Selected Regression Algorithms Applied to Spatial Computation

Anna Cavender, Martha Mercaldi

December 13, 2004

Abstract: *For this project, we have formulated an architecture problem as a machine learning problem. The problem, stemming from the WaveScalar project, is to map operations in a dataflow graph onto a grid of processing elements (where they are executed). An unsolved and widely applicable problem in computer architecture involves finding a mapping that performs best when executed. A function that evaluates a mapping could be used as a heuristic to optimize application layout. This is an apt problem for machine learning as analytical models previously developed to predict performance leave room for improvement (because the exact performance characteristics of this new processor design are unknown).*

We study five separate machine learning algorithms to model our simulation data. We observed four features of each mapping: operand latency, operand bandwidth, distributed data cache miss rate, and contention of processing elements. These four properties became our independent variables used as bases for the learning algorithms that attempted to predict our dependent variable: IPC (instructions per cycle).

We found that all five regression learning algorithms: Regression Trees, Cascade Correlation, Multivariate Adaptive Regression Splines (MARS), Multiple Regression, and Weighted Contribution, gave very similar correlations of predicted to actual IPC. Since a broad range of regression algorithms were used, this similarity indicates a limiting factor that is not part of any regression algorithm, but rather stems from the application.

1 Introduction

For the past several years technlogical innovation has provided processor designers with an enormous quantity of raw computational resources. Computer architects are exploring how to convert this opportunity into improvements in application performance. Despite differences in overall approach, raw technology, and execution models, five recently proposed architectures: nanoFabrics [1], TRIPS [2], RAW [3], SmartMemories [4], and WaveScalar [5] share the common trait of mapping large portions, sometimes even all, of an application onto a distributed collection of processing elements. Once mapped, the application executes "in place", explicitly communicating between these computational elements. Researchers call this form of computation distributed Instruction Level Parallelism (ILP) [2, 3, 5] or spatial computing [1].

Each of these systems must be programmed to accommodate and to take advantage of the distributed nature of the architecture. Our own research shows that performance can vary by as much as a factor of ten depending upon the application mapping used. How do developers, compiler writers, or micro-architects tune the layout of an application so that it executes quickly? To begin developing algorithms that construct or optimize layouts, one needs an accurate model of how these systems behave. Such a model provides the researcher with a foundation for understanding his system, and can also serve as an objective function for any optimization algorithm.

We focus on a particular architecture, WaveScalar, from this class of systems. The model comprises four components: inter-instruction operand latency, bandwidth constraints between producer and consumer instructions, interactions between instructions and the distributed data cache system, and resource contention for processing elements. Section 2 describes the WaveCache architecture and these four model components in more detail. This problem is well-suited to machine learning techniques, because these spatial computing architectures are relatively new and unstudied. While designers are able to identify factors (features) which should influence overall performance it is not known how they interact with each other.

2 Background

2.1 WaveCache Architecture

Instruction set architecture: WaveScalar is a dataflow architecture. Like all dataflow architectures (e.g. [6, 7, 8, 9, 10, 11, 12, 13]), its binary is a program's dataflow graph. Each node in the graph is a single instruction which computes a value and sends it to the instructions that consume it. Instructions execute after all input operand values have arrived, according to a principle known as the *dataflow firing rule* [6, 7]. Unlike other dataflow architectures, however, WaveScalar provides a program with a correct global ordering of all its memory operations. The details of this mechanism were previously introduced in [5].

Microarchitecture: Each static instruction in a program binary executes in a separate processing element (PE). Clearly, building a PE for each static instruction is both im-

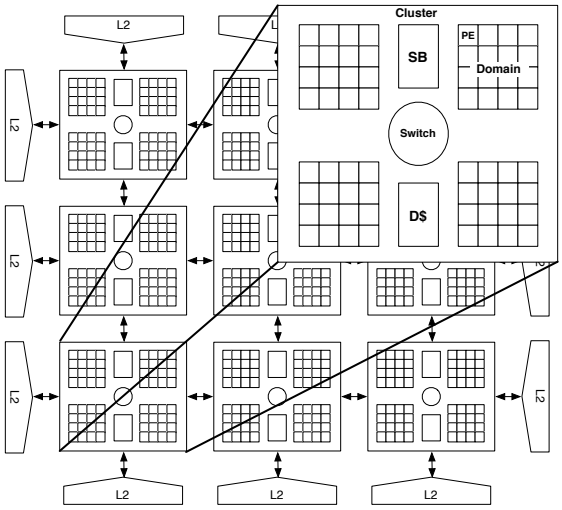


Figure 1: **The WaveCache and cluster:** A 3x3 WaveCache with nine clusters, and details of one cluster.

possible and wasteful, so, in practice, we dynamically bind multiple instructions to a fixed-number of PEs, and swap them in and out when more room is needed. We say that the PEs *cache* the working set of the application. Hence, the microarchitecture that executes WaveScalar binaries, essentially a grid of simple processing elements, is called a *WaveCache*. Figure 10 illustrates how a WaveScalar program can be mapped into a WaveCache.

To reduce communication costs within the grid, PEs are organized hierarchically, as depicted in Figure 1. PEs are first grouped into domains; within a domain, instructions execute and send their results to a consuming PE within a single cycle.

Four domains are then grouped into a cluster, which also contains wave-ordered memory hardware and a traditional L1 data cache. A single cluster, combined with an L2 cache and traditional main memory is sufficient to run any WaveScalar program. To build larger machines, multiple clusters are connected by an on-chip network and cache coherence is maintained by a simple, directory-based protocol that supports a single cache owner with no sharing. The coherence directory and the L2 cache are distributed around the edge of the grid of clusters.

2.2 Features

We have selected four features to present to the learning algorithms. We briefly present them now. More detailed descriptions and evaluations of the chosen features can be found in Appendix D.

Operand Latency An estimate of how many network hops are required for each operand to travel the network from producer to consumer. Each trip is weighted by the number of times it is made during program execution (according to the profile)

Operand Bandwidth Indicates the quality of bandwidth usage. It is calculated based on the expected load at each network link.

Distributed Data Cache Behavior Estimated L1 miss rate of the distributed data cache.

Processing Element Contention Rough estimate of the number of expected WaveCache misses. The WaveCache is designed to cache the working set of an application. A miss occurs when program execution requires an instruction which is not currently resident in the WaveCache and it must be fetched from memory.

3 Methodology

In this section we present our methodology for evaluating several machine learning algorithms when applied to the problem described in Section 2.

The data on which we train and evaluate each regression technique was gathered using six benchmarks (from a variety of suites including Spec2000 [14] and Splash2 [15]). Each application was mapped onto the WaveCache in eight different ways. The layout algorithms are described in more detail in the Appendix B. This resulted in fourty-eight datapoints. For each datapoint the four feature values were calculated. Each of these feature metrics, described in more detail in the Appendix D, has been demonstrated to correlate with the actual value of the performance component they were designed to model. The application was then simulated on a detailed cycle-by-cycle simulator of the WaveCache architecture to measure its performance in IPC (Instructions Per Cycle). [1]

Each of five regression techniques is described in Section 4 were applied to these data. For each regression technique we cross-validated our model, separating our 48 data points into training and test sets. In each case the test set consisted of the data for one of the six benchmark programs. For example we would learn a model on five of the six benchmarks and then evaluate it on the sixth. We chose to divide the data in this way to simulate how the model would perform on a benchmark it has never seen before. We learned six models using each regression technique, each one with a different benchmark "knocked out", on which the model was evaluated.

To evaluate the quality of a model we used the standard statistical correlation metric computed between a model's predicted application performance and actual simulated application performance. We chose this over other possible

[1] This data was originally gathered for a conference submission earlier this quarter. Feature selection, feature evaluation and all simulation was performed prior to this course project. For more details see Appendix A

measures (such as average absolute error) because of the intended use for these models. The ultimate purpose is to compare two potential layouts and choose the best of the two (for example this is done at each step while hill climbing or during simulated annealing). For this purpose it is more important that the relationship between the predicted and actual values be conserved, as opposed to the precision of the estimates. For this reason we use correlation as our evaluation function where 1 is perfect linear correlation and 0 is no correlation.

Evaluation and discussion of the results follows in Section 5.

4 Regression Algorithms

All regression learning algorithms build models that represent some set of training data and then use these models to predict values for future observations.

4.1 Regression Trees

Regression tree algorithms build decision tree models based on training data that can be used to predict the outcome of test data. Data is divided at each level in the tree so that each node represents a range of values learned from the independent variables in the training data. A path from the root to a leaf represents a series of tests on features of the data ending in a leaf node that represents a value for a particular outcome (dependent variable). Thus outcomes are predicted by starting at the root node and following branches left or right based on feature variables. Since both our feature values (Bandwidth, Contention, Data, and Latency) and our return value (IPC) are continuous rather than discrete, we used a variance splitting method which splits the data at each node so as to minimize the sum of the squared errors in the child nodes.

To test regression trees on our data, we used DTREG v3.5 [16]. This program allows the generation of Single Tree, Tree Boost, and Decision Tree Forest Models. The Single Tree model is as described above.

The Tree Boost Model builds several trees in a series. The first tree in the series is fitted to the data. It then feeds its residuals (error values) into the next tree in an attempt to resolve them. This is repeated so that a chain of successive trees are generated. Then, predicted outcomes are created by adding the weighted sum of each of the trees.

The Decision Tree Forest Model builds several independent trees: each one uses a subset of observations from the training data. To predict the outcome of test data, the average outcome from each of the trees in the forest is returned.

4.2 Cascade Correlation

Cascade Correlation is a learning algorithm that can be used on various neural networks to train the network to optimize its outcome. A neural network consists of layers of processing elements (nodes). The nodes in the first layer are fed the input variables and the node(s) in the last layer output the value of the outcome. All other layers are called hidden layers. The connections between nodes are weighted and these weights are adjusted until the outcome best matches the desired outcome from the training set. After the best possible weights are found, a new node is added. The cascade correlation algorithm adds only one hidden node at a time and that node is chosen from several candidate nodes that are linked to the input nodes and to all other hidden nodes in the network. Weights are optimized for these candidate nodes and then the best one is chosen to be incorporated into the network. This process repeats until the best possible network with the best possible weights is found.

Instead of waiting for this ideal situation, we limited the number of iterations to 100. Trail and error indicated that this was a good number as overfitting seemed to occur shortly after that.

We used the ThinksPro v1.05 [17] program to test Cascade Correlation using the Cascade architecture.

4.3 Multivariate Adaptive Regression Splines (MARS)

MARS builds a model of training data by fitting together several piecewise linear regressions so as to best match the data. This simulates a non-linear regression without the time and computational expense of non-linear regression algorithms, such as neural nets. The intervals are found by evaluating the significance of each feature (independent variable) one by one and building basis functions. Then the algorithm chooses best possible linear interactions between variables.

Training of our data in MARS was done with MARS v2.0 by Salford Systems [18].

4.4 Multiple Regression

Multiple Regression [19] finds the relationship between the outcome (dependent variable) and several features (independent variables) in the training set by defining a plane in n-space (where n is the number of features) that best fits the data points. Outcome predictions are then estimated by comparing features in the test set to the plane. The plane is found by testing the influence of each variable in turn. The influence of a variable v is determined by comparing the square of the correlation coefficient of all variables except v with the square of the correlation coefficient of all variables. The bigger the difference, the more significant v is. For this project,

we used Huberts method for the influence function with Hubert Constant = 1.345.

In addition to testing the influence of each individual variable, curvilinear and interactive relationships are also found. Curvilinear relationships include the relationship of each variable to itself by creating a new variable that is the square or cube of the variable being tested. In 2-way curvilinear relationships, individual variables are compared with the squares of those variables. In 3-way curvilinear relationships, individual variables are compared with the square and the cube of those variables. An n-way curvilinear relationship compares all variables with all n powers of each variable. Interactive relationships are similar to curvilinear except that instead of multiplying the values of variables with the same variable, variables are multiplied by all other variables in the data set. An n-way interaction compares all n combinations of variables in the data set.

This process estimates the coefficients (weights) of each variable and thus defines the plane of the model.

Multiple Regression was performed using the NCSS (Number Cruncher Statistical System) Trial Version from NCSS [20].

4.5 Weighted Contributions

This model simply combines features linearly with each feature value weighted by its *contribution*. Contribution is the importance of a feature to measured performance. [2] Higher contribution values suggest that the feature is an important factor in performance. Low contribution values suggest the component has no real bearing on performance.

5 Experimental Results

We used cross-validation to test the accuracy of each learning algorithm. Each learning algorithm was tested on five of the six different benchmarks and the resulting model was used to estimate the remaining benchmark for a total of six different tests per algorithm. Upon evaluation each regression method produced an estimate that had a correlation coefficient of .83-.87 with actual measured IPC. Because such a broad range of regression techniques are all so consistent with one another, we believe that what limits the performance of these techniques is probably not the techniques themselves, but rather the problem domain. It is possible that given this set of benchmarks and these four features, one cannot improve on .87 correlation.

5.1 Regression Trees

We created three different tree models of our data: Simple Trees, TreeBoost, and Decision Tree Forest. The Simple Tree provided us with a graphical representation of the resulting model, shown in Figure 4 (the other two are difficult to visualize and so were not generated). Each node shows the node number, the group of observations from the training set that it represents, the number of nodes in that group (N) and the sum of their weights (W), the IPC value at that node (our dependent variable), and the standard deviation for the mean IPC value. Note that we elected not to place weights on any of the features and thus N and W are the same in every node.

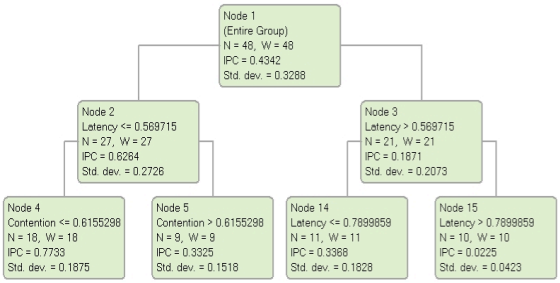


Figure 4: **Regression Tree Model:** Single regression tree trained on all six benchmarks. Each node shows the node number, the group of observations from the training set that it represents, the number of nodes in that group (N) and the sum of their weights (W), the IPC value at that node (the dependent variable), and the standard deviation for the mean IPC value.

During evaluation six different trees were produced, and across them all there is a common theme: Latency and Contention are the most significant variables in determining the IPC value with Data and Bandwidth rarely being used in the decision trees. This is consistent across all of the regression techniques which rank the significance of features. The pie charts in Figure 2 indicate the extent to which Latency and Contention were significant in the analysis of these algorithms. The algorithms vary primarily in how much share they award to Bandwidth and Data.

This remained a common theme in both the TreeBoost and Decision Tree Forest algorithms: Latency and Contention are found to be the most significant variables with Data and Bandwidth consistently less significant.

5.2 Cascade Correlation

We allowed the Cascade Correlation algorithm to run for 100 iterations (100 hidden nodes were added and weighted appropriately). While there is no easy visualization of the resulting model, this algorithm did reasonably well with an average correlation of predicted and actual IPC of 0.83. Although slightly lower, this correlation does not differ significantly from any of the other models. Despite being the most sophisticated of the techniques with which we experimented, these Neural Nets fared no better. This lends further credence to our conclusion that the problem is the factor lim-

[2] These values had been calculated previously, as stated in Appendix A.

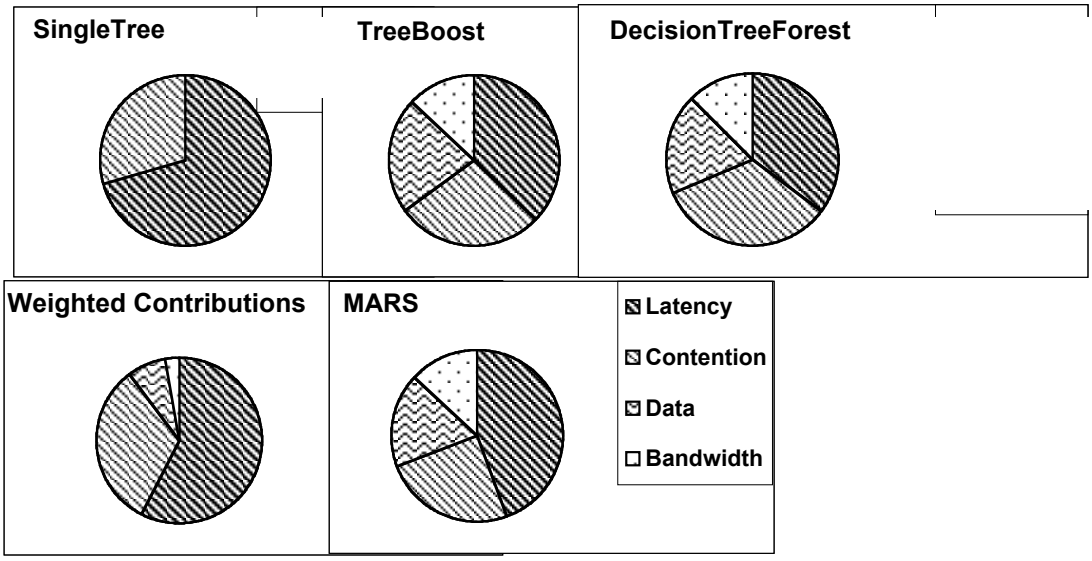
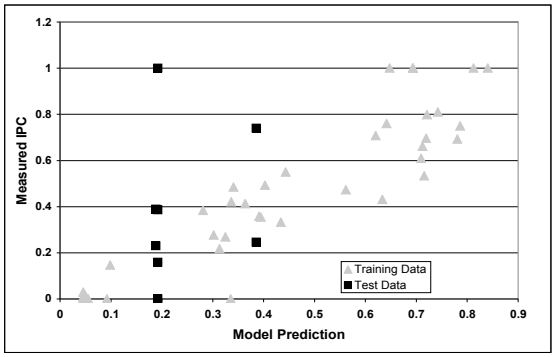
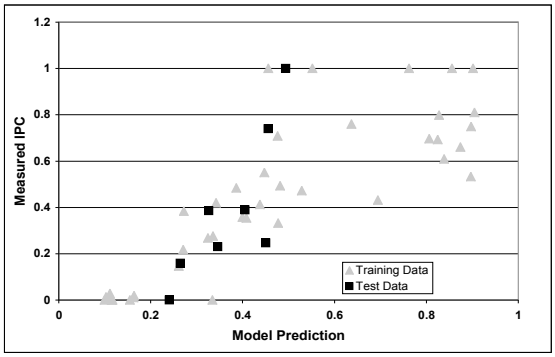


Figure 2: **Relative Feature Importance Scores**



	Correlation Coefficient					
	Single Tree		Tree Boost		Decision Tree Forest	
Benchmark	Training Data	Test Data	Training Data	Test Data	Training Data	Test Data
fft	0.92	0.00	0.93	0.19	0.93	0.19
ocean	0.90	0.82	0.94	0.75	0.94	0.78
equake	0.93	0.93	0.92	0.92	0.92	0.93
art	0.86	0.62	0.94	0.79	0.92	0.83
gzip	0.70	0.92	0.93	0.78	0.94	0.92
mcf	0.89	0.88	0.93	0.90	0.93	0.89
Average		0.70		0.72		0.76

Figure 3: **Regression Tree Data** On the bottom is a table showing the evaluation of six models on their training data and their test data. Above it is a graph detailing the correlation between predicted IPC, on the x-axis, and actual IPC, on the y-axis, for both the training data (all benchmarks except *fft*, gray triangles) and the test data (*fft*, black squares).



	Correlation Coefficient	
Benchmark	**Training Data**	**Test Data**
fft	0.90	0.81
ocean	0.91	0.70
equake	0.88	0.94
art	0.87	0.70
gzip	0.87	0.96
mcf	0.89	0.89
Average		0.83

Figure 5: **Cascade Correlation Data** On the right is a table showing the evaluation of six neural networks on their training and test data. The graph on the left is a plot of Actual IPC v. Predicted IPC for the model evaluated in the first row of the table: training data is all benchmarks except *fft* (gray triangles), test data is *fft* (black squares).

iting the quality of these models.

5.3 Multivariate Adaptive Regression Splines (MARS)

The models that we obtained by running our data with the MARS algorithm consist of piecewise linear regressions. The regressions shown in Figure 7, represent the model created using all of the data (without cross-validation). Notice that Latency, Contention, and Bandwidth all have inverse relationship with the target variable (IPC) (while Data is directly related) when the algorithm finds them to be significant contributors.

5.4 Multiple Regression

We experimented with 1-way, 2-way, and 3-way Multiple Regression (with both curvilinear and interactive relationships). The results of these experiments are shown in the table in Figure 8. As we increase the expressiveness of the model (moving from 1-way to 3-way) we see improving correlation of the model to the training data (from approximately .9 correlation to .98) but a degrading correlation of the model to the test data (from .87 to .36). This is a clear sign of overfitting, with the model predicting excellently the values on which it was trained but lacking the generality to predict anything about previously unseen data. Furthermore, on inspection of each 3-way model, one sees a much larger range in the coefficients than one sees in the 1-way models (Appendix F). This is a sign of generality in the 1-way models that is lacking in the 3-way models.

5.5 Weighted Contributions

Contribution is the importance of the component to performance. We computed this by calculating the variance of the simulated IPC dividing it by the average IPC. In measuring these IPC values the simulator was configured such that all architectural components, except those effecting the feature we sought to measure, were idealized. Higher contribution values suggest that component, independent of the performance model, is an important factor in performance. Low contributions values suggest the component has no real bearing on performance.

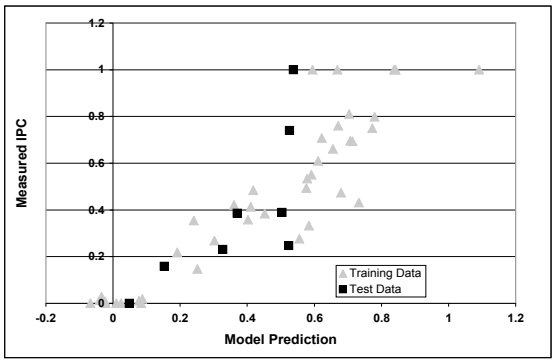
$$\begin{aligned} IPC \quad = \quad & Contrib_{latency} \times Comp_{Latency} \\ & Contrib_{bandwidth} \times Comp_{bandwidth} \\ & Contrib_{data} \times Comp_{data} \\ & Contrib_{PeContention} \times Comp_{PeContention} \end{aligned} \tag{1}$$

Based on experimental measures it was found that found that $Contrib_{latency} = 0.57$, $Contrib_{bandwidth} = 0.03$, $Contrib_{data} = 0.07$, and $Contrib_{PeContention} = 0.32$.

6 Future Work

One approach for future research would be to apply some of these techniques to learn the equations for the features. It will be trickier deciding on what features to base this learning, however the feature calculations shown in Appendix D are acknowledged simplifications of actual program behavior. The reason they are simplified is because architects are sure how to describe the more complex behaviors. Perhaps machine learning could provide some insights in this domain.

While this project certainly did not attempt to comprehensively explore all machine learning algorithms in the field, it



	Correlation Coefficient	
Benchmark	**Training Data**	**Test Data**
fft	0.92	0.74
ocean	0.91	0.82
equake	0.84	0.94
art	0.88	0.78
gzip	0.89	0.83
mcf	0.82	0.90
Average		0.83

Figure 6: **MARS Data** On the right is a table showing the evaluation of MARS models on their training and test data. The graph on the left is a plot of Actual IPC v. Predicted IPC for the model evaluated in the first row of the table: training data is all benchmarks except *fft* (gray triangles), test data is *fft* (black squares).

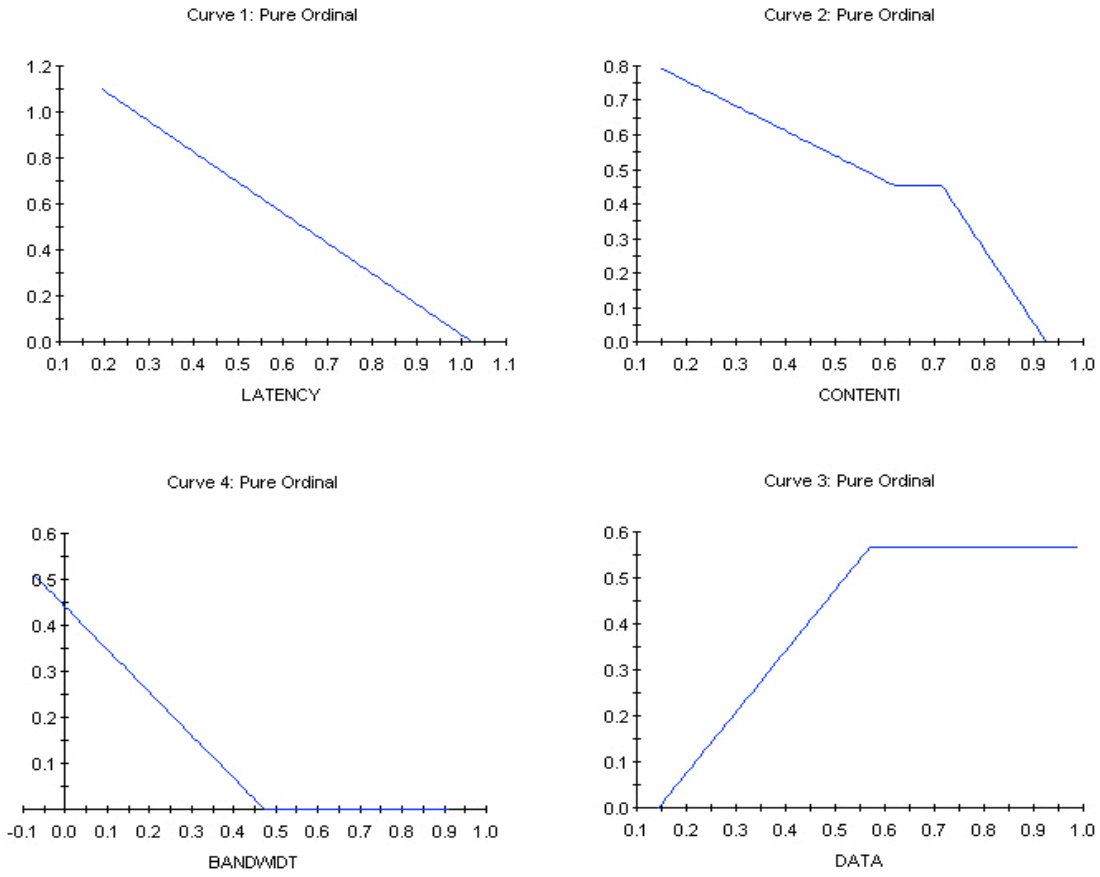
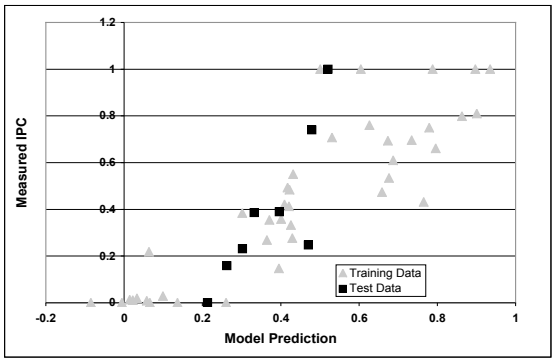
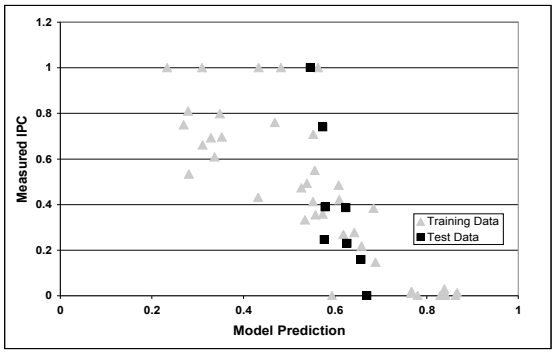


Figure 7: **Piecewise Linear Features:** This output from the MARS software graphs the relationship of each of the four features to IPC.



	Correlation Coefficient					
	One Way		Two Way		Three Way	
Benchmark	**Training Data**	**Test Data**	**Training Data**	**Test Data**	**Training Data**	**Test Data**
fft	0.90	0.83	0.94	0.74	0.99	0.05
ocean	0.91	0.78	0.92	0.87	0.98	0.62
equake	0.87	0.95	0.92	0.82	0.98	-0.37
art	0.89	0.77	0.90	0.62	0.98	0.69
gzip	0.87	0.98	0.92	0.38	0.98	0.25
mcf	0.88	0.92	0.92	0.79	0.98	0.91
Average		0.87		0.70		0.36

Figure 8: **Multiple Regression Data** On the bottom is a table showing the evaluation of eighteen models on both their training data and their test data. Moving from left to right the models increase in their expressiveness. With this we witness an increase in training data correlation. However we also see a marked degradation in test data correlation, indicating that the more expressive model tends to overfit the training data. Above is a graph detailing the correlation between "One Way" predicted IPC, on the x-axis, and actual IPC, on the y-axis, for both the training data (all benchmarks except *fft*, gray triangles) and the test data (*fft*, black squares).



	Correlation Coefficient	
Benchmark	**Training Data**	**Test Data**
fft	-0.86	-0.83
ocean	-0.89	-0.68
equake	-0.83	-0.95
art	-0.84	-0.88
gzip	-0.85	-0.87
mcf	-0.84	-0.90
Average		-0.85

Figure 9: **Weighted Contributions Data** On the right is a table evaluating the quality of the Weighted Contribution combination of features. On the left is a plot of Actual IPC v. Predicted IPC for the model evaluated in the first row of the table: training data is all benchmarks except *fft* (gray triangles), test data is *fft* (black squares). Note that this regression technique produced a strong but inverse correlation with respect to the others.

is interesting that each of the algorithms used here resulted in very similar correlations. Further research could evaluate other algorithms that may be better suited to this type of problem or confirm our hypothesis that perhaps this problem can only be learned to the extent that we have shown here.

Also, many of the software packages used in this project provided several parameters that could be tuned to best suit a particular problem or data set. On several occasions, the author accepted default setting for the purpose of brevity. A more in depth evaluation of each program (as opposed to the breadth study presented here) would find best possible parameters using a more in depth understanding of the program. Examples of parameters that could influence results are the number of allowable nodes in the regression trees or the cascade correlation network. Also, the number of iterations for which the cascade correlation is allowed to run would effect both accuracy and overfitting.

Finally the six benchmark programs used for this project were chosen in order to incorporate a broad collection of application properties. If more benchmark programs with different characteristics were chosen, the learning algorithms may benefit from the increased knowledge of a bigger and wider training set.

7 Conclusion

We have applied a broad selection of regression algorithms to a problem inspired by recently proposed, but relatively unstudied, spatial computation architectures. We found the range predictive powers of each algorithm (as measured by correlation of predicted to actual performance) to be quite small, with correlation coefficients varying from .83 (MARS and Cascade Correlation) to .87 (1-Way Multiple Regression). The only time the correlation exceeded .87 was for overfit models run on their training data. Based on this small variance across algoriths we believe that prediction quality is not limited by any specific regression algorithm, but rather by the inherent difficulty of the architecture problem.

References

[1] S. C. Goldstein and M. Budiu, "Nanofabrics:spatial computing using molecular electronics," in *Proceedings of the 28th annual international symposium on Computer architecture*, pp. 178–191, 2001.

[2] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," in *Proceedings of the 30th annual international symposium on Computer architecture*, 2003.

[3] W. Lee *et al.*, "Space-time scheduling of instruction-level parallelism on a Raw machine," in *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS-VIII*, October 1998.

[4] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. Dally, and M. Horowitz, "Smart memories: A modular reconfigurable architecture," in *International Symposium on Computer Architecture*, 2002.

[5] S. Swanson, K. Michelson, A. Schwerin, and M. Oskin, "WaveScalar," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, p. 291, 2003.

[6] J. B. Dennis, "A preliminary architecture for a basic dataflow processor," in *Proceedings of the 2nd Annual Symposium on Computer Architecture*, 1975.

[7] A. L. Davis, "The architecure and system method of DDM1: A recursively structured data driven machine," in *Proceedings of the 5th Annual Symposium on Computer Architecture*, (Palo Alto, California), pp. 210–215, IEEE Computer Society and ACM SIGARCH, April 3–5, 1978.

[8] S. Sakai, y. Yamaguchi, K. Hiraki, Y. Kodama, and T. Yuba, "An architecture of a dataflow single chip processor," in *Proceedings of the 16th annual international symposium on Computer architecture*, pp. 46–53, ACM Press, 1989.

[9] T. Shimada, K. Hiraki, K. Nishida, and S. Sekiguchi, "Evaluation of a prototype data flow processor of the sigma-1 for scientific computations," in *Proceedings of the 13th annual international symposium on Computer architecture*, pp. 226–234, IEEE Computer Society Press, 1986.

[10] J. R. Gurd, C. C. Kirkham, and I. Watson, "The manchester prototype dataflow computer," *Communications of the ACM*, vol. 28, no. 1, pp. 34–52, 1985.

[11] M. Kishi, H. Yasuhara, and Y. Kawamura, "Dddp-a distributed data driven processor," in *Conference Proceedings of the tenth annual international symposium on Computer architecture*, pp. 236–242, IEEE Computer Society Press, 1983.

[12] V. G. Grafe, G. S. Davidson, J. E. Hoch, and V. P. Holmes, "The epsilon dataflow processor," in *Proceedings of the 16th annual international symposium on Computer architecture*, pp. 36–45, ACM Press, 1989.

[13] G. Papadopoulos and D. Culler, "Monsoon: An explicit token-store architecture," in *Proceedings of the 17th International Symposium on Computer Architecture*, May 1990.

[14] SPEC, "Spec CPU 2000 benchmark specifications." SPEC2000 Benchmark Release, 2000.

[15] D. Buell *et al.*, *Splash 2: FPGAs in a Custom Computing Machine*. IEEE Computer Society, 1996.

[16] P. Sherrod, "Dtreg software package," 2004. http://www.dtreg.com/.

[17] I. Logical Designs Consulting, "Thinkspro software package," 2004. http://www.sigma-research.com/bookshelf/rtthinks.htm.

[18] S. Systems, "Mars software package," 2004. http://www.salford-systems.com/mars.php.

[19] M. Irani and P. Anandan, "Robust multi-sensor image alignment," in *ICCV*, pp. 959–966, 1998.

[20] N. C. S. S. N. Company, "Ncss and pass data analysis systems," 2004. http://www.ncss.com/.

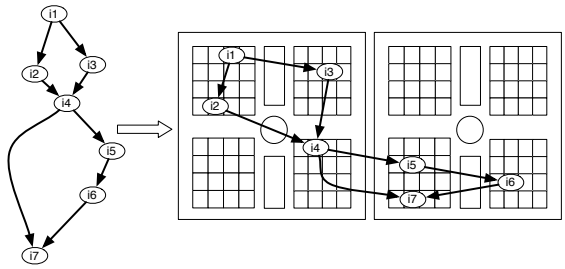


Figure 10: **WaveScalar Application Layout:**

APPENDIX

A Contributors

All data was gathered prior to this course project. This includes

1. Layout algorithm design and implementation
2. Feature selection and validation (Appendix D
3. Simulation of each benchmark with each possible application layout

The raw data with which we began this work is shown in Appendix E, Table 1.

For this project we split all tasks evenly. Anna gathered data for three regression algorithms (Regression Trees, MARS, Cascade Correlation) while Martha covered (Multiple Regression and Weighted Contributions). Writing, data preparation and analysis of results was also divided evenly.

B Instruction Layouts

In the general case an application layout is a mapping of computation elements composing the application to specific locations in a regular computational substrate. In the specific case of WaveScalar, a small application layout is a mapping of WaveScalar instructions (dataflow nodes) onto a specific processing element in the WaveCache.

Figure 10 illustrates an application and a sample layout. For this study we concentrate only on *static* instruction layouts. In a static layout each instruction is assigned to a processing element prior to program execution, and this assignment does not change during execution. The WaveCache demand loads instructions and for this study always assigns them to the location chosen for the layout. As more instructions can be assigned to a single location than that location can physically handle, instructions are swapped out of locations using an LRU algorithm.

This study is based on simulation results from eight different instruction placements. Here we briefly describe the placements and the algorithm used to generate them:

random Each instruction is assigned to a randomly chosen PE anywhere in the WaveCache.

packed-random Each instruction is assigned to a randomly chosen PE from a restricted set of contiguous domains. The size of this set is the minimal number of domains required to hold all of the program instructions.

static-stripe Instructions are assigned to PEs in static program order, creating stripes across the WaveCache.

depth-first-stripe This is a depth-first search based algorithm which computes a pre-order traversal ordering of the instructions in the DFG. It then assigns each segment of 16 instructions to a PE. The goal of this algorithm is to place strands of sequential, data-dependent instructions all in the same PE, as these are instructions which would not be able to fire in parallel and therefore can best share a PE.

dynamic-stripe : Instructions are assigned in dynamic program order to each processing element.

over-2-DFS This is the same depth first search algorithm used for DEPTH-FIRST-STRIPE, except that twice as many instructions are assigned to PEs as the hardware can hold at any point in time.

over-4-DFS This is the same as OVER-2-DFS except four times as many instructions are used.

over-8-DFS This is the same as OVER-8-DFS except that eight times as many instructions are used.

We should note that these layout algorithms are used for a variety of reasons in this study. The RANDOM and RANDOM-PACKED "algorithms" represent very naive layouts which perform very poorly. The STATIC-STRIPE layout represents something that a prefetching algorithm could easily accomplish. The DEPTH-FIRST-STRIPE algorithm attempts to utilize the sequential nature of execution of dependent instructions in a productive way for layout. The DYNAMIC-STRIPE is our current overall best performing placement algorithm. Finally, the over-subscribed DFS algorithms are used to explore contention effects in the architecture.

C Application Profile

An application profile of a WaveScalar binary is an annotated dataflow graph of the program. Figure 11 illustrates an application profile. The nodes represent program instructions, and the directed edges that connect them indicate operand producer-consumer relationships. These edges are annotated with the number of times an operand was passed from producer to consumer during profiled execution.

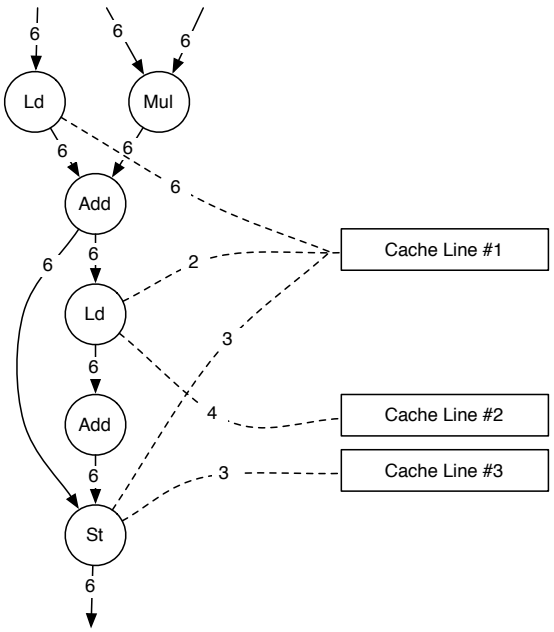


Figure 11: **WaveScalar Application Profile**

The profile also adds a second type of node, an address node, each of which represents an address (of cache-line size granularity) in memory that was accessed by the application. Address nodes are connected to the instructions which access them by directed edges. These edges also have weights indicating the number of profiled accesses.

D Feature Details

In describing the four layout features we use the following common set of variables:

n is the number of instructions in the application.

i refers to the i'th instruction, j refers to the j'th instruction.

C_i is the x, y physical location of the cluster containing instruction i. Cx_i, and Cy_i are the components individually. Similarly, D_i and P_i are the location of the domain and processing element, respectively, where instruction i is placed.

$T_{i,j}$ refers the amount of communication (or traffic) between instructions i and j.

A is the set of cache-line size addresses the application uses.

a refers to a particular cache-line size address.

$M_{i,a}$ refers to the number of memory accesses between instruction i and memory address a.

$M_{C,a}$ refers to the number of memory accesses between all instructions in cluster C and memory address a. $M_{C,a}$ can be defined in terms of $M_{i,a}$: $M_{C,a} = \sum_{i \in C} M_{i,a}$.

Many times throughout the text it important to refer to the distance between two items. In these cases, we use the Manhattan-distance in terms of clusters, domains or processing elements. This distance, $||C_i - C_j||$ is simply calculated as $|Cx_i - Cx_j| + |Cy_i - Cy_j|$.

D.1 Feature: Operand Latency

The latency of operand traffic in this micro-architecture can be modeled as the distance between the producer and consumer of the operand. Considering any two instructions, i and j, the latency between them is dependent on their position:

$$Latency_{i,j} = \begin{cases} 0 & \text{if } D_i = D_j, \\ C_{i,j} \cdot (||C_i - C_j|| + 2) & \text{otherwise.} \end{cases} \quad \text{(A-1)}$$

The total latency incurred by operand traffic is therefore the summation of this value for each pair of instructions multiplied by the quantity of communication between them:

$$Latency = \sum_i \sum_j T_{i,j} \times Latency_{i,j} \quad \text{(A-2)}$$

D.2 Feature: Operand Bandwidth

We estimate the bandwidth demands at each network link in the following way: Given the routing algorithm, when instruction i sends an operand to instruction j many possible links between their respective locations (C_i and C_j) could be used. Each operand creates demand on a rectangular region of network switches. Overlaying all of these rectangles for all operand communication in the application, we can estimate the total expected bandwidth requirements at each switch.

Across many messages, the aggregate bandwidth utilization across these links is a predictable function of the layout and profile data. The expected load, $Load_{x,y,i,j}$, at each switch is simply the sum of incoming link loads:

$$Load_{x,y,i,j} = \begin{cases} 0 \\ \quad \text{when} \quad x,y \notin BoundingBox(C_i,C_j) \\ T_{i,j} \\ \quad \text{when} \quad x,y = C_i \vee x,y = C_j \\ Load_{x-1,y,i,j} + Load_{x,y-1,i,j}/2 \\ \quad \text{when} \quad x = Cx_j, y \neq Cy_j \\ Load_{x-1,y,i,j}/2 + Load_{x,y-1,i,j} \\ \quad \text{when} \quad x \neq Cx_j, y = Cy_j \\ Load_{x-1,y,i,j}/2 + Load_{x,y-1,i,j}/2 \\ \quad \text{when} \quad x \neq Cx_j, y \neq Cy_j \end{cases} \tag{A-3}$$

In this set of equations, $BoundingBox(C_i, C_j)$ is simply the set of network switches that traffic may flow over from C_i to C_j. We have also simplified things slightly by assuming that C_i is the upper-left and C_j is the lower right corners of this box. A similar (not shown) set of equations are used to compute the $Load$ when this is not the case. The origin is placed at the upper left of the WaveCache.

Using this model we can then estimate the total amount of network traffic that passes through a particular network switch:

$$Load_{x,y} = \sum_{i,j} Load_{x,y,i,j}$$

Experiments have shown that the best scoring function for a set of estimate link work loads is PEAK-ABOVE-AVERAGE: the maximum amount by which any one expected workload exceeds the average workload ($max_{x,y}(Load_{x,y}) - AverageLoad$).

D.3 Feature: Distributed Data Cache Behavior

$$Misses_a = \sum_C \begin{cases} 1 & \text{if } M_{C,a} > 0, \\ 0 & otherwise. \end{cases} \tag{A-4}$$

$$Hits_a = \left(\sum_C M_{C,a} \right) - Misses_a \tag{A-5}$$

$$Misses = \sum_a Misses_a \tag{A-6}$$

$$Hits = \sum_a Hits_a \tag{A-7}$$

D.4 Feature: Processing Element Contention

We use a simple model to estimate the amount of contention there will be among instructions for available slots in their PEs. The estimate is the number of instructions which cannot be resident in the WaveCache at any time. In the following equations $PeCapacity$ is the number of instructions which any PE can hold, and I_p is the set of instructions assigned to processing element P.

$$PeContention = \sum_P \begin{cases} |I_P| - PeCapacity \\ \quad \text{when} \quad |I_P| > PeCapacity \\ 0 \\ \quad \text{otherwise} \end{cases} \tag{A-8}$$

Despite the model's simplicity, it is highly effective at modeling resource contention in the WaveCache.

E Data Tables

F Multiple Regression Models

F.1 One Way Models

Trained on all data:

$$\begin{aligned} Estimate \quad &= 1.35757923490309 \\ &-2.22283902899033 \times L \\ &-.364895145130047 \times B \\ &+1.58087577607372 \times D \\ &-.625146295399346 \times C \end{aligned} \tag{A-9}$$

Trained on all except *fft*:

$$\begin{aligned} Estimate \quad &= 1.30060757902123 \\ &-2.1098895690747 \times L \\ &-.365835704298298 \times B \\ &+1.4405402246403 \times D \\ &-.50383304413062 \times C \end{aligned} \tag{A-10}$$

Trained on all except *ocean*:

$$\begin{aligned} Estimate \quad &= 1.32869546348535 \\ &-2.05498256078302 \times L \\ &-.408708364098947 \times B \\ &+1.38739858978484 \times D \\ &-.515705049197833 \times C \end{aligned} \tag{A-11}$$

Trained on all except *equake*:

$$\begin{aligned} Estimate \quad &= 1.4605725268469 \\ &-2.2077997138043 \times L \\ &-.43019502557154 \times B \\ &+1.61817106184067 \times D \\ &-.755147396431784 \times C \end{aligned} \tag{A-12}$$

Trained on all except *art*:

$$\begin{aligned} Estimate \quad &= 1.33370858312509 \\ &-2.45740966634696 \times L \\ &-.20464809314502 \times B \\ &+1.78114142673435 \times D \\ &-.733231179130816 \times C \end{aligned} \tag{A-13}$$

		Feature Values				
Benchmark	**Layout**	**Latency**	**Bandwidth**	**Data Cache**	**PE Contention**	**IPC**
FFT	exp4	0.569552716	0.542023309	0.589768726	0.463870778	1
FFT	dfs	0.578689802	0.652664031	0.599188127	0.45292937	0.739741817
FFT	stripe	0.580717902	0.6646903	0.599398556	0.45292937	0.246423266
FFT	randomPacking	0.674783883	0.39734024	0.639316305	0.518886897	0.389060865
FFT	randomSpreading	0.730801538	0.542023309	0.669835961	0.453856608	0.230488046
FFT	dfs2	0.571921426	0.6646903	0.58400388	0.723064698	0.385712822
FFT	dfs4	0.572739964	0.68772699	0.589794031	0.858565073	0.158649659
FFT	dfs8	0.570716293	0.698765047	0.57861794	0.925820733	0
OCEAN	exp4	0.661067928	0.585547383	0.666793063	0.466243844	0.483918805
OCEAN	dfs	0.526540395	0.632220824	0.569861786	0.560035451	1
OCEAN	stripe	0.462099295	0.585547383	0.57359705	0.560035451	0.472554418
OCEAN	randomPacking	0.91339994	0.632220824	0.808988323	0.560035451	0.018661675
OCEAN	randomSpreading	0.920985774	0.61626494	0.808361912	0.560035451	0.010578864
OCEAN	dfs2	0.505887256	0.632220824	0.560662827	0.560035451	0.707741438
OCEAN	dfs4	0.483729634	0.632220824	0.506594291	0.757795086	0.358080975
OCEAN	dfs8	0.474753604	0.632220824	0.453604575	0.92424764	0
equake	exp4	0.205390632	0.370417422	0.284701991	0.489470013	1
equake	dfs	0.293800159	0.315053659	0.244241454	0.489470013	0.692720505
equake	stripe	0.274768371	0.430118034	0.288290508	0.489470013	0.695912381
equake	randomPacking	0.849170354	0.429271973	0.880802573	0.461286401	0.146418565
equake	randomSpreading	1.022943704	0.428015414	0.962800912	0.486880168	0
equake	dfs2	0.291670876	0.462691405	0.247769517	0.371904091	0.609225312
equake	dfs4	0.252215446	0.472966353	0.247764071	0.31305584	0.660470714
equake	dfs8	0.195382817	0.476808099	0.228971333	0.283805821	0.809910163
art	exp4	0.547229025	0.165696588	0.570116849	0.611538911	1
art	dfs	0.626063774	0.868331361	0.660845125	0.627298194	0.382813322
art	stripe	0.547359277	0.852848047	0.630590273	0.627298194	0.276166422
art	randomPacking	1.005997719	0.741239159	0.918136837	0.64058152	0
art	randomSpreading	0.986354852	0.762717294	0.963457136	0.639701441	0.01282984
art	dfs2	0.59290902	0.528750525	0.630547946	0.715540789	0.420058292
art	dfs4	0.569877286	0.591716002	0.600281	0.767547595	0.267816045
art	dfs8	0.547172813	0.91166479	0.4489886	0.793457123	0.217352313
gzip	exp4	0.446996087	0.611542865	0.505061575	0.148579657	1
gzip	dfs	0.529031331	0.567415087	0.529538068	0.61952071	0.549878445
gzip	stripe	0.520499185	0.509947844	0.498242141	0.61952071	0.332026228
gzip	randomPacking	0.997844271	0.632297638	0.96995167	0.61952071	0.008076393
gzip	randomSpreading	0.99704772	0.629970796	0.97352276	0.61952071	0
gzip	dfs2	0.481256584	0.631924028	0.466471586	0.61952071	0.492770986
gzip	dfs4	0.44909271	0.635484507	0.468915913	0.751944868	0.412106407
gzip	dfs8	0.428947417	0.632132539	0.439011591	0.852587229	0.354426237
mcf	exp4	0.240202669	-0.070698534	0.262256578	0.528150663	1
mcf	dfs	0.357182615	0.677782218	0.410885158	0.528150663	0.759378517
mcf	stripe	0.333085402	0.536827159	0.435472252	0.528150663	0.431175115
mcf	randomPacking	0.9724043	0.792326256	0.891093863	0.507974414	0
mcf	randomSpreading	1.005944484	0.682818906	0.989576064	0.521686066	0.028526346
mcf	dfs2	0.273616372	0.438711028	0.350761743	0.41094957	0.798626174
mcf	dfs4	0.272906994	0.225653701	0.216536403	0.352227928	0.748907567
mcf	dfs8	0.244963277	0.416885379	0.143724052	0.323016147	0.533080168

Table 1: **Feature and IPC Data:** normalized per application

Trained on all except *gzip*:

$$\begin{aligned} Estimate &= 1.32403189804916 \\ &-2.13454211367373 \times L \\ &-.421754386750501 \times B \\ &+1.52693757990869 \times D \\ &-.551681836347069 \times C \end{aligned} \tag{A-14}$$

Trained on all except *mcf*:

$$\begin{aligned} Estimate &= 1.42500499246307 \\ &-2.31806965197578 \times L \\ &-.355304559780754 \times B \\ &+1.64061514333014 \times D \\ &-.691967173027319 \times C \end{aligned} \tag{A-15}$$

F.2 Two Way Models

Trained on all data:

$$\begin{aligned} Estimate &= .646846726382973 \\ &-4.79998581607504 \times L \\ &+.176845337194203 \times B \\ &+4.88318811674013 \times D \\ &+.995075188754996 \times C \\ &-12.2070798918943 \times L \times L \\ &+2.4360578096496 \times L \times B \\ &+22.9714475551078 \times L \times D \\ &+5.38752108312524 \times L \times C \\ &+.711761275075665 \times B \times B \\ &-2.51071872533098 \times B \times D \\ &-2.10981799250765 \times B \times C \\ &-12.1509595565399 \times D \times D \\ &-3.80031888334303 \times D \times C \\ &-1.20546840747366 \times C \times C \end{aligned} \tag{A-16}$$

Trained on all except *fft*:

$$\begin{aligned} Estimate &= .552995986891075 \\ &-3.98553196729973 \times L \\ &+.289959992199797 \times B \\ &+4.31052298441198 \times D \\ &+1.03183895921818 \times C \\ &-12.8598438724197 \times L \times L \\ &+1.81972041157642 \times L \times B \\ &+24.8830377940944 \times L \times D \\ &+4.34780916931938 \times L \times C \\ &+.895718470427828 \times B \times B \\ &-2.06412020037136 \times B \times D \\ &-2.4316046346246 \times B \times C \\ &-13.5271371846747 \times D \times D \\ &-2.81218215685294 \times D \times C \\ &-1.09507477718901 \times C \times C \end{aligned} \tag{A-17}$$

Trained on all except *ocean*:

$$\begin{aligned} Estimate &= .569095473436304 \\ &-4.9757995755376 \times L \\ &+.458843091170079 \times B \\ &+4.80577123632488 \times D \\ &+1.07174055132507 \times C \\ &-8.54289020338866 \times L \times L \\ &+1.03964420258117 \times L \times B \\ &+15.9362237070587 \times L \times D \\ &+7.34993033639456 \times L \times C \\ &+.685803545713162 \times B \times B \\ &-1.02740557628383 \times B \times D \\ &-2.78360441864369 \times B \times C \\ &-8.40808457123213 \times D \times D \\ &-6.25324958308374 \times D \times C \\ &-.549639589652148 \times C \times C \end{aligned} \tag{A-18}$$

Trained on all except *equake*:

$$\begin{aligned} Estimate &= 1.25344423992606 \\ &-4.91804639663961 \times L \\ &+.114583279647884 \times B \\ &+4.99350398979705 \times D \\ &-.859171350291234 \times C \\ &-21.0345257441464 \times L \times L \\ &+5.54963962355569 \times L \times B \\ &+41.8332328284203 \times L \times D \\ &+.792737791472825 \times L \times C \\ &+1.1780490915126 \times B \times B \\ &-7.34730383709171 \times B \times D \\ &-1.06964849396972 \times B \times C \\ &-21.8169651259802 \times D \times D \\ &+1.82757178942486 \times D \times C \\ &-.85504598655848 \times C \times C \end{aligned} \tag{A-19}$$

Trained on all except *art*:

$$\begin{aligned} Estimate &= .856067789756287 \\ &-.979157164517814 \times L \\ &-.38368675695592 \times B \\ &+.877297195292073 \times D \\ &+.915797588427185 \times C \\ &-13.9955464922535 \times L \times L \\ &-5.45197829276944 \times L \times B \\ &+29.9806307306872 \times L \times D \\ &+1.87717337142577 \times L \times C \\ &+.198322636226673 \times B \times B \\ &+6.93156643446531 \times B \times D \\ &-1.32295400573736 \times B \times C \\ &-17.315762348572 \times D \times D \\ &-1.69348813360628 \times D \times C \\ &-.978534803573938 \times C \times C \end{aligned} \tag{A-20}$$

Trained on all except *gzip*:

$$
\begin{aligned}
Estimate \quad &= .355506850134312 \\
&-4.42472971596117 \times L \\
&-.162272097245777 \times B \\
&+4.00189469730502 \times D \\
&+2.93679058475312 \times C \\
&-18.9676637014647 \times L \times L \\
&+4.1034221146722 \times L \times B \\
&+36.7542762635053 \times L \times D \\
&+3.24665865384726 \times L \times C \\
&+.299299830175597 \times B \times B \\
&-4.50226629281317 \times B \times D \\
&-.47865908473717 \times B \times C \\
&-19.1189540422086 \times D \times D \\
&-.719749244745123 \times D \times C \\
&-3.89415483937944 \times C \times C
\end{aligned}
\tag{A-21}
$$

Trained on all except *mcf*:

$$
\begin{aligned}
Estimate \quad &= 1.60863738059912 \times C \\
&-16.0730518798983 \times L \times L \\
&+4.3526228804108 \times L \times B \\
&+35.1917120135489 \times L \times D \\
&+6.851001483146 \times L \times C \\
&+1.30784287321571 \times B \times B \\
&-3.55091180534156 \times B \times D \\
&-1.78275283938774 \times B \times C \\
&-20.7851190396177 \times D \times D \\
&-6.54037036267939 \times D \times C \\
&-1.31392600238978 \times C \times C
\end{aligned}
\tag{A-22}
$$

F.3 Three Way Models

Trained on all data:

$$
\begin{aligned}
Estimate \quad &= \\
&-5.58277372399634 \\
&-6.26463481290477 \times L \\
&+.104091496811669 \times B \\
&+13.7915581694172 \times D \\
&+33.0181783684145 \times C \\
&-22.2948524114537 \times L \times L \\
&+106.559605887342 \times L \times B \\
&-63.7468571048466 \times L \times D \\
&-4.82714210110338 \times L \times C \\
&+40.2289060571519 \times B \times B \\
&-136.720036317505 \times B \times D \\
&-78.1156004322207 \times B \times C \\
&+59.2604860587323 \times D \times D \\
&+73.4907266029237 \times D \times C \\
&-49.3400152178979 \times C \times C \\
&+496.289170413332 \times L \times L \times L \\
&-480.258949343188 \times L \times L \times B \\
&-1470.94424028762 \times L \times L \times D \\
&+420.215292526403 \times L \times L \times C \\
&-364.930687409122 \times L \times B \times B \\
&+1193.70212290249 \times L \times B \times D \\
&+396.217775912385 \times L \times B \times C \\
&+1627.2503076241 \times L \times D \times D \\
&-1265.75911481958 \times L \times D \times C \\
&-17.5726976085011 \times L \times C \times C \\
&+18.1060918781153 \times B \times B \times B \\
&+348.438505850582 \times B \times B \times D \\
&-113.275651123336 \times B \times B \times C \\
&-687.341008897365 \times B \times D \times D \\
&-368.893968911273 \times B \times D \times C \\
&+192.336057286245 \times B \times C \times C \\
&-653.012432967894 \times D \times D \times D \\
&+860.805885401155 \times D \times D \times C \\
&-72.9054324762932 \times D \times C \times C \\
&-17.0676171233299 \times C \times C \times C
\end{aligned}
\tag{A-23}
$$

Trained on all except *fft*:

$$
\begin{aligned}
Estimate \;=\; & \\
& -6.70489911359652 \\
& -77.358183002975 \times L \\
& -30.7258748027209 \times B \\
& +66.4207340166738 \times D \\
& +81.9712477215603 \times C \\
& -215.507398558395 \times L \times L \\
& +150.614048279455 \times L \times B \\
& +295.306217110512 \times L \times D \\
& +225.844533295334 \times L \times C \\
& +109.183829017832 \times B \times B \\
& -214.535196528692 \times B \times D \\
& -65.1890297446678 \times B \times C \\
& -89.0708245170768 \times D \times D \\
& -98.7305317507319 \times D \times C \\
& -163.859283263678 \times C \times C \\
& +612.267491927846 \times L \times L \times L \\
& -446.225450347633 \times L \times L \times B \\
& -2001.34915368313 \times L \times L \times D \\
& +981.896951753312 \times L \times L \times C \\
& -207.780831359504 \times L \times B \times B \\
& +919.005500103044 \times L \times B \times D \\
& +182.092085685427 \times L \times B \times C \\
& +2328.04837094334 \times L \times D \times D \\
& -2119.53766627545 \times L \times D \times C \\
& -215.506945374313 \times L \times C \times C \\
& +7.9299037204519 \times B \times B \times B \\
& +233.2247865247 \times B \times B \times D \\
& -232.652226914959 \times B \times B \times C \\
& -445.522035810119 \times B \times D \times D \\
& -191.65011646169 \times B \times D \times C \\
& +305.599848022955 \times B \times C \times C \\
& -943.969067130086 \times D \times D \times D \\
& +1134.99240349335 \times D \times D \times C \\
& +121.267586921882 \times D \times C \times C \\
& +9.7942331125198 \times C \times C \times C
\end{aligned}
$$

(A-24)

Trained on all except *ocean*:

$$
\begin{aligned}
Estimate \;=\; & 5.24714036506066 \\
& -18.5166161851985 \times L \\
& -16.7576803251652 \times B \\
& +25.8384594001202 \times D \\
& -7.06302119890658 \times C \\
& -212.810640207234 \times L \times L \\
& +127.474850452569 \times L \times B \\
& +200.695463513735 \times L \times D \\
& +99.3886098178124 \times L \times C \\
& +7.71302757230586 \times B \times B \\
& -148.03172929518 \times B \times D \\
& +38.0662789624442 \times B \times C \\
& +32.1157150233328 \times D \times D \\
& -119.669212920719 \times D \times C \\
& -2.62265180718772 \times C \times C \\
& -746.75042526607 \times L \times L \times L \\
& +126.71384505734 \times L \times L \times B \\
& +1500.63823789427 \times L \times L \times D \\
& +650.096260275838 \times L \times L \times C \\
& -388.300358597183 \times L \times B \times B \\
& +23.5286081966848 \times L \times B \times D \\
& +378.633020903096 \times L \times B \times C \\
& -519.02657938056 \times L \times D \times D \\
& -1617.04560856973 \times L \times D \times C \\
& -143.532082837902 \times L \times C \times C \\
& +23.6515774635998 \times B \times B \times B \\
& +357.357423363327 \times B \times B \times D \\
& -54.8190530746558 \times B \times B \times C \\
& -130.108464299666 \times B \times D \times D \\
& -330.216202389011 \times B \times D \times C \\
& +19.7962588592307 \times B \times C \times C \\
& -255.813571200576 \times D \times D \times D \\
& +969.895846709106 \times D \times D \times C \\
& +124.693762507772 \times D \times C \times C \\
& -2.2006142692742 \times C \times C \times C
\end{aligned}
$$

(A-25)

Trained on all except *equake*:

$$
\begin{aligned}
Estimate \;=\; & \\
& -1.1058296145392 \\
& -12.9145696370589 \times L \\
& -31.742240839475 \times B \\
& +63.6137341125083 \times D \\
& -14.2192226935603 \times C \\
& +606.067590620911 \times L \times L \\
& -48.5925377736995 \times L \times B \\
& -1477.69197974513 \times L \times D \\
& +333.217446548324 \times L \times C \\
& +9.94393245899634 \times B \times B \\
& +58.7848819365562 \times B \times D \\
& +63.2350861194432 \times B \times C \\
& +788.422084183634 \times D \times D \\
& -314.234760869072 \times D \times C \\
& -14.8797824674288 \times C \times C \\
& +485.015954726155 \times L \times L \times L \\
& -188.61740261143 \times L \times L \times B \\
& -1685.26773893091 \times L \times L \times D \\
& -707.056520899603 \times L \times L \times C \\
& -159.246333603269 \times L \times B \times B \\
& +573.987949523436 \times L \times B \times D \\
& +210.295927476846 \times L \times B \times C \\
& +2168.53116165039 \times L \times D \times D \\
& +1142.07459614386 \times L \times D \times C \\
& -246.415492391879 \times L \times C \times C \\
& +9.21109117374485 \times B \times B \times B \\
& +90.5074867714206 \times B \times B \times D \\
& +23.1245831532279 \times B \times B \times C \\
& -324.492262614972 \times B \times D \times D \\
& -225.720545230833 \times B \times D \times C \\
& -51.0389788667802 \times B \times C \times C \\
& -953.75190355329 \times D \times D \times D \\
& -418.695192305601 \times D \times D \times C \\
& +217.253077449922 \times D \times C \times C \\
& +27.7289345853318 \times C \times C \times C
\end{aligned}
$$

(A-26)

Trained on all except *art*:

$$
\begin{aligned}
Estimate \;=\; & \\
& -14.0042737012478 \\
& +176.315332958374 \times L \\
& +180.939063181863 \times B \\
& -208.21991825287 \times D \\
& -39.2009157458858 \times C \\
& -110.083250361565 \times L \times L \\
& -208.863380268163 \times L \times B \\
& +33.3351020143411 \times L \times D \\
& -275.376509841926 \times L \times C \\
& -387.723229201401 \times B \times B \\
& +357.045224151836 \times B \times D \\
& -111.410248840629 \times B \times C \\
& -4.12069581582178 \times D \times D \\
& +425.745491518281 \times D \times C \\
& +47.4124732347632 \times C \times C \\
& +488.894182406868 \times L \times L \times L \\
& -1173.01838063143 \times L \times L \times B \\
& -1381.93680038383 \times L \times L \times D \\
& +1218.21343109496 \times L \times L \times C \\
& -1355.01746841742 \times L \times B \times B \\
& +3208.58027013734 \times L \times B \times D \\
& +2423.73020138451 \times L \times B \times C \\
& +1521.57047951747 \times L \times D \times D \\
& -3477.73337874829 \times L \times D \times C \\
& -522.973763746664 \times L \times C \times C \\
& +18.3623009246042 \times B \times B \times B \\
& +1514.6039256421 \times B \times B \times D \\
& +544.862788635241 \times B \times B \times C \\
& -2096.79513743946 \times B \times D \times D \\
& -2901.25111327832 \times B \times D \times C \\
& -201.670105264077 \times B \times C \times C \\
& -601.629846704226 \times D \times D \times D \\
& +2361.87765651616 \times D \times D \times C \\
& +532.951818426539 \times D \times C \times C \\
& +30.1522792811675 \times C \times C \times C
\end{aligned}
$$

(A-27)

Trained on all except *gzip*:

$$
\begin{aligned}
Estimate \;=\; & \\
& -4.06517947955519 \\
& -39.2928010637235 \times L \\
& -30.3640899135772 \times B \\
& +78.3958062599992 \times D \\
& +29.6968892664377 \times C \\
& -223.304864603133 \times L \times L \\
& +122.091281555359 \times L \times B \\
& +350.555465104108 \times L \times D \\
& +88.0506715287155 \times L \times C \\
& +99.7391747810439 \times B \times B \\
& -288.492040357919 \times B \times D \\
& +28.5012499643061 \times B \times C \\
& -123.45335132393 \times D \times D \\
& -48.5577569991573 \times D \times C \\
& -85.2175295422557 \times C \times C \\
& +885.688199296099 \times L \times L \times L \\
& -856.577135296056 \times L \times L \times B \\
& -2846.65960674198 \times L \times L \times D \\
& +1454.39892063375 \times L \times L \times C \\
& -526.543484697141 \times L \times B \times B \\
& +1958.46844382149 \times L \times B \times D \\
& +706.368412763451 \times L \times B \times C \\
& +3312.02569953026 \times L \times D \times D \\
& -3608.73982973051 \times L \times D \times C \\
& -116.650176540365 \times L \times C \times C \\
& -2.90915044620492 \times B \times B \times B \\
& +551.532345533153 \times B \times B \times D \\
& -204.078987290418 \times B \times B \times C \\
& -1039.38441796659 \times B \times D \times D \\
& -583.973204734535 \times B \times D \times C \\
& +163.747112223819 \times B \times C \times C \\
& -1363.281086793 \times D \times D \times D \\
& +2109.46164076051 \times D \times D \times C \\
& +54.1043974091502 \times D \times C \times C \\
& +.96062193880829 \times C \times C \times C
\end{aligned}
\tag{A-28}
$$

Trained on all except *mcf*:

$$
\begin{aligned}
Estimate \;=\; & 7.94077359300507 \\
& +2.81788651820168 \times L \\
& -24.8073949422957 \times B \\
& +19.3425610524692 \times D \\
& -20.970106977387 \times C \\
& +160.51792246553 \times L \times L \\
& +.899412808285314 \times L \times B \\
& -390.911519254012 \times L \times D \\
& +24.2528267300188 \times L \times C \\
& +7.32688513828638 \times B \times B \\
& +4.39141805668798 \times B \times D \\
& +41.9817299360341 \times B \times C \\
& +190.291419262007 \times D \times D \\
& -21.5156220526788 \times D \times C \\
& +5.41732496211429 \times C \times C \\
& +511.049843424096 \times L \times L \times L \\
& -85.0438493984723 \times L \times L \times B \\
& -1503.43505893887 \times L \times L \times D \\
& -273.972067123187 \times L \times L \times C \\
& -183.285928163177 \times L \times B \times B \\
& +182.781174934111 \times L \times B \times D \\
& +395.502587006318 \times L \times B \times C \\
& +1569.00448812019 \times L \times D \times D \\
& +441.900197140683 \times L \times D \times C \\
& -167.565073187283 \times L \times C \times C \\
& -5.57422792430857 \times B \times B \times B \\
& +177.608039111264 \times B \times B \times D \\
& -.886645874148615 \times B \times B \times C \\
& -111.083032422834 \times B \times D \times D \\
& -358.734329476738 \times B \times D \times C \\
& -28.0178619471141 \times B \times C \times C \\
& -565.021042419176 \times D \times D \times D \\
& -132.938480772543 \times D \times D \times C \\
& +114.045385778895 \times D \times C \times C \\
& +17.4431966682632 \times C \times C \times C
\end{aligned}
\tag{A-29}
$$