

UNIVERSITY OF CALIFORNIA, SAN DIEGO

**Performance Prediction and Ordering of Supercomputers using a
Linear Combination of Benchmark Measurements**

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Omid Khalili

Committee in charge:

Professor Allan Snavely, Chair
Professor Scott Baden, Co-Chair
Professor Charles Elkan

2007

Copyright

Omid Khalili, 2007

All rights reserved.

The thesis of Omid Khalili is approved:

Co-Chair

Chair

University of California, San Diego

2007

TABLE OF CONTENTS

Signature Page	iii
Table of Contents	iv
List of Figures	vi
List of Tables	vii
Acknowledgements	x
Abstract of the Thesis	xi
Chapter 1 Introduction	1
Chapter 2 Previous and Related Work	4
2.1 Simple Metrics Vs. Combination of Metrics	4
2.2 Ranking of Supercomputers	5
2.3 Neural Networks for Performance Modeling	6
2.4 More Related Work	8
2.5 Extensions to Previous Work	9
Chapter 3 Least Squares Regression with Basis Reduction	11
3.1 Least Squares Regression	11
3.2 Basis Reduction	12
3.3 Experimental Methodology	13
Chapter 4 HPC Challenge Benchmarks	16
Chapter 5 HPC Challenge and TopCrunch	19
5.1 TopCrunch Applications	20
5.2 Reduced Metrics	20
5.3 Results	21
Chapter 6 HPC Challenge and TI-06 Applications	27
6.1 Applications	28
6.2 Reduced Metrics	29
6.3 Results	30
Chapter 7 A Different Basis Set for TI-06 Applications	32
7.1 Benchmarks	32
7.2 Reduced Metrics	33
7.3 Results	34
7.4 MultiMaps	36

Chapter 8	HPC Challenge and FFT and DGEMM	41
8.1	Reduced Metrics	41
8.2	Results	42
8.3	Using Optimized HPCC Results for FFT	44
8.3.1	Reduced Metrics	47
8.3.2	Results	48
Chapter 9	Limitations	50
Chapter 10	Application of the Thesis	52
10.1	Future Work	53
Chapter 11	Conclusion	56
Appendix A	TopCrunch System Information	59
Appendix B	TI-06 System Information	86
Bibliography	106

LIST OF FIGURES

Figure 2.1: Normalized runtimes for a subset of TI-06 applications run on their respective machines.	7
Figure 5.1: 3 car collision residual plots for Random Access, and Random Ring Bandwidth on 32 processors.	26
Figure 7.1: Maps bandwidth measurements for an IBM p690 node across growing array sizes.	33
Figure 7.2: Avus residual plots for L2-strided and main memory strided access on 128 processors.	39
Figure 7.3: Avus residual plots for L2-random access and network bandwidth on 128 processors.	40
Figure 8.1: FFT residual plots for HPL and PTRANS.	45
Figure 8.2: FFT residual plots for Stream Triad and Random Ring Bandwidth.	46

LIST OF TABLES

Table 5.1: Average absolute relative error for TopCrunch applications using FLOPS, the full HPCC suite, and the reduced HPCC suite.	22
Table 5.2: Average number of threshold inversions for TopCrunch applications using FLOPS, the full HPCC suite, and the reduced HPCC suite. ($\alpha = .01$ and $\beta = .001$)	23
Table 5.3: The effects of varying α and β on the average number of threshold inversions for the TopCrunch applications using the HPC Challenge benchmarks.	23
Table 5.4: Linear regression coefficients and p-values for neon_refined on 64 processors using the full HPC Challenge set of benchmarks.	25
Table 6.1: Average absolute relative error for TI-06 applications using FLOPS, the full HPCC suite, and the reduced HPCC suite.	31
Table 6.2: Average number of threshold inversions for TI-06 applications using FLOPS, the full HPCC suite, and the reduced HPCC suite. ($\alpha = .01$ and $\beta = .001$)	31
Table 6.3: The effects of varying α and β on the average number of threshold inversions for the TI-06 applications using the HPC Challenge benchmarks.	31
Table 7.1: Average absolute relative error for TI-06 applications using FLOPS, the full HPCC suite, the reduced HPCC suite, the full Maps+Netbench set and the reduced Maps+Netbench set.	37
Table 7.2: Average number of threshold inversions for TI-06 applications using FLOPS, the full HPCC suite, the reduced HPCC suite, the full Maps+Netbench set and the reduced Maps+Netbench set. ($\alpha = .01$ and $\beta = .001$)	38
Table 7.3: The effects of varying α and β on the average number of threshold inversions for the TI-06 applications using the Maps and Netbench benchmarks, averaged over all applications.	38
Table 7.4: Linear regression coefficients and p-values for Avus run on 12 machines on 128 processors using the full Maps and Netbench set of benchmarks.	38
Table 8.1: Average absolute relative errors (AARE) and average number of threshold inversions for FFT and DGEMM, including standard deviations (stdev). When counting threshold inversions, $\alpha = .01$ and $\beta = .001$	43
Table 8.2: Number of threshold inversions for FFT and DGEMM when varying α and β	43

Table 8.3: Linear regression coefficients and p-values for neon_refined on 64 processors using the full HPC Challenge set of benchmarks. . . .	47
Table 8.4: Average absolute relative errors (AARE) and average number of threshold inversions for FFT, including standard deviations (stdev), using the optimized HPC Challenge benchmark results. When counting threshold inversions, $\alpha = .01$ and $\beta = .001$	48
Table 8.5: Number of threshold inversions for FFT using the Optimized HPC Challenge results when varying α and β	49
Table 10.1: Correlations for TI-06 systems based on their Maps and Net-bench measurements.	54
Table 10.2: System information for those listed in Table 10.1.	54
Table 10.3: Example of threshold inversions versus max inversion distance.	55
Table A.1: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 22).	60
Table A.2: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 22).	61
Table A.3: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 22).	62
Table A.4: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 22).	63
Table A.5: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 21 and 22 of 22).	64
Table A.6: 3 Car Collisions on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 20).	65
Table A.7: 3 Car Collisions on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 20).	66
Table A.8: 3 Car Collisions on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 20).	67
Table A.9: 3 Car Collisions on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 20).	68
Table A.10: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 33).	69
Table A.11: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 33).	70
Table A.12: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 33).	71
Table A.13: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 33).	72
Table A.14: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 21-25 of 33).	73

Table A.15: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 26-30 of 33).	74
Table A.16: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 31-33 of 33).	75
Table A.17: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 26).	76
Table A.18: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 26).	77
Table A.19: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 26).	78
Table A.20: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 26).	79
Table A.21: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 21-25 of 26).	80
Table A.22: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (System 26 of 26).	80
Table A.23: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 23).	81
Table A.24: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 23).	82
Table A.25: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 23).	83
Table A.26: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 23).	84
Table A.27: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 21-23 of 23).	85
Table B.1: TI-06 system information and matched HPCC Challenge systems.	86
Table B.2: TI-06 Maps benchmark measurements for strided access.	89
Table B.3: TI-06 Maps benchmark measurements for random access.	90
Table B.4: TI-06 Netbench network measurements.	92
Table B.5: AVUS runtimes on different processor counts on TI-06 systems.	93
Table B.6: CTH runtimes on different processor counts on TI-06 systems.	95
Table B.7: GAMESS runtimes on different processor counts on TI-06 systems.	96
Table B.8: HYCOM runtimes on different processor counts on TI-06 systems.	98
Table B.9: LAMMPS runtimes on different processor counts on TI-06 systems.	99
Table B.10: OOCORE runtimes on different processor counts on TI-06 systems.	101

Table B.11: OVERFLOW runtimes on different processor counts on TI-06 systems.	102
Table B.12: WRF runtimes on different processor counts on TI-06 systems.	104

ACKNOWLEDGEMENTS

I owe many thanks to Allan Snaveley for taking me in as a Masters student, and advising me these last few years. Laura Carrington, Mustafa M. Tikir and Tzu-Yi Chen have been very helpful in understanding the TI-06 data set and guiding my initial approach. My committee members, Scott Baden and Charles Elkan deserve thanks for their watchful eyes over my research. I am very grateful of Claudia Padula's support throughout my college years. My parents and brother deserve the most thanks for all of their support my whole life.

This work was supported in part by a grant from the National Science Foundation entitled "The Cyberinfrastructure Evaluation Center" and by NSF grant #CCF-0446604. This work was sponsored in part by the Department of Energy Office of Science through SciDAC2 award "PERI: The Performance Engineering Research Institute".

ABSTRACT OF THE THESIS

**Performance Prediction and Ordering of Supercomputers using a
Linear Combination of Benchmark Measurements**

by

Omid Khalili

Master of Science in Computer Science

University of California, San Diego, 2007

Professor Allan Snavely, Chair

Professor Scott Baden, Co-Chair

This thesis presents a performance prediction model and method of ordering supercomputer performance using a linear combination of benchmark measurements. Benchmark measurements for different resources—for example floating point operations per second (FLOPS), memory bandwidth, and network measurements—are collected on a set of machines and reduced to a set of non-correlated measurements. A least squares regression is used to obtain the weights for the reduced set mea-

measurements that minimize the squared error estimate for an application's runtime on the machines. The performance model uses these weights to make predictions for an application's performance on new systems, given their benchmark measurements. This is extended to make performance predictions for many systems and to order them based on the predicted performance.

For all ten applications tested, using a linear combination of benchmark metrics gives better predictions for the application runtimes, compared to FLOPS alone. The average of the absolute values of the relative errors, over all applications, of 72.5%, using only FLOPS, was reduced to 39.5%, using a combination of metrics. In addition, the ordering of supercomputers is better, in terms of threshold inversions, using this methodology. An inversion occurs when an ordering says one machine is faster than the other, but actual measurements suggest the opposite; thresholds are used to account for the variability in measurements. Out of a maximum of 10 threshold inversions when ordering 5 machines, there were an average, over all applications, of 4.05, using only FLOPS, compared to 3.05, using a combination of metrics.

Chapter 1

Introduction

When users look for the best system for their application, many first look to the Top500 ranking of supercomputers. The Top500 uses the Linpack benchmark [26] and measures the peak floating point operations per second (FLOPS) of the system. It is assumed that the system that performs the best on the Linpack benchmark will also have the best performance for all applications. However, as noted in the Linpack description, “This performance does not reflect the overall performance of a given system, as no single number ever can. It does, however, reflect the performance of a dedicated system for solving a dense system of linear equations.” [27].

The performance of today’s systems are no longer bottlenecked by the CPU, and are instead limited by the memory hierarchy and interconnect. A better approach to ranking supercomputers might consider benchmark measurements for all resources in the system, for example peak system FLOPS, cache and memory bandwidths, network bandwidth and latency, and disk I/O bandwidth. A combination of these measurements could give users a more accurate representation of

how the systems they can chose from will perform.

Nevertheless, a single measurement, or combination of measurements, may not be the best metric of ranking for all applications. Some applications are compute heavy, whereas others are memory intensive. The combination of measurements should be done with respect to the application in question so that, for example, compute heavy applications give more weight to the FLOPS of the system, and memory intensive ones give more weight to the cache and memory bandwidths.

This thesis explores a method that tries to consider all aspects of a supercomputer by using a linear combination of the benchmark measurements for different resources. For a given application, a least squares regression is used to get weights for different benchmark measurements that minimizes the squared error for predicting the application's runtime on a set machines. These weights are then used to make a prediction for the application's runtime on a new set of machines. Furthermore, the predicted runtimes for the new machines are used to order the machines. For all the applications tested, using a linear combination of benchmark measurements, compared to the peak system FLOPS, not only provides more accurate predictions for the runtime of the application, but also gives rankings that more closely resemble the true ranking using the measured application runtimes.

The utility of this method is demonstrated using a pair of applications from the TopCrunch benchmarks—mechanical and aerospace applications that simulate car crashes [5]—as well as eight applications from the Technical Insertion 2006 (TI-06) [6] ranging from chemical simulations, turbulence analysis, a weather forecasting

model, and more. All applications were analyzed using the High Performance Computing Challenge (HPC Challenge [2]) suite of benchmarks. The HPC Challenge benchmarks measure FLOPS, memory bandwidths and network bandwidth and latency using various kernels that mimic application behavior. In addition, the TI-06 applications were analyzed using a separate set of benchmarks that consisted of more detailed measurements of the memory cache hierarchy as well as network bandwidth and latency.

The next chapter describes related work and describes *threshold inversions*, a metric for comparing different system orderings. Chapter 3 reviews the least square regression, describes the method for reducing a full set of benchmarks and describes the experimental methodology used in the rest of the thesis. Chapter 4 gives a more detailed overview of the nine HPC Challenge benchmarks followed by the analysis for two TopCrunch applications (Chapter 5) and eight Technical Insertion 2006 (TI-06) applications (Chapter 6). In Chapter 7 the analysis for the TI-06 applications is revisited using a different benchmark set consisting of cache hierarchy and network measurements. In Chapter 8 the analysis is repeated for two of the HPC Challenge benchmarks that resemble full applications (FFT and DGEMM, a parallel matrix multiply) using the rest of the HPC Challenge benchmarks as the full set. Chapter 9 describes some limitations this methodology may face. Finally, Chapter 10 briefly describes the Java tool written to make performance predictions and order machines using this research and describes some future work to make the tool more useful.

Chapter 2

Previous and Related Work

2.1 Simple Metrics Vs. Combination of Metrics

Carrington *et al.* investigated the predictive power of a single metric versus a combination of metrics [11] for predicting an application's performance. They found that when the performance prediction model used a single metric, performance was always worse than when it used a combination of metrics. The best single metric, memory random access giga-updates per second (GUPS), had a relative error of 33%. The predictive model using a combination of system FLOPS, Maps memory bandwidths, Netbench network measurements, as well as loop and control-flow dependencies proved to have the best predictive power with a relative error of 18%.

The predictive model required tracing the application to measure operation counts, memory references, and network communications. Tracing the application to acquire this information would cause a slowdown of up to 30x, compared to running it without tracing; a time consuming process for applications that can run

between 1-4 hours. The traced operation counts along with measured rates for the processor, memory bandwidth, network bandwidth and latency were combined to predict the runtime of the application.

2.2 Ranking of Supercomputers

Chen *et al.* explored the ranking of supercomputers using a combination of metrics, as in [11], and introduced a metric, *threshold inversions*, for comparing rankings of supercomputers using different methods [12]. They found that in addition to predicting application runtimes more accurately, a combination of metrics is better at ranking supercomputers than a single metric.

An inversion occurs when a ranking says machine A is better than B, but measured runtimes of the application shows the opposite. However, there is usually some variance in measured runtimes of applications, as well as in benchmarks measurements; therefore, *threshold inversions* are used to account for such variances. The α parameter is used to account for variances in application runtimes, and β is used for benchmark measurements ($0 \leq \alpha, \beta \leq 1$). An $\alpha = .01$ means that a difference of 1% in the application runtimes is insignificant. For example, let the predicted runtimes of A and B be \hat{RT}_A and \hat{RT}_B and the measured runtimes be RT_A and RT_B . If the predicted runtimes $\hat{RT}_A < \hat{RT}_B$, then A would be ranked better than B, but, if the measured runtimes $RT_A > RT_B$, then there is an inversion. Yet, when using threshold inversions with α , that would only count as

an inversion if $RT_A > (1 + \alpha) \times RT_B$. β is used in a similar fashion to allow for variance in benchmark measurements, and is usually set to be less than α .

In addition to introducing threshold inversions, Chen showed that for different applications, different machines would give the best performance. This is exemplified in Figure 2.1 using a subset of the applications described in this thesis. Figure 2.1 shows the runtimes for a subset of the Technical Insertion 2006 (TI-06) [6] applications, normalized by each application’s longest runtime (note that not all applications were run on all machines, so some normalized runtimes are 0). The figure shows that across all applications, no single machine is the slowest or the fastest; therefore, it is unlikely that any single metric is a strong predictor of the application’s performance.

2.3 Neural Networks for Performance Modeling

Using performance samples of an application running on the target machine, Ipek *et al.* trained a multi-layer neural network to make performance predictions for the application using different parameter spaces [18]. Performance samples of the application, SMG2000, are taken to cover its input parameter space, six parameters describing processor workload and topology, and number in the thousands. Noise and variance in the samples, due to operating system overhead for example, were cleaned up using Stratification and Bagging. The trained multi-layer neural network served as their performance model and resulted in relative errors of 5-7%

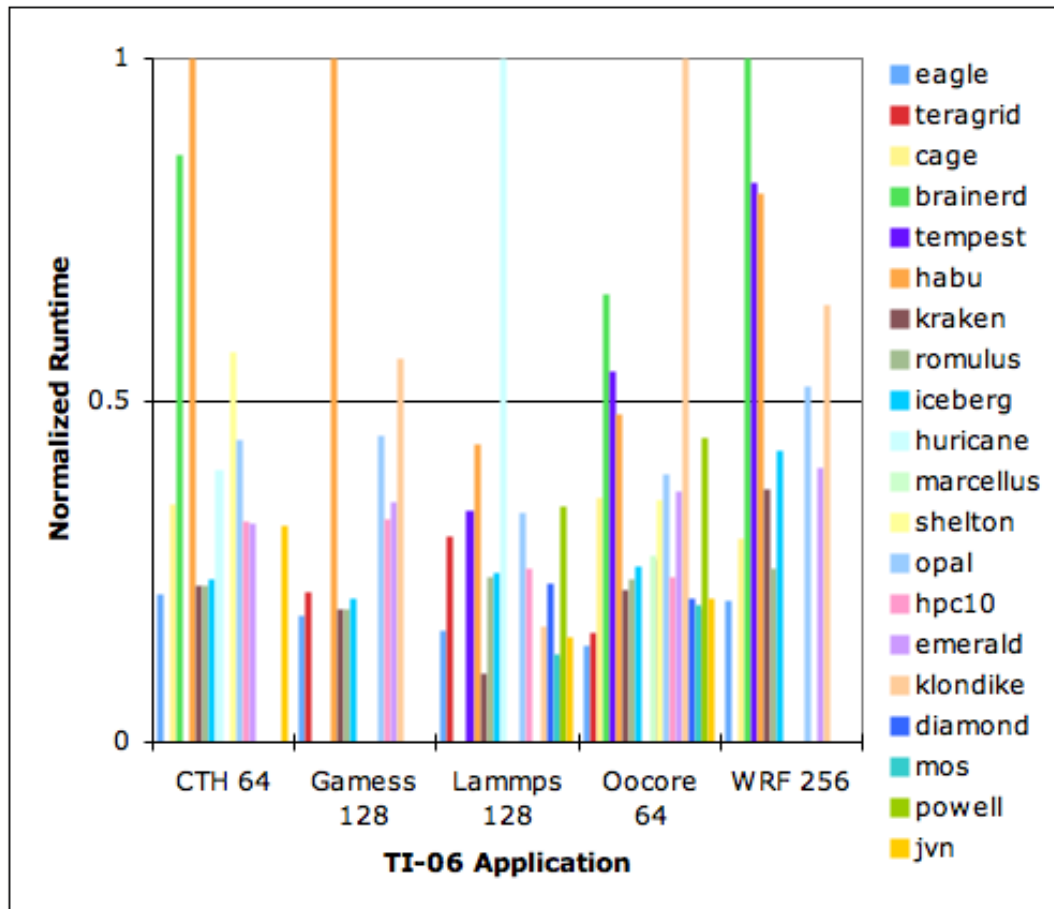


Figure 2.1: Normalized runtimes for a subset of TI-06 applications run on their respective machines.

on a randomly selected test set, using thousands of training inputs. This methodology can find nonlinear patterns in the training input in order to make accurate performance predictions; however, may be difficult to use to make predictions on a new system.

2.4 More Related Work

Using a full run of an application on a reference system, along with partial application runtimes on the reference and a target system, Yang could predict the full application performance using the relative performance of the short runs with an accuracy of 97% or higher [28]; however, as they mentioned, such an approach could miss computation behavior that changes over the runtime of the application, and accuracy was reduced when using the partial runtime to predict the application using a different problem size or number of processors. One way to remedy this is to use cycle accurate simulation of the application [23, 9, 10, 22, 25, 20]; however because of the level of detail in the simulations, it could take 1,000,000 times as long to simulate the application than run it. Other methods create a model of the application from compiler information [7, 14], or from instrumenting it at a higher level [13, 16] in order to predict performance. These methods can give accurate performance predictions, without the huge slowdown of cycle accurate simulations.

The HPC Challenge [2], NAS Parallel [8] and the SPEC [4] suite of benchmarks are used to measure performance of HPC systems. These suites consist of different

application-like kernels, but are more difficult to use as a performance predictor than a single metric. McCalpin [21] and Gustafson [17] showed the application-like kernels correlate and relate to applications, but did not extend the work to parallel and large scale applications. Different approaches have been taken to combine such measurements to make performance predictions and then order them [12] (as described in section 2.2) or to order systems based on a normalization of benchmark measurements [15]. One drawback of the IDC methodology [15] is that weights for measurements are not given with respect to the application in question, and may not generalize well for different applications.

2.5 Extensions to Previous Work

The goal of the thesis is to try provide a higher level method of combining benchmark measurements for system resources to give accurate performance predictions and orderings of systems. It adds to the previous work by attempting to remove the need for tracing an application, which is a time consuming process, in order to model an application. It may be faster and easier to run a portable application on many different systems, at the same time, than to trace it on one system. The question then becomes how well a predictive model using a linear regression on the runtimes of the application across system can predict the runtime for a new machine. In addition, how such a model would serve as a method for ordering a set of machines is investigated.

Other works [11, 24] looked at using a linear regression but did not consider requiring the basis set of measurements to be orthogonal across the test machines. This thesis shows that using a combination of a reduced set of metrics typically provides better performance than the full set of metrics not only in predicting the application runtime but also in ordering a set of systems. Furthermore, a combination of metrics regularly outperforms any single metric, particularly FLOPS, in prediction and ordering performance.

The initial approach in this thesis was to use a multi-layered neural network. Inputs to the network consisted of the benchmarks taken on a system, and the target value for training was an application’s runtime on the system. A single machine was used as the test set to get performance predictions, or a set of machines were used as the test set in order to order the machines based on predicted performance. Performance of the network depended on finding a machine with similar benchmark measurements to include in the validation set, to avoid over-generalization, and wasn’t always the case when the validation set was randomly selected. Unfortunately, the lack of many training inputs—the number of machines with benchmark measurements and application runtimes were no more than 20—made it difficult to find nonlinear patterns between the benchmark measurements and application runtimes; relative errors of performance prediction and the number of threshold inversions were high. For such cases, it is therefore better to focus on using a linear combination of the measurements.

Chapter 3

Least Squares Regression with Basis Reduction

3.1 Least Squares Regression

A least squares regression is used to obtain the weights for a set of benchmark measurements that minimize the squared error estimate for an application's runtime on a set machines. Benchmarks for a set of machines are given in a matrix M consisting of benchmark measurements for different resources, in columns, on different machines, in rows. For example, M in Equation 1 shows a matrix containing benchmark measurements for b resources on n machines. The b resources make up the *full basis set*. The application runtimes are given in a vector \mathbf{P} . This vector has application runtimes for the n machines in M run using the same number of processors on each of the machines.

Getting the weights for the benchmark measurements is equivalent to solving for \mathbf{w} in $M \times \mathbf{w} = \mathbf{P}$ (Equation 1). It is assumed that $n > b$ and therefore least squares regression is used to find the \mathbf{w} that minimizes the 2-norm of the residual.

Algorithm 1 Use least squares regression to solve for the benchmark weights, \mathbf{w} , given benchmarks on machines, in M , and application runtimes on machines, in \mathbf{P} .

$$M \times \mathbf{w} = \begin{pmatrix} m_{1,1} & \dots & m_{1,b} \\ m_{2,1} & \dots & m_{2,b} \\ m_{3,1} & \dots & m_{3,b} \\ \cdot & & \cdot \\ \cdot & \dots & \cdot \\ \cdot & & \cdot \\ m_{n,1} & \dots & m_{n,b} \end{pmatrix} \times \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \dots \\ w_b \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \cdot \\ \cdot \\ \cdot \\ p_n \end{pmatrix} = \mathbf{P} \quad (3.1)$$

The Java tool written to run the analysis uses the Colt Library [1] to compute the least squares regression.

After \mathbf{w} is obtained, it can be used to make predictions for a new set of machines, M_{new} with benchmark measurements for the b resources. Therefore, the runtime predictions for the application on the new set of machines is $\hat{\mathbf{P}}_{new} = M_{new} \times \mathbf{w}$. Finally, the new machines are ordered by sorting their respective application runtimes in $\hat{\mathbf{P}}_{new}$.

3.2 Basis Reduction

A problem that arises from using full basis set of benchmark measurements is that they may not be orthogonal for a given set of machines. If two or more benchmarks have highly correlated measurements in M , then M stores redundant information.

For example, consider a simple case: predicting a person's shoe size given their

weight and height. Weight and height are highly correlated, so which is a better predictor of the shoe size? A linear regression can give the persons height a higher weight, and thus their weight a lower weight, or the opposite. It would be better to use only one of the predictors, eliminating this correlation.

Similarly, redundant information from M could be removed. Basis reduction, therefore, first looks at the correlations of all pairs of measurements on all machines in M . Then, for highly correlated pairs of predictors where the correlation coefficient is greater than .8, or less than -.8, one of the predictors is dropped. This removes the correlation in the least squares regression predictors and should help generalize better for new machines not in M .

3.3 Experimental Methodology

Both experiments in this thesis present cross-validation results. M and \mathbf{P} are constructed for a given application. All measurements must have the same unit of measurement; thus the inverse of any latency measurements is taken. Next, the inverse of all measurements are taken, and each column is normalized by its maximum measurement. This not only puts all measurements in comparable units but also on a comparable scale.

When the measurements are in comparable units, the correlation matrix of the benchmark measurements, M 's columns, is constructed. For pairs of predictors that are highly correlated, one of the predictors is dropped. M is reduced to M_r ,

where M_r has the same number of rows as M , but fewer columns.

The first experiment uses cross-validation to test the prediction model. One at a time, each machine is removed from M_r and its application runtime is removed from \mathbf{P} , and placed in $M_{r_{val}}$ and \mathbf{P}_{val} . The remaining machines are stored in $M_{r_{train}}$ and \mathbf{P}_{train} . Least squares regression is used to get the weights, \mathbf{w}_{train} for how the reduced set of benchmarks affect the application’s runtime in the training set. These weights are then used to make a prediction for the runtime on the validation machine, $\hat{\mathbf{P}}_{val} = M_{r_{val}} \times \mathbf{w}_{train}$ and the relative error of the prediction is calculated as $RelErr = \frac{\hat{\mathbf{P}}_{val} - \mathbf{P}_{val}}{\mathbf{P}_{val}}$ where \mathbf{P}_{val} is the measured runtime on the validation machine. This is repeated for each machine and the average absolute relative error is reported over all the machines. The average absolute relative error is the average of the absolute values of the relative errors.

The second experiment uses cross-validation to test how well this method orders a set of machines. A set of v machines are randomly selected and moved from M_r to $M_{r_{val}}$ along with their runtimes from \mathbf{P} to \mathbf{P}_{val} . The remaining machines are stored in $M_{r_{train}}$ and \mathbf{P}_{train} . The number of randomly chosen machines, v , is picked so that there are enough machines in $M_{r_{train}}$ to compute the least squares regression. The weights, \mathbf{w}_{train} , are computed as before and used to predict the runtimes for the validation machines, $\hat{\mathbf{P}}_{val} = M_{r_{val}} \times \mathbf{w}_{train}$. Finally, the number of threshold inversions [12] between the sorted predicted runtimes, \hat{P}_{val} , and sorted true runtimes, P_{val} , are calculated. This is repeated 5000 times, each time choosing a random set of v machines to order and count threshold inversions for, to get a

strong mix of randomly selected validation machines and the average number of threshold inversions is reported.

Chapter 4

HPC Challenge Benchmarks

The HPC Challenge is a suite of nine benchmarks aimed at measuring different aspects of today's supercomputers [2]. In addition to the High Performance Linpack (HPL), the HPC Challenge benchmarks includes kernels with different memory and network access patterns that more closely mimic HPC applications. Users of systems can download the HPC Challenge benchmarks, found at [2], and run them on their systems. The program that runs the benchmarks verifies the results, which are then submitted and displayed on the results page, [3]. Below is a description of the nine benchmarks, from [3]:

HPL Solves a randomly generated dense linear system of equations in double floating-point precision (IEEE 64-bit) arithmetic using MPI. The linear system matrix is stored in a two-dimensional block-cyclic fashion and multiple variants of code are provided for computational kernels and communication patterns. The solution method is LU factorization through Gaussian elimination with partial row pivoting followed by a backward substitution.

PTRANS Implements a parallel matrix transpose for two-dimensional block-cyclic storage. It is an important benchmark because it exercises the communications of the computer heavily on a realistic problem where pairs of processors communicate with each other simultaneously. It is a useful test of the total communications capacity of the network.

Random Access Global RandomAccess, also called GUPs, measures the rate at

which the computer can update pseudo-random locations of its memory - this rate is expressed in billions (giga) of updates per second (GUP/s).

FFT Global FFT performs the same test as FFT but across the entire system by distributing the input vector in block fashion across all the nodes.

Stream Triad The Embarrassingly Parallel STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth and the corresponding computation rate for simple numerical vector kernels. It is run in embarrassingly parallel manner - all computational nodes perform the benchmark at the same time, the arithmetic average rate is reported.

Stream Sys This benchmark gives to system sustainable bandwidth, given by $STREAMTriad \times MPIProcesses$.

DGEMM The Embarrassingly Parallel benchmark measures the floating-point execution rate of double precision real matrix-matrix multiply performed by the DGEMM subroutine from the BLAS (Basic Linear Algebra Subprograms). It is run in embarrassingly parallel manner - all computational nodes perform the benchmark at the same time, the arithmetic average rate is reported.

Random Ring Bandwidth Randomly Ordered Ring Bandwidth, reports bandwidth achieved in the ring communication pattern. The communicating nodes are ordered randomly in the ring (with respect to the natural ordering of the MPI default communicator). The result is averaged over various random assignments of processes in the ring.

Random Ring Latency Randomly-Ordered Ring Latency, reports latency in the ring communication pattern. The communicating nodes are ordered randomly in the ring (with respect to the natural ordering of the MPI default communicator) in the ring. The result is averaged over various random assignments of processes in the ring.

Results for the benchmarks were downloaded from the results page [3]. In general, during basis reduction, it was found that PTRANS (a parallel matrix transpose), and FFT were highly correlated. This was expected as FFT does a matrix transpose. In addition, both of the Stream benchmarks (Triad and Sys) were highly correlated; Stream Sys is a linear function of Stream Triad and the

correlation was expected. Similarly, network bandwidth and latency, tend to be highly correlated, as the latency is a function of the bandwidth and number of bytes sent. Depending on the set of machines a particular application was run on, what was reduced during basis reduction varied slightly; the details are described in their respective sections.

Chapter 5

HPC Challenge and TopCrunch

TopCrunch [5] is an HPC benchmark that uses real software engineering applications, instead of synthetic benchmarks, to acquire performance measurements. The benchmark includes different applications, two of which are described below, and has been run on many HPC systems. The systems that the applications were run on were matched, with regards to the processor, interconnect, and number of CPUS, as closely as possible to systems benchmarked using the HPC Challenge. HPC Challenge benchmarks for the matched systems were used to predict the runtime, and order systems, for two of the TopCrunch applications.

Similar to the HPC Challenge benchmarks, users download the TopCrunch benchmarks to run on their systems, then upload the results to the TopCrunch site. Refer to Appendix A for system information as well as application runtimes on their respective processors. Appendix A also shows which TopCrunch systems were matched to which HPC Challenge systems. Note that some TopCrunch systems were matched to the same HPC Challenge system; without doing so, there would not have been enough systems to run a least squares regression and

order more than 2 systems.

5.1 TopCrunch Applications

The TopCrunch benchmarks are taken from the mechanical and aerospace communities and reflects computations for structural dynamics (LS-DYNA), fluid flow (CTH) and materials science (SPaSM). The benchmarks address the challenging aspects of scaling including message passing, load balancing and dynamic memory allocations suitable for today’s supercomputers. The two benchmarks analyzed are based on the LS-DYNA code.

3 Car Crash This benchmark simulates a “van crashing into the rear of a compact car, which, in turn, crashes into a midsize car”. The runtimes for this benchmark on 32 and 64 processors for all previous years were downloaded and used in the analysis.

neon.refined This benchmark simulates a frontal crash with an initial speed of 31.5 mph based on vehicle crash analysis of the 1996 Plymouth Neon. The runtimes of the benchmark on 8, 32 and 64 processors for all previous years were downloaded and used.

5.2 Reduced Metrics

The applications were run on different sets of systems, with a small overlap; thus basis reduction was done separately for each application. On the machines 3 Car Crash was run on, HPL, Random Access, PTRANS, and FFT were highly correlated, and thus on Random Access was kept. In addition, Stream Triad and Stream Sys were highly correlated and only Stream Triad was kept. Finally,

the network bandwidth and network latency were highly correlated, and only the bandwidth was kept. The reduced basis set for 3 Car Crash is:

< RandomAccess, StreamTriad, DGEMM, RandomRingBandwidth >

On the machines neon_refined was run on the correlations were similar to those seen for 3 Car Crash, except HPL was not correlated with any other benchmark. Therefore, the reduced basis set for neon_refined is:

< HPL, RandomAccess, StreamTriad, DGEMM, RandomRingBandwidth >

In both cases, different combination of which measurements to remove were briefly investigated and produced small changes in the final results. In the end, the measurements chosen to be removed were those that were initially picked.

5.3 Results

Results for the prediction and ordering performance are presented in Tables 5.1 and 5.2 (bold number show the best performance and orderings). For all of the applications, the predicted performance was better when using a reduced basis set, compared to FLOPS alone. For all of the applications, except for neon_refined on 32 processors, the number of threshold inversions in ordering the predicted runtimes is less than the ordering using FLOPS alone (with $((\alpha, \beta) = (.01, .001))$). Overall, results show that using a reduced basis set of benchmarks, versus just FLOPS, not only predicts the application's performance more accurately, but also

gives an ordering of systems that more closely resembles the true ordering based on measured runtimes.

The middle columns of Tables 5.1 and 5.2 show that using the full basis set can produce poor performance predictions. However, even though the performance predictions results are better with the reduced basis set, the full basis set is still able to provide an ordering that is much closer to the true ordering. Therefore, using a combination of metrics gives better orderings than using FLOPS alone.

Table 5.3 shows the effects of the average number of threshold inversions as α and β grow, averaged over all applications. When using FLOPS, the full basis set and the reduced basis set (reduced with respect to each application), the average number of threshold inversions always decrease as α and β increase. In all cases, however, using a combination of measurements provides more accurate results than just FLOPS alone.

Table 5.1: Average absolute relative error for TopCrunch applications using FLOPS, the full HPCC suite, and the reduced HPCC suite.

	FLOPS	Full HPCC Suite	Reduced HPCC Suite
neon_refined-8	72.43	28.69	34.96
neon_refined-32	80.61	52.13	44.59
neon_refined-64	81.18	300+	65.08
3 Car Collision-32	89.86	300+	38.45
3 Car Collision-64	71.15	237.49	28.01
Average	79.05	183.66+	42.22

The cases where the absolute relative error for the application is 300%+ deserve more examination; the case for 3 car collision on 32 processors is examined in

Table 5.2: Average number of threshold inversions for TopCrunch applications using FLOPS, the full HPCC suite, and the reduced HPCC suite. ($\alpha = .01$ and $\beta = .001$)

	FLOPS	Full HPCC Suite	Reduced HPCC Suite
neon_refined-8	5.47	2.29	4.44
neon_refined-32	4.14	4.81	4.83
neon_refined-64	5.33	3.52	4.47
3 Car Collision-32	4.95	3.13	4.03
3 Car Collision-64	5.42	3.92	4.15
Average	5.06	3.54	4.39

Table 5.3: The effects of varying α and β on the average number of threshold inversions for the TopCrunch applications using the HPC Challenge benchmarks.

	$(\alpha, \beta) =$			
	$(.01, .001)$	$(.1, .01)$	$(.2, .02)$	$(.5, .05)$
FLOPS	5.06	4.35	3.93	2.76
Full Basis Set	3.54	3.09	2.78	2.22
Reduced Basis Set	4.39	4.01	3.44	2.34

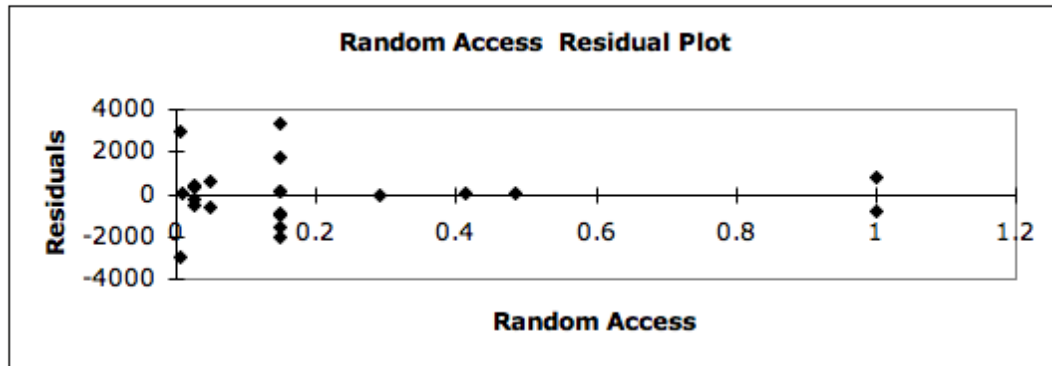
more detail. Table 5.4 shows the linear regression coefficient and their p-values for this case using the full HPC Challenge set of benchmarks over all machines. P-values for PTRANS, Random Access, StreamSys, DGEMM, and Random Ring Latency are $< .05$; thus with 95% confidence, they are not due to random chance and are statistically significant. Figure 5.1 shows the residual plots for Random Access, and Random Ring Bandwidth. Not all residual plots are presented here, but others show similar patterns. The decreasing spread in the x suggests that the variance is not constant, most likely due to the fact that some systems were matched to the same HPC Challenge system and used the same benchmarks. The figure also shows that some influential points exist (such as the single point on the right side of the random ring bandwidth plot). When removing influential points, the regression coefficients are likely to change. For example, for the random ring bandwidth, there is only one point that is considerably large, and is an influential point for that system; when that system is removed from the data set, and the new coefficients are used to make a prediction for its runtime, the relative error is huge: $1.08 \times 10^{17}\%$. Only a few such cases like this exist, and they skew the average absolute relative error over all machines.

If the predictors with insignificant coefficients were removed, results may improve. After removing the predictors with insignificant coefficients the average absolute relative error dropped to 27%, from the previous best of 33% with the reduced HPC Challenge set. The average number of threshold inversions were 3.8, compared to the previous best of 3.22 using the full set of HPC Challenge bench-

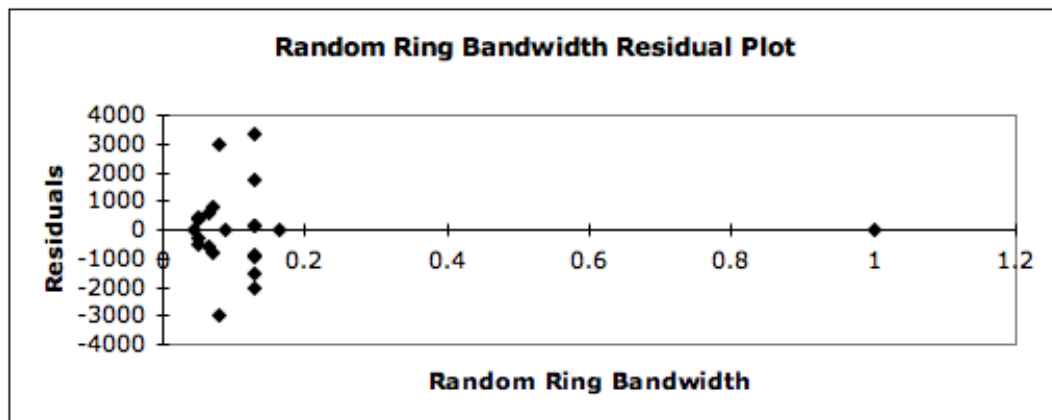
marks. Interestingly enough, even though using the full HPC Challenge set gives a really high average absolute relative error, due to four machines with ridiculous relative errors on their performance predictions, it still does a better job with the relative ordering of systems. Removing predictors with insignificant coefficients only helped improve performance predictions.

Table 5.4: Linear regression coefficients and p-values for neon_refined on 64 processors using the full HPC Challenge set of benchmarks.

Benchmark	Coefficient	p-values
HPL	-93243.02093	0.08668995
PTRANS	17352.88777	0.008263595
RandomAccess	-43148.39381	0.006826258
FFT	7579.175326	0.847472886
StreamSys	104232.7533	0.02105699
StreamTriad	-10295.28251	0.457212807
DGEMM	24842.74408	0.024944794
RandomRingBW	12554.09282	0.657025152
RandomRingLat	-14630.57972	0.046846147



(a)



(b)

Figure 5.1: 3 car collision residual plots for Random Access, and Random Ring Bandwidth on 32 processors.

Chapter 6

HPC Challenge and TI-06 Applications

The Department of Defense’s Technical Insertion 2006 (TI-06) [6] includes a set of eight applications run on various HPC systems on various processor counts. Each application was run by a user on their system using the DoD ”Standard” input and the final application runtime was collected. The inputs used for each application were the same for runs on all systems.

Next, the systems the TI-06 applications were run on were matched to systems benchmarked using the HPC Challenge, similar to how the TopCrunch systems were matched. Appendix B shows the system information and which TI-06 systems were matched to which HPC Challenge systems. Note that some TI-06 systems were matched to the same HPC Challenge system; without doing so, there would not have been enough systems to run a least squares regression and order more than 2 systems. Unfortunately, some of the matched systems were not benchmarked using the full HPCC suite; the missing benchmarks (Random Access, FFT and DGEMM) were dropped from the analysis. The remaining HPC

Challenge benchmarks served as the basis set and were used to predict the runtime, and order systems, for each of the TI-06 applications.

6.1 Applications

The eight TI-06 applications are described in more detail below. There were a total of 20 machines in the set. Although not all applications were run on all machines, each application was typically run on a set of about 15 machines with various processor counts from 16 to 384. Refer to Appendix B for the application runtimes on their respective systems and processors.

AVUS Developed by the Air Force Research Laboratory, AVUS is used to determine the fluid flow and turbulence of projectiles and air vehicles. The parameters used calculates 100 time-steps of fluid flow and turbulence for a wing, flap, and end plates using 7 million cells.

CTH CTH measures effects of multi-material, large deformation, strong shock wake, solid mechanics and was developed by the Sandia national Laboratories. CTH models multi-phase, elastic viscoplastic, porous and explosive materials on 3D and 2D rectangular grids, as well as 1D rectilinear, cylindrical, and spherical meshes.

GAMESS Developed by the Gordon research group at Iowa State University, computes *ab initio* molecular quantum chemistry.

HYCOM Models all of the world's oceans as one global body of water at a resolution of one-fourth of a degree measured at the Equator. It was developed by the Naval Research Laboratory, Los Alamos National Laboratory and the University of Miami.

LAMMPS LAMMPS, developed by the Sandia National Laboratories, is generally used as a parallel particle simulator for particles at the mesoscale or continuum levels.

OOCORE OOCORE is an out-of-core matrix solver developed by the SCALAPACK group at The University of Tennessee at KnoKnoxville. OOCORE has been included in past benchmark suites and is typically I/O bound.

OVERFLOW NASA Langley and NASA Ames developed OVERFLOW to solve CFD equations on a set of overlapped, adaptive grids, so that the resolution near an obstacle is higher than other portions of the scene. With this approach, computations of both laminar and turbulent fluid flows over geometrically complex non-stationary boundaries can be solved.

WRF WRF is a weather forecasting model. It uses multiple dynamical cores and a 3D variational data assimilation system with the ability to scale to many processors. WRF was developed by a partnership between the National Center for Atmospheric Research, the National Oceanic and Atmospheric Administration, the Air Force Weather Agency, the Naval Research Laboratory, Oklahoma University, and the Federal Aviation Administration.

6.2 Reduced Metrics

The TI-06 applications were run on the same set of 20 systems, although not all applications were run on all systems. Thus, it was simpler to do basis reduction over all systems, as opposed to once for each application’s set of systems. Half of the 10 machines, when matched to similar machines benchmarked by the HPC Challenge, did not have the full Challenge run on them; thus, the “full” basis set consisted of:

$\langle HPL, PTRANS, StreamTriad, StreamSys, RandRingBW, RandRingLat \rangle$

The correlation matrix showed that only Random Ring Bandwidth and Latency were highly correlated. Removing one or the other produced small changes in the final results, and thus Random Ring Latency was removed, as it was the first one picked. The reduced basis set consisted of:

$\langle HPL, PTRANS, StreamSys, StreamTriad, RandomRingBandwidth \rangle$

6.3 Results

The results in Tables 6.1 show that the performance prediction results using the reduced basis set are more accurate than the full set, although not as accurate as when FLOPS was used alone. In addition, FLOPS provides better orderings (Table 6.2) than when using a combination of metrics (with $\alpha = .01$ and $\beta = .001$). Table 6.3 shows that as α and β are increased, the average number of threshold inversions, averaged over all TI-06 applications, decrease. This was expected as some inversions are ignored when more variance between measurements is being allowed. Even when using larger α s and β s, FLOPS outperforms a combination of metrics. This suggests that either the missing HPC Challenge benchmarks are important, the HPC Challenge suite does not represent the TI-06 applications, or FLOPS is the best predictor for the TI-06 Applications. On the other hand, a different story is seen in Chapter 7, when a different set of benchmarks is used.

Although there are some cases where the average absolute relative error is fairly large, their regression analysis are not investigated in further detail as the full HPC Challenge suite of benchmarks was not able to be used.

Table 6.1: Average absolute relative error for TI-06 applications using FLOPS, the full HPCC suite, and the reduced HPCC suite.

	FLOPS	Full HPCC Suite	Reduced HPCC Suite
avus	73.9	200.93	203.62
cth	61.24	92.23	40.68
gamess	72.62	170.91	73.89
hycomm	67.56	226.7	129.72
lammgs	58.40	74.15	80.47
oocore	62.9	103.83	66.85
overflow	74.84	68.0	62.14
wrf	58.22	96.611	78.05
Average	66.21	129.18	91.92

Table 6.2: Average number of threshold inversions for TI-06 applications using FLOPS, the full HPCC suite, and the reduced HPCC suite. ($\alpha = .01$ and $\beta = .001$)

	FLOPS	Full HPCC Suite	Reduced HPCC Suite
cth	2.94	3.61	3.61
lammgs	2.91	3.75	3.48
oocore	3.11	3.33	2.75
overflow	3.31	3.06	2.88
wrf	3	4.86	4.23
Average	3.05	3.72	3.39

Table 6.3: The effects of varying α and β on the average number of threshold inversions for the TI-06 applications using the HPC Challenge benchmarks.

	$(\alpha, \beta) =$			
	(.01, .001)	(.1, .01)	(.2, .02)	(.5, .05)
FLOPS	3.05	2.47	2.2	1.69
Full Basis Set	3.72	3.27	3.02	2.33
Reduced Basis Set	3.39	3.15	2.93	2.36

Chapter 7

A Different Basis Set for TI-06 Applications

The systems the TI-06 applications were run on have also been benchmarked using Membench and Netbench. As the bottleneck for today's applications is memory bandwidth, a basis set consisting solely of memory and network benchmarks is compared to that of the HPC Challenge. Unfortunately some of the HPC Challenge benchmarks were missing for the machines of the TI-06 dataset, so the comparison is not complete.

7.1 Benchmarks

The Maps benchmark in Membench was used to measure bandwidths to L1, L2, L3 and main memory for both strided access and random access; Netbench was used to measure network bandwidth and latency. Maps measures strided and random access bandwidth to all levels of the cache hierarchy, compared to STREAM and Random Access, which measure bandwidth only to main memory.

Maps and Netbench were run by users on their systems. Maps measures bandwidths were for different array sizes for both strided and random access to the array. An example of such measurements for strided access are shown in Figure 7.1 for an IBM p690 node. As the array size grows, it cannot fit into the smaller faster caches and bandwidths drop show a plateau for each cache region. The regions for L1, L2 L3 caches and main memory are identified for each machine and the measurements are averaged to obtain a single metric for each region. The final set of benchmarks on all TI-06 systems are presented in Appendix B.

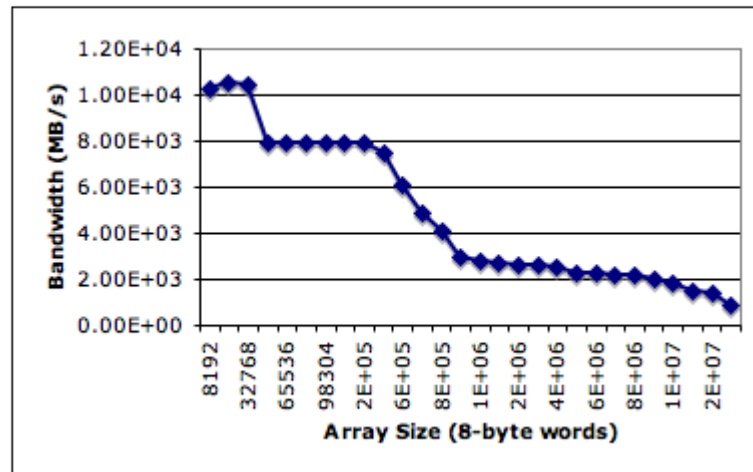


Figure 7.1: Maps bandwidth measurements for an IBM p690 node across growing array sizes.

7.2 Reduced Metrics

As in section 6.2, basis reduction was run on all machines, instead of for each application's set of machines. The full basis set consisted of:

$$\langle L1_{stride}, L2_{stride}, L3_{stride}, MainMem_{stride}, \\ L1_{rand}, L2_{rand}, L3_{rand}, MainMem_{rand}, NetBW, NetLat \rangle$$

Measurements for strided access to L1 and L2 caches were highly correlated, and only L1-strided bandwidths were kept. In addition, both strided access to L3 and main memory along with random access to L3 and main memory were highly correlated. This is because not many of the system had an L3 cache, and as a result the average bandwidth for L3 cache was close to that of main memory for both strided and random access; thus, only main memory measurements were kept. Network bandwidth and latency were not correlated for this set of machines. As before, different combinations of which measurements to reduced were explored, resulting in minor differences in the results; those measurements removed and explored further were the initially chosen ones. The reduced basis set consisted of:

$$\langle L1_{stride}, MainMem_{stride}, L1_{rand}, L2_{rand}, MainMem_{rand}, NetBW, NetLat \rangle$$

7.3 Results

As shown in Tables 7.1 and 7.2, using a reduced basis set consisting mostly of memory metrics and some network measurements provides both better runtime predictions and better system orderings. There is a 44% improvement in both the performance predictions and average number of threshold inversions compared to FLOPS alone. These results give strong support to adding more detailed memory measurements to the HPC Challenge. Moreover, unlike the analysis for TopCrunch

and the TI-06 applications that used the HPC Challenge benchmarks as the basis set, the using the reduced basis set provides both more accurate performance predictions and more accurate system orderings. Nonetheless, with or without the reducing the full basis set, the system orderings are always more accurate than using the full and reduced HPCC basis sets. Table 7.3 further illustrates this: as values for α and β increase, the number of threshold inversions drop for all sets of measurements, yet the reduced basis set always provides the fewest amount. Although FLOPS is a better predictor of system orderings, compared a combination of the HPC Challenge benchmarks, a combination of memory benchmarks outperforms FLOPS.

Avus is the only application with a large average absolute relative error when using the full Maps and Netbench basis set, and its regression is explored further. Avus was run on at most 12 system using different processor counts from 32 to 256 and only the set of runtimes on 128 processors (the maximum processor count where Avus was run on 12 systems) is considered. Table 7.4 shows the linear regression coefficients and their p-values for the full set of 12 machines. Only L2 random access has a p-value $< .05$; all other coefficient are statistically insignificant. Figures 7.2 and 7.3 show the residual plots along L2-strided, main memory strided, L2-random, and network bandwidth. These figures show some decrease in spread, although not as evident as with the TopCrunch 3 car collision residuals, suggesting a non constant variance, and some possible influential points also exist. Other residual plots for Avus on 128 processors show similar behavior and are not

shown, but this analysis was not repeated for Avus on other processor counts. In any case, there are only 2 more measurements than there are independent variables, which is probably not enough to draw any strong conclusions.

The fact that some of the predictors' coefficients are insignificant suggests that reducing them may provide better results. The measurements for L2-random access are kept, and the cross-validation experiments are run again; the average absolute relative error improved to 39%, from the previous best of 49% using the reduced Maps and Netbench set. Over all processor counts that Avus was run on, the average absolute relative error was 33%, compared to the previous best of 43% using the reduced Maps and Netbench set. The average number of inversions over all processor counts using just L2-random access measurements was 1.9, but this test could not be repeated using other basis sets as there were not enough machines to remove 5 and still be able to do the linear regression.

7.4 MultiMaps

The Stream and Random Access benchmarks in the HPC Challenge measure bandwidth to main memory, but not the caches. Seeing as how threshold inversions are less when using a more detailed memory benchmark, the HPC Challenge suite could benefit from including such a benchmark.

MultiMaps extends membench by testing many different stride sizes, for many different array sizes. This causes the benchmark to run for a long time, and needs

to be shortened before being included in the HPC Challenge. Measurements for strides 2, 4 and 8 were highly correlated on a test set of five machines, so only the stride-4 test is included in the HPCC version of MultiMaps. In addition, the number of iterations for each array size is dynamically throttled down based on a few quick tests before the main benchmark is run. The benchmark now runs in 6-12 minutes, compared to 10-30 minutes, and is an acceptable time for it to be included in the HPC Challenge. Furthermore, the results are almost exactly the same as the non-throttled version of MultiMaps, and thus the predictive power will not change.

Table 7.1: Average absolute relative error for TI-06 applications using FLOPS, the full HPCC suite, the reduced HPCC suite, the full Maps+Netbench set and the reduced Maps+Netbench set.

	FLOPS	Full HPCC Suite	Reduced HPCC Suite	Full Maps+ Netbench Set	Red. Maps+ Netbench Set
avus	73.9	200.93	203.62	213.4	43.54
cth	61.24	92.23	40.68	64.84	36.02
gamess	72.62	170.91	73.89	–	54.96
hycomm	67.56	129.72	35.71	70.8	60.4
lammgs	58.4	74.15	80.47	54.27	32.35
oocore	62.9	103.83	66.85	33.27	24.03
overflow	74.84	68.0	62.14	23.25	27.88
wrf	58.22	96.61	78.05	45.44	17.41
Average	66.214	129.18	91.92	72.41	37.07

Table 7.2: Average number of threshold inversions for TI-06 applications using FLOPS, the full HPCC suite, the reduced HPCC suite, the full Maps+Netbench set and the reduced Maps+Netbench set. ($\alpha = .01$ and $\beta = .001$)

	FLOPS	Full HPCC Suite	Reduced HPCC Suite	Full Maps+Netbench Set	Red. Maps+Netbench Set
cth	2.94	3.61	3.61	–	3.23
lammps	2.91	3.75	3.48	3.63	3.12
oocore	3.11	3.33	2.75	3.14	2.43
overflow	3.31	3.06	2.88	2.22	2.11
wrf	3	4.86	4.23	3.63	1.91
Average	3.05	3.72	3.39	3.16	2.56

Table 7.3: The effects of varying α and β on the average number of threshold inversions for the TI-06 applications using the Maps and Netbench benchmarks, averaged over all applications.

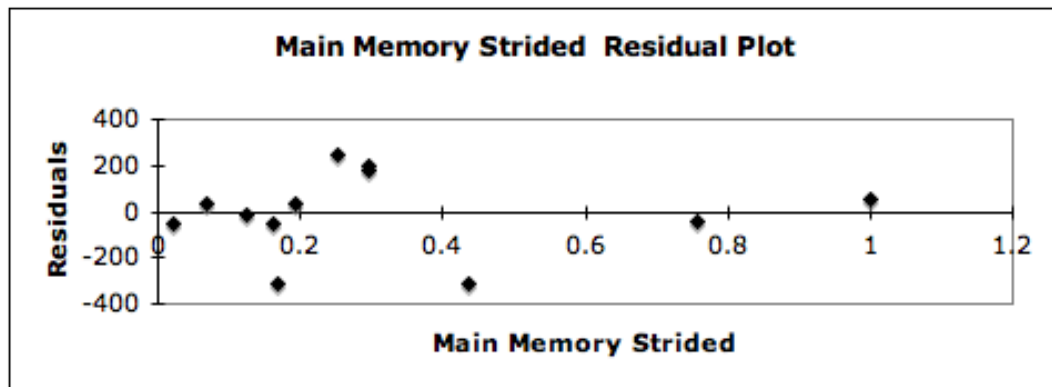
	$(\alpha, \beta) =$			
	(.01, .001)	(.1, .01)	(.2, .02)	(.5, .05)
FLOPS	3.05	2.47	2.2	1.69
Full Basis Set	3.16	2.93	2.81	2.29
Reduced Basis Set	2.56	2.3	2.08	1.65

Table 7.4: Linear regression coefficients and p-values for Avus run on 12 machines on 128 processors using the full Maps and Netbench set of benchmarks.

Benchmark	Coefficient	p-values
L1S	4066.975348	0.563613274
L2S	17081.0812	0.201369372
L3S	-10375.17224	0.27326202
MainMemS	8673.568115	0.078753478
L1R	10590.96679	0.207770254
L2R	-22858.84335	0.017804815
L3R	47250.90852	0.190804182
MainMemR	-9928.709421	0.199353407
NetBW	-6779.979694	0.183109667
NetLat	-2205.009471	0.782457834

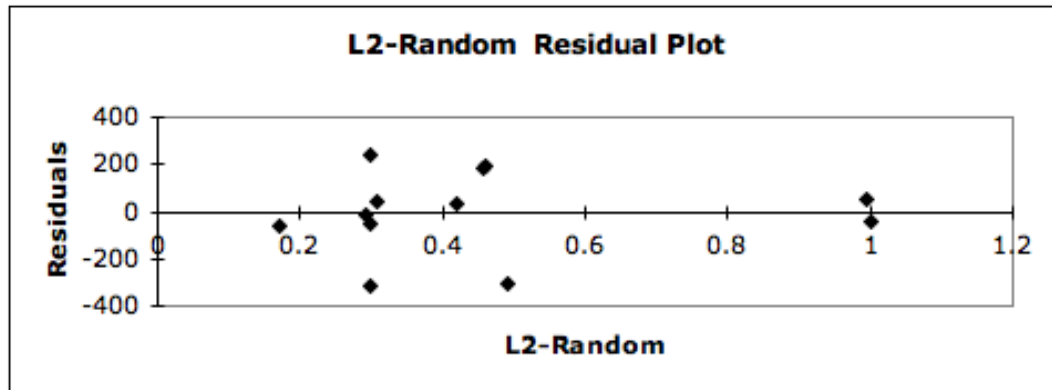


(a)

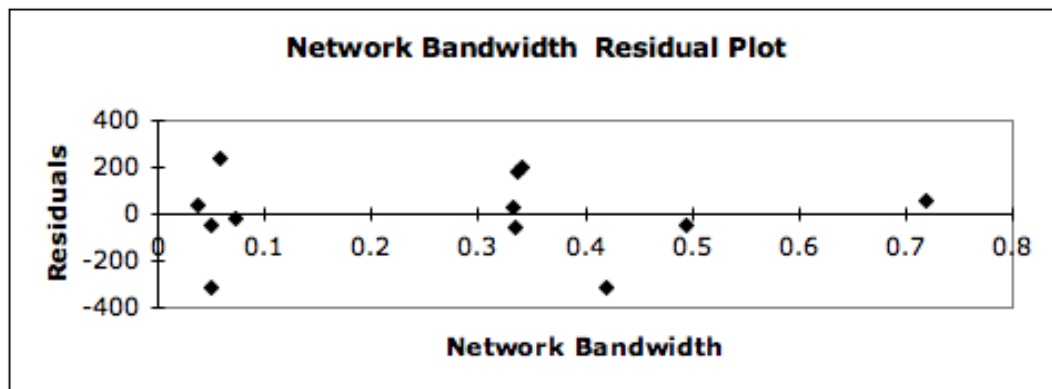


(b)

Figure 7.2: Avus residual plots for L2-strided and main memory strided access on 128 processors.



(a)



(b)

Figure 7.3: Avus residual plots for L2-random access and network bandwidth on 128 processors.

Chapter 8

HPC Challenge and FFT and DGEMM

The FFT and DGEMM (a parallel matrix multiply) benchmarks included in the HPC Challenge can be considered as applications on their own. This chapter describes two experiments that predict runtimes and order machines for FFT and DGEMM on their own. For each experiment the full basis set consists of the nine HPC Challenge benchmarks except for the “application” in question. The basis set of eight benchmarks was reduced and the cross validation was carried out as described in section 3.3.

8.1 Reduced Metrics

Results for the 90 machines that had the full HPC Challenge suite run on them were downloaded from [3]. Please refer to [3] for these measurements as they are too long to list here. The full basis set consisted of the nine HPC Challenge benchmarks, except either FFT or DGEMM for their respective tests. For both

sets there were no correlations greater than .8 or less than $-.8$ over the set of 90 machines; thus the full basis set was used in each case.

8.2 Results

Table 8.1 shows the performance prediction and ordering results for FFT and DGEMM. For both benchmarks, the system FLOPS is relatively a better predictor of runtime, yet it is a poor predictor for ranking the systems.

In the ordering cross-validation tests, 45 of the 90 machines are randomly chosen to be ordered (for a total of 990 possible inversions), while the other 45 serve as the training set to learn the benchmark weights. Although the linear combination of the metrics is a poor predictor of performance, it is much more useful for ordering the systems than FLOPS. For FFT there is a 65% drop in the number of threshold inversions and a 15% drop of threshold inversions for DGEMM. Table 8.2 shows that as α and β are increased, the average number of threshold inversions decrease. In addition, using a combination of metrics, for both FFT and DGEMM, provides more accurate orderings of the systems.

The average absolute relative error for FFT is huge, and its regression analysis is further investigated. There are many more data points than independent variables, which should help. Table 8.3 shows the linear regression coefficients and their p-values. The table shows that PTRANS, Random Ring Bandwidth and Random Ring Latency have p-values < 0.5 and are statistically significant. Figures 8.1 and

Table 8.1: Average absolute relative errors (AARE) and average number of threshold inversions for FFT and DGEMM, including standard deviations (stdev). When counting threshold inversions, $\alpha = .01$ and $\beta = .001$.

	FFT	
	AARE (stdev)	Ave. Inversions (stdev)
Basis Set	2942.16 (17171.57)	295.75 (55.38)
FLOPS	153.46 (303.57)	852.24 (18.23)
	DGEMM	
	AARE (stdev)	Ave. Inversions (stdev)
Basis Set	179.15 (307.318)	433.6 (69.66)
FLOPS	99.54 (97.64)	504.8 (37.27)

Table 8.2: Number of threshold inversions for FFT and DGEMM when varying α and β .

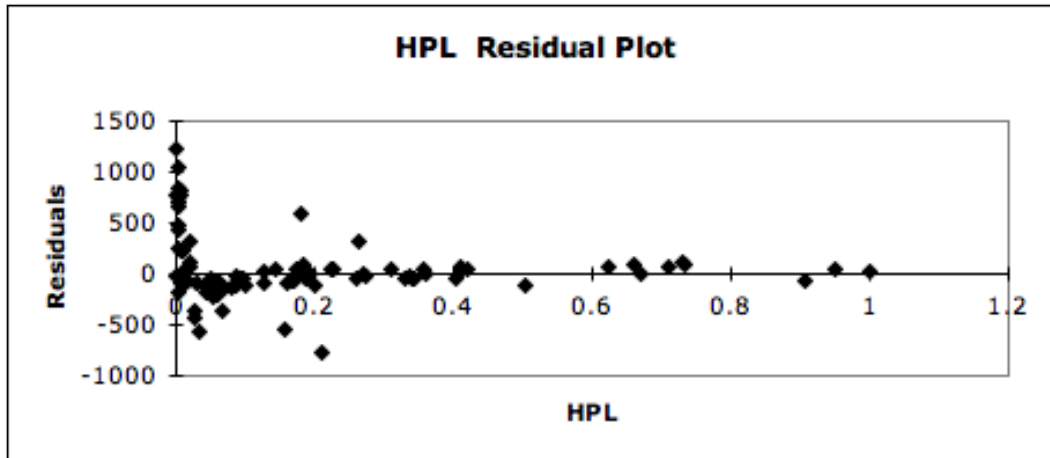
	FFT, $(\alpha, \beta) =$			
	(.01, .001)	(.1, .01)	(.2, .02)	(.5, .05)
Basis Set	295.75	276.57	263.18	227.43
FLOPS	852.24	834.47	820.01	773.61
	DGEMM, $(\alpha, \beta) =$			
	(.01, .001)	(.1, .01)	(.2, .02)	(.5, .05)
Basis Set	433.6	369.83	327.87	254.08
FLOPS	504.8	438.51	388.36	292.45

8.2 show the residual plots for HPL, PTRANS, Stream Triad and Random Ring Bandwidth (other plots are not shown but have similar patterns). Although these plots have a little more of an even spread, they still tend to decrease in spread in the x direction (suggesting non-constant variance), and also exhibit some possible influential points. The non-constant variance within a measurement suggests that the method of normalizing the measurements, as described in Section 3.3 is not so helpful. Other methods, such as centering each set of measurements to have a mean of 0, should be explored and may provide better results.

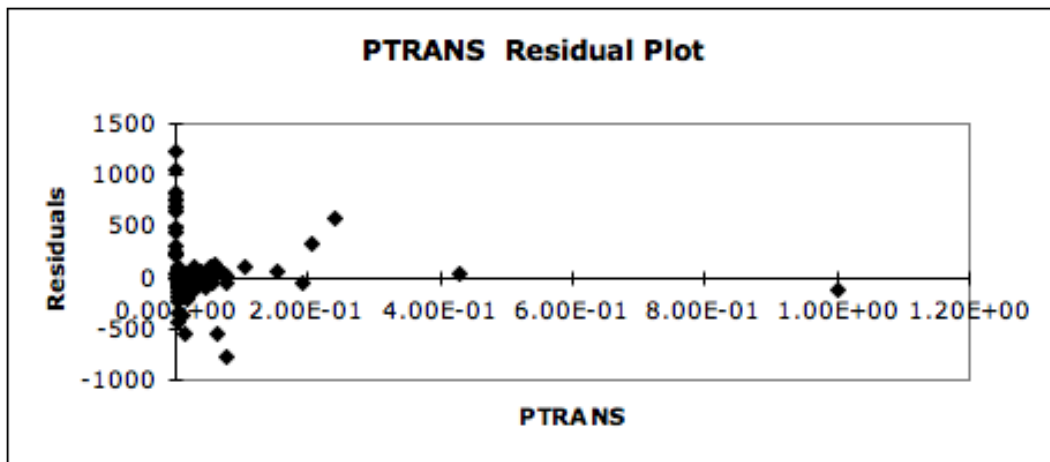
If the predictors with insignificant coefficients were removed, results may improve. The average absolute relative error drops a little, from 2900% to 2500%, when using only the predictors with significant coefficients. Looking over some of the machine's relative errors, there are a handful in the tens of thousands, which are likely skewing the mean. On the other hand, the average number of threshold inversions dropped from 291, using the full set, to 239. Both the average absolute relative error improved (although a relative error of 2500% is still high), as did the average number of inversions when using predictors with significant coefficients.

8.3 Using Optimized HPCC Results for FFT

Users are allowed to optimize the HPC Challenge benchmarks for their systems and submit optimized results. There are 15 systems that have had the full HPC Challenge suite optimized and run on them; the FFT cross-validation analysis

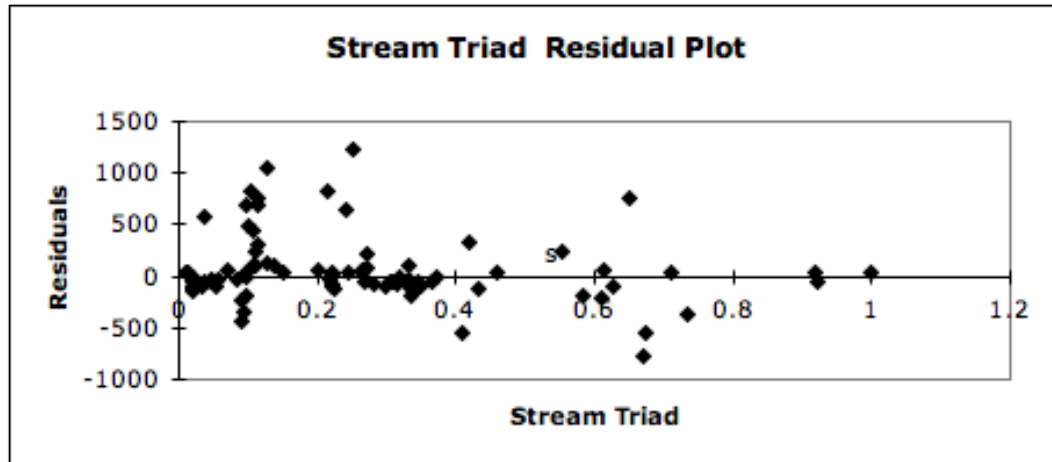


(a)

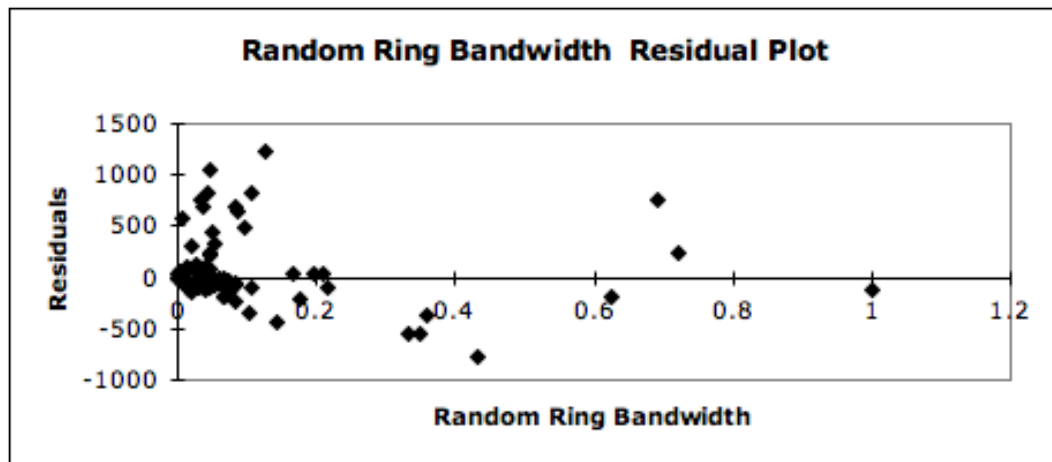


(b)

Figure 8.1: FFT residual plots for HPL and PTRANS.



(a)



(b)

Figure 8.2: FFT residual plots for Stream Triad and Random Ring Bandwidth.

Table 8.3: Linear regression coefficients and p-values for neon_refined on 64 processors using the full HPC Challenge set of benchmarks.

Benchmark	Coefficient	p-values
HPL	-13.08757805	0.955643497
PTRANS	-2769.487536	7.0643E-07
RandomAccess	28.88521559	0.942032021
StreamSys	416.4619221	0.437271326
StreamTriad	144.6159179	0.715718447
DGEMM	-567.7888123	0.203517157
RandomRingBW	2130.072314	2.89853E-07
RandomRingLat	754.0150452	0.003664446

above has been run on these systems. Five systems were chosen for the ordering cross-validation tests, for a maximum of ten inversions.

8.3.1 Reduced Metrics

The full basis set consisted of the nine HPC Challenge benchmarks, minus FFT. The correlation matrix showed that HPL and PTRANS were highly correlated, so HPL was removed. In addition, PTRANS and Stream Triad were highly correlated, so Stream Triad was reduced. Finally DGEMM and the network bandwidth were highly correlated, so DGEMM was dropped. As before, different combinations of which measurements to remove produce small changes in the results; thus, the removed benchmarks were the initially chosen ones. The reduced basis set consisted of the optimized results of:

$$\langle PTRANS, RandomAccess, StreamSys, NetBandwidth, NetLatency \rangle$$

8.3.2 Results

Table 8.4 shows the results. Similar to the results above, FLOPS is a better predictor of performance, relative to the reduced basis set, yet is a very poor predictor for orderings, achieving almost all possible threshold inversions. The reduced basis set improved the orderings of the systems for FFT by 70% compared to FLOPS alone. Furthermore, using the full basis set the performance prediction results are a little better (although a 1500% relative error is not good in general), but the threshold inversions in the ordering is slightly higher. Nonetheless, using a combination of metrics provides a better ordering than FLOPS alone. Table 8.5 shows that increasing α and β typically reduces the average number of threshold inversions. In addition, the average number of threshold inversions when using a combination of metrics is always less than using FLOPS alone, suggesting that a combination of metrics gives more accurate orderings than FLOPS alone.

Table 8.4: Average absolute relative errors (AARE) and average number of threshold inversions for FFT, including standard deviations (stdev), using the optimized HPC Challenge benchmark results. When counting threshold inversions, $\alpha = .01$ and $\beta = .001$.

	FFT	
	AARE (stdev)	Ave. Inversions (stdev)
Reduced Optimized Set	1746.02 (3915.11)	2.7 (2.04)
Full Optimized Set	1502.5 (2754.97)	3.21 (2.09)
FLOPS	188.84 (411.92)	9.1 (.88)

Table 8.5: Number of threshold inversions for FFT using the Optimized HPC Challenge results when varying α and β .

	$(\alpha, \beta) =$			
	$(.01, .001)$	$(.1, .01)$	$(.2, .02)$	$(.5, .05)$
Reduced Optimized Set	2.7	1.6	1.8	2
Full Optimized Set	3.21	3.13	3.06	3.3
FLOPS	9.1	9.08	8.82	8.9

Chapter 9

Limitations

The accuracy for performance prediction and system orderings is limited by the choice of the initial basis set, as well as the inputs to the applications at runtime. For example, the performance predictions and orderings of systems using the HPC Challenge benchmarks with the TI-06 applications was worse than using FLOPS alone; however when a different basis set was used, consisting of the Maps memory bandwidths and Netbench network measurements, both the performance predictions and system orderings were more accurate. The basis set should include measurements for resources that the application uses. With an incomplete basis set, measurements for resources that the application uses won't contribute to the final performance prediction, and system orderings are likely to suffer.

In addition, this methodology is limited to making performance predictions and order systems for a static set of application parameters. As shown in [19], performance of a blocked matrix multiply with a fixed block size on different matrix sizes varies, depending on the underlying cache hierarchy. Similarly, if an application uses a different set of inputs, for example a different input file, then the runtime

is likely to change, and vary across different machines and processor counts. This methodology may not be able to predict such performance without training data for the application run using the different inputs.

Chapter 10

Application of the Thesis

A Java command line tool, LSO, has been written to carry out the analysis. Benchmarks and application runtimes on different processor counts are given in comma separated value (CSV) files. The user can view the correlation matrix of the benchmarks to allow the user to pick which benchmarks to reduce. The Colt library [1] is used to handle all matrix operations, including the least squares regression (the weights given by the least squares regression have been verified using Matlab). The tool then runs the cross validation tests for performance analysis and threshold inversions in ordering. With minor changes the tool can provide an ordering for a set of systems.

A researcher may have an application written using MPI that can easily be ported to different systems and can collect runtimes for it on different systems using standard input parameters. Benchmarks for the systems the application was run on could be collected by the researchers, for example using the HPC Challenge benchmarks download [2], other benchmark packages [8, 4], or matching the systems to HPC Challenge systems already benchmarked, similar to what was

done in this thesis. With the system benchmarks and application runtimes, LSO can be used to reduce the basis set, if needed, and run cross-validation tests to provide a reference point for how runtime predictions and system orderings could perform. Finally, the researcher can collect performance predictions and system orderings for new systems they are interested in.

10.1 Future Work

This tool could benefit from some future work to make it more useful. First, as there are small differences when removing different correlated measurements, an automated approach to pick which measurements to reduce could be implemented. This would save users the trouble of having to look at correlation matrices to decide which measurements to reduce.

Second, the method of reducing columns may be extended to the set of machines for the training set. Table 10.1 shows the correlations of a subset of the systems the TI-06 application were run on based on their Maps and Netbench measurements and Table 10.2 shows some of their specifications. Eagle, Teragrid and Cage are systems that are not similar to the others, and that is reflected by the fact that their highest correlation to other systems is .437 (eagle's correlation to brainerd). On the other hand, brainerd and tempest are similar, as are kraken and iceberg, and that is reflected in their correlations: .953 correlation between brainerd and tempest; and .954 correlation between kraken and iceberg. Using such a method

could help identify a minimum set of machines that application should be run on but further research needs to be done to see if that helps prediction and ordering accuracy.

Table 10.1: Correlations for TI-06 systems based on their Maps and Netbench measurements.

	eagle	teragrid	cage	brainerd	tempest	kraken	iceberg
eagle	1.000	0.400	0.231	0.437	0.358	-0.012	0.034
teragrid	0.400	1.000	0.414	0.255	0.166	-0.141	-0.134
cage	0.231	0.414	1.000	-0.032	0.018	0.236	0.133
brainerd	0.437	0.255	-0.032	1.000	0.953	0.515	0.632
tempest	0.358	0.166	0.018	0.953	1.000	0.652	0.771
kraken	-0.012	-0.141	0.236	0.515	0.652	1.000	0.954
iceberg	0.034	-0.134	0.133	0.632	0.771	0.954	1.000

Table 10.2: System information for those listed in Table 10.1.

	Site	Vendor	Processor	Frequency
eagle	ASC	SGI	Altix	1.6GHz
teragrid	SDSC	IBM	IA64	1.5GHz
cage	ARL	IBM	Opteron	2.2GHz
brainerd	ARL	IBM	P3	0.375GHz
tempest	MHPCC	IBM	P3	0.375GHz
kraken	NAVO	IBM	p655	1.7GHz
iceberg	ARSC	IBM	p655	1.5GHz

Simply cutting measurements based on correlations may cause a loss of information. Instead, principal components analysis (PCA) could be used to reduce the the basis set using a linear transformation based on the components with the highest variances. LSO has implemented PCA, but the results have not yet been verified with Matlab, and thus, are not presented here.

Finally, different metrics for judging the accuracy of system orderings may be useful in addition to threshold inversions. One such metric may be to look at the maximum inversion distance, and can help provide some more information. Table 10.3 shows an example. Using threshold inversions, the two orderings appear to be good orderings, but Predicted Ordering 2 mis-ordered system A and E, the best and worst systems, respectively. Maximum inversion distance would help catch grave mis-orderings such as this and would provide more useful information for judging the accuracy of orderings.

Table 10.3: Example of threshold inversions versus max inversion distance.

	Predicted Order 1	Predicted Order 2	True Order
	B	E	A
	A	B	B
	D	C	C
	C	D	D
	E	A	E
Num. Threshold Inversions	2	1	–
Max Inversion Distance	1	4	–

Chapter 11

Conclusion

Previous work has shown that any single metric is not only poor for predicting application performance, but also poor for ranking systems, compared to using a combination of metrics. This thesis explored a simple method to combine different benchmark measurements to more accurately predict application performance and accurately order systems for the application's performance and has shown that a combination of metrics is almost always a better predictor of performance and orderings of systems, than FLOPS alone.

Given runtimes for an application on different systems and benchmark measurements for different resources on those systems, benchmark measurements that are highly correlated are first removed. A least square regression is used to get weights for the reduced set of benchmark measurements that minimize the squared error estimate for the application's runtime on the set machines. For a new system with measurements for the same resources, the weights are used to predict the application's performance on the new system. Moreover, this was extended to make performance predictions for many new systems, then order those systems based

on the performance predictions. Using a reduced set of measurements always improved performance predictions, compared to using the full set of measurements; making sure the set of measurements are orthogonal helped generalize better for new systems.

For all ten applications tested, using a linear combination of benchmark metrics gives better predictions for the application runtimes, compared to FLOPS alone. The average absolute relative error, over all applications, was 72.5%, using only FLOPS; this was reduced to 39.5%, using a combination of metrics. In addition, the ordering of supercomputers is better, in terms of threshold inversions, using this methodology. Out of a maximum of 10 threshold inversions when ordering 5 machines, there were an average, over all applications, of 4.05, using only FLOPS, compared to 3.05, using a combination of metrics. Furthermore when ordering 45 systems based on the FFT and DGEMM benchmarks of the HPC Challenge suite, out of a maximum number of 990 inversions, there were an average of 852.24 for FFT and 504.8 for DGEMM using FLOPS alone. This was cut down to 295.75 for FFT and 433.6 for DGEMM when using a combination of benchmark measurements.

There are some cases where the average absolute relative error for performance predictions are very large and exploring the residual plots for some of these cases showed two things: a pattern of reduced spread as the measurement for a benchmark grows and possible influential points. The decrease in spread suggest that the variance of the measurements for a benchmark are not constant and that nor-

malizing the measurement by the highest one may not work as well as expected. When predicting machines that contain an influential point, the weights are likely different than if the system was included in the regression, resulting in poor performance predictions for that system. There are few such cases, but when they happen, their large relative errors skew the overall average absolute relative error.

Appendix A

TopCrunch System Information

Table A.1: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 22).

System Number	1	2	3	4	5
Processor	Intel Xeon Dual Core 5160 EM64T	Intel Dualcore Xeon 3.0 GHz DL140	AMD Dualcore Opteron 2.6 GHz DL145	Intel Dualcore Xeon 3.0 GHz DL140	Intel Xeon Dual Core 5160 EM64T
Computer System	S5000PAL	CP3000/Linux	Opteron CP4000	CP3000/Linux	S5000PAL
OS	Redhat EL4 update 2	Red Hat EL 4.3	Red Hat EL 4.3	Red Hat EL 4.3	Redhat EL4 update 2
Interconnect	Infiniband	InfiniBand DDR	InfiniBand	InfiniBand DDR	Infiniband
Vendor	Intel	HP	HP	HP	Intel
MPI	Intel MPI 2.0.1	HP-MPI	HP-MPI	HP-MPI	Intel MPI 2.0.1
Runtime	4298	4693	5883	5905	5955
Matched to HPC Challenge System	Dell PowerEdge 2650 Cluster Intel Xeon INFINIBAND INTERCONN	HP XC3000 Intel Core 2	HP XC4000 AMD Opteron	HP XC3000 Intel Core 2	Dell PowerEdge 2650 Cluster Intel Xeon INFINIBAND INTERCONN

Table A.2: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 22).

System Number	6	7	8	9	10
Processor	AMD Opteron 2.6 GHz DL145	POWER5	AMD Opteron 275 (2.2 GHz)	AMD Dual Core Opteron 2.2 GHz	Intel/ Itanium 2 1.6 GHz
Computer System	Opteron CP4000	1.9 GHz	Altus 1300	CRAY XD1	Altix3700/Bx2
OS	Red Hat 3.0	eserver p5 575	Scyld ClusterWare	SuSE Linux 9	Linux64/SGI ProPack 3.3
Interconnect	Topspin Infini-Band	AIX 5.2, eserver HPS	Infiniband	Rapid Array	NUMALINK
Vendor	HP	IBM	Penguin Computing	CRAY Inc.	SGI
MPI	HP-MPI	POE	MPICH 1.2.5	CRAY XD1 MPI	SGI MPT 1.11
Runtime	6347	6379	6380	6652	6672
Matched to HPC Challenge System	HP XC4000 AMD Opteron	IBM p5-575 Power5	Dell PowerEdge 1950 Cluster Intel Xeon EM64T	Cray Inc. XD1 AMD Opteron	SGI Altix 3700 Intel Itanium 2

Table A.3: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 22).

System Number	11	12	13	14	15
Processor	AMD Opteron 2.4 GHZ	AMD Dualcore Opteron 2.6 GHz DL145	Dual Core Opteron 2.2 GHz DL145	AMD Opteron 2.2 HGZ	AMD Dual Core Opteron 2.2 GHZ
Computer System	CRAY XD1	Opteron CP4000	Opteron CP4000	CRAY XD1	CRAY XD1
OS	SuSE Linux 9	Red Hat EL 4.3	Red Hat 4.0	Suse Linux 8	SuSE Linux 9
Interconnect	Rapid Array	InfiniBand	Voltaire	Rapid Array	Rapid Array
Vendor	CRAY Inc.	HP	HP	Cray	CRAY Inc.
MPI	CRAY XD1 MPI	HP-MPI	HP-MPI	Cray XD1 MPI	CRAY XD1 MPI
Runtime	6878	6902	6989	7504	7591
Matched to HPC Challenge System	Cray Inc. XD1 AMD Opteron	HP XC4000 AMD Opteron	HP XC4000 AMD Opteron	Cray Inc. XD1 AMD Opteron	Cray Inc. XD1 AMD Opteron

Table A.4: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 22).

System Number	16	17	18	19	20
Processor	AMD Opteron 2.4GHz DL145	Dual Core Opteron 2.2 GHz DL145	1.5GHz Itanium2 rx2600	Xeon 3.6Ghz	AMD Opteron 2.2GHz DL145
Computer System	Opteron CP4000	Opteron CP4000	Itanium2 CP6000	GT4000	Opteron CP4000/ Infini-Band Voltaire
OS	Red Hat 3.0	Red Hat 4.0	HP-UX 11.23	Linux	Red Hat 3.0
Interconnect	Myrinet	Voltaire	Infiniband TopSpin	Infiniband	InfiniBand Voltaire
Vendor	HP	HP	HP	Galactic Computing (Shenzhen) Ltd.	HP
MPI	HP-MPI	HP-MPI	HP-MPI	MPICH VMI-2.1	HP-MPI
Runtime	8024	8042	8189	8433	9650
Matched to HPC Challenge System	HP XC4000 AMD Opteron	HP XC4000 AMD Opteron	HP XC Intel Itanium 2	Dell PowerEdge 1850 cluster Intel Xeon EM64T	HP XC4000 AMD Opteron

Table A.5: 3 Car Collisions on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 21 and 22 of 22).

System Number	21	22
Processor	2 GHz Opteron	1.5 GHz Itanium2 RX2600
Computer System	Opteron Cluster	Itanium2 Cluster
OS	SuSE SLES8 SP3	HP-UX 11.23
Interconnect	InfiniCon Infiniband	Infiniband
Vendor	Appro, Rackable and Verari	HP
MPI		
Runtime	11233	14182
Matched to HPC Challenge System	HP XC4000 AMD Opteron	HP XC Intel Itanium 2

Table A.6: 3 Car Collisions on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 20).

System Number	1	2	3	4	5
Processor	Intel Xeon Dual Core 5160 EM64T	Intel Dual-core Xeon 3.0 GHz DL140	AMD Dual Core Opteron 2.2 GHZ	Intel Xeon Dual Core 5160 EM64T	Intel/ Itanium 2 1.6 GHz
Computer System	S5000PAL	CP3000/ Linux	CRAY XD1	S5000PAL	Altix3700/ BX2
OS	Redhat EL4 update 2	Red Hat EL 4.3	SuSE Linux 9	Redhat EL4 update 2	Linux64/ SGI ProPack3.3
Interconnect	Infiniband	InfiniBand DDR	Rapid Array	Infiniband	NUMALINK
Vendor	Intel	HP	CRAY Inc.	Intel	SGI
MPI	Intel MPI 2.0.1	HP-MPI	CRAY XD1 MPI	Intel MPI 2.0.1	SGI MPT 1.11
Runtime	2539	2762	3393	3402	3572
Matched to HPC Challenge System	Dell PowerEdge 2650 Cluster Intel Xeon INFINIBAND INTERCONN	HP XC3000 Intel Core 2	Cray Inc. XD1 AMD Opteron	Dell PowerEdge 2650 Cluster Intel Xeon INFINIBAND INTERCONN	SGI Altix 3700 Intel Itanium 2

Table A.7: 3 Car Collisions on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 20).

System Number	6	7	8	9	10
Processor	POWER5	AMD Dualcore Opteron 2.6 GHZ DL145	AMD Dual Core Opteron 2.2 GHZ	AMD Opteron 2.4 GHZ	Intel Dualcore Xeon 3.0 GHZ DL140
Computer System	1.9 GHZ	Opteron CP4000	CRAY XD1	CRAY XD1	CP3000/Linux
OS	eserver p5 575	Red Hat EL 4.3	SuSE Linux 9	SuSE Linux 9	Red Hat EL 4.3
Interconnect	AIX 5.2, eserver HPS	InfiniBand	Rapid Array	Rapid Array	InfiniBand DDR
Vendor	IBM	HP	CRAY Inc.	CRAY Inc.	HP
MPI	POE	HP-MPI	CRAY XD1 MPI	CRAY XD1 MPI	HP-MPI
Runtime	3638	3797	3846	3946	4118
Matched to HPC Challenge System	IBM p5-575 Power5	HP XC4000 AMD Opteron	Cray Inc. XD1 AMD Opteron	Cray Inc. XD1 AMD Opteron	HP XC3000 Intel Core 2

Table A.8: 3 Car Collisions on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 20).

System Number	11	12	13	14	15
Processor	AMD Dual-Core Opteron Model 275 (2.2GHz)	Xeon 3.6Ghz	AMD Dualcore Opteron 2.6 GHz DL145	AMD Dual-Core Opteron Model 275 (2.2GHz)	AMD Opteron 2.2 GHZ
Computer System	Emerald	GT4000	Opteron CP4000	Emerald	CRAY XD1
OS	ROCKS 4.0.0 (RHEL4 U1)	Linux	Red Hat EL 4.3	ROCKS 4.0.0 (RHEL4 U1 kernel)	Suse Linux 8
Interconnect	PathScale Infini-Path/Silverstorm Infini-Band switch	Infiniband	InfiniBand	PathScale Infini-Path/Silverstorm Infini-Band switch	Rapid Array
Vendor	Rackable Systems	Galactic Computing (Shenzhen) Ltd.	HP	Rackable Systems	CRAY
MPI	PathScale MPICH	MPICH VMI-2.1	HP-MPI	PathScale MPICH	CRAY XD1 MPI
Runtime	4167	4176	4236	4296	4619
Matched to HPC Challenge System	PathScale	Dell PowerEdge 1850 cluster Intel Xeon EM64T	HP XC4000 AMD Opteron	PathScale	Cray Inc. XD1 AMD Opteron

Table A.9: 3 Car Collisions on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 20).

System Number	16	17	18	19	20
Processor	Xeon 3.6Ghz	1.5GHz Itanium2 rx2600	Opteron 275	2 GHz Opteron	1.5 GHz Itanium2 RX2600
Computer System	GT4000	Itanium2 CP6000	1122Hi-81	Opteron Cluster	Itanium2 Cluster
OS	Linux	HP-UX 11.23	SuSE SLES 9	SuSE SLES8 SP3	HP-UX 11.23
Interconnect	Infiniband	Infiniband TopSpin	Level 5 Networks - 1 Gb Ethernet NIC	InfiniCon Infini- band	Infiniband
Vendor	Galactic Com- puting (Shen- zhen) Ltd.	HP	Appro	Appro, Rack- able, and Verari	HP
MPI	MPICH VMI-2.1	HP-MPI	Scali v4.4.2		
Runtime	4732	5152	5356	7238	8106
Matched to HPC Challenge System	Dell Pow- erEdge 1850 clus- ter Intel Xeon EM64T	HP XC Intel Itanium 2	lustervision BV Beastie AMD Opteron	HP XC4000 AMD Opteron	HP XC Intel Itanium 2

Table A.10: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 33).

System Number	1	2	3	4	5
Processor	Intel Xeon Dual Core 5160 EM64T	AMD Opteron 156 3.0 GHz	AMD Opteron 156 3.0 GHz	Intel 5160 Woodcrest DC 3.0GHz	AMD Opteron 2.6 GHz
Computer System	S5000PAL	Sun Fire X2100	Sun Fire X2100	Altix XE1200 Compute Cluster	Microway Navion
OS	Redhat EL4 update 2	64-bit SUSE SLES 9 SP 3	64-bit SUSE SLES 9 SP 3	SUSE Linux Enterprise Server 10 (x86_64)	Fedora Core 3
Interconnect	Infiniband	Infiniband (Topspin)	Infiniband (Voltaire)	Voltaire IB 410 HCA PCIe card with firmware v1.2.0	PathScale Infini-Path/Silverstorm IB switch
Vendor	Intel	Sun Microsystems	Sun Microsystems	SGI	PathScale
MPI	Intel MPI 2.0.1	MVAPICH	MVAPICH	Intel MPI Runtime v2	PathScale MPI
Runtime	1138	1283	1321	1436	1477
Matched to HPC Challenge System	Dell PowerEdge 2650 Cluster Intel Xeon	Sun Fire V20z Cluster AMD Opteron	Sun Fire V20z Cluster AMD Opteron	NO GOOD MATCH	PathScale

Table A.11: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 33).

System Number	6	7	8	9	10
Processor	AMD Opteron 2.6 GHz DL145	Intel Xeon Dual Core 5160 EM64T	Opteron	AMD Dual Core Opteron 2.2 GHZ	AMD Opteron 2.4 GHZ
Computer System	Opteron CP4000	S5000PAL	2.4 GHz	CRAY XD1	CRAY XD1
OS	Red Hat 3.0	Redhat EL4 update 2	e326, SUSE LINUX 9	SuSE Linux 9	SuSE Linux 9
Interconnect	Topspin Infini-Band	Infiniband	Myrinet	Rapid Array	Rapid Array
Vendor	HP	Intel	IBM	CRAY Inc.	CRAY Inc.
MPI	HP-MPI	Intel MPI 2.0.1	mpich-gm	CRAY XD1 MPI	CRAY XD1 MPI
Runtime	1486	1548	1597	1607	1647
Matched to HPC Challenge System	HP XC4000 AMD Opteron	Dell PowerEdge 2650 Cluster Intel Xeon	Cluster-vision BV Beastie AMD Opteron	Cray Inc. XD1 AMD Opteron	Cray Inc. XD1 AMD Opteron

Table A.12: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 33).

System Number	11	12	13	14	15
Processor	Intel Xeon Dual Core 5160 EM64T	AMD Opteron 2.4GHz DL145	Intel Xeon Dual Core 5160 EM64T	AMD Opteron 2.2 GHZ	AMD Dual Core Opteron 2.2 GHZ
Computer System	Relion 1600	Opteron CP4000	Relion 1600	CRAY XD1	CRAY XD1
OS	Scyld ClusterWare 4	Red Hat 3.0	Scyld ClusterWare 4	Suse Linux 8	SuSE Linux 9
Interconnect	Silverstorm Infini-band	Myrinet	Gigabit Ethernet	Rapic Array	Rapid Array
Vendor	Penguin Computing	HP	Penguin Computing	CRAY	CRAY Inc.
MPI	MPICH 1.2.5	HP-MPI	MPICH 1.2.5	CRAY XD1 MPI	CRAY XD1 MPI
Runtime	1662	1740	1776	1795	1820
Matched to HPC Challenge System	Dell PowerEdge 2650 Cluster Intel Xeon	HP XC4000 AMD Opteron	Dell PowerEdge 1950 Cluster Intel Xeon EM64T	Cray Inc. XD1 AMD Opteron	Cray Inc. XD1 AMD Opteron

Table A.13: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 33).

System Number	16	17	18	19	20
Processor	1.5GHz Itanium2 rx2600	AMD Opteron 2.2GHz DL145	1.5 GHz Itanium2 RX2600	AMD Opteron 2.0 GHz	2 GHz Opteron
Computer System	Itanium2 CP6000	Opteron CP4000	Itanium2 Cluster	2.0 GHz	Opteron Cluster
OS	HP-UX 11.23	Red Hat 3.0	HP-UX 11.23	e325 cluster, SuSE SLES-8	SuSE SLES8 SP3 w MPI/Pro
Interconnect	Infiniband TopSpin	InfiniBand Voltaire	Infiniband	Myrinet	InfiniCon Infini- band
Vendor	HP	HP	HP	IBM	Appro, Rack- able and Verari
MPI	HP-MPI	HP-MPI			
Runtime	1872	1950	2088	2108	2196
Matched to HPC Challenge System	HP XC Intel Itanium 2	HP XC4000 AMD Opteron	HP XC Intel Itanium 2	lustervision BV Beastie AMD Opteron	HP XC4000 AMD Opteron

Table A.14: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 21-25 of 33).

System Number	21	22	23	24	25
Processor	AMD Opteron	AMD Athlon 64 3500+ (2.2 GHz)	1.5 GHz Itanium2 RX2600	AMD Dualcore Opteron 2.2 GHz DL145	POWER4+
Computer System	2.0 GHz	AMD64 Cluster	HP-UX Itanium2 Cluster/ GigE	CP4000/ Windows CCS	1.7 GHz
OS	e325 cluster, SuSE SLES-8	Fedora Core 3	HP-UX 11.23	Microsoft Windows Compute Cluster Server 2003	IBM p655
Interconnect	Gigabit Ethernet	Gigabit Ethernet		Voltaire Infini-Band SDR	AIX 5.1
Vendor	IBM	Engineering Research Nordic AB	HP	HP	IBM
MPI		LAM MPI		HP-MPI	
Runtime	2454	2493	2550	2605	2635
Matched to HPC Challenge System	Cluster-vision BV Beastie AMD Opteron	NO GOOD MATCH	HP XC Intel Itanium 2	HP XC4000 AMD Opteron	IBM p655 Power4+

Table A.15: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 26-30 of 33).

System Number	26	27	28	29	30
Processor	Xeon 3.4GHz	Intel Xeon 3.06GHz	Intel Xeon	Intel Dual Xeon 3.066 GHz	Intel Dual Xeon 3.066 GHz
Computer System	Xeon Desktop	RLX 3000ix Server- Blade	HP Pro- Liant dl360 G3	IBM x335	IBM x335
OS	Windows XP	Red Hat Enter- prise Linux 3 Update 3	Red Hat Enter- prise Linux 3 WS update 4	Red Hat EL 3.0 WS	Red Hat EL 3.0 WS
Interconnect	GigE	Voltaire Infini- Band	Gigabit Ethernet	Myrinet	Gigabit Ethernet
Vendor	Dell	RLX	Hewlett- Packard	IBM	IBM
MPI		MVAPICH	LAM 6.5.9		
Runtime	2725	3150	3210	3669	3885
Matched to HPC Challenge System	Dell Pow- erEdge 1950 Cluster Intel Xeon EM64T	Dell Pow- erEdge 1850 clus- ter Intel Xeon EM64T	Dell Pow- erEdge 1950 Cluster Intel Xeon EM64T	Dell Pow- erEdge 2650 Cluster Intel Xeon	Dell Pow- erEdge 1950 Cluster Intel Xeon EM64T

Table A.16: neon_refined on 8 processors system information, runtimes and matched HPC Challenge systems (Systems 31-33 of 33).

System Number	31	32	33
Processor	AMD Athlon MP 2600+	AMD Athlon MP 2000+	POWER3
Computer System	Athlon Cluster	Athlon Cluster	Blue Horizon
OS	White Box Enterprise Linux 3.0	RedHat Enterprise Linux 3	AIX 5.1
Interconnect	1 Gbit Ethernet	100 Mbit ethernet	
Vendor	Appro	Appro	IBM
MPI			
Runtime	6173	7188	12285
Matched to HPC Challenge System	NO GOOD MATCH	NO GOOD MATCH	IBM RS/6000 SP Power3

Table A.17: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 26).

System Number	1	2	3	4	5
Processor	Intel Xeon Dual Core 5160 EM64T	AMD Opteron 156 3.0 GHz	Intel Xeon Dual Core 5160 EM64T	Intel 5160 Woodcrest DC 3.0GHz	AMD Opteron 2.6 GHz
Computer System	S5000PAL	Sun Fire X2100	S5000PAL	Altix XE1200 Compute Cluster	Microway Navion
OS	Redhat EL4 update 2	64-bit SUSE SLES 9 SP 3	Redhat EL4 update 2	SUSE Linux Enterprise Server 10 (x86_64)	Fedora Core 3
Interconnect	Infiniband	Infiniband (Topspin)	Infiniband		Voltaire IB 410 HCA PCIe card with firmware v1.2.0
Vendor	Intel	Sun Microsystems	Intel	SGI	PathScale
MPI	Intel MPI 2.0.1	MVAPICH	Intel MPI 2.0.1	Intel MPI Runtime v2	PathScale MPI
Runtime	364	433	451	452	480
Matched to HPC Challenge System	Dell PowerEdge 2650 Cluster Intel Xeon	Sun Fire V20z Cluster AMD Opteron	Dell PowerEdge 2650 Cluster Intel Xeon	NO GOOD MATCH	PathScale

Table A.18: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 26).

System Number	6	7	8	9	10
Processor	AMD Dual Core Opteron 2.2 GHZ	AMD Opteron 2.6 GHz DL145	AMD Opteron 2.4 GHZ	AMD Dual Core Opteron 2.2 GHZ	Opteron
Computer System	CRAY XD1	Opteron CP4000	CRAY XD1	CRAY XD1	2.4 GHz
OS	SuSE Linux 9	Red Hat 3.0	SuSE Linux 9	SuSE Linux 9	e326, SUSE LINUX 9
Interconnect	PathScale Infini-Path/Silverstorm IB switch	Rapid Array	Topspin Infini-Band	Rapid Array	Rapid Array
Vendor	CRAY Inc.	HP	CRAY Inc.	CRAY Inc.	IBM
MPI	CRAY XD1 MPI	HP-MPI	CRAY XD1 MPI	CRAY XD1 MPI	mpich-gm
Runtime	527	529	541	569	586
Matched to HPC Challenge System	Cray Inc. XD1 AMD Opteron	HP XC4000 AMD Opteron	Cray Inc. XD1 AMD Opteron	Cray Inc. XD1 AMD Opteron	Clustervision BV Beastie AMD Opteron

Table A.19: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 26).

System Number	11	12	13	14	15
Processor	AMD Opteron 2.2 GHZ	AMD Opteron 2.4GHz DL145	1.5GHz Itanium2 rx2600	Xeon 3.6Ghz	1.5 GHz Itanium2 RX2600
Computer System	CRAY XD1	Opteron CP4000	Itanium2 CP6000	GT4000	Itanium2 Cluster
OS	Suse Linux 8	Red Hat 3.0	HP-UX 11.23	Linux	HP-UX 11.23
Interconnect	Myrinet	Rapic Array	Myrinet	Infiniband TopSpin	Infiniband
Vendor	CRAY	HP	HP	Galactic Computing (Shenzhen) Ltd.	HP
MPI	CRAY XD1 MPI	HP-MPI	HP-MPI	MVAPICH 0.9.5	
Runtime	607	655	688	699	734
Matched to HPC Challenge System	Cray Inc. XD1 AMD Opteron	HP XC4000 AMD Opteron	HP XC Intel Itanium 2	Dell PowerEdge 1850 cluster Intel Xeon EM64T	HP XC Intel Itanium 2

Table A.20: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 26).

System Number	16	17	18	19	20
Processor	AMD Opteron 2.2GHz DL145	AMD Dualcore Opteron 2.2 GHz DL145	AMD Opteron	2 GHz Opteron	POWER4+
Computer System	Opteron CP4000/ Infini-Band Voltaire	CP4000/ Windows CCS	2.0 GHz	Opteron Cluster	1.7 GHz
OS	Red Hat 3.0	Microsoft Windows Compute Cluster Server 2003	e325 cluster, SuSE SLES-8	SuSE SLES8 SP3	IBM p655
Interconnect	InfiniBand Voltaire	Voltaire Infini-Band SDR	Myrinet	InfiniCon Infini-band	AIX 5.1
Vendor	HP	HP	IBM	Appro, Rack-able and Verari	IBM
MPI	HP-MPI				
Runtime	797	810	828	836	885
Matched to HPC Challenge System	HP XC4000 AMD Opteron	HP XC4000 AMD Opteron	Cluster-vision BV Beastie AMD Opteron	HP XC4000 AMD Opteron	IBM p655 Power4+

Table A.21: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (Systems 21-25 of 26).

System Number	21	22	23	24	25
Processor	1.5 GHz Itanium2 RX2600	AMD Opteron	Intel Dual Xeon 3.066 GHz	Intel Dual Xeon 3.066 GHz	Power 4 1.3 GHz
Computer System	HP-UX Itanium2 Cluster / GigE	2.0 GHz	IBM x335	IBM x335	IBM P690
OS	HP-UX 11.23	e325 cluster, SuSE SLES-8	Red Hat EL 3.0 WS	Red Hat EL 3.0 WS	AIX 5
Interconnect		Gigabit Ethernet	Myrinet	Gigabit Ethernet	
Vendor	HP	IBM	IBM	IBM	IBM
MPI					
Runtime	1041	1141	1161	1422	1469
Matched to HPC Challenge System	HP XC Intel Itanium 2	lustervision BV Beastie AMD Opteron	Dell PowerEdge 2650 Cluster Intel Xeon	Dell PowerEdge 2650 Cluster Intel Xeon GigE	NO GOOD DATA

Table A.22: neon_refined on 32 processors system information, runtimes and matched HPC Challenge systems (System 26 of 26).

System Number	26
Processor	POWER3
Computer System	Blue Horizon
OS	AIX 5.1
Interconnect	
Vendor	IBM
MPI	
Runtime	3993
Matched to HPC Challenge System	NO GOOD DATA

Table A.23: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 1-5 of 23).

System Number	1	2	3	4	5
Processor	Intel Xeon Dual Core 5160 EM64T	Intel Xeon Dual Core 5160 EM64T	AMD Dual Core Opteron 2.2 GHZ	AMD Opteron 2.4 GHZ	AMD Dual Core Opteron 2.2 GHZ
Computer System	S5000PAL	S5000PAL	CRAY XD1	CRAY XD1	CRAY XD1
OS	Redhat EL4 update 2	Redhat EL4 update 2	SuSE Linux 9	SuSE Linux 9	SuSE Linux 9
Interconnect	Infiniband	Infiniband	Rapid Array	Rapid Array	Rapid Array
Vendor	Intel	Intel	CRAY Inc.	CRAY Inc.	CRAY Inc.
MPI	Intel MPI 2.0.1	Intel MPI 2.0.1	CRAY XD1 MPI	CRAY XD1 MPI	CRAY XD1 MPI
Runtime	244	289	315	327	342
Matched to HPC Challenge System	Dell PowerEdge 2650 Cluster Intel Xeon	Dell PowerEdge 2650 Cluster Intel Xeon	Cray Inc. XD1 AMD Opteron	Cray Inc. XD1 AMD Opteron	Cray Inc. XD1 AMD Opteron

Table A.24: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 6-10 of 23).

System Number	6	7	8	9	10
Processor	Intel 5160 Woodcrest DC 3.0GHz	AMD Dual-Core Opteron Model 275 (2.2GHz)	AMD Opteron 2.2 GHZ	AMD Dual-Core Opteron Model 275 (2.2GHz)	AMD Opteron 2.2 GHZ
Computer System	Altix XE1200 Compute Cluster	Emerald	CRAY XD1	Emerald	CRAY XD1
OS	SUSE Linux Enterprise Server 10 (x86_64)	ROCKS 4.0.0 (RHEL4 U1)	Suse Linux 8	ROCKS 4.0.0 (RHEL4 U1)	Suse Linux 8
Interconnect	Voltaire IB 410 HCA PCIe card with firmware v1.2.0	PathScale Infini-Path/Silverstorm Infini-Band switch	Rapid Array	PathScale Infini-Path/Silverstorm Infini-Band switch	Rapid Array
Vendor	SGI	Rackable Systems	CRAY	Rackable Systems	CRAY
MPI	Intel MPI Runtime v2	PathScale MPICH	CRAY XD1 MPI	PathScale MPICH	CRAY XD1 MPI
Runtime	353	373	380	394	405
Matched to HPC Challenge System	NO GOOD MATCH	PathScale	Cray Inc. XD1 AMD Opteron	PathScale	Cray Inc. XD1 AMD Opteron

Table A.25: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 11-15 of 23).

System Number	11	12	13	14	15
Processor	Xeon 3.6Ghz	1.5GHz Itanium2 rx2600	AMD Opteron 2.2 GHZ	Opteron	1.5 GHz Itanium2 RX2600
Computer System	GT4000	Itanium2 CP6000	CRAY XD1	2.4 GHz	Itanium2 Cluster
OS	Linux	HP-UX 11.23	Suse Linux 8	e326, SUSE LINUX 9	HP-UX 11.23
Interconnect	Infiniband	Infiniband TopSpin	Rapic Ar- ray	Myrinet	Infiniband
Vendor	Galactic Com- puting (Shen- zhen) Ltd.	HP	CRAY	IBM	HP
MPI	MVAPICH 0.9.5	HP-MPI	CRAY XD1 MPI	mpich-gm	
Runtime	418	421	429	452	456
Matched to HPC Challenge System	Dell Inc. / Scali AS Pow- erEdge 1855 clus- ter Intel Xeon EM64T	HP XC Intel Itanium 2	Cray Inc. XD1 AMD Opteron	Cluster- vision BV Beastie AMD Opteron	HP XC Intel Itanium 2

Table A.26: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 16-20 of 23).

System Number	16	17	18	19	20
Processor	Opteron 275	AMD Dualcore Opteron 2.2 GHz DL145	2 GHz Opteron	POWER4+	Power 4 1.3 GHz
Computer System	1122Hi-81	CP4000/Windows CCS	Opteron Cluster	1.7 GHz	IBMP690
OS	SuSE SLES 9	Microsoft Windows Compute Cluster Server 2003	SuSE SLES8 SP3	IBM p655	AIX 5.3
Interconnect	Level 5 Networks - 1 Gb Ethernet NIC	Voltaire Infini-Band SDR	InfiniCon Infini-band	AIX 5.1	
Vendor	Appro	HP	Appro, Rack-able and Verari	IBM	IBM
MPI	Scali v4.4.2	HP-MPI			
Runtime	492	579	584	667	945
Matched to HPC Challenge System	Cluster-vision BV Beastie AMD Opteron	HP XC4000 AMD Opteron	HP XC4000 AMD Opteron	IBM p655 Power4+	NO GOOD MATCH

Table A.27: neon_refined on 64 processors system information, runtimes and matched HPC Challenge systems (Systems 21-23 of 23).

System Number	21	22	23
Processor	Intel Dual Xeon 2.8GHz	Intel Dual Xeon 2.2 GHz	POWER3
Computer System	Linux Cluster	Linux Cluster	Blue Horizon
OS	Linux RedHat80	Linux	AIX 5.1
Interconnect	3D SCI		
Vendor	Self-made (SKIF program)	ACT	IBM
MPI			
Runtime	1150	1711	2734
Matched to HPC Challenge System	NO GOOD MATCH	Dell PowerEdge 2650 Cluster Intel Xeon	NO GOOD MATCH

Appendix B

TI-06 System Information

Table B.1: TI-06 system information and matched HPC

Challenge systems.

Name	Site	Ven- dor	Type	Freq.	Procs.	Matched to HPC Challenge System
eagle	ASC	SGI	Altix	1.6 GHz	2000p	SGI Altix 3700 Bx2 Intel Itanium 2
teragrid	SDSC	IBM	IA64	1.5 GHz	512p	HP XC Intel Ita- nium 2
cage	ARL	IBM	Opteron	2.2 GHz	2304p	HP XC4000 AMD Opteron
brainerd	ARL	IBM	P3	0.375 GHz	1024p	IBM RS/6000 SP Power3

Table B.1: Continued

Name	Site	Vendor	Type	Freq.	Procs.	Matched to HPC Challenge System
tempest	MHPCC	IBM	P3	0.375 GHz	736p	IBM RS/6000 SP Power3
habu	NAVO	IBM	P3	0.375 GHz	928p	IBM RS/6000 SP Power3
kraken	NAVO	IBM	p655	1.7 GHz	2832p	IBM p655 Power4+
romulus	NAVO	IBM	p655	1.7 GHz	464p	IBM p655 Power4+
iceberg	ARSC	IBM	p655	1.5 GHz	784p	IBM p655 Power4+
hurricane	MHPCC	IBM	p690	1.3 GHz	320p	IBM p690 Power4
marcellus	NAVO	IBM	p690	1.3 GHz	1328p	IBM p690 Power4
shelton	ARL	IBM	p690	1.7 GHz	128p	IBM p690 Power4
opal	ERDC	HP	SC40	0.833 GHz	488p	HP SC-40 Alpha 21264B

Table B.1: Continued

Name	Site	Ven- dor	Type	Freq.	Procs.	Matched to HPC Challenge System
hpc10	ASC	HP	SC45	1.0 GHz	768p	HP AlphaServer SC45 Alpha 21264B
emerald	ERDC	HP	SC45	1.0 GHz	488p	HP AlphaServer SC45 Alpha 21264B
klondike	ARSC	Cray	X1	0.8 GHz	504p	Cray Inc. X1 Cray MSP
diamond	ERDC	Cray	X1	0.8 GHz	240p	Cray Inc. X1 Cray MSP
mos	AHPCRC	Cray	X1E	1.13 GHz	960p	Cray Inc. X1 Cray E
powell	ARL	LNX	Xeon	3.06 GHz	256p	Linux Networx Powell Intel Xeon
jvn	ARL	LNX	Xeon	3.6 GHz	2048p	Dell Inc. / Scali AS PowerEdge 1855 cluster Intel Xeon EM64T

Table B.2: TI-06 Maps benchmark measurements for strided access.

System Name	L1 Strided	L2 Strided	L3 Strided	Main Mem. Strided
eagle	2.91E+04	2.92E+04	1.90E+04	3.01E+03
teragrid	8.69E+03	8.92E+03	8.40E+03	2.79E+03
cage	1.25E+04	6.19E+03	2.02E+03	1.94E+03
brainerd	4.63E+03	2.74E+03	2.90E+03	4.96E+02
tempest	4.63E+03	2.78E+03	2.90E+03	4.87E+02
habu	5.32E+03	2.89E+03	2.90E+03	3.76E+02
kraken	1.34E+04	1.07E+04	4.77E+03	2.31E+03
romulus	1.35E+04	1.07E+04	5.07E+03	2.23E+03
iceberg	1.19E+04	9.38E+03	4.93E+03	1.83E+03
hurricane	1.03E+04	7.93E+03	4.06E+03	1.40E+03
marcellus	1.04E+04	7.89E+03	4.50E+03	1.10E+03
shelton	1.35E+04	1.04E+04	3.86E+03	1.49E+03
opal	6.43E+03	6.55E+03	5.03E+03	8.64E+02
hpc10	7.74E+03	7.85E+03	7.32E+03	1.27E+03
emerald	7.74E+03	7.89E+03	7.35E+03	1.27E+03

Table B.2: Continued

System Name	L1	L2	L3	Main
	Strided	Strided	Strided	Mem.
				Strided
klondike	3.96E+03	5.37E+03	5.63E+03	5.64E+03
diamond	4.10E+03	5.38E+03	5.61E+03	5.51E+03
mos	5.10E+03	7.03E+03	7.38E+03	7.28E+03
powell	4.87E+03	4.89E+03	1.64E+03	1.59E+03
jvn	2.21E+04	2.28E+04	2.06E+04	1.82E+04

Table B.3: TI-06 Maps benchmark measurements for random access.

System Name	L1 Ran-	L2 Ran-	L3 Ran-	Main
	dom	dom	dom	Mem.
				Random
eagle	2.69E+03	2.67E+03	1.48E+03	1.12E+02
teragrid	4.32E+03	3.94E+03	1.40E+03	1.60E+02
cage	3.07E+03	1.85E+03	3.65E+02	2.30E+02
brainerd	2.35E+03	7.78E+02	1.56E+02	1.97E+01

Table B.3: Continued

System Name	L1 Ran- dom	L2 Ran- dom	L3 Ran- dom	Main Mem. Random
tempest	2.34E+03	7.78E+02	1.57E+02	1.95E+01
habu	2.41E+03	7.82E+02	1.57E+02	2.17E+01
kraken	9.59E+03	2.61E+03	2.89E+02	5.26E+01
romulus	9.65E+03	2.61E+03	2.87E+02	5.19E+01
iceberg	8.28E+03	1.86E+03	3.07E+02	4.34E+01
hurricane	7.35E+03	1.98E+03	1.67E+02	2.86E+01
marcellus	7.26E+03	1.99E+03	1.37E+02	2.58E+01
shelton	9.64E+03	2.59E+03	2.07E+02	3.13E+01
opal	4.54E+03	1.58E+03	3.35E+02	7.39E+01
hpc10	5.39E+03	1.70E+03	3.77E+02	9.53E+01
emerald	5.46E+03	1.70E+03	3.81E+02	9.55E+01
klondike	1.90E+03	2.29E+03	7.90E+02	7.51E+02
diamond	2.37E+03	2.53E+03	1.13E+03	9.74E+02
mos	2.86E+03	3.27E+03	1.44E+03	1.20E+03
powell	4.75E+03	4.61E+03	4.40E+01	3.27E+01
jvn	5.74E+03	4.52E+03	5.03E+02	1.04E+02

Table B.4: TI-06 Netbench network measurements.

System Name	Network Band- width	Network Latency
eagle	1.13E+03	1.80E-05
teragrid	2.14E+02	9.00E-06
cage	2.46E+02	8.00E-06
brainerd	1.65E+02	3.50E-05
tempest	2.56E+02	2.80E-05
habu	1.14E+02	3.50E-05
kraken	1.63E+03	7.00E-06
romulus	1.66E+03	7.00E-06
iceberg	1.54E+03	7.00E-06
hurricane	8.17E+01	2.88E-05
marcellus	2.46E+02	2.90E-05
shelton	1.40E+03	7.00E-06
opal	1.95E+02	8.00E-06
hpc10	2.40E+02	1.00E-05
emerald	2.43E+02	7.00E-06
klondike	4.33E+03	1.10E-05

Table B.4: Continued

System Name	Network Band- width	Network Latency
diamond	2.16E+03	1.10E-05
mos	2.16E+03	1.10E-05
powell	2.43E+02	8.00E-06
jvn	2.44E+02	8.00E-06

Table B.5: AVUS runtimes on different processor counts
on TI-06 systems.

System Name	Processor Count						
	32	64	96	128	192	256	384
eagle	5224	2403	1601	1201	830	676	482
teragrid	0	0	0	0	0	0	0
cage	4720	2396	1626	1230	831	707	472
brainerd	18310	9195	6172	4848	3354	2775	2005
tempest	0	0	0	0	0	0	0
habu	20633	10116	6742	5020	3343	2517	1713

Table B.5: Continued

System Name	Processor Count						
	32	64	96	128	192	256	384
kraken	5585	2793	1871	1409	945	727	496
romulus	5565	2774	1849	1387	928	699	0
iceberg	0	0	0	0	0	0	0
hurricane	0	0	0	0	0	0	0
marcellus	0	0	0	0	0	0	0
shelton	7865	3896	2531	1998	0	0	0
opal	12758	6208	4289	3190	2181	1602	1136
hpc10	9246	4496	3041	2291	1556	1165	837
emerald	9330	4579	3159	2404	1619	1259	899
klondike	0	0	0	0	0	0	0
diamond	43453	22569	15523	12417	8715	0	0
mos	0	0	0	0	0	0	0
powell	0	0	0	0	0	0	0
jvn	4576	2278	1487	1149	834	663	562

Table B.6: CTH runtimes on different processor counts

on TI-06 systems.

	Processor Count			
System Name	16	32	64	96
eagle	10488	5532	3281	2392
teragrid	0	0	0	0
cage	17600	9458	5270	0
brainerd	45646	23324	13057	0
tempest	0	0	0	0
habu	52739	27844	15179	11808
kraken	12899	6756	3494	0
romulus	12953	6699	3470	2372
iceberg	13508	6939	3625	0
hurricane	11748	10970	6037	5131
marcellus	20900	11380	0	0
shelton	0	0	0	0
opal	22331	12447	6700	5251
hpc10	16006	9009	4914	0
emerald	16369	8902	4865	3836
klondike	0	0	0	0

Table B.6: Continued

	Processor Count			
System Name	16	32	64	96
diamond	0	0	0	0
mos	0	0	0	0
powell	33731	0	0	6739
jvn	16366	8759	4822	3589

Table B.7: GAMESS runtimes on different processor counts on TI-06 systems.

	Processor Count				
System Name	32	48	64	96	128
eagle	7172	5456	4855	3902	3160
teragrid	7744	5764	4913	3909	3778
cage	0	0	0	0	0
brainerd	0	0	0	0	0
tempest	0	0	0	0	0
habu	55229	38434	29797	21416	17165
kraken	8982	6434	5149	3941	3318

Table B.7: Continued

System Name	Processor Count				
	32	48	64	96	128
romulus	9177	6571	5258	3966	3312
iceberg	10199	7332	5922	4440	3611
hurricane	10696	7820	6355	0	0
marcellus	0	0	0	0	0
shelton	0	0	0	0	0
opal	19306	13943	11328	8911	7698
hpc10	15114	11083	8624	6666	5602
emerald	15687	11297	9112	7072	6050
klondike	0	0	16779	11823	9624
diamond	38590	0	0	22466	0
mos	0	0	0	0	0
powell	0	0	0	0	0
jvn	0	0	0	0	0

Table B.8: HYCOM runtimes on different processor counts on TI-06 systems.

System Name	Processor Count						
	24	47	59	80	96	111	124
eagle	0	0	0	0	0	0	0
teragrid	0	0	0	0	0	0	0
cage	4306	2482	1936	1460	1268	1116	1031
brainerd	17741	8941	8369	0	0	0	3489
tempest	0	0	0	0	0	0	0
habu	19933	10066	7129	5244	4420	3829	3348
kraken	4814	2622	2010	1478	1267	1132	990
romulus	4751	2565	1970	0	0	0	914
iceberg	5249	2847	2201	0	0	0	1062
hurricane	0	0	0	0	0	0	0
marcellus	7136	4345	3364	2818	2472	2266	2031
shelton	4964	3003	2451	0	0	0	0
opal	11241	6503	4660	3602	3270	2907	2608
hpc10	8679	5004	3459	2809	2492	2166	1943
emerald	10589	0	0	3071	2643	2276	2222
klondike	0	0	0	0	1354	0	0

Table B.8: Continued

	Processor Count						
System Name	24	47	59	80	96	111	124
diamond	0	0	0	0	1487	0	0
mos	0	0	0	0	3304	0	0
powell	8511	4890	3788	0	0	0	0
jvn	0	0	0	0	0	0	0

Table B.9: LAMMPS runtimes on different processor counts on TI-06 systems.

	Processor Count				
System Name	16	32	48	64	128
eagle	23511	11769	8009	6326	3465
teragrid	20900	12029	9212	8042	6376
cage	0	0	0	0	0
brainerd	0	0	0	0	0
tempest	42259	20943	15035	11955	7135
habu	43004	24419	16800	13616	9235
kraken	12729	6410	4724	3489	2150

Table B.9: Continued

System Name	Processor Count				
	16	32	48	64	128
romulus	14449	8287	7358	5698	5091
iceberg	15619	9162	7860	5896	5246
hurricane	20693	12403	0	0	0
marcellus	0	0	0	0	0
shelton	0	0	0	0	0
opal	35103	15609	12392	9833	7096
hpc10	27969	13305	9931	8033	5381
emerald	28043	13378	9781	8125	0
klondike	27466	13701	9329	6959	3578
diamond	36103	17908	12680	10600	4923
mos	21034	10369	7124	5305	2707
powell	20273	9667	6570	5801	0
jvn	12899	7248	5117	4332	3270

Table B.10: OOCORE runtimes on different processor counts on TI-06 systems.

System Name	Processor Count				
	16	32	48	64	128
eagle	5631	2846	2026	1931	977
teragrid	9391	4509	3040	2172	1197
cage	19482	10530	0	4889	0
brainerd	32164	18142	11784	8928	5253
tempest	28070	14855	9579	7406	4199
habu	27094	14639	0	6552	0
kraken	12446	6578	4238	3049	0
romulus	12787	6646	4450	3236	0
iceberg	15684	7502	4847	3489	0
hurricane	19451	8723	6384	0	0
marcellus	16836	8204	5210	3739	0
shelton	15091	8158	6270	4836	0
opal	24703	12090	8523	5347	0
hpc10	14947	7472	5609	3305	0
emerald	21294	9178	6394	5016	0
klondike	16844	0	0	0	2228

Table B.10: Continued

	Processor Count				
System Name	16	32	48	64	128
diamond	14560	5856	4481	2870	0
mos	10746	0	3782	2744	1674
powell	0	0	0	0	0
jvn	11286	5754	3917	2871	0

Table B.11: OVERFLOW runtimes on different processor counts on TI-06 systems.

	Processor Count			
System Name	16	32	48	64
eagle	4032	2120	1609	1159
teragrid	4624	2576	1996	1477
cage	8893	4047	2978	2082
brainerd	24252	13399	0	7282
tempest	0	13834	10259	7168
habu	26267	14039	10212	7136
kraken	0	4686	3499	2329

Table B.11: Continued

System Name	Processor Count			
	16	32	48	64
romulus	9401	4670	3484	2335
iceberg	10302	5138	3828	2591
hurricane	0	7720	5763	4615
marcellus	0	0	0	0
shelton	9896	5906	0	3004
opal	14116	7357	5964	4169
hpc10	10674	0	4321	3316
emerald	11302	5777	4411	3345
klondike	6124	3086	2473	1744
diamond	5801	3122	2335	1666
mos	5329	0	2198	1904
powell	28064	12938	9424	6329
jvn	7856	3598	2633	1908

Table B.12: WRF runtimes on different processor counts
on TI-06 systems.

System Name	Processor Count							
	16	32	48	64	96	128	192	256
eagle	5460	2744	2014	1583	1082	847	649	546
teragrid	0	0	0	0	0	0	0	0
cage	6843	3615	2732	2095	1497	1189	897	788
brainerd	24520	12489	8621	6746	4697	3820	2913	2637
tempest	0	0	0	0	0	3442	0	2161
habu	27348	13645	9096	6862	4660	3678	2626	2115
kraken	9020	4700	3263	2511	1749	1469	1114	972
romulus	8786	4468	3052	2313	1573	1222	849	671
iceberg	10072	5289	3634	2779	1948	1618	1243	1125
hurricane	0	5833	0	3504	0	3280	0	0
marcellus	0	0	0	0	0	0	0	0
shelton	0	0	0	0	0	0	0	0
opal	13878	7284	5063	3948	2818	2343	1751	1374
hpc10	10139	5376	3669	2863	1996	1653	0	0
emerald	10348	5519	3813	3015	2203	1757	1300	1061
klondike	22373	11309	7700	5815	3957	3024	2146	1686

Table B.12: Continued

	Processor Count							
System Name	16	32	48	64	96	128	192	256
diamond	21981	11280	7612	5771	3857	2972	2103	0
mos	18387	9617	6949	5005	3449	0	2156	0
powell	29594	16207	11359	9493	6121	4832	3503	0
jvn	0	4058	2965	2406	0	1297	0	0

Bibliography

- [1] Colt Library. <http://dsd.lbl.gov/hoschek/colt/index.html>.
- [2] HPC Challenge Benchmarks. <http://icl.cs.utk.edu/hpcc/>.
- [3] HPC Challenge Results. http://icl.cs.utk.edu/hpcc/hpcc_results.cgi.
- [4] SPEC. <http://www.spec.org/>.
- [5] TopCrunch. <http://www.topcrunch.org/>.
- [6] Department of Defense High Performance Computing Modernization Program. Technology Insertion - 06 (TI-06). <http://www.hpcmo.hpc.mil/Htdocs/TI/TI06>, May 2005.
- [7] Vikram Adve and Rizos Sakellariou. Application Representations for Multiparadigm Performance Modeling of Large-Scale Parallel Scientific Codes. *International Journal of High Performance Computing Applications*, 2000.
- [8] D. Bailey, J. Barton, T. Lasinski, and H. Simon. The NAS Parallel Benchmarks. *International Journal of Supercomputer Applications*, 1991.
- [9] L.T. Boland, G.D. Granito, A.V. Marcotte, B.V. Messina, and J.W. Smith. The IBM System 360/Model9:Storage System. *IBM J. Res. And Develop*, 11:54–79, 1967.
- [10] D. Burger, T.M. Austin, and S. Bennett. Evaluating Future Microprocessors: The SimpleScalar Tool Set. *Tech. Rep. CS-TR-1996-1308*, University of Wisconsin-Madison 1996.
- [11] Laura C. Carrington, Michael Laurenzano, Allan Snavely, Roy L. Campbell, Jr., and Larry P. Davis. How Well Can Simple Metrics Represent the Performance of HPC Applications? *Proceedings of Supercomputing (SC—05)*, November 2005.

- [12] Tzu-Yi Chen, Meghan Gunn, Beth Simon, Laura Carrington, and Allan Snavely. Metrics for Ranking the Performance of Supercomputers. *CTWatch Quarterly*, 2(4B), November 2006.
- [13] David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice Santos, and Ramesh Subramonian. Logp: Towards a Realistic Model of Parallel Computation. *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 1993.
- [14] Pedro C. Diniz and Jeremy Abramson. SLOPE - A Compiler Approach to Performance Prediction and Performance Sensitivity Analysis for Scientific Codes. *CTWatch Quarterly*, 2(4B), November 2006.
- [15] Ad Emmen. IDC Reports Latest Supercomputer Rankings Based on the IDC Balance Rating Test. *Primeur Monthly*, May 2002.
- [16] Marcio Faerman, Alan Su, Richard Wolski, and Francine Berman. Adaptive Performance Prediction for Distributed Data-Intensive Applications. *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, 1999.
- [17] J. Gustafson and R. Todi. Conventional Benchmarks as a Sample of the Performance Spectrum. *Hawaii International Conference on System Sciences*, 1998.
- [18] E Ipek, B. R. de Supinski, M.Schulz, and S. A. *Euro-Par 2005 Parallel Processing*, volume Volume 3648/2005, chapter An Approach to Performance Prediction for Parallel Applications, pages 196–205. Springer Berlin / Heidelberg, 2005.
- [19] Monica S. Lam, Edward E. Rothberg, and Michael E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, April 1991.
- [20] J. Lo, S. Egger, J. Emer, H. Levy, R. Stamm, and D. Tullsen. Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading. *ACM Transaction on Computer System*, August 1997.
- [21] J. McCalpin. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Technical Committee on Computer Architecture Newsletter*.
- [22] J.O. Murphey and R.M. Wad. The IBM 360/195. *Datamation*, 16(4):72–79, 1970.

- [23] Ballansc R.S., J.A. Cocke, , and H.G. Kolsky. *The Lookahead Unit, Planning a Computer System*. McGraw-Hill, New York, 1962.
- [24] Allan Snavely, Laura Carrington, Mustafa M. Tikir, Roy L. Campbell, Jr., and Tzu-Yi Chen. Solving the Convolution Problem in Performance Modeling. 2006.
- [25] G.S. Tjaden and M.J. Flynn. Detection and Parallel Execution of Independent Instruction. *IEEE Trans. Comptrs.*, C-19:889–895, 1970.
- [26] Top500 Supercomputer Sites. <http://www.top500.org>.
- [27] Top500 Linpack. <http://top500.org/about/linpack>.
- [28] Leo T. Yang, Xiaosong Ma, and Frank Mueller. Cross Platform Performance Prediction of Parallel Applications Using Partial Execution. *Proceedings of Supercomputing (SC–05)*, November 2005.