) 1MB L2.

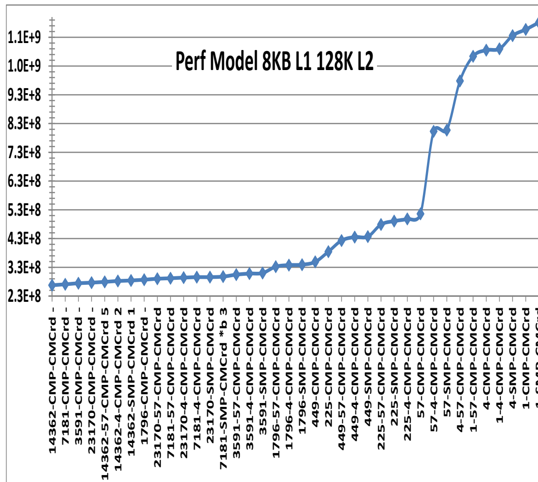


(e) 2MB L2.

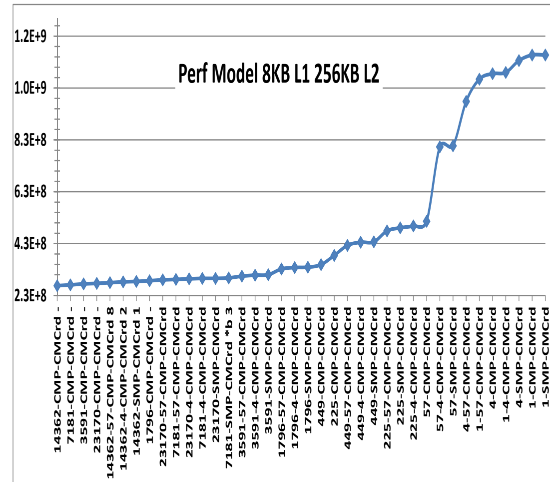


(f) 4MB L2.

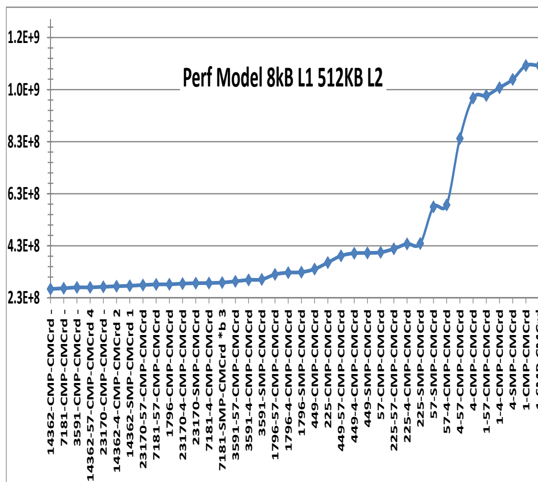
Figure 6.25: Ordered PIN Performance Model results for MOLDYN under the **auto** problem with $4kB$ L1 cache and $128kB - 4MB$ L2 caches.



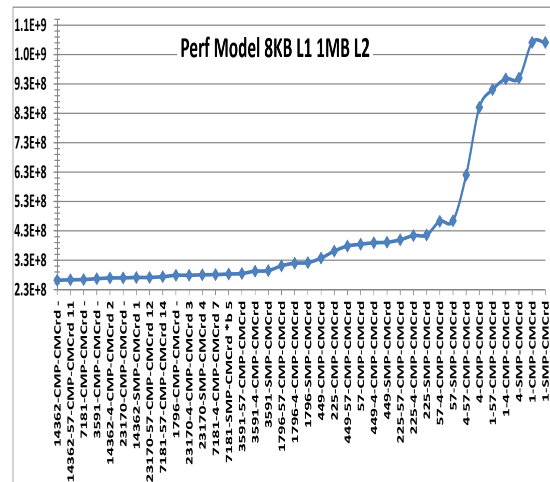
(a) 128kB L2.



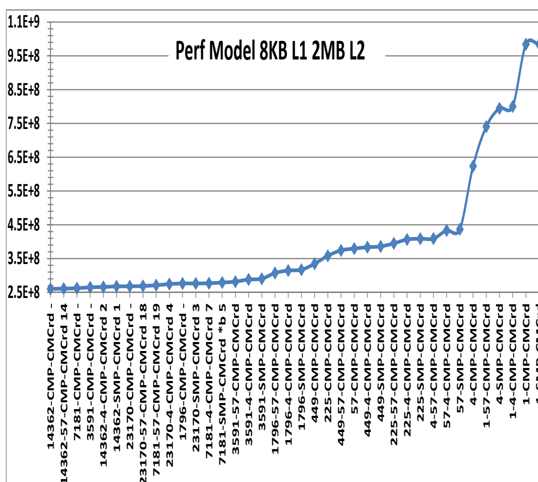
(b) 256kB L2.



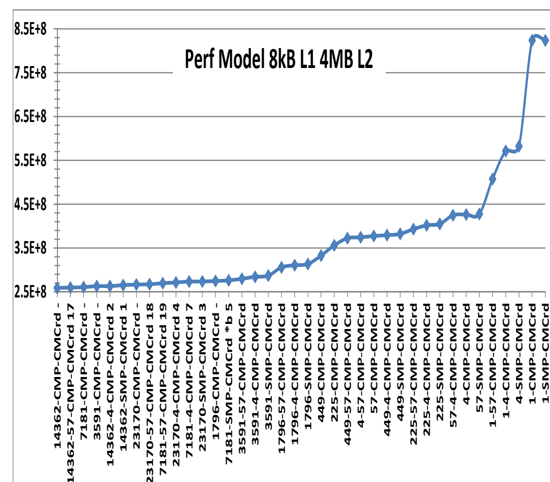
(c) 512kB L2.



(d) 1MB L2.

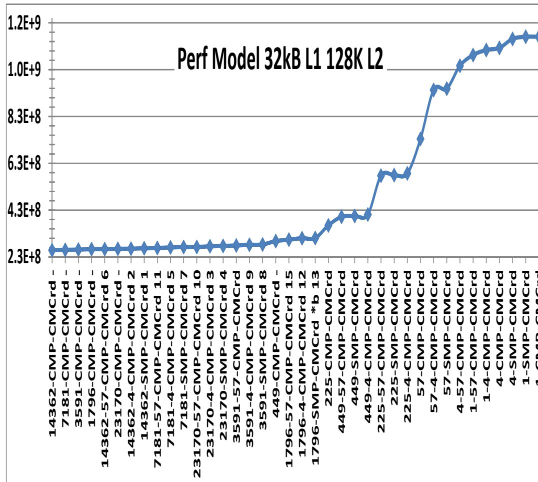


(e) 2MB L2.

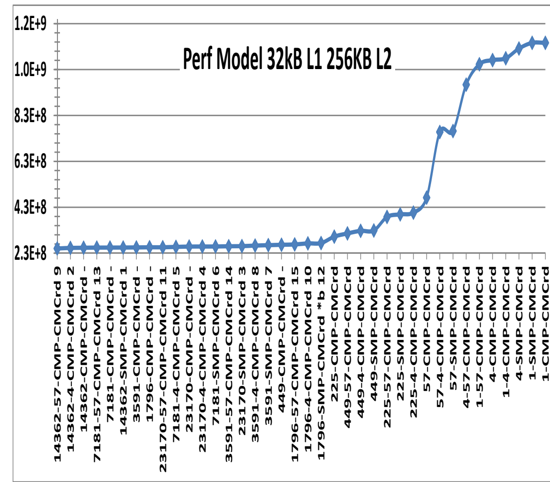


(f) 4MB L2.

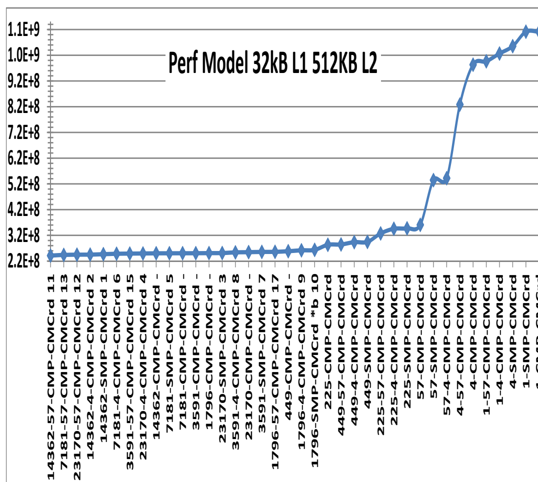
Figure 6.26: Ordered PIN Performance Model results for MOLDYN under the **auto** problem with $8kB$ L1 cache and $128kB - 4MB$ L2 caches.



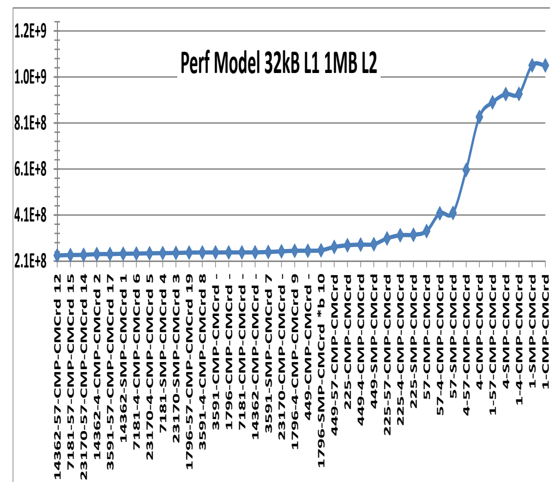
(a) 128kB L2.



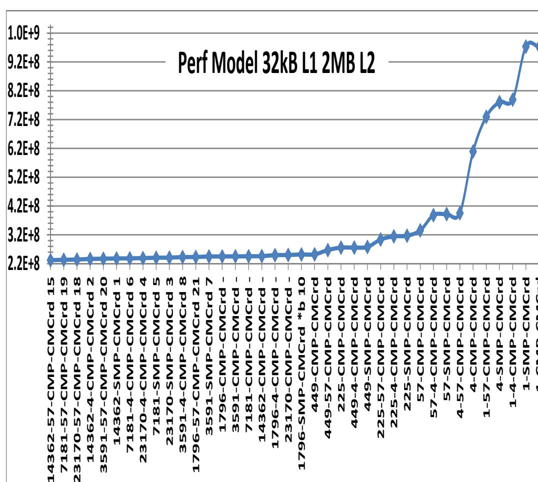
(b) 256kB L2.



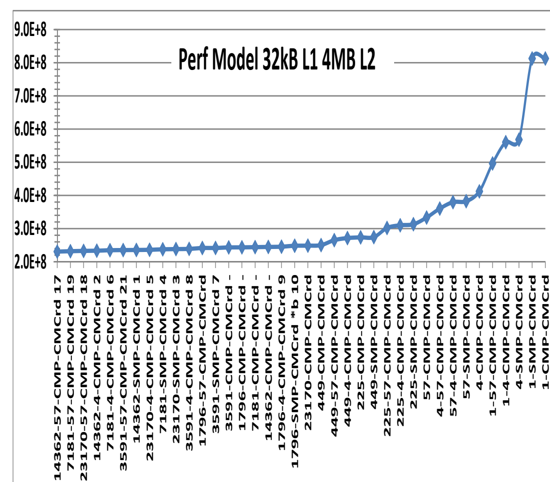
(c) 512kB L2.



(d) 1MB L2.



(e) 2MB L2.



(f) 4MB L2.

Figure 6.27: Ordered PIN Performance Model results for MOLDYN under the **auto** problem with $32kB$ L1 cache and $128kB - 4MB$ L2 caches.

Table 6.15: Summary of Partition Size prediction using PIN against the best selected by M5 for the NBF benchmark with each of the three input data set under all L1 and L2 cache size selections.

L2/L1	4kB	8kB	32kB
144			
128kB	(11224,5612) (4,2)	(11224,5612) (3,2)	(11224,5612) (5,2)
256kB	(11224,11224) (7,1)	(11224,11224) (8,1)	(11224,5612) (5,3)
512kB	(11224,2806) (3,3)	(11224,5612) (2,2)	(11224,5612) (3,3)
1MB	(11224,11224) (7,1)	(11224,11224) (7,1)	(11224,11224) (10,1)
2MB	(11224,11224) (9,1)	(11224,11224) (12,1)	(11224,5612) (14,2)
4MB	(11224,2806) (11,3)	(11224,2806) (12,2)	(11224,11224) (16,1)
m14b			
128kB	(4329,4329) (4,1)	(4329,4329) (2,1)	(8658,8658) (5,1)
256kB	(4329,4329) (4,1)	(4329,4329) (6,1)	(4329,8658) (2,2)
512kB	(4329,4329) (2,1)	(4329,4329) (2,1)	(4329,8658) (3,2)
1MB	(4329,4329) (6,1)	(4329,4329) (8,1)	(4329,4329) (13,1)
2MB	(4329,4329) (8,1)	(4329,4329) (15,1)	(4329,8658) (18,2)
4MB	(4329,4329) (9,1)	(4329,4329) (15,1)	(8658,4329) (19,2)
auto			
128kB	(17332,17332) (3,1)	(8667,17332) (2,2)	(8667,17332) (3,3)
256kB	(17332,17332) (4,1)	(8667,17332) (3,3)	(8667,17332) (2,2)
512kB	(17332,8667) (2,2)	(8667,17332) (2,2)	(8667,17332) (2,2)
1MB	(17332,17332) (3,1)	(8667,17332) (3,3)	(8667,17332) (5,3)
2MB	(17332,17332) (4,1)	(8667,17332) (6,2)	(17332,17332) (7,1)
4MB	(17332,17332) (5,1)	(8667,17332) (7,2)	(17332,17332) (9,1)

Table 6.16: Summary of Partition Size prediction using PIN against the best selected by M5 for the MOLDYN benchmark with each of the three input data set under all L1 and L2 cache size selections.

L2/L1	4kB	8kB	32kB
144			
128kB	(18565,9283) (4,2)	(9283,18565) (6,2)	(9283,18565) (10,2)
256kB	(18565,18565) (1,1)	(9283,18565) (6,4)	(9283,18565) (11,2)
512kB	(18565,18565) (1,1)	(9283,18565) (6,4)	(9283,18565) (10,2)
1MB	(18565,18565) (1,1)	(9283,18565) (6,4)	(9283,18565) (8,2)
2MB	(18565,18565) (1,1)	(9283,18565) (6,4)	(9283,18565) (10,2)
4MB	(18565,18565) (1,1)	(9283,18565) (4,4)	(9283,18565) (4,2)
m14b			
128kB	(7055,7055) (1,1)	(7055,7055) (3,1)	(7055,7055) (5,1)
256kB	(7055,7055) (1,1)	(7055,7055) (4,1)	(7055,23170) (5,2)
512kB	(7055,7055) (1,1)	(7055,7055) (5,1)	(7055,23170) (5,2)
1MB	(7055,7055) (1,1)	(7055,7055) (5,1)	(7055,23170) (5,2)
2MB	(7055,7055) (1,1)	(7055,7055) (5,1)	(7055,23170) (6,2)
4MB	(7055,7055) (1,1)	(7055,23170) (5,2)	(7055,7055) (9,1)
auto			
128kB	(14362,14362) (12,1)	(14362,14362) (5,1)	(14362,14362) (6,1)
256kB	(14362,14362) (4,1)	(14362,14362) (8,1)	(14362,14362) (9,1)
512kB	(14362,14362) (1,1)	(14362,14362) (4,1)	(14362,14362) (11,1)
1MB	(14362,14362) (1,1)	(14362,14362) (11,1)	(14362,14362) (12,1)
2MB	(14362,14362) (1,1)	(14362,14362) (14,1)	(14362,14362) (15,1)
4MB	(14362,14362) (1,1)	(14362,14362) (17,1)	(14362,14362) (17,1)

Table 6.17: A summary of the Ranks obtained by PIN's top prediction for each benchmarks.

	Irreg		NBF		Moldyn		Total	
Rank	Count	% of Total	Count	% of Total	Count	% of Total	Count	% of Total
1	1	1.9			15	27.8	16	9.9
2	0	0	10	18.5	0	0	10	6.2
3	14	25.9	9	16.7	1	1.9	24	14.8
4	2	3.7	5	9.3	6	11.1	13	8.0
5	3	1.9	5	3.1	9	5.6	17	10.5
6	3	1.9	3	1.9	7	4.3	13	8.0
7	6	3.7	5	3.1	0	0	11	6.8
8	3	1.9	3	1.9	2	1.2	8	4.9
9	5	3.1	3	1.9	2	1.2	10	6.2
10	5	3.1	1	0.6	3	1.9	9	5.6
11	1	0.6	1	0.6	3	1.9	5	3.1
12	3	1.9	2	1.2	2	1.2	7	4.3
13	3	1.9	1	0.6	0	0	4	2.5
14	0	0	1	0.6	1	0.6	2	1.2
15	1	0.6	2	1.2	1	0.6	4	2.5
16	3	1.9	1	0.6	0	0	4	2.5
17	0	0	0	0	2	1.2	2	1.2
18	0	0	1	0.6	0	0	1	0.6
19	1	0.6	1	0.6	0	0	2	1.2
Total	54	100	54	100	54	100	162	100

Table 6.18: A summary of the ranks achieved using our "Run Three" optimization to enhance the accuracy of the prediction of the PIN model. The graph partitioning size remains unaltered but the ranking improves significantly as shown below.

	IRREG		NBF		MOLDYN		Total	
Rank	Count	% of Total	Count	% of Total	Count	% of Total	Count	% of Total
1	29	53.7	29	53.7	36	66.7	94	58.0
2	8	14.8	17	31.5	13	24.1	38	23.5
3	6	11.1	8	14.8	0	0	14	8.6
4	5	9.3	0	0	5	9.3	10	6.2
5	3	5.6	0	0	0	0	3	1.9
7	2	3.7	0	0	0	0	2	1.2
11	1	1.9	0	0	0	0	1	0.6
Total	54	100	54	100	54	100	162	100

Chapter 7

Conclusion and Future Work

In this Chapter, we will summarize our work, then we outline the future directions beyond this thesis.

7.1 Summary

There are two thrusts for this thesis. The first is a design space exploration for CMPs with several variables. The main two variables are parallelization strategies and locality optimizations. The benchmarks used in this thesis are graph problems.

We have tried several parallelization strategies. The first baseline parallelization is the legacy parallelization which divides the problem into P parts, where P is the number of cores. We investigate three other parallelization techniques one of them was proposed in the proposal stage of this thesis and we added two other further optimizations that aim at reducing the working set and increase the sharing among the cores.

To optimize locality for graph problems the accept method is to apply graph partitioning. We used metis [97] to derive graph partitioning. We estimated the partition sizes by computing the total size of the data for the input data set and then we compute an estimate for the partition size by dividing the total data size by the cache size that we want the partitions to fit into. We compute several of

such partitions to the maximum partition size that metis allows us to have. We also show the optimization results for several different L1 caches and L2 caches. We run a set of three benchmarks each with three input data sets. We report gains relative to a baseline setting. The baseline represents the partition size that we estimate to fit in the L1 cache. We note that the baseline performance is a lot better than the un-optimized parallel version without any graph partitioning applied. This should explain in part why in some cases the gains relative to the baseline can be only a few percent points. Yet, we see gains in excess of 19%. In a few cases the gain is literally 0%, which means that the baseline case is the best case.

The second thrust of this thesis is building and verifying an execution time performance model based on reuse distance profiles computed using a PIN tool. To our model, this thesis is the first to use multicore concurrent reuse distance profiles to derive parallel program locality optimizations. We use the CMCs of the CRD and PRD profiles to compute estimate to the different miss counts for the different levels of the memory hierarchy. We develop a model to predict the execution time yet the goal of the model is not the prediction of the execution time but to be able to rank order different parallelizations and locality optimizations relative to each other to be able to predict what the PIN model estimates to be the best performing combination of a parallelization and locality optimization. We have shown that the tool can predict the partition sizes far better than the best parallelization. If are allowed to run on a simulator or a real machine three or four runs we can increase the accuracy of the model to very large values compared to the around 50% of the maximum M5 gain that we can achieve.

7.2 Future Work

Controlled Sharing, through Controlled Hierarchical Partition Creation

As we have outlined and shown in this thesis and others have shown that sharing has become a good aspect of CMPs. We think that we can increase sharing by designing partitioners that take sharing at heart. One idea we have is to try to assemble together the small partitions that metis finds into larger partitions in a hierarchical fashion in such a way that the edges coming out of that larger assembled partition is minimal to other partitions but the intra-partition edges are not of concern at all since they represent shared data item among the cores that will process that partition. The assembled partition can be designed to be of different sizes either P parts, where P is the core count or an arbitrary size which could be chosen so that the partition would fit in the shared LLC. One advantage of this method if successful is that it would enhance sharing or control it instead of the method we are using which does not control the sharing at all.

Deriving other Multicore Compiler Optimizations

We have applied Multicore Reuse Distance analysis for choosing locality optimizations that require knowing the input data set where we run a profile once on any machine and can predict the performance and the best optimization on any other machine. This methodology can be used in any compiler infrastructure to store information with the binary or the distributable output as to what to do with different caches and memory hierarchies.

Program Analysis & Programmer Assistance Tool

The reuse distance model can give us a lot of insight to the spectrum of gains that we would expect from any possible optimizations and what can the programmer expect or should focus on. The multicore reuse distance profiles really provide a framework that can be used in as an analysis infrastructure to give us an overall architecture independent look at how a benchmark behaves under different caches sizes and under different input data sets. We think that this framework can give the developer important insights. For example, under a given hardware whether or not optimizations should be used or not and should we invest efforts into optimization the benchmark more or the benchmark in its current state is in good shape. If the process of computing the multicore reuse distance profiles can be accelerated using time distance analysis [18, 19, 60, 158, 159], the overhead and slowness of obtaining the profiles would make it a more daily used infrastructure for program analysis tool. What we envision is a Vtune [188] like tool that give results for any cache after running the tool only once on top of a particular system. So, the model would be analyze once and use forever.

Application to New and Contemporary Problems and Quantifying Power Efficiency

Thus far, we have used the multicore reuse distance analysis model for some relatively older and well-known graph problems. We would want to take a step ahead and try to do a similar study on graphs like the social network graphs and see how much gain can we get at what costs. This would be an interesting challenge. We would also like to quantify the energy and power efficiency of locality optimizations. We would like to estimate the total energy cost to create and use the optimization

compared to the gains and compute what is the number of iteration where we would break-even at. We have used the serial version of the metis library. We would like to use the parallel version of metis which is called ParMetis [105] and compare, the cost-benefit tradeoff and the energy and power efficiency. We have only done all of our study on a relatively small core count ($P = 4$). We would want to try on a larger core count to study the gains possible and how accurate the multicore reuse distance model can predict the graph partitionings and the parallelizations.

Teaching Computer Architecture, Compiler Design and Program Optimization

Interest in the academia for teaching tools that can give the students more insight and more visualized sense of the different aspects of Computer Architecture, Compiler Design and Program Optimization. It can be easily envisioned that we can use the profiles in showing the students how does loop fusion and loop fission affect program behavior and execution with such profiles in a vivid eye opening manner. We can show how different tile sizes for a tiled simple array computation can make a huge difference in performance visually. We can use this framework and tool to show to the students what does a working set mean and how significant the concept of reuse distance is.

Bibliography

[1] Manuel E. Acacio, Jose Gonzalez, Jose M. Garcia, and Jose Duato. A two-level directory architecture for highly scalable cc-numa multiprocessors. *IEEE Transaction Parallel Distributed Systems*, 16(1):67–79, January 2005.

[2] Anant Agarwal, Liewei Bao, John Brown, Bruce Edwards, Matt Mattina, Chyi-Chang Miao, Carl Ramey, and David Wentzlaff. Tile processor: embedded multicore for networking and multimedia. In *Proceedings of the Symposium on High Performance Chips*, 2007.

[3] Anant Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz. An evaluation of directory schemes for cache coherence. In *Proceedings of the 15th International Symposium on Computer Architecture*, 1988.

[4] I. Al-Furaih and S. Ranka. Memory hierarchy management for iterative graph structures. In *Proceedings of the 12th International Parallel Processing Symposium*, Orlando, FL, April 1998.

[5] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the American Federation of Information Processing Societies (AFIPS) Spring Joint Computing Conference*, volume 30, pages 483–485, April 1967.

[6] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Solid-State Circuits Society Newsletter, IEEE*, 12(3):19–20, 2007. Reprinted from the AFIPS Conference Proceedings, Vol. 30 (Atlantic City, N.J., Apr. 18–20), AFIPS Press, Reston, Va., 1967, pp. 483–485, when Dr. Amdahl was at International Business Machines Corporation, Sunnyvale, California.

[7] C. Scott Ananian, Krste Asanovic, Bradley C. Kuszmaul, Charles E. Leiserson, and Sean Lie. Unbounded transactional memory. In *Proceedings of the Eleventh International Symposium on High-Performance Computer Architecture*, pages 316–327. Feb 2005.

[8] ARM. Arm announces power-optimized dual-core arm cortex-a15 hard macro, May 2013.

[9] Todd Austin, Eric Larson, and Dan Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[10] Moshe (Maury) Bach, Mark Charney, Robert Cohn, Elena Demikhovsky, Tevi Devor, Kim Hazelwood, Aamer Jaleel, Chi-Keung Luk, Gail Lyons, Harish Patil, and Ady Tal. Analyzing parallel programs with pin. *Computer*, 43(3):34–41, March 2010.

[11] Abdel-Hameed A. Badawy. *The Efficacy of Software Prefetching and Locality Optimizations on Future Memory Systems.* Ms.c. thesis, ECE Dept., University of Maryland, 2002.

[12] Abdel-Hameed A. Badawy, Aneesh Agarwala, Donald Yeung, and Chau-Wen Tseng. Evaluating the Impact of memory system performance on software prefetching and locality optimizations. In *Proceedings of the 15th Annual International Conference on Supercomputing*, Sorrento, Italy, June 2001. ACM.

[13] Abdel-Hameed A. Badawy, Aneesh Aggarwal, Donald Yeung, and Chau-Wen Tseng. The efficacy of software prefetching and locality optimizations on future memory systems. *J. Instruction-Level Parallelism*, Vol. 6, 2004.

[14] Abbas BanaiyanMofrad, Nikil Dutt, and Gustavo Girão. Modeling and analysis of fault-tolerant distributed memories for networks-on-chip. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1605–1608, San Jose, CA, USA, 2013. EDA Consortium.

[15] Abbas BanaiyanMofrad, Gustavo Girão, and Nikil Dutt. A novel noc-based design for fault-tolerance of last-level caches in cmps. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '12, pages 63–72, New York, NY, USA, 2012. ACM.

[16] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A scalable architecture based on single-chip multiprocessing, 2000.

[17] Luiz André Barroso. The price of performance. *ACM Queue*, 3(7):48–53, 2005.

[18] Erik Berg and Erik Hagersten. Efficient data-locality analysis of long-running applications. Technical Report TR 2004-021, University of Uppsala, 2004.

[19] Erik Berg and Erik Hagersten. Statcache: a probabilistic approach to efficient and accurate data locality analysis. In *Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, 2004.

[20] Sion Berkowits. Pin – a dynamic binary instrumentation tool, June 2012.

[21] Kristof Beyls and Erik H. D'Hollander. Reuse distance as a metric for cache behavior. In *In Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems*, pages 617–662, 2001.

[22] Kristof Beyls and Erik H. D'Hollander. Platform-independent cache optimization by pinpointing low-locality reuse. In *Proceedings of the International Conference on Computational Science*, 2004.

[23] Kristof Beyls and Erik H. D'Hollander. Discovery of locality-improving refactoring by reuse path analysis. In *Proceedings of the International Conference on High Performance Computing and Communication*, 2006.

[24] Kristof Beyls and Erik H. D'Hollander. Intermediately executed code is the key to find refactorings that improve temporal data locality. In *Proceedings of the 3rd Conference on Computing Frontiers*, 2006.

[25] Kristof Beyls and Erik H. D'Hollander. Refactoring for data locality. *IEEE Computer*, 2009.

[26] Kristof Beyls, Erik H. D'Hollander, and Frederik Vandeputte. Rdvis: a tool that visualizes the causes of low locality and hints program optimizations. In *Proceedings of the International Conference on Computational Science*, 2005.

[27] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, Aug 2011.

[28] Nathan Binkert, Ronald Dreslinski, Lisa Hsu, Kevin Lim, Ali Saidi, and Steven Reinhardt. The M5 simulator: modeling networked systems. *IEEE Micro*, 2006.

[29] Nathan L. Binkert, Ronald G. Dreslinski, Lisa R. Hsu, Kevin T. Lim, Ali G. Saidi, and Steven K. Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro special issue on Computer Architecture Simulation*, July/August. 2006.

[30] Nathan L. Binkert, Erik G. Hallnor, and Steven K. Reinhardt. Network-oriented full-system simulation using M5. *6th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, February 2003.

[31] Shekhar Y. Borkar, Pradeep Dubey, Kevin C. Kahn, David J. Kuck, Hans Mulder, Stephen S. Pawlowski, and Justin R. Rattner. Platform 2015: `Intel` Processor and Platform Evolution for the Next Decade. *Intel White Paper, Platform 2015*, Edited by R. M. Ramanathan and Vince Thomas, Intel Corporation, August 2005.

[32] Bernard R Brooks, Charles L Brooks, Alexander D MacKerell, Lennart Nilsson, RJ Petrella, Benoît Roux, Youngdo Won, Georgios Archontis, Christoph Bartels, Stefan Boresch, et al. Charmm: the biomolecular simulation program. *Journal of computational chemistry*, 30(10):1545–1614, 2009.

[33] Bernard R Brooks, Robert E Bruccoleri, Barry D Olafson, S Swaminathan, Martin Karplus, et al. Charmm: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of computational chemistry*, 4(2):187–217, 1983.

[34] Doug Burger and Todd M. Austin. The SimpleScalar Tool Set, Version 2.0. CS TR 1342, University of Wisconsin-Madison, June 1997.

[35] Harold W. Cain and Mikko H. Lipasti. Edge chasing delayed consistency: pushing the limits of weak memory models. In *Proceedings of the 2012 ACM workshop on Relaxing synchronization for multicore and manycore scalability*, RACES '12, pages 15–24, New York, NY, USA, 2012. ACM.

[36] B. Calder, C. Krintz, S. John, and T. Austin. Cache-conscious data placement. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, San Jose, CA, October 1998.

[37] David Callahan, Ken Kennedy, and Allan Porterfield. Software prefetching. *SIGOPS Operating System Review*, 25(Special Issue):40–52, April 1991.

[38] David Callahan, Ken Kennedy, and Allan Porterfield. Software prefetching. *SIGARCH Computer Architecture News*, 19(2):40–52, April 1991.

[39] David Callahan, Ken Kennedy, and Allan Porterfield. Software prefetching. In *Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, ASPLOS IV, pages 40–52, New York, NY, USA, 1991. ACM.

[40] David Callahan, Ken Kennedy, and Allan Porterfield. Software prefetching. *SIGPLAN Notices*, 26(4):40–52, April 1991.

[41] M. Carlisle, A. Rogers, J. Reppy, and L. Hendren. Early experiences with Olden. In *Proceedings of the Sixth Workshop on Languages and Compilers for Parallel Computing*, Portland, OR, August 1993.

[42] David Chaiken, John Kubiatowicz, and Anant Agarwal. LimitLESS directories: a scalable cache coherence scheme. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 1991.

[43] Siew Yin Chan, Teck Chaw Ling, and Eric Aubanel. The impact of heterogeneous multi-core clusters on graph partitioning: an empirical study. *Cluster Computing*, 15(3):281–302, September 2012.

[44] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2005.

[45] Jichuan Chang and Guri Sohi. Cooperative caching for chip multiprocessors. In *33rd International Symposium on Computer Architecture (ISCA 2006), Boston, MA, USA*, 2006.

[46] P. Chaparro, J. Gonzalez, G. Magklis, Cai Qiong, and A. Gonzalez. Understanding the thermal implications of multi-core architectures. *Parallel and Distributed Systems, IEEE Transactions on*, 18(8):1055–1065, 2007.

[47] G. Chen and M. Kandemir. Optimizing inter-processor data locality on embedded chip multiprocessors. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 227–236, New York, NY, USA, 2005. ACM Press.

[48] T. Chen, R. Raghavan, J.N. Dale, and E. Iwata. Cell broadband engine architecture and its first implementation – a performance view. *IBM Journal of Research and Development*, 51(5):559–572, 2007.

[49] T. Chilimbi, M. Hill, and J. Larus. Cache-conscious structure layout. In *Proceedings of the SIGPLAN '99 Conference on Programming Language Design and Implementation*, Atlanta, GA, May 1999.

[50] Zeshan Chishti, Michael D. Powell, and T. N. Vijaykumar. Optimizing replication, communication, and capacity allocation in CMPs. In *32nd International Symposium on Computer Architecture (ISCA 2005), Madison, Wisconsin, USA*, pages 357–368, 2005.

[51] Markus Christen, Philippe H Hünenberger, Dirk Bakowies, Riccardo Baron, Roland Bürgi, Daan P Geerke, Tim N Heinz, Mika A Kastenholz, Vincent Kräutler, Chris Oostenbrink, et al. The gromos software for biomolecular simulation: Gromos05. *Journal of computational chemistry*, 26(16):1719–1751, 2005.

[52] S. Coleman and K. S. McKinley. Tile size selection using cache organization and data layout. In *Proceedings of the SIGPLAN '95 Conference on Programming Language Design and Implementation*, La Jolla, CA, June 1995.

[53] A.M. Dani, Y.N. Srikant, and B. Amrutur. Efficient cache exploration method for a tiled chip multiprocessor. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–6, 2012.

[54] R. Das, M. Uysal, J. Saltz, and Y.-S. Hwang. Communication optimizations for irregular scientific computations on distributed memory architectures. *Journal of Parallel and Distributed Computing*, 22(3):462–479, September 1994.

[55] John D. Davis, Cong Fu, and James Laudon. The RASE (rapid, accurate simulation environment) for chip multiprocessors. *SIGARCH Computer Architecture News*, 33(4):14–23, 2005.

[56] C. Ding and K. Kennedy. Improving cache performance of dynamic applications with computation and data layout transformations. In *Proceedings of the SIGPLAN '99 Conference on Programming Language Design and Implementation*, Atlanta, GA, May 1999.

[57] Chen Ding and Trishul Chilimbi. All-window profiling of concurrent executions. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2008.

[58] Chen Ding and Trishul Chilimbi. A composable model for analyzing locality of multi-threaded programs. Technical Report MSR-TR-2009-107, Microsoft Research, 2009.

[59] Chen Ding and Yutao Zhong. Predicting whole-program locality through reuse distance analysis. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*. ACM, 2003.

[60] David Eklov and Erik Hagersten. Statstack: efficient modeling of lru caches. In *Proceedings of the International Symposium on Performance Analysis of Systems Software*, 2010.

[61] N.D. Enright Jerger, Li-Shiuan Peh, and M.H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pages 35–46, 2008.

[62] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Power challenges may end the multicore era. *Communications of the ACM*, 56(2):93–102, February 2013.

[63] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th International Symposium on Computer Architecture*, 2011.

[64] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th International Symposium on Computer Architecture*, 2011.

[65] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Power limitations and dark silicon challenge the future of multicore. *ACM Transaction on Computer Systems*, 30(3):11:1–11:27, August 2012.

[66] Yi Feng and Emery D. Berger. A locality-improving dynamic memory allocator. In *MSP '05: Proceedings of the 2005 workshop on Memory system performance*, pages 68–77, New York, NY, USA, 2005. ACM Press.

[67] Fujitsu. Multi-core multi-thread processor sparc64 series.

[68] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: An analytical representation of cache misses. In *Proceedings of the 1997 ACM International Conference on Supercomputing*, Vienna, Austria, July 1997.

[69] Michael Gschwind. Chip multiprocessing and the cell broadband engine. In *Proceedings of the 3rd conference on Computing frontiers, CF'06*, pages 1–8, New York, NY, USA, 2006. ACM Press.

[70] Rajiv Gupta and Mary Lou Soffa. Region Scheduling: An Approach for Detecting and Redistributing Parallelism. *IEEE Transactions on Software Engineering*, 16(4):421–431, April 1990.

[71] J. L. Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533, May 1988.

[72] Lance Hammond, Basem A. Nayfeh, and Kunle Olukotun. A single-chip multiprocessor. *IEEE Computer*, 30(9):79–85, 1997.

[73] Lance Hammond, Vicky Wong, Mike Chen, Brian D. Carlstrom, John D. Davis, Ben Hertzberg, Manohar K. Prabhu, Honggo Wijaya, Christos Kozyrakis, and Kunle Olukotun. Transactional memory coherence and consistency. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, page 102. IEEE Computer Society Press., Jun 2004.

[74] H. Han and C.-W. Tseng. A comparison of locality transformations for irregular codes. In *Proceedings of the 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, Rochester, NY, May 2000.

[75] H. Han and C.-W. Tseng. Improving locality for adaptive irregular codes. In *Proceedings of the Thirteenth Workshop on Languages and Compilers for Parallel Computing*, White Plains, NY, August 2000.

[76] Hwansoo Han and Chau-Wen Tseng. A comparison of parallelization techniques for irregular reductions. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01), San Francisco, CA, April 23-27*, page 27. IEEE Computer Society, 2001.

[77] Hwansoo Han and Chau-Wen Tseng. Exploiting locality for irregular scientific codes. *IEEE Transactions on Parallel and Distributed Systems*, 17(7):606–618, 2006.

[78] Sarojadevi Hande. *Processor-Directed Cache Coherence Mechanism: Architectural Support for Early and Local Cache Coherence in DSM Systems*. LAP Lambert Academic Publishing, Germany, 2012.

[79] V. Hanumaiah, S. Vrudhula, and K.S. Chatha. Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(11):1677–1690, 2011.

[80] Nikolaos Hardavellas, Stephen Somogyi, Thomas F. Wenisch, Roland E. Wunderlich, Shelley Chen, Jangwoo Kim, Babak Falsafi, James C. Hoe, and Andreas G. Nowatzyk. SIMFLEX: a fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *SIGMETRICS Performance Evaluation Review*, 31(4):31–34, 2004.

[81] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. Reactive NUCA: near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th International Symposium on Computer Architecture*, 2009.

[82] Sarah Harris. Synergistic Caching in Single-Chip Multiprocessors., PhD Thesis. Technical report, Stanford University, March 2005.

[83] Allan Hartstein, Viji Srinivasan, Thomas R. Puzak, and Philip G. Emma. Cache miss behavior: is it sqrt(2)? In *Proceedings of the Conference Computing Frontiers*, 2006.

[84] Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *IEEE Computer*, 2008.

[85] Yatin Hoskote, Sriram Vangal, Nitin Borkar, and Shekhar Borkar. Teraflop Prototype Processor with 80 Cores. In *Proceedings of the Symposium on High Performance Chips*, 2007.

[86] Lisa Hsu, Ravi Iyer, Srihari Makineni, Steve Reinhardt, and Donald Newell. Exploring the cache design space for large scale CMPs. *SIGARCH Computer Architecture News*, 33(4):24–33, 2005.

[87] http://tilera.com/products/processors. Processors from Tilera Corporation.

[88] Wei Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M.R. Stan. Hotspot: a compact thermal modeling methodology for early-stage vlsi design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(5):501–513, 2006.

[89] Christopher J. Hughes, Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve. RSIM: Simulating shared-memory multiprocessors with ilp processors. *IEEE Computer*, 35(2):40–49, Feb. 2002.

[90] Intel. Intel public roadmap for desktop, mobile, data center.

[91] Aamer Jaleel, Matthew Mattina, and Bruce Jacob. Last-level cache (LLC) performance of data-mining workloads on a CMP–a case study of parallel bioinformatics workloads. In *12th International Conference on High-Performance Computer Architecture (HPCA-12 2005), Austin TX, USA*, pages 88–98, February 2006.

[92] Yunlian Jiang, Eddy Z. Zhang, Kai Tian, and Xipeng Shen. Is reuse distance applicable to data locality analysis on chip multiprocessors? In *Proceeding of Compiler Construction*, 2010.

[93] Mahmut Kandemir. Data locality enhancement for cmps. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, ICCAD '07, pages 155–159, Piscataway, NJ, USA, 2007. IEEE Press.

[94] Mahmut Kandemir and *et al.*. Compiler algorithm for increasing sharing. In *Proceedings of the 2010 IEEE Microarchitecture Conference*, MICRO '10, New York, NY, USA, 2010. IEEE.

[95] Mahmut Kandemir, Taylan Yemliha, SaiPrashanth Muralidhara, Shekhar Srikantaiah, Mary Jane Irwin, and Yuanrui Zhnag. Cache topology aware computation mapping for multicores. In *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '10, pages 74–85, New York, NY, USA, 2010. ACM.

[96] G. Karypis, V. Kumar, and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[97] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. In *Proceedings of the 1995 International Conference on Parallel Processing, August 14-18, 1995, Urbana-Champain, Illinois, USA.*, pages 113–122.

[98] George Karypis and Vipin Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *Siam Review*, 41(2):278–300, 1999.

[99] Chi keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa, and Reddi Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Programming Language Design and Implementation*, 2005.

[100] M. Y. Kim and A. N. Tantawi. Asynchronous disk interleaving: Approximating access delays. *IEEE Transactions on Computers*, 40(7), July 1991.

[101] Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, 2005.

[102] Anil Krishna, Ahmad Samih, and Yan Solihin. Impact of data sharing on cmp design: a study based on analytical modeling. In *Proceedings of the Second International Workshop on New Frontiers in High-performance and Hardware-aware Computing*, 2011.

[103] Milind Kulkarni, Vijay Pai, and Derek Schuff. Towards architecture independent metrics for multicore performance analysis. *ACM SIGMETRICS Performance Evaluation Review*, 2010.

[104] M. Lam, E. Rothberg, and M. E. Wolf. The cache performance and optimizations of blocked algorithms. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, Santa Clara, CA, April 1991.

[105] Dominique LaSalle and George Karypis. Multi-threaded graph partitioning. In *Proceedings of the 27th IEEE International Parallel & Distributed Processing Symposium*, IPDPS, 2013.

[106] Jian Li and José F. Martínez. Power-performance considerations of parallel computing on chip multiprocessors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2(4):397–422, 2005.

[107] Yingmin Li, David Brooks, Zhigang Hu, and Kevin Skadron. Performance, energy, and thermal considerations for SMT and CMP architectures. In *11th International Conference on High-Performance Computer Architecture (HPCA-11 2005), San Francisco, CA, USA*, pages 71–82, 2005.

[108] Jack L. Lo, Susan J. Eggers, Henry M. Levy, Sujay S. Parekh, and Dean M. Tullsen. Tuning compiler optimizations for simultaneous multithreading. In *MICRO 30: Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 114–124, Washington, DC, USA, 1997. IEEE Computer Society.

[109] Jack L. Lo, Susan J. Eggers, Henry M. Levy, Sujay S. Parekh, and Dean M. Tullsen. Tuning compiler optimizations for simultaneous multithreading. *International Journal of Parallel Programming*, 27(6):477–503, 1999.

[110] H. Lu, A. Cox, S. Dwarkadas, R. Rajamony, and W. Zwaenepoel. Compiler and software distributed shared memory support for irregular applications. In *Proceedings of the Sixth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Las Vegas, NV, June 1997.

[111] Gregory Lueck, Harish Patil, and Cristiano Pereira. Pinadx: an interface for customizable debugging with dynamic instrumentation. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization*, CGO '12, pages 114–123, New York, NY, USA, 2012. ACM.

[112] Peter Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, and Gustav Hallberg. SIMICS: A full system simulation platform. *IEEE Computer*, 35(2):50–58, Feb. 2002.

[113] Milo M. K. Martin, Mark D. Hill, and David A. Wood. Token coherence: Decoupling performance and correctness. In *Proceedings of the 30th Annual International Symposium on Computer Architecture, June 2003.* 2003.

[114] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and

David A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Computer Architecture News*, 33(4):92–99, 2005.

[115] R.L. Mattson, J. Gecsei, D.R. Slutz, and I.L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[116] Collin McCurdy and Charles Fischer. Using pin as a memory reference generator for multiprocessor simulation. *SIGARCH Computer Architecture News*, 2005.

[117] Austen McDonald, JaeWoong Chung, Hassan Chafi, Chi Cao Minh, Brian D. Carlstrom, Lance Hammond, Christos Kozyrakis, and Kunle Olukotun. Characterization of TCC on chip-multiprocessors. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques.* Sept 2005.

[118] K. S. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems*, 18(4):424–453, July 1996.

[119] J. Mellor-Crummey, D. Whalley, and K. Kennedy. Improving memory hierarchy performance for irregular applications. In *Proceedings of the 1999 ACM International Conference on Supercomputing*, Rhodes, Greece, June 1999.

[120] J.E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. Graphite: A distributed parallel simulator for multicores. In *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA'10)*, pages 1–12, 2010.

[121] N. Mitchell, L. Carter, and J. Ferrante. Localizing non-affine array references. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Newport Beach , LA, October 1999.

[122] Nicholas Mitchell, Karin Högstedt, Larry Carter, and Jeanne Ferrante. Quantifying the multi-level nature of tiling interactions. *International Journal of Parallel Programming*, 26(6):641–670, 1998.

[123] Matteo Monchiero, Ramon Canal, and Antonio González. Design space exploration for multicore architectures: a power/performance/thermal view. In *Proceedings of the 20th annual international conference on Supercomputing*, 2006.

[124] Irene Moulitsas and George Karypis. Architecture aware partitioning algorithms. In *Proceedings of the 8th international conference on Algorithms and Architectures for Parallel Processing*, ICA3PP '08, pages 42–53, Berlin, Heidelberg, 2008. Springer-Verlag.

[125] Shubhendu S. Mukherjee, Joel S. Emer, and Steven K. Reinhardt. The soft error problem: An architectural perspective. In *11th International Conference on High-Performance Computer Architecture (HPCA-11 2005), 12-16 February 2005, San Francisco, CA, USA*, pages 243–247. IEEE Computer Society.

[126] Shubhendu S. Mukherjee and Mark D. Hill. An evaluation of directory protocols for medium-scale shared-memory multiprocessors. In *Proceedings of the International Conference on Supercomputing*, 1994.

[127] B. Nayfeh and K. Olukotun. Exploring the design space for a shared cache multiprocessor. In *In Proceedings of the 21st Annual International Symposium on Computer Architecture, pages 166–175, Chicago, Illinois, April 1994.* IEEE Computer Society Press., 1994.

[128] Basem A. Nayfeh, Lance Hammond, and Kunle Olukotun. Evaluation of Design Alternatives for a Multiprocessor Microprocessor. In *Proceedings of 23rd International Symposium on Computer Architecture (ISCA-23)*, pages 67–77, Philadelphia, PA, May 1995. ACM.

[129] Dimitrios S. Nikolopoulos. Code and Data Transformations for Improving Shared Cache Performance on SMT Processors. In Alexander V. Veidenbaum, Kazuki Joe, Hideharu Amano, and Hideo Aiso, editors, *Proceedings of the Fifth International Symposium on High Performance Computing*, volume 2858 of *Lecture Notes in Computer Science*, pages 54–69, Tokyo-Odaiba, Japan, Oct 2003. Springer Verlag. **Awarded Best Paper**.

[130] Dimitrios S. Nikolopoulos. Dynamic Tiling for Effective Use of Shared Caches on Multithreaded Processors. *International Journal of High Performance Computing and Networking*, 1(4), Spring 2004.

[131] Ralph Nissler. Power7+ systems are now available in the ibm virtual loaner program (vlp), 2013.

[132] Taecheol Oh, Hyunjin Lee, Kiyeon Lee, and Sangyeun Cho. An analytical model to study optimal area breakdown between cores and caches in a chip multiprocessor. In *Proceedings of the IEEE Computer Society Symposium on VLSI*, 2009.

[133] Kunle Olukotun and Lance Hammond. The future of microprocessors. *ACM Queue*, 3(7):26–29, 2005.

[134] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. In *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, ASPLOS VII, pages 2–11, New York, NY, USA, 1996. ACM.

[135] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. *SIGPLAN Notices*, 31(9):2–11, September 1996.

[136] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. *SIGOPS Operating System Review*, 30(5):2–11, September 1996.

[137] Jeffrey Oplinger, David Heine, Shih-Wei Liao, Basem A. Nayfeh, Monica S. Lam, and Kunle Olukotun. Software and Hardware for Exploiting Speculative Parallelism with a Multiprocessor. CSL-TR 97-715, Stanford University, February 1997.

[138] Isil Oz, Haluk Rahmi Topcuoglu, Mahmut Kandemir, and Oguz Tosun. Reliability-aware core partitioning in chip multiprocessors. *Journal of Systems Architecture*, 58(3-4):160–176, March 2012.

[139] Guoteng Pan, Qiang Dou, and Lunguo Xie. A two-level directory organization solution for cc-numa systems. In *Proceedings of the 7th international conference on Algorithms and architectures for parallel processing*, ICA3PP'07, pages 142–152, Berlin, Heidelberg, 2007. Springer-Verlag.

[140] R. Panda, H. Nakamura, N. Dutt, and A. Nicolau. Augmenting loop tiling with data alignment for improved cache performance. *IEEE Transactions on Computers*, 48(2):142–149, February 1999.

[141] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing representative portions of large intel itanium programs with dynamic instrumentation. In *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on*, pages 81–92, 2004.

[142] Apan Qasem and Ken Kennedy. Evaluating a model for cache conflict miss prediction. Technical Report CS-TR05-457, Rice University, 2005.

[143] Qualcomm. Qualcomm snapdragon 800 processor first to use tsmc's 28hpm advanced process technology: Snapdragon 800 processor on 28nm hpm process delivers high performance and low power for mobile devices, 2013.

[144] Arun Raghavan, Colin Blundell, and Milo M. K. Martin. Token tenure: Patching token counting using directory-based cache coherence. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 47–58, Washington, DC, USA, 2008. IEEE Computer Society.

[145] Arun Raghavan, Colin Blundell, and Milo M. K. Martin. Token tenure and patch: A predictive/adaptive token-counting hybrid. *ACM Transaction Architecture Code Optimization*, 7(2):6:1–6:31, Oct 2010.

[146] Ravi Rajwar, Maurice Herlihy, and Konrad Lai. Virtualizing transactional memory. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 494–505. IEEE Computer Society, Jun 2005.

[147] Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos. SESC simulator, January 2005. http://sesc.sourceforge.net.

[148] G. Rivera and C.-W. Tseng. Data transformations for eliminating conflict misses. In *Proceedings of the SIGPLAN '98 Conference on Programming Language Design and Implementation*, Montreal, Canada, June 1998.

[149] G. Rivera and C.-W. Tseng. A comparison of compiler tiling algorithms. In *Proceedings of the 8th International Conference on Compiler Construction (CC'99)*, Amsterdam, The Netherlands, March 1999.

[150] G. Rivera and C.-W. Tseng. Tiling optimizations for 3d scientific computations. In *Proceedings of SC'00*, Dallas, TX, November 2000.

[151] David P. Rodgers. Improvements in multiprocessor system design. In *Proceedings of the 12th annual international symposium on Computer architecture*, ISCA '85, pages 225–231, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.

[152] Brian Rogers, Anil Krishna, Gordon Bell, Ken Vu, Xiaowei Jiang, and Yan Solihin. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. In *Proceedings of the 36th International Symposium on Computer Architecture*, 2009.

[153] Alberto Ros, Manuel E. Acacio, and José M. García. A novel lightweight directory architecture for scalable shared-memory multiprocessors. In *Proceedings of the 11th international Euro-Par conference on Parallel Processing*, Euro-Par'05, pages 582–591, Berlin, Heidelberg, 2005. Springer-Verlag.

[154] Alberto Ros, Manuel E. Acacio, and José M. García. A scalable organization for distributed directories. *J. Syst. Archit.*, 56(2-3):77–87, February 2010.

[155] Daniel Sanchez, George Michelogiannakis, and Christos Kozyrakis. An analysis of on-chip interconnection networks for large-scale chip multiprocessors. *ACM Transactions on Architecture and Code Optimization*, 2010.

[156] Derek L. Schuff, Milind Kulkarni, and Vijay S. Pai. Accelerating multicore reuse distance analysis with sampling and parallelization. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, 2010.

[157] Derek L. Schuff, Benjamin S. Parsons, and Jivay S. Pai. Multicore-aware reuse distance analysis. Technical Report TR-ECE-09-08, Purdue University, 2009.

[158] Xipeng Shen and Jonathan Shaw. Scalable implementation of efficient locality approximation. In *Proceedings of the International Workshop on Languages and Compilers for Parallel Computing*, 2008.

[159] Xipeng Shen, Jonathan Shaw, Brian Meeker, and Chen Ding. Locality approximation using time. In *Proceedings of the 34th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2007.

[160] Xudong Shi, Feiqi Su, Jih-Kwon Peir, Ye Xia, and Zhen Yang. Modeling and single-pass simulation of cmp cache capacity and accessibility. In *Proceedings of the International Symposium on Performance Analysis of Systems Software*, 2007.

[161] Y. Song and Z. Li. New tiling techniques to improve cache temporal locality. In *Proceedings of the SIGPLAN '99 Conference on Programming Language Design and Implementation*, Atlanta, GA, May 1999.

[162] Evan Speight, Hazim Shafi, Lixin Zhang, and Ramakrishnan Rajamony. Adaptive mechanisms and policies for managing cache hierarchies in chip multiprocessors. In *32nd International Symposium on Computer Architecture (ISCA 2005), Madison, Wisconsin, USA*, pages 346–356, 2005.

[163] Lawrence Spracklen and Santosh G. Abraham. Chip multithreading: Opportunities and challenges. In *11th International Conference on High-Performance Computer Architecture (HPCA-11 2005), 12-16 February 2005, San Francisco, CA, USA*, pages 248–252. IEEE Computer Society, 2005.

[164] Jayanth Srinivasan, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *32nd International Symposium on Computer Architecture (ISCA 2005), Madison, Wisconsin, USA*, pages 520–531, 2005.

[165] Phillip Stanley-Marbell, Victoria Caparros Cabezas, and Ronald Luijten. Pinned to the walls: impact of packaging and application properties on the memory and power walls. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, ISLPED '11, pages 51–56, Piscataway, NJ, USA, 2011. IEEE Press.

[166] Michelle Mills Strout, Larry Carter, and Jeanne Ferrante. Compile-time composition of run-time data and iteration reorderings. In *PLDI'03: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation 2003, San Diego, California, USA, June 9-11*, pages 91–102, 2003.

[167] Michelle Mills Strout and Paul D. Hovland. Metrics and models for reordering transformations. In *Proceedings of the 2004 workshop on Memory System Performance, Washington, DC, June 8*, pages 23–34. ACM, 2004.

[168] G. Edward Suh, Srinivas Devadas, and Larry Rudolph. Analytical cache models with applications to cache partitioning. In *Proceedings of International Conference on Supercomputing*, 2001.

[169] Guangyu Sun, Christopher J. Hughes, Changkyu Kim, Jishen Zhao, Cong Xu, Yuan Xie, and Yen-Kuang Chen. Moguls: a model to explore the memory hierarchy for bandwidth improvements. In *Proceedings of the 38th International Symposium on Computer Architecture*, 2011.

[170] Hyojin Sung, Rakesh Komuravelli, and Sarita V. Adve. Denovond: efficient hardware support for disciplined non-determinism. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '13, pages 13–26, New York, NY, USA, 2013. ACM.

[171] Tilera. Tilera announces tile-gx72, the worlds highest-performance and highest-efficiency manycore processor, 2013.

[172] D. N. Truong, François Bodin, and André Seznec. Improving cache behavior of dynamically allocated data structures. In *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques, 12-18 October, 1998, Paris, France. IEEE Computer Society*, pages 322–.

[173] Dean Tullsen, Susan Eggers, and Henry Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 392–403, Santa Margherita Ligure, Italy, June 1995. ACM.

[174] Dean M. Tullsen, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, and Rebecca L. Stamm. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In *Proceedings of the 1996 International Symposium on Computer Architecture*, Philadelphia, May 1996.

[175] Dean M. Tullsen, Jack L. Lo, Susan J. Eggers, and Henry M. Levy. Supporting Fine-Grained Synchronization on a Simultaneous Multithreading Processor. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture*, January 1999.

[176] W. F. van Gunsteren and H. J. C. Berendsen. GROMOS: GROningen MOlecular Simulation software. Technical report, Laboratory of Physical Chemistry, University of Groningen, Nijenborgh,The Netherlands, 1988.

[177] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob. DRAMSIM: a memory system simulator. *SIGARCH Computer Architecture News*, 33(4):100–107, 2005.

[178] Thomas F. Wenisch, Roland E. Wunderlich, Mike Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe. SIMFLEX: Statistical sampling of computer architecture simulation. *IEEE Micro special issue on Computer Architecture Simulation*, July/August. 2006.

[179] David Wentzlaff, Nathan Beckmann, Jason Miller, , and Anant Agarwal. Core count vs cache size for manycore architectures in the cloud. Technical Report MIT-CSAIL-TR-2010-008, Massachusetts Institute of Technology, 2010.

[180] M. Wolf and M. Lam. A data locality optimizing algorithm. In *Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation*, Toronto, Canada, June 1991.

[181] Meng-Ju Wu. *Reuse Distance Analysis for Large-Scale Chip Multiprocessors*. Ph.D. dissertation, ECE Dept., University of Maryland, 2012.

[182] Meng-Ju Wu and D. Yeung. Coherent profiles: Enabling efficient reuse distance analysis of multicore scaling for loop-based parallel programs. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 264–275, 2011.

[183] Meng-Ju Wu and Donald Yeung. Scaling single-program performance on large-scale chip multiprocessors. Technical Report UMIACS-TR-2009-16, University of Maryland, November 2009.

[184] Meng-Ju Wu and Donald Yeung. Identifying optimal multicore cache hierarchies for loop-based parallel programs via reuse distance analysis. In *Proceedings of the 2012 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, MSPC '12, pages 2–11, New York, NY, USA, 2012. ACM.

[185] Meng-Ju Wu and Donald Yeung. Understanding Multicore Cache Behavior of Loop-based Parallel Programs via Reuse Distance Analysis. Technical Report UMIACS-TR-2012-01, University of Maryland, Jan 2012.

[186] Meng-Ju Wu and Donald Yeung. Efficient reuse distance analysis of multicore scaling for loop-based parallel programs. *ACM Transactions on Computer Systems*, 31(1):1:1–1:37, Feb 2013.

[187] Meng-Ju Wu, Minshu Zhao, and Donald Yeung. Studying multicore processor scaling via reuse distance analysis. In *the 40th International Symposium on Computer Architecture*, ISCA-XL, June 2013.

[188] www.intel.com/software/products/vtune/. Intel vtune performance profiler and analyze.

[189] Xiaoya Xiang, Bin Bao, Tongxin Bai, Chen Ding, and Trishul Chilimbi. All-window profiling and composable models of cache sharing. In *Proceedings of the 16th ACM symposium on Principles and Practice of Parallel Programming*, 2011.

[190] Xiaoya Xiang, Bin Bao, Chen Ding, and Yaoqing Gao. Linear-time modeling of program working set in shared cache. In *Proceedings of the 20th International Symposium on Parallel Architectures and Compilation Techniques*, PACT'11, 2011.

[191] Di Xu, Chenggang Wu, and Pen-Chung Yew. On mitigating memory bandwidth contention through bandwidth-aware scheduling. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT '10, pages 237–248, New York, NY, USA, 2010. ACM.

[192] Yi Xu, Yu Du, Youtao Zhang, and Jun Yang. A composite and scalable cache coherence protocol for large scale cmps. In *Proceedings of the international conference on Supercomputing*, ICS '11, pages 285–294, New York, NY, USA, 2011. ACM.

[193] Eddy Z. Zhang, Yunlian Jiang, and Xipeng Shen. Does cache sharing on modern cmp matter to the performance of contemporary multithreaded programs? In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '10, pages 203–212, New York, NY, USA, January 2010. ACM. Best Paper Award.

[194] Eddy Z. Zhang, Yunlian Jiang, and Xipeng Shen. Does cache sharing on modern cmp matter to the performance of contemporary multithreaded programs? *ACM SIGPLAN Notices*, 45(5):203–212, January 2010.

[195] Eddy Z. Zhang, Yunlian Jiang, and Xipeng Shen. The significance of cmp cache sharing on contemporary multithreaded applications. *IEEE Trans. Parallel & Distributed Systems*, 23(2):367–374, February 2012.

[196] Michael Zhang and Krste Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *32nd International Symposium on Computer Architecture (ISCA 2005), Madison, Wisconsin, USA*, pages 336–345, 2005.

[197] Michael Zhang and Krste Asanovic. Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *Proceedings of the 32nd International Symposium on Computer Architecture*, 2005.

[198] Youtao Zhang and Rajiv Gupta. Data compression transformations for dynamically allocated data structures. In *Proceedings Compiler Construction, 11th International Conference, CC 2002, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12*, volume 2304 of *Lecture Notes in Computer Science*, pages 14–28. Springer, 2002.

[199] Li Zhao, Ravi Iyer, Srihari Makineni, Jaideep Moses, Ramesh Illikkal, and Donald Newell. Performance, area and bandwidth implications on large-scale

CMP cache design. In *Proceedings of the Workshop on Chip Multiprocessor Memory Systems and Interconnect*, 2007.

[200] Chuanlei Zheng, P. Shukla, Shuai Wang, and Jie Hu. Exploring hardware transaction processing for reliable computing in chip-multiprocessors against soft errors. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on*, pages 92–97, 2012.

[201] Yutao Zhong, Steven G. Dropsho, and Chen Ding. Miss rate prediction across all program inputs. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, 2003.

[202] Yutao Zhong, Steven G. Dropsho, Xipeng Shen, Ahren Studer, and Chen Ding. Miss rate prediction across program inputs and cache configurations. *IEEE Transactions on Computers*, 2007.

[203] Yutao Zhong, Maksim Orlovich, Xipeng Shen, and Chen Ding. Array regrouping and structure splitting using whole-program reference affinity. In *PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, pages 255–266, New York, NY, USA, 2004. ACM Press.

Abdel-Hameed A. Badawy CV

Education

University of Maryland, College Park, MD, USA
Electrical and Computer Engineering Department
Ph.D., Electrical and Computer Engineering, Expected August 2013
Dissertation Title: "Reuse Distance for Compiler Optimization Selection and Program Analysis of Sharing, Locality, and Prefetching on Chip-Multiprocessors for Graph-based Problems".

University of Maryland, College Park, MD, USA
Electrical and Computer Engineering Department
M.S. Electrical and Computer Engineering, August 2002
Thesis Title: "Evaluating the Impact of Memory System Performance on Software Prefetching and Locality Optimizations".

Information Technology Institute (ITI), Giza, Egypt
Information Decision Support Center (IDSC), Egyptian Cabinet
Software Skills Development Graduate Program
Software Engineering Diploma, 1997.
Graduation Project Title: "Mobile Aglets (Agent) System".
Ranked among the top 5 trainees in a class of more than 200 trainees.

Mansoura University, Mansoura, Egypt
Computers and Systems Department
B.Sc. Electronics Engineering, 1996.
Degree conferred with Distinction and Honor.
Major: Computer Engineering and Automatic Control.
Capstone Senior Graduation Project: "Computer-controlled Robot Arm System".
Ranked 2nd on a class of 85 students.

Research Interests

Computer Architecture, Performance Modeling, High Performance Computing, Locality Optimizations, Compile-time Optimization Techniques, Profile-guided Optimizations, Computational Intelligence Techniques for Compilers, Adaptive Computing Systems, Cognitively Enabled Architectures, Applications for Computational Intelligence, Computational Intelligence Techniques for Biomedical Applications, Computer Engineering Education, Computer Science Education, Scholarship of Teaching and Learning, and Jurisprudence.

Honors, Awards, & Grants

1. **Distinguished Student Excellence Award** in every year of the the B.Sc. Study, 1991 – 1996.

2. **Egyptian Cabinet Scholarship** for a training program at the Information

Technology Institute (ITI), Giza, Egypt, August 1996 - July 1997.

3. **Award of Excellence** and a **Grant of** $200 in the University of Maryland, Graduate Research Interaction Day (GRID'2004) (An interdisciplinary research competition attended by more than 200 presenters) for the poster titled: "Prefetch-aware Memory Controllers", April 2004.

4. **CTE Lilly-East Conference Grant** from UMD's Center for Teaching Excellence (CTE) to attend the Conference with the theme: "Learning by Design", held at the University of Delaware, April $16 - 18, 2008$.

5. **CTE Lilly-East Conference Grant** from UMD's Center for Teaching Excellence (CTE) to attend the Conference with the theme: "Millennial Learning: Teaching in the 21st Century", held at the University of Delaware, April $15 - 17, 2009$.

6. **Noyce Scholarship** to attend the 2010 National Center for Science & Civic Engagement (NCSCE) Symposium and Capitol Hill poster Session held in Washington DC and the University of Maryland, College Park, April 2010. **Only five scholarships were awarded**.

7. **CTE Lilly-DC Conference Grant** from the Center for Teaching Excellence (CTE) to attend Conference with the theme: "Evidence based Teaching and Learning", held in Washington DC, June $3 - 5, 2010$.

8. **Teaching Assistants Training and Development (TATD) Fellow** at the ECE Dept., University of Maryland, for the $2010 - 2011$ academic year.

9. **TATD Fellow Conference Travel Grant** of $250 in recognition for service to the ECE Dept., University of Maryland, August 2010.

10. **Best Student Paper Award** and a **Grant of** $200 for the paper titled: "One-Class Support Vector Machines vs. Two-class Support Vector Machines for Normal Mammogram Detection", presented at the IEEE Workshop on Applied Imagery Pattern Recognition (AIPR' 2010), Washington, DC, October 2010.

11. **U**niversity of Maryland Graduate School Goldhaber Conference Travel Grant of $400 to present a paper at the International Symposium on Biomedical Imaging (ISBI'2011), March 2011.

12. **NIH-funded Conference Travel Grant** of $300 to attend the International Symposium on Biomedical Imaging (ISBI'2011), March 2011.

13. **CTE Lilly-DC Conference attendance Grant** from UMD's Center for Teaching Excellence (CTE) to attend the Conference with the theme: "Evidence based Teaching and Learning", held in Washington DC, June $3-5, 2011$.

14. **Poster presentation Invitation** at the 8th Era of Hope Conference, Breast Cancer Research Program (BCRP), Department of Defense, held in Orlando, FL, August $2-5, 2011$.
15. **Teaching Assistants Training and Development (TATD) Fellow** at the ECE Dept., University of Maryland, for the $2011-2012$ academic year.
16. **Teaching Assistants Training and Development (TATD) Fellow Conference Travel Grant** of $250 in recognition for service to the ECE Dept., University of Maryland, August 2011.
17. **CTE-Lilly Graduate Fellow**, a cohort of nine graduate students to do a relevant teaching and learning study to the University of Maryland and its students. Nine graduate students only are selected for this fellowship for the $2011-2012$. A summary of the developed project is available here.
18. **CTE-Lilly Graduate Fellows Honorary Stipend** of $1000 funded by the University of Maryland Graduate School and CTE, September 2011.
19. **CTE-Lilly Graduate Fellows Conference Travel Grant** of $500 funded by the the University of Maryland Graduate School and CTE to present and participate at the 31st Annual International Lilly Conference on College Teaching held at Miami University, Oxford, OH, November $17-20, 2011$.
20. **Best Student Poster Award** and a **Grant of** $500 in the University of Maryland, Graduate Research Interaction Day (GRID'2012) (An interdisciplinary research competition attended by more than 500 presenters) for the poster titled: "Building a Tool for pre-assessing Student Expectations", April 2012.
21. **CTE Lilly-DC Conference attendance Grant** from UMD's Center for Teaching Excellence (CTE) to attend the Lilly DC Conference with the theme: "Evidence based Teaching and Learning", held in Washington DC, June $1-3, 2012$.
22. **Nominated Member of CIRTL's task-force** established at UMD for the Center for Integration of Research, Teaching and Learning (CIRTL) at UMD. This task-force is established after UMD had participated in a 25 million dollar grant submitted to the National Science Foundation (NSF).
23. **CTE Lilly-DC Conference attendance Grant** from UMD's Center for Teaching Excellence (CTE) to attend the Lilly DC Conference with the theme: "Evidence based Teaching and Learning", held in Washington DC, June $1-3, 2013$.
24. **CTE Distinguished Teaching Assistant Honoree** at the Center for Teaching Excellence (CTE) annual CTE DTA Ceremony in recognition for the completion of the "University Teaching and Learning Program" (UTLP), May 10th, 2013.

25. **Nominated to Marquis Who's Who in America**, August, 2013.

Publications

1. Abdel-Hameed A. Badawy *et.al.*, "Computer-controlled Robot Arm System", Technical Report, Capstone Graduation Project Report, Computer and Systems Dept., School of Engineering, Mansoura University, Mansoura, Egypt, June 1996.

2. Abdel-Hameed A. Badawy *et.al.*, "Mobile Intelligent Agent (Aglet) System based on the UNIMEM Algorithm for the Student Admissions Decision Problem", Technical Report, SSDP-81 graduation project report published by the Information Technology Institute, Giza, Egypt, June 1997.

3. Aneesh Aggrawal, Abdel-Hameed A. Badawy, Donald Yeung, and Chau-Wen Tseng, "Evaluating the Impact of Memory System Performance on Software Prefetching and Locality Optimizations", Technical Reports of the Computer Science Department & UMIACS, $CS-TR-4169$; $UMIACS-TR-2000-57$, August 2000. [Preliminary results Technical Report, 51 ***Downloads***, 1 ***Citation*** Google Scholar]

4. Abdel-Hameed A. Badawy, Aneesh Aggrawal, Donald Yeung, and Chau-Wen Tseng, "Evaluating the Impact of Memory System Performance on Software Prefetching and Locality Optimizations", Appeared in the proceedings of ACM's International Conference on Supercomputing 2001 (ICS'2001), Pages $486-500$, Sorrento, Italy, June 2001. [Conference ***Acceptance Rate*** 34%; 45 out of 133 submissions, 38 ***Citations***, Google Scholar, 434 ***Downloads*** from the ACM Digital Library]

5. Abdel-Hameed A. Badawy, "Evaluating the Impact of Memory System Performance on Software Prefetching and Locality Optimizations", Technical Reports of the Computer Science Department & UMIACS, CS-TR-4392; UMIACS-TR-2002−72, September 2002. [Master's Thesis Technical Report, 362 ***Downloads***]

6. Abdel-Hameed A. Badawy, Aneesh Aggrawal, Donald Yeung, and Chau-Wen Tseng, "The Efficacy of Software Prefetching and Locality Optimizations on Future Memory Systems", The Journal of Instruction-Level Parallelism, Volume 6, Number 7, July 2004. [25 ***Citations*** Google Scholar, JILP paper PDF]

7. Abdel-Hameed A. Badawy, and Michelle Hugue, "The Effectiveness of Blackboard and Discussion Boards in Junior Computer Science/Engineering Class: Survey and Students' Reflections", Accepted at the 17th International Conference on Learning, Hong Kong Institute of Education, Hong Kong, Paper # $L10P1185$, July $6-9$, 2010. Abstract, PowerPoint Slides, Youtube video Part I, Part II

8. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, "Using Local Binary Pattern Features for Normal Mammogram Detection", Appeared in proceedings of the 23rd IEEE International Symposium on Computer-Based Medical Systems (CBMS'2010), Perth, Australia, October $12-15$, 2010. (http://goo.gl/DbYRT)

9. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, "One-Class Support Vector Machines vs. Two-class Support Vector Machines for Normal Mammogram Detection", Appeared in proceedings of IEEE Workshop on Applied Imagery Pattern Recognition (AIPR' 2010), Washington DC, October $13-15$, 2010. [**Best Student Paper Award**] (http://goo.gl/f7odu)

10. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, and Mohamed F. Chouikha, "Selection of Enhancement Techniques for Mammograms based on Breast Tissue Density", Accepted for publication at the 10th IEEE International Conference on Signal Processing (ICSP'10), Beijing, China, October $24-28$, 2010.

11. Abdel-Hameed A. Badawy, and Michelle Hugue, "Evaluating Discussion Boards on BlackBoard as a Collaborative Learning Tool: A Students' Survey and Reflections", Appeared in proceedings of the IEEE 2010 International Conference on Education and Management Technology (ICEMT'2010), Cairo, Egypt, November $2-4$, 2010. (10.1109/ICEMT.2010.5657540 and arXiv: 1210.1230)

12. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, "Detection of Normal Mammograms based on Breast Tissue Density using GLCM Features", Appeared in proceedings of the 8th IASTED and IEEE EMBS (Engineering in Medicine and Biology Society) International Conference on Biomedical Engineering (BioMed'2011), Innsbruck, Austria, February 2011. (http://goo.gl/ow5sO)

13. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, "Effect of Breast Density in Selecting Features for Normal Mammogram Detection" Appeared at the 8th IEEE International Symposium on Biomedical Imaging (ISBI'2011), Chicago, IL, March 30 - April 2, 2011. (http://goo.gl/4MioG)

14. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, and Mohamed F. Chouikha, "Selection of Enhancement Techniques for Mammograms based on Breast Tissue Density", Accepted for publication at the IASTED International Symposia on Imaging and Signal Processing in Healthcare and Technology (ISPHT'2011), Washington, DC, May 2011.

15. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, "Using Gray Level Co-occurrence Matrix Features for

Normal Mammogram Detection" Appeared at the Society of Imaging Informatics in Medicine Annual Meeting (SIIM'2011), Washington DC, June 2011. (http://goo.gl/voo8n)

16. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, "Screening-out Normal Mammograms using Breast Density Information", Appeared in proceedings of the 8th Era of Hope Workshop, Department of Defense (DoD) Breast Cancer Research Program (BCRP), Paper No. $BC004067-3386$, Poster $P17-7$, Orlando, FL, August $2-5, 2011$. (http://goo.gl/GGOxz)

17. Abdel-Hameed A. Badawy, "Students' Perception of the Effectiveness of Discussion Boards: What can we get from our students for a freebie point?", Appeared in the Science and Information Organization (SAI), International Journal of Advanced Computer Science and Applications(IJACSA), Volume 3, Issue 9, Pages 136 – 144, October, 2012. [Paper on arXiv, 2 ***Citations*** Google Scholar]

18. Karl B. Schmitt, Abdel-Hameed A. Badawy, Sabrina Kramer, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Mara Dougherty, Matthew Miller, Artesha Taylor, Breanne Robertson, Alexis Williams, and Spencer Benson, "Student Expectations from CS and other STEM Courses: They aren't Like CS-Majors! or ($CS \neq STEM - CS$)", *To appear* in proceedings of the Consortium of Computer Science in Colleges, North East Conference, (CCSCNE'2013), Albany, NY, April 2013.

19. Karl B. Schmitt, Abdel-Hameed A. Badawy, Sabrina Kramer, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Mara Dougherty, Matthew Miller, Artesha Taylor, Breanne Robertson, Alexis Williams, and Spencer Benson, "Student Expectations from CS and other STEM Courses: They aren't Like CS-Majors! or ($CS \neq STEM - CS$)", *Appears* in the Journal of Computing Sciences in Colleges, Volume 28, Issue 6, Pages 100 – 108, Consortium for Computing Sciences in Colleges , USA, June 2013. [ACM digital library, Paper PDF]

20. Karl Schmitt, Elise Larsen, Matthew Miller, Abdel-Hameed A. Badawy, Breanne Robertson, Mara Dougherty, Katie Marie Hrapczynski, Andrea Andrew, Artesha Taylor, Alexis Williams, Sabrina Kramer, and Spencer Benson, "Building a Tool for pre-assessing Student Expectations", *To Appear* to the Journal of Microbiology & Biology Education (JMBE) (In Press).

21. Abdel-Hameed A. Badawy, Karl B. Schmitt, Sabrina Kramer, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Mara Dougherty, Matthew Miller, Artesha Taylor, Breanne Robertson, Alexis Williams, and Spencer Benson, "Expectations of Computing and other STEM Students: A Comparison for different Class Levels, or (CSE $\neq$ STEM - CSE)$|_{Course\ Level}$", *To Appear* at

the 43rd Annual IEEE Frontiers in Education (FIE) Conference to be held in Oklahoma City, Oklahoma, October 2013 (In Press).

Under Revision

1. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, “On Using Gray Level Co-occurrence Matrix Features and Tissue Density for Normal Mammogram Detection”, **invited paper** *in review* to the International Journal of Computational Bioscience, International Association of Science and Technology for Development (IASTED), ACTA Press.

Under Submission

1. Abdel-Hameed A. Badawy, Meng-Ju Wu, and Donald Yeung, “Profile-guided Optimization Selection via Reuse Distance Coherent Profiles”, *in submission* to the 2013 International Conference on Parallel & Distributed Processing Techniques & Applications (PDPTA’13).

2. Abdel-Hameed A. Badawy, Meng-Ju Wu, and Donald Yeung, “Sharing and Locality Optimizations on Tiled-CMPs”, *in submission* to TBD’2013.

3. Abdel-Hameed A. Badawy, Karl Schmitt, Breanne Robertson, Mara Dougherty, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Matthew Miller, Artesha Taylor, Alexis Williams, Sabrina Kramer, and Spencer Benson, “The Design and Analysis of a Tool for Assessing Student Expectations in STEM Courses”, *in submission* to International Journal of College Science Teaching.

4. Abdel-Hameed A. Badawy, Karl B. Schmitt, Sabrina Kramer, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Mara Dougherty, Matthew Miller, Artesha Taylor, Breanne Robertson, Alexis Williams, and Spencer Benson, “Comparing Student Expectations from Computer Science/Engineering and other STEM Courses: $(\text{CS} \neq \text{STEM - CS})|_{Course\ Level}$”, *in submission* to the Journal of Teaching and Learning, Georgia Southern University, Statesboro, GA.

5. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, ”Tissue Type based Pre-CAD Normal Mammogram Detection”, invited paper *in submission* to the American Journal of Biomedical Engineering, Scientific & Academic Publishing.

Presentations, Talks, & Posters

1. Abdel-Hameed A. Badawy, and Donald Yeung, “Evaluating the Impact of Memory System Performance on Software Prefetching and Locality Optimizations”, ***talk*** presented at Research Review Day, University of Maryland, College Park, February 2001.

2. Abdel-Hameed A. Badawy, "Evaluating the Impact of Memory System Performance on Software Prefetching and Locality Optimizations", ***talk*** presented at the ACM's International Conference on Supercomputing (ICS'01), Sorrento, Italy, June 2001.

3. Abdel-Hameed A. Badawy, and Donald Yeung, "Studying the effects of Software Prefetching and Locality Optimizations on System Energy and Power", ***poster*** presented at Research Review Day, University of Maryland, College Park, March 2003.

4. Abdel-Hameed A. Badawy, "Prefetch-aware Memory-controllers", ***poster*** presented at the Graduate Research Interaction Day (GRID'2004), April 2004. [**Received the Award of Excellence**].

5. Meng-Ju Wu, Abdel-Hameed A. Badawy (Presenter), Inseok Choi, Xu Yang, and Donald Yeung, "Scalability of Multicore Processors", ***poster*** presented at the ECE Research Review Day, October 2009.

6. Abdel-Hameed A. Badawy, and Michelle Hugue, "Determining the Effectiveness of Discussion Board Interactivity", ***talk*** presented at the Innovations in Teaching and Learning Conference (ITL'2010), University of Maryland, College Park, April 2010.

7. Abdel-Hameed A. Badawy, "Lessons from the Blackboard's Automatic Usage Statistics", ***talk*** presented at the 10th Annual Lilly-TC, Conference on College and University Teaching and Learning, Traverse City, MI, September 2010.

8. Abdel-Hameed A. Badawy, "Students' Perceptions of the Effectiveness of Discussion Boards", ***poster*** presented at the 10th Annual Lilly-TC, Conference on College and University Teaching and Learning, Traverse City, MI, September 2010.

9. Abdel-Hameed A. Badawy, "ELMS, the Facebook of the Classroom: Using ELMS and its Discussion Boards as an Environment for Active and Collaborative Learning", ***talk*** presented as a training workshop at the Electrical and Computer Engineering Department as part of the Graduate Teaching Assistant Training and Development Program, College Park, MD, October 2010.

10. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, "One-Class Support Vector Machines vs. Two-class Support Vector Machines for Normal Mammogram Detection", ***poster*** presented at the IEEE Workshop on Applied Imagery Pattern Recognition (AIPR' 2010), Washington, DC, October 2010. [**Best Student Paper Award**].

11. Abdel-Hameed A. Badawy, "Discussion Boards on Blackboard as a Collaborative Learning Tool: A Survey and Students' Reflections", ***talk*** accepted for presentation at the 30th Annual International Lilly Conference on College Teaching, Oxford, OH, November 2010.

12. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, "Effect of Breast Density in Selecting Features for Normal Mammogram Detection", ***poster*** presented at the 8th IEEE International Symposium on Biomedical Imaging (ISBI'2011), Chicago, IL, April 2011.

13. Abdel-Hameed A. Badawy, "What can we get from our students for a freebie point?", ***poster*** presented at the Annual Lilly-DC Conference on College and University Teaching and Learning, Washington, DC, June 2011.

14. Mona Y. Elshinawy, Abdel-Hameed A. Badawy, Wael W. Abdelmageed, and Mohamed F. Chouikha, "Screening-out Normal Mammograms using Breast Density Information", ***poster*** presented at the 8th Era of Hope Workshop, Department of Defense (DoD) Breast Cancer Research Program (BCRP), Orlando, FL, August 2011.

15. Abdel-Hameed A. Badawy, "Discussion Boards on Blackboard as a Collaborative Learning Tool: A Survey and Students' Reflections: What would the students do for a freebie point?", ***talk*** to be presented at the 31st Annual International Lilly Conference on College Teaching, Oxford, OH, November 2011.

16. Abdel-Hameed A. Badawy, "Blackboard: Online Learning for the Facebook Generation", ***talk*** presented as a training workshop at the Electrical and Computer Engineering Department as part of the Graduate Teaching Assistant Training and Development Program, College Park, MD, December 2011.

17. Abdel-Hameed Badawy, Karl Schmitt, Breanne Robertson, Mara Dougherty, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Matthew Miller, Artesha Taylor, Alexis Williams, Sabrina Kramer, and Spencer Benson, "Building a Tool for pre-assessing Student Expectations", ***poster*** presented at the Graduate Student Government, Graduate Research Interaction Day (GRID' 2012), April 2012. [**Best Student Poster Award**]

18. Abdel-Hameed Badawy, Karl Schmitt, Breanne Robertson, Mara Dougherty, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Matthew Miller, Artesha Taylor, Alexis Williams, Sabrina Kramer, and Spencer Benson, "From Seed to STEM: Cultivating Understanding of Student and Faculty Classroom Expectations", ***talk*** to be presented at the Lilly Fellows Annual Meeting, Center for Teaching Excellence, University of Maryland, College Park, MD, April 2012.

19. Abdel-Hameed Badawy, Karl Schmitt, Breanne Robertson, Mara Dougherty, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Matthew Miller, Artesha Taylor, Alexis Williams, Sabrina Kramer, and Spencer Benson, "What Do The Students Expect? A Tool for Analyzing Student Expectations", ***poster*** to be presented at the Innovations in Teaching and Learning (ITL) Conference , University of Maryland, College Park, MD, April 2012.

20. Abdel-Hameed Badawy, Karl Schmitt, Breanne Robertson, Mara Dougherty, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Matthew Walker Miller, Artesha Taylor, Alexis Williams, Sabrina Kramer, and Spencer Benson, “What Do The Students Expect? A Tool for Analyzing Student Expectations”, ***talk*** to be presented at the the Innovations in Teaching and Learning (ITL) Conference, University of Maryland, College Park, MD, April 2012.

21. Abdel-Hameed Badawy, Katie Hrapczynski, Mara Dougherty, Karl Schmitt, Breanne Robertson, Artesha Taylor, Alexis Williams, Sabrina Kramer, and Spencer Benson, “What Do They Expect? Assessment of Student Expectations in the Classroom and Applications for Faculty”, ***talk*** presented at the Annual Lilly-DC Conference on College and University Teaching and Learning, Washington, DC, June 2012.

22. Abdel-Hameed Badawy, and Karl Schmitt, “Student Expectations”, ***talk*** presented at the University of Maryland New Graduate Teaching Assistants Orientation, College Park, MD, August 2012.

23. Karl B. Schmitt, Abdel-Hameed A. Badawy, Sabrina Kramer, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Mara Dougherty, Matthew Miller, Artesha Taylor, Breanne Robertson, Alexis Williams, and Spencer Benson, “Student Expectations from CS and other STEM Courses: They aren’t Like CS-Majors! or $(CS \neq STEM-CS)$”, ***talk*** to be presented at the Consortium of Computer Science in Colleges, North East Conference, (CCSCNE’2013), Albany, NY, April 2013. [Presentation Slides]

24. Abdel-Hameed A. Badawy, Karl B. Schmitt, Sabrina Kramer, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Mara Dougherty, Matthew Miller, Artesha Taylor, Breanne Robertson, Alexis Williams, and Spencer Benson, “On the Expectations of Computer Science and other STEM Students”, ***Poster session talk presented*** at the Annual Innovation in Teaching and Learning (ITL) Conference, University of Maryland, College Park, April 2013. [Abstract, Presentation Slides]

25. Abdel-Hameed A. Badawy, Elise Larsen, Karl B. Schmitt, Sabrina Kramer, Katie Marie Hrapczynski, Andrea Andrew, Mara Dougherty, Matthew Miller, Artesha Taylor, Breanne Robertson, Alexis Williams, and Spencer Benson, “The Student-Faculty Chasm: Looking at where student and faculty expectations meet and change”, ***talk*** presented at the Annual Lilly-DC Conference on College and University Teaching and Learning, Washington, DC, June 2013. Presentation Slides]

26. Abdel-Hameed A. Badawy, Karl B. Schmitt, Sabrina Kramer, Katie Marie Hrapczynski, Elise Larsen, Andrea Andrew, Mara Dougherty, Matthew Miller, Artesha Taylor, Breanne Robertson, Alexis Williams, and Spencer Benson, “Expectations of Computing and other STEM Students: A Comparison for

different Class Levels, or (CSE $\neq$ STEM - CSE)$|_{Course\ Level}$", ***talk*** to be presented at the 43rd Annual IEEE Frontiers in Education (FIE) Conference to be held in Oklahoma City, Oklahoma, October 2013.

Teaching & Learning Professional Development

1. Participant in Tomorrow's Professor Blog.
2. Participant at the Lilly-East Conference held at the University of Delaware, April $16 - 18, 2008$.
3. Participant at the Lilly-East Conference held at the University of Delaware, April $15 - 17, 2009$.
4. Enrolled in a seminar style course about active learning "BIOL608C: Active Learning" during spring 2009. [**Grade: A**].
5. Enrolled in "UNIV798A: Special Topics Colloquium on University Teaching and Learning: Introduction to University Teaching" spring 2010. [**Grade: A**].
6. Presenter at the Innovations in Teaching and Learning Conference (ITL'2010) held at the UMD, College Park, MD, April, 2010.
7. Participant at the Annual Lilly-DC Conference held in Washington, DC, June $3 - 5, 2010$.
8. Presenter at the 10th Annual Lilly-Traverse City Conference held in Traverse City, MI, September 2010.
9. Member of the TLT Group's Online Institute since June 2010.
10. Member of the TLT-SWG (Improving teaching and learning by using available technology) since June 2010.
11. Regular participant in TLT Group's FridayLive! Online teaching and learning professional development seminars since June 2010.
12. Led a training seminar for TAs at the Electrical and Computer Engineering Department in October 2010 as part of being a TA Training and Development (TATD) Fellow.
13. Mentor for one new graduate teaching assistant as part of being a TA Training and Development Fellow, $2010 - 2011$ academic year.
14. Mentor for three new graduate teaching assistants as part of being a TA Training and Development Fellow, for the $2011 - 2012$ academic year.

15. Led a training seminar for TAs at the Electrical and Computer Engineering Department in October 2011 as part of being a TA Training and Development (TATD) Fellow.

16. Presenter at the Annual Lilly-DC Conference held Washington, DC, June $3-5, 2011$.

17. Presenter at the 31st Annual International Lilly Conference on College Teaching held at Miami University, Oxford, OH, November $17-20, 2011$.

18. Presenter at the Innovations in Teaching and Learning Conference (ITL'2012) held at the University of Maryland, College Park, MD, April, 2012.

19. Presenter and Poster Session Judge at the Annual Lilly-DC Conference held Washington, DC from June $1-3, 2012$.

20. Participant in the CIRTL Network activities at UMD including the CIRTL Coffee Hour Series.

21. Presenter at the Innovations in Teaching and Learning Conference (ITL'2013) held at the University of Maryland, College Park, MD, April, 2013.

22. Presenter at the Annual Lilly-DC Conference held Washington, DC from June $1-3, 2013$.

23. Completed the Center for Teaching Excellence (CTE) at UMD, University Teaching and Learning Program (UTLP), May 2013.

Professional Activities & Service

- Associate editor for the Journal of Learning, August 2010.
- Technical Reviewer for the Society of Imaging Informatics in Medicine (SIIM) Journal of Digital Imaging (JDI) since November 2010.
- Technical Reviewer for the 3rd International Conference on Machine Learning and Computing (ICMLC'2011) held in Singapore, February 2011.
- Technical Committee Member for the 3rd International Conference on Mechanical and Electrical Technology (ICMET'2011) held in China, August 2011.
- Technical Reviewer for the International Journal of Computer and Electrical Engineering (IJCEE), October 2011.
- Technical Review Board member for the International Journal of Advanced Computer Science and Applications (IJACSA), March 2012.
- Technical Program Committee member for the 4th International Conference on Software Technology and Engineering (ICSTE'2012), Phuket, Thailand, September 2012.

- Session Convener, the 6th Annual Innovation in Teaching and Learning (ITL) Conference, University of Maryland, College Park, MD, April 2012.
- Poster Session Judge, Annual Lilly-DC Conference on College & University Teaching, Bethesda, MD, June 2012.
- Technical Reviewer for technical papers and posters for the Special Interest Group Computer Science Education (SIGCSE) 2013 conference, Denver, CO.
- Core Committee member for the Science and Information (SAI) Conference 2013, London, UK.
- Technical Program Committee member for the Science and Information (SAI) Conference 2013, London, UK.
- Technical Reviewer for the International Journal of New Computer Architectures and their Applications (IJNCAA), October 2012.
- Peer book reviewer for books published by Common Ground Publishing, November 2012.
- Technical Review for the following Consortium of Computer Science in Colleges (CCSC) regional conferences:
  - Technical Reviewer for the South-Central Region Conference (CCSC-SC) 2013 conference, held at University of Louisiana, Shreveport, LU.
  - Technical Reviewer for the South-West Region Conference (CCSC-SW) 2013 conference, held at University of California, San Marcos, CA.
  - Technical Reviewer for the North-East Region Conference (CCSC-NE) 2013 conference, held at Sienna College, Albany, NY.
- Technical Reviewer for NED University Journal of Research – Applied Sciences, December 2012.
- Technical Program Committee member for the 5th International Conference on Software Technology and Engineering (ICSTE'2013), September 2013, Bandar Seri Begawan, Brunei Darussalam, organized by IACSIT and Lecture Notes on Software Engineering (LNSE).
- Technical Program Committee member for the prestigious IEEE 43rd Annual IEEE Frontiers in Education Conference (FIE'13), October 2013, Oklahoma City, OK, USA.

Memberships

- IEEE, IEEE Computer Society, ACM since 1998.

- Served as the Graduate Student member of the Undergraduate Affairs Committee, 2001 – 2004.
  The committee's task was to work on re-accreditation requirements of ABET for the Electrical and Computer Engineering Department, University of Maryland, College Park, MD.

- Member, International Association of Computer Science and Information Technology (IACSIT), June 2010.

- Member, Science and Engineering Institute (SCIEI), November 2011.

- Member, Society of Digital Information and Wireless Communications (SDIWC), October 2012.