

Modeling Multiprocessor Interconnects Using Support Vector Regression

Mel Tsai, Scott Weber
University of California, Berkeley
{mtsai, sjweber}@eecs.berkeley.edu

Abstract

Previous studies have used performance tuning to drive the synthesis of multiprocessor interconnect parameters. The large design space of multiprocessor interconnects makes exhaustive simulation of possibilities impractical. Furthermore, analytical models of multiprocessor interconnects do not often capture the entire parameter space.

This paper proposes a method to learn the design-space of multiprocessor interconnects using a non-linear regression method called support vector regression (SVR). By training a support vector machine (SVM) using real simulation data, this technique creates a model of the interconnect parameter space that can predict final interconnect performance orders of magnitude faster than simulation, and is more accurate than purely analytical techniques. The SVM model can now be used in a generalized flow to quickly and accurately explore the entire design space of multiprocessor interconnects. Our results indicate that training with a small number of simulation vectors produces a model that can predict average message latency to within 25% of the true simulated value across three benchmark applications.

1. Introduction

The performance of a multiprocessor machine depends heavily on the design of its interconnection network. Thus, in order to maximize performance, an interconnection network must efficiently implement the styles of communication inherent to a given set of applications. Furthermore, with the recent trend towards the design of application-specific parallel computers such as IBM's Deep Blue [1], the NEC Earth Simulator [2], and the GRAPE-6 stellar dynamics supercomputer [3], understanding how a specific interconnection network behaves under a specific workload is also important.

As machine sizes scale, "flat" single-bus interconnection networks become inefficient. Thus, the move to variable-latency point-to-point network introduces a

wide array of design choices. For example, for a given application and machine size, what is the optimal buffer depth of a network switch? Should virtual channels be implemented, and how many are optimal? What deadlock-free routing algorithm gives the best performance/cost tradeoff? Should links be half-duplex or full duplex and at what bandwidth? These design-tradeoffs must be fully understood by the system architect in order to make the best design decisions.

In general, exhaustive design-space exploration of interconnect parameters using simulation is impractical. Not only can simulation run-times be prohibitively long, there is little guarantee that an optimal (or near-optimal) set of parameters will be found. Some studies [4][19] attempt to mathematically abstract interconnect parameters to generate an intuition about the design trade-offs. However, while analytical methods may facilitate efficient exploration, it can easily fail to capture all the important aspects of the system.

In recent years, a powerful learning technique using *non-linear regression* (support vector regression, or SVR) has demonstrated its ability to model a complex non-linear system of parameters. This study presents a proof-of-concept that the design-space of multiprocessor interconnects can be learned and abstracted using such a technique. Once abstracted, the model allows system architects to efficiently explore the *entire* design-space of multiprocessor interconnects without exhaustive simulation. In addition, our results suggest that this technique is also faster and more accurate than analytical system modeling. This is because the tools (not the user) automatically develop the model and there is high confidence that the model captures all the important aspects of the system.

The remainder of this paper is organized as follows. Section 2 outlines the general flow of this new approach for design-space exploration. Section 3 presents details on FlexSim 1.2, our chosen network interconnect simulator. Section 4 introduces regression analysis techniques, with an overview of support vector regression. Sections 5 and 6 describe the experi-

mental setup and results of this work. Section 7 outlines some related work in network interconnect design-space exploration and learning-based techniques. Sections 9 and 10 give conclusions and suggested directions for future work.

2. Design-Space Exploration

The design space of multiprocessor interconnect architectures has a number of axes. These axes include (but are not limited to):

- *Network type (e.g. direct or indirect, k-ary n-cubes, fat tree, butterfly, irregular), lossless or best-effort*
- *Connectedness (e.g. mesh, torus)*
- *Link bandwidth, latency, uni- or bi-directional, half- or full-duplex*
- *Switch latency, buffer depths, virtual channels, routing algorithm(s), prevents/allows deadlock, prevents/allows livelock*
- *Message injection buffer depth*

Through detailed simulation, the system architect wishing to perform design-space exploration is able to determine the performance of a certain interconnect network for a given application workload. Typically, such performance is measured as average message latency or bandwidth. However, other measures of performance such as worst-case message latency, cost of fabrication (area), and power consumption may also be important.

As motivated in the introduction, exhaustive simulations of interconnect parameters in order to find the optimal configuration is prohibitively expensive. Therefore, this approach creates a statistical model of the interconnect architecture. Figure 1 shows the proposed flow. In the learning phase, the results of a limited number of simulations (varying in their interconnect configurations) become *simulation vectors* for an SVM regression analysis tool. This tool captures (learns) the network design space and produces an abstracted model of the network. Interconnect parameters can be fed into the model, which will produce an estimate of the performance (i.e. average message latency and bandwidth) of this configuration. Because this model can produce these estimates *orders of magnitude* faster than simulation, the space of possible interconnect configurations can be quickly explored.

The performance of an interconnect network is uninteresting when it is infeasible to implement or too expensive. Thus, the role of a *cost model* (see bottom half of Figure 1) is to weigh the different design trade-offs, assigning different costs (weights) to different design choices. Thus, once a statistical model of the network has been produced, the interconnect parameter space is efficiently explored using a cost model of the network and a non-linear optimizer (such as simulated annealing) to find the optimal or near-optimal performance/cost. Because the focus of the project is to understand the ability to use statistical learning to create a model, we do not discuss the optimization portion of the methodology. Various non-linear optimization packages [5] could be plugged in to complete the flow.

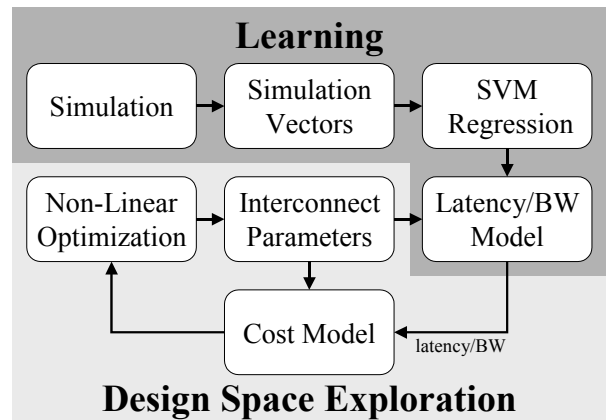


Figure 1. Design-Space Exploration.

Clearly, the success of this approach depends on the accuracy of the simulator, the production of good simulation vectors, and the degree to which the SVM analysis tool can abstract the system. In the following two sections, the chosen interconnect simulator (FlexSim) and the support vector regression technique are described.

3. Interconnect Simulator

FlexSim 1.2 [6] is a flit-level network simulator that simulates direct k -ary d -cube torus networks with various switch architectures, routing functions, link characteristics, network faults, and synthetic traffic generation. The conceptual switch architecture of FlexSim is shown in Figure 2. The basic units of data in FlexSim are *flits* and *messages*; the basic unit of time is *network cycles*. Messages in our experiments are wormhole-routed rather than circuit-switched.

Through command-line invocation, FlexSim accepts a set of interconnect and workload parameters and

simulates the network for a specified number of network cycles. FlexSim produces various statistics from the simulation, including average communication distance (hops and latency), average message throughput, number of misroutes, and message queuing time.

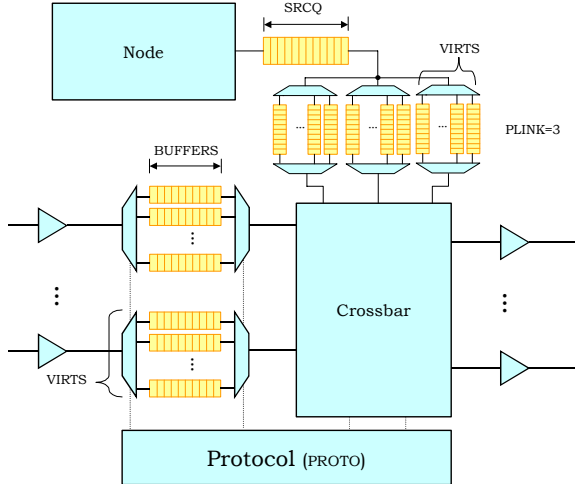


Figure 2. FlexSim's Switch Architecture.

Describing every parameter supported by FlexSim is beyond the scope of this paper. Thus, the following is the list of parameters that concern the interests of this work:

3.1.1. Network Topology. FlexSim's `D` and `SIZE` parameters fix the dimension (n) and size (k) for a k -ary n -cube torus network. For simplicity, the number of processing nodes in the system is fixed at 64, thus three different network topologies were explored in our experiments: an 8-ary 2-cube, a 4-ary 3-cube, and a 2-ary 6-cube.

3.1.2. Link Properties. The `HALF` parameter determines whether links operate in half-duplex or full-duplex mode. The `DEMAND` parameter specifies whether flits traverse links in a demand-driven or time-slice multiplexed fashion.

3.1.3. Switch Properties. The `VIRTS` and `BUFFERS` parameters specify the number of virtual channels per incoming link and the buffer depth (in flits) of each virtual channel (Figure 2). The `SRCQ` parameter determines the size of the (lossy) message inject queue; when this queue becomes full, subsequent messages are dropped at the node. The `PLINK` parameter determines the number of independent paths between the switch and the processing node. The `PROTO` and `USEVIRTS` parameters determine the routing algo-

rithm implemented by the switch. Four different routing algorithms were chosen and varied in our experiments. First, Ecube [7] routing is the standard deadlock-free dimension-ordered routing algorithm. Second, Duato [8] is an adaptive deadlock-free routing algorithm. Third, Dimension Reversal [9] is another adaptive deadlock-free routing algorithm by Dally. Finally, Disha [10] is a fully adaptive routing algorithm with deadlock-recovery. Disha routing requires another parameter, `USEVIRTS`, which specifies the true number of virtual channels used by the algorithm (because one virtual channel is used to mimic a deadlock buffer).

3.1.4. Synthetic Traffic Generation. Processing nodes within FlexSim can be instructed to inject messages into the network at a specified rate and communication pattern, simulating a synthetic traffic workload. The `PER` parameter is the average message injection period (in network cycles) according to a Poisson distribution. The `MSGL` parameter specifies the size (in flits) of each injected message. Optionally, two different message sizes can be injected: short messages and long messages (by setting `HYBRID = 1`). The length of short messages is set by `MSGL`, long message length is set by `LMSGL`, and the percentage of long messages versus short messages is set by `LMSGPCT`. The `DIST` parameter determines the selection function of source-destination pairs (the communication pattern). Possible values of `DIST` are randomized communication, bit-reversed, matrix transpose, destinations at a fixed distance, and destinations within a bounded distance. Alternately, additional communication patterns are easily added by modifying the FlexSim code.

The following parameters were fixed (held constant at one cycle) throughout our experiments: routing intra-node header delay, the acknowledgement flit intra-node delay, the intra-node delay, and the link arbitration time for half-duplex link reversal. Because the Ecube routing cannot handle network faults, none were simulated.

4. Introduction to SVR

The previous section described FlexSim, the interconnect simulator used in our experiments. FlexSim allows us to generate simulation vectors from which we train a support vector machine (SVM) to create an abstract model of the system. In this section, the basic principles behind regression are introduced, and are provided for readers who are not familiar with this ma-

terial. This section summarizes the beginning of a tutorial on support vector regression [11], borrowing equations and diagrams where applicable. For more interested readers, this tutorial and alternately a book on SVMs [12] are recommended.

4.1. Linear Regression

Basic linear regression is perhaps the simplest of the statistical modeling techniques. Given training data $\{(x_1, y_1), \dots, (x_l, y_l)\} \subset \mathfrak{R}^n \times \mathfrak{R}$, one wants to solve the following optimization problem – the Euclidean norm is minimized subject to allowing prediction errors of $\pm \varepsilon$:

$$\begin{aligned} f(x) &= \langle w, x \rangle + b \text{ with } w \in \mathfrak{R}^n, b \in \mathfrak{R} \\ \text{minimize } & \frac{1}{2} \|w\|^2 \\ \text{subject to } & \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon \end{cases} \end{aligned}$$

4.1.1. LMS Algorithm. The Least-Mean Square (LMS) algorithm formulated below provides a simple method to solve the linear regression problem [13]. The online version, which is useful when the training set is large, uses steepest descent to find w . The offline version uses the normal equation to find w . The b term is assumed to be zero for simplicity of illustration.

“Online” versionⁱⁱⁱ:

$$w^{(t+1)} = w^{(t)} + \mu \sum_{i=1}^l (y_i - w^{(t)T} x_i) x_i \text{ where } \mu = 1 / \lambda_{\max} [X^T X]$$

“Offline” version:

$$w = (X^T X)^{-1} X^T y$$

4.1.2. ε -Insensitive Loss Algorithm. This is an extension to the linear regression problem that allows for deviations from ε to be tolerated. Setting $C > 0$ provides a means to tune the tolerance to deviations.

ⁱ $\langle w, x \rangle$ is the dot product of w and x .

ⁱⁱ $\|w\|^2$ is the Euclidean norm.

ⁱⁱⁱ $\mu = 1 / \lambda_{\max} [X^T X]$ is the inverse of the maximum eigenvalue of $X^T X$.

$$\begin{aligned} \text{minimize } & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^l (\xi_i + \xi_i^*) \\ \text{subject to } & \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \end{aligned}$$

The formulation uses the ε -insensitive loss function $|\xi|_\varepsilon$ as shown in Figure 3.

$$|\xi|_\varepsilon := \begin{cases} 0 & \text{if } |\xi| \leq \varepsilon \\ |\xi| - \varepsilon & \text{otherwise} \end{cases}$$

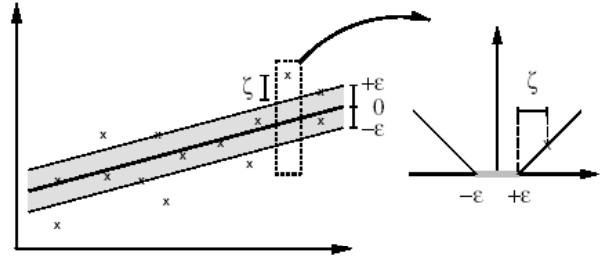


Figure 3. ε -Insensitive Loss Function.

4.1.3. Dual Formulation. The dual of the ε -insensitive loss algorithm is the Lagrange function shown below. The problem can now be solved using quadratic programming. Note that w is now a function of the number of training patterns not the dimensionality of the input space. The calculation of b uses the Karush-Kuhn-Tucker (KKT) conditions and is described in [11].

$$\begin{aligned} \text{maximize } & \left\{ -\frac{1}{2} \sum_{i,j=1}^l (\alpha_i - \alpha_i^*) (\alpha_j - \alpha_j^*) \langle x_i, x_j \rangle \right\} \\ \text{subject to } & \begin{cases} -\varepsilon \sum_{i=1}^l (\alpha_i + \alpha_i^*) + \sum_{i=1}^l y_i (\alpha_i - \alpha_i^*) \\ \sum_{i=1}^l (\alpha_i - \alpha_i^*) = 0 \\ \alpha_i, \alpha_i^* \in [0, C] \end{cases} \end{aligned}$$

The solution of the model is used to create the regression model as shown below.

$$\begin{aligned} w &= \sum_{i=1}^l (\alpha_i - \alpha_i^*) x_i \\ f(x) &= \sum_{i=1}^l (\alpha_i - \alpha_i^*) \langle x_i, x \rangle + b \\ b &= y_i - \langle w, x_i \rangle - \varepsilon \quad \text{for } \alpha_i \in (0, C) \\ b &= y_i - \langle w, x_i \rangle + \varepsilon \quad \text{for } \alpha_i^* \in (0, C) \end{aligned}$$

4.2. Non-Linear Regression Using Kernels

We did not believe that the interconnect parameter space was linear so we used non-linear regression in the form of support vector regression. To make support vector regression non-linear, kernels are introduced. The idea is to introduce a mapping Φ from feature space (i.e. the parameters of the system) to a higher dimensional space. The mapping allows linear regression in a higher dimensional space where the translation between input and the space of Φ can be nonlinear. An example of Φ is shown below.

$$\begin{aligned}\Phi(x_1, x_2) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \\ k(x, x') &:= \langle \Phi(x), \Phi(x') \rangle \\ &= \langle (x_1^2, \sqrt{2}x_1x_2, x_2^2), (x_1'^2, \sqrt{2}x_1'x_2', x_2'^2) \rangle \\ &= \langle x, x' \rangle^2\end{aligned}$$

The kernel function $k(x, x')$ takes the dot product in the Φ mapped space. $k(x, x')$ can replace the dot product in the dual formulation and now we have SVM regression as shown below.

$$\begin{aligned}\text{maximize} & \left\{ \begin{aligned} & -\frac{1}{2} \sum_{i,j=1}^l (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) k(x_i, x_j) \\ & -\varepsilon \sum_{i=1}^l (\alpha_i + \alpha_i^*) + \sum_{i=1}^l y_i (\alpha_i - \alpha_i^*) \end{aligned} \right\} \\ \text{subject to} & \left\{ \begin{aligned} & \sum_{i=1}^l (\alpha_i - \alpha_i^*) = 0 \\ & \alpha_i, \alpha_i^* \in [0, C] \end{aligned} \right\}\end{aligned}$$

The solution of the model is used to create the SVM regression model as shown below.

$$\begin{aligned}w &= \sum_{i=1}^l (\alpha_i - \alpha_i^*) \Phi(x_i) \\ f(x) &= \sum_{i=1}^l (\alpha_i - \alpha_i^*) k(x_i, x) + b \\ b &= y_i - \langle w, x_i \rangle - \varepsilon \quad \text{for } \alpha_i \in (0, C) \\ b &= y_i - \langle w, x_i \rangle + \varepsilon \quad \text{for } \alpha_i^* \in (0, C)\end{aligned}$$

The theory of what constitutes as a kernel and what are kernels is out of the scope of this paper, but information can be found at [11] and [12]. Below are the four kernels that we explored.

$$\begin{aligned}k(x, x') &= \langle x, x' \rangle^p && \text{Linear} \\ k(x, x') &= (\langle x, x' \rangle + c)^p && \text{Polynomial} \\ k(x, x') &= \tanh(s * a * b + r) && \text{Sigmoidal} \\ k(x, x') &= e^{-\frac{\|x-x'\|^2}{2\sigma^2}} && \text{Gaussian}\end{aligned}$$

4.3. SVMTorch

We used SVMTorch [14] to learn the latency and bandwidth models. SVMTorch implements SVM classification and regression using the Sequential Minimal Optimization (SMO) method. The SMO method allows for very large (~10000s) training sets with high dimensional (~100s) input space.

5. Experimental Setup

To generate simulation vectors for SVMTorch, FlexSim was run repeatedly by varying the network interconnect parameters described in Section 3. A simulation vector is defined as a set of input network parameters and the resulting performance (in latency and bandwidth) indicated by the simulator. Scripts were used to randomly vary the input network parameters within the ranges shown in Table 1.

Parameter	Range
D, SIZE	0-2 (8-ary 2-cube, 4-ary 3-cube, 2-ary 6-cube)
DEMAND	0-1
HALF	0-1
PLINK	1-8
SRCQ	1-64
BUFFERS	2-64
VIRTS	2, 4, 8, 16, or 32
USEVIRTS	1-29
PROTO	0-3 (Ecube, Dimension Reversal, Duato, Disha)

Table 1. FlexSim Network Parameters.

Due to the long runtimes of FlexSim, fifteen Linux-based machines were used simultaneously to generate simulation vectors. Scripts were used to farm out FlexSim processes to each machine, and results were collected into a centralized location. These scripts also killed (timed-out) simulations that ran over 10 minutes; this was necessary because approximately 43% of all simulations required more than 10 minutes to com-

plete. Without this time-out period, very few simulation vectors could be generated because certain pathological simulations required well over one hour to converge, and would render the machine ineffective without manual intervention. For simulations that did not time-out, the average simulation run-time was 2.6 minutes on an Athlon 1.33 GHz machine. The aggregate simulation speed of our 15-machine setup was approximately 380 vectors per hour. If a simulation was not killed, it terminated (converged) whenever the average message latency of the most-recent 640 messages had a standard deviation that varied by 5% or less^{iv}.

All 64 nodes in the FlexSim simulation are assumed to execute 2.0 GFLOPs with 1 MB four-way set associative caches and 64-byte cache lines. One flit is assumed to carry 32-bits of data. This information was used to generate parameters for the synthetic traffic workloads, presented in the next section.

5.1. Synthetic Workloads

To simulate realistic traffic workloads, the parameters of Section 3.1.4 were chosen based on three different real applications: Barnes-Hut, FFT, and VORTEX. Thus, because three different application workloads were used, three separate experiments were run using simulation vectors and SVMTorch.

Part of the SPLASH2 benchmark suite [15], Barnes-Hut is a 3D particle interaction solver that uses the hierarchical N-body method. It has an inherently unstructured communication pattern and typically sends messages of size $O(\text{one cache line})$. Thus, based on known message volume rates for a 64-node machine (as indicated by [15]), a random communication pattern with $\text{PER} = 78$ network cycles and $\text{MSG} = 16$ flits was chosen.

Also part of SPLASH2, the FFT kernel implements a 1-dimensional Fast Fourier Transform on 64K complex data points. According to [16], each processor sends sub-matrix messages of size $(n^{0.5}/P)^2$, equal to 256 bytes ($\text{MSG} = 64$) for a 64-node machine and a 64K problem size. FFT’s communication pattern is staggered all-to-all, thus each processor M sends a message to $(M+1) \bmod M$, $(M+2) \bmod M$, and so on. Because FlexSim did not have a `DIST` parameter that matches this pattern, we manually implemented this communication pattern within FlexSim. Based on

^{iv} The 640 messages were grouped into ten bins of 64 messages each. The standard deviation was taken between the averages among the bins.

message volume rates indicated by [15], PER was fixed at 228 network cycles.

VORTEX, the final application used in this study, models 2-D fluid flow among vortices. According to [17], VORTEX uses a “ring” pattern of communication that was again manually implemented within FlexSim. Based on information in [17], a hybrid traffic pattern with $\text{MSG} = 4$, $\text{LMSG} = 321$, $\text{LMSGPCT} = 97$, and $\text{PER} = 6802$ was chosen.

In order to calculate reasonable PER values for the above applications, different assumptions about link bandwidths were assumed. FlexSim parameters for Barnes-Hut and VORTEX assume 12.8 Gbit/sec links (for one direction) between network switches, a modest bandwidth estimate for a modern large-scale multi-processor. However, due to limitations of FlexSim, the same assumption for FFT causes an explosion in simulation run-time. To alleviate this problem, the PER value in FFT assumes 51 Gbit/sec links (in one direction). This is the bandwidth achieved by next-generation on-chip links such as AMD’s HyperTransport I/O technology [18]. Because each experiment is treated as an independent validation of the learning approach, we assume differences in the underlying network assumptions (between experiments) are acceptable.

5.2. Support Vector Regression

An SVM requires a set of vectors to train and a different set of vectors to validate (test) the regression model. We used 1000 vectors for training and 1000 vectors for testing. A vector is composed of the 9 FlexSim parameters described in Table 1 and the latency or bandwidth as reported by FlexSim for these parameters. This reported latency/bandwidth value is known as the *target* value. Support vector regression uses the training vectors to create the latency or bandwidth model. The model has the form as described in Section 4.2.

In order to guarantee robustness of the model, 10-fold cross-validation was applied during training. 10-fold cross-validation splits the training vector set into 10 equally sized sets. Training is then performed using 9 of the sets. The resulting model is tested on the remaining set to determine the mean average error with reference to the target value found using FlexSim. This process is repeated 10 times where a different set is left out for testing on each run. The best model is then used. If the variance among the quality of the models is small, then the model is robust and generalizes well. In addition to 10-fold cross-validation, we

also tested the model against the 1000 test vectors to further verify the robustness of the model. The results of testing against the 1000 test vectors are used in Section 6.

SVR models for each of the three applications, Barnes-Hut, FFT, and VORTEX, were created. The simulation vectors varied according to Table 1. The vectors were normalized to have a mean of 0 and a variance of 1. In addition to generating 1000 test and 1000 training vectors for each application, for Barnes-Hut we also generated 10000 training vectors to determine if increasing the number of training vectors has an effect on model accuracy.

In order to tune the learning process to find the best model, a total of approximately 1000 combinations of C , ϵ , and the kernel k (as discussed in Section 4.2) were attempted for each of the three applications.

5.3. Model Comparisons

In addition to SVMTorch, we implemented LMS (using Matlab) for comparison with SVR. We also compare the SVM method to the analytical performance estimation method presented by K. Johnson in [19]. In this interconnect network model, the predicted average message latency T_m is computed directly as:

$$T_m = n \cdot k_d \cdot T_h + B$$

where $n = \#$ of dimensions, k_d is the average number of hops per dimension, T_h is the average per-hop latency, and B is the flit size. For $k_d \geq 1$,

$$T_h = 1 + \left(\frac{\rho \cdot B}{1 - \rho} \right) \cdot \left(\frac{k_d - 1}{k_d^2} \right) \cdot \left(\frac{n + 1}{n} \right)$$

$$\text{where } \rho = r_m \cdot B \cdot k_d / 2$$

$$\text{otherwise } T_h = 1 \text{ for } k_d < 1$$

According to [19], this model provides a simple and accurate way to predict average message latencies for a given set of interconnect- and application-specific parameters. In the next section we compare the results of our SVM approach to this model as well as our Matlab implementation of LMS linear regression.

6. Results

We compared the ability of SVR, LMS regression, and the Johnson model to predict the average message latency and throughput of various network configurations. In each of the three applications, SVR predicted

latency and throughput with the highest accuracy. Runtimes for each of the methods were also analyzed to determine the practicality of the approaches. SVR and LMS regression run two orders of magnitude faster than brute force simulation.

6.1. Latency Model Comparisons

As shown in Figure 4, for each of the three applications, the SVR models have the smallest mean percentage error in latency prediction compared to actual simulated latency found using FlexSim. Using 1000 training vectors, the SVR predicts latency to within a mean factor of 22% for Barnes-Hut, 25% for FFT, and 3% for VORTEX. Histograms showing the latency prediction errors are shown in Appendix A. In the case of Barnes-Hut, using a larger training set (10000 vectors instead of 1000) results in a factor of two improvement in prediction accuracy (10% on average). Due to time constraints, larger training sets were not tried for FFT and VORTEX.

Surprisingly, the Johnson model with full-duplex links predicts only slightly worse than the SVR models (the Johnson model has trouble accounting for half-duplex links). In fact, if the standard deviation from the actual value is used as the comparison metric, then the Johnson model is superior.

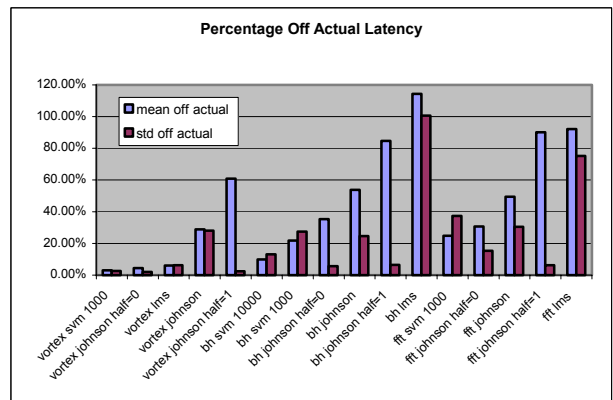


Figure 4. Latency Predictions (1000 and 10000 indicate the number of training vectors).

6.2. Throughput Model Comparisons

Figure 5 indicates that SVR predicts throughput only slightly better than LMS regression. Except for VORTEX, the standard deviations are probably too large to faithfully use the models. Unlike latency, throughput does not appear to be amenable to the learning methods we have applied.

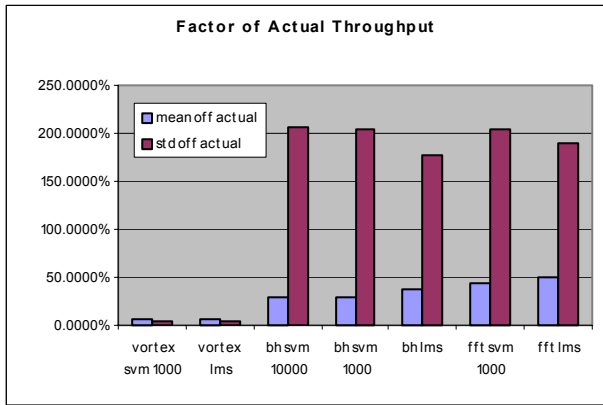


Figure 5. Throughput Predictions.

6.3. Runtime Comparisons

Runtimes required to explore merely 1% of the parameter space, as shown in Figure 6, indicate that SVR and LMS regression are two orders of magnitude faster than brute-force simulation for Barnes-Hut. SVR runtimes are clearly dominated by training, which requires running FlexSim to generate the training vectors. Using one machine for simulations of Barnes-Hut, the generation of 1000 FlexSim training vectors plus the SVR training requires approximately nine days.

In contrast to training times, the *testing* times (i.e. the time required to generate latency and bandwidth predictions) for the Johnson, LMS, and SVR models are equivalent and trivially small. In fact, the entire FlexSim parameter range shown in Table 1 can be exhaustively searched in less than one hour using a single computer, compared to 250 years using FlexSim simulations directly. Moreover, only the optimizer in the design space exploration loop (see Figure 1) will incur the testing time. Thus, if the time required to generate training vectors and train on this data is tolerable, then SVR can provide an accurate model. One should note that it probably took K. Johnson longer than 9 days to develop the model in [19] (which could be classified as training time).

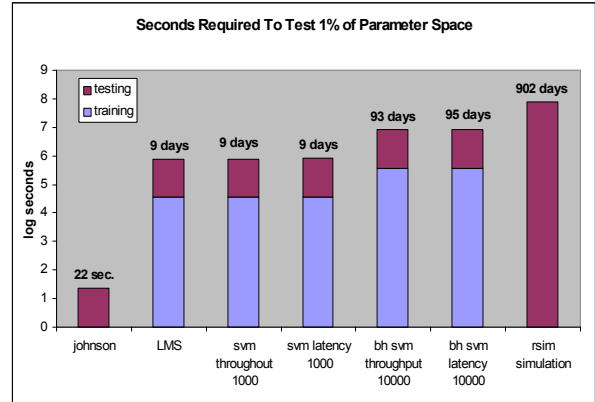


Figure 6. Runtime Comparison for Barnes-Hut.

7. Related Work

A large body of work exists in the study of design trade-offs in multiprocessor interconnects. However, in general there has been little work in the area of automated design-space exploration of multiprocessor interconnects using learning. In [20], three different learning techniques (Markov prediction, linear prediction, and time-delay neural networks) were used to predict the memory access patterns of a few parallel processing applications. Recurrent neural networks were used in [21] to synthesize a good interconnect topology based on a set of desired communication routes.

Several studies have used performance results to analytically drive the selection of interconnection network parameters. For example, [22] uses ILP optimization of performance constraints to drive the synthesis of multiprocessor interconnection networks. The work in [23] uses system-level constraints on memory bandwidth to optimize the connectivity between memory ports and global buses in data-intensive applications. The work in [24] varies link bandwidths over a range of applications in order to track the “knee” where network overheads begin to dominate total execution time, allowing them to predict optimal link bandwidth requirements. In [25], the sources of overhead in a parallel program were modeled using “lost cycles” analysis in order to predict performance. The effects of virtual channels in message-passing networks were modeled in [26]. The impact of switch selection functions across a range of communication patterns was shown in [27]. The work in [28] tunes network switching schemes to match application communication characteristics.

Purely analytical models of interconnection networks have been proposed. A precursor to [19],

Agarwal gives an analytical model of the limits on interconnection network performance in [29]. Agarwal presents a model of network latency that captures the effects of contention, packet size, and communication locality in buffered direct networks. His main results give insight into the effects of communication locality verses network topology. LogP [30] defines four parameters of a machine-independent network that helps application writers choose good communication strategies.

8. Design Space Exploration

The focus of the project was to analyze the ability to learn models of interconnect networks. We have not yet incorporated this model into the larger design-space exploration flow. For example, we assume existing optimization packages [5] could be used to select a good network configuration based on the SVM model. We would only need to develop a cost model for the optimization package.

8.1. Cost Model

Development of an accurate cost model of the network depends on technological factors. For example, the cost of implementing a network switch with a certain number of virtual channels is associated with a certain cost (in terms of power or silicon area). Thus, a cost model can use weighting factors to emphasize a certain evaluation criteria over another.

Using FlexSim parameters (with T = number of inter-node links and N = number of nodes), the following is our prototype network cost model:

$$\begin{aligned}
 Cost = & 2 \cdot T \cdot c_1 \cdot (VIRTS \cdot (BUFFERS + c_2 \cdot VIRTS^{c_3})) \\
 & + c_4 \cdot \left\{ \begin{array}{l} 4^2, \text{Network} = 8 \times 8 \\ 6^2, \text{Network} = 4 \times 4 \times 4 \\ 12^2, \text{Network} = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \end{array} \right\} \\
 & + N \cdot c_5 \cdot PLINK + T \cdot \left\{ \begin{array}{l} c_6, HALF = 0 \\ c_7, HALF = 1 \end{array} \right\} \\
 & + N \cdot \left\{ \begin{array}{l} c_8, PROTO = E \\ c_9, PROTO = C \\ c_{10}, PROTO = R \\ c_{11}, PROTO = X \end{array} \right\} + N \cdot c_{12} \cdot SRCQ \\
 & + T \cdot \left\{ \begin{array}{l} c_{13}, DEMAND = 0 \\ c_{14}, DEMAND = 1 \end{array} \right\} \\
 & + c_{15} \cdot Latency + c_{16} \cdot BW
 \end{aligned}$$

To use the cost model, the constants $c_1:c_{16}$ must be adjusted to reflect the technology and weighting of evaluation criteria for a particular network.

9. Future Work

As explained in Section 8, one should add the optimizer into the design-space exploration flow. This will allow us to better understand the number of configurations that need to be explored. If the number is a substantial percentage of the entire parameter space, then the use of learning techniques become more attractive.

Using density estimation, we could remove outliers from the model to improve the models. Using trace-driven traffic and a more complete model of the processing nodes in FlexSim will help to improve the authenticity of the simulation vectors. Alternately, rather than fix the application parameters of Section 3.1.4 for each experiment, they could become free parameters in the model themselves. Effectively, this would extend the learned model to provide latency and bandwidth predictions for a full range of applications. However, the increased complexity of this approach is unknown.

Finally, we have seen that the Johnson model is surprisingly accurate. It would be worthwhile to see if the model could be adjusted to become competitive with the model found by SVR. It might also be interesting to incorporate the structure of the Johnson model to help the learning process.

10. Conclusion

SVMs can effectively learn the structure of multiprocessor interconnect networks and predict latency and bandwidth to learn the space of multiprocessor interconnects. The SVR models outperform the Johnson model and LMS regression in terms of predicting message latency. However, the Johnson model predicts well for full-duplex network configurations, and the Johnson model has a slightly lower standard deviation of the mean factor off the actual latency. With some small changes, it might be possible for the Johnson model to perform as well as the SVR. However, adjusting the Johnson model might require learning.

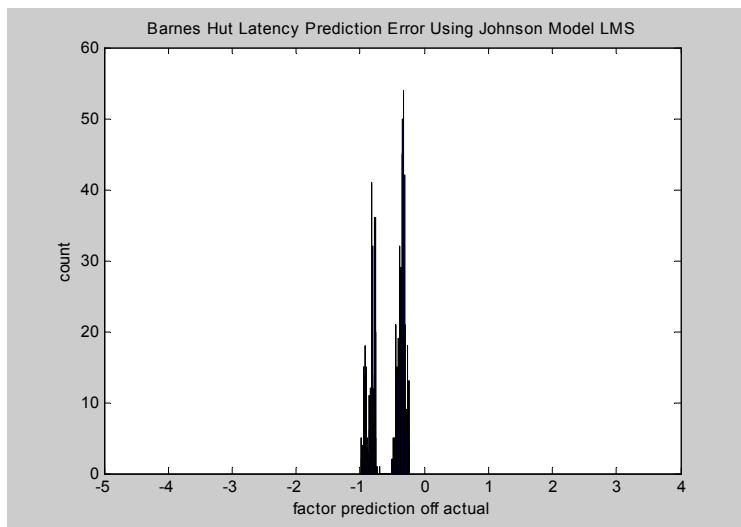
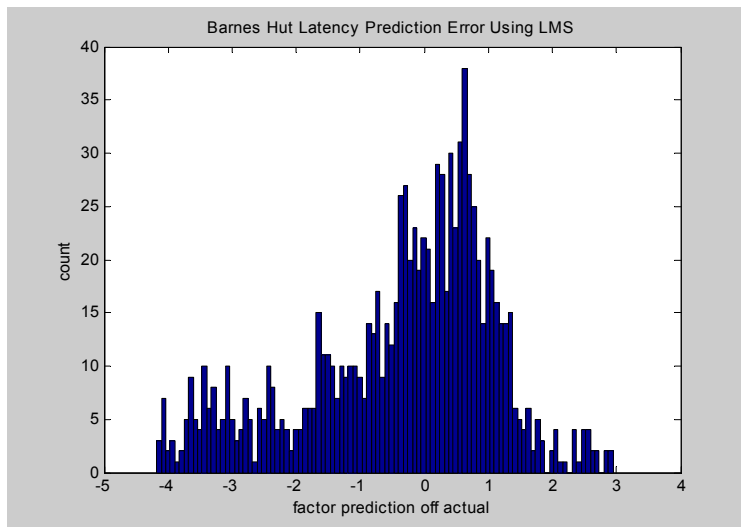
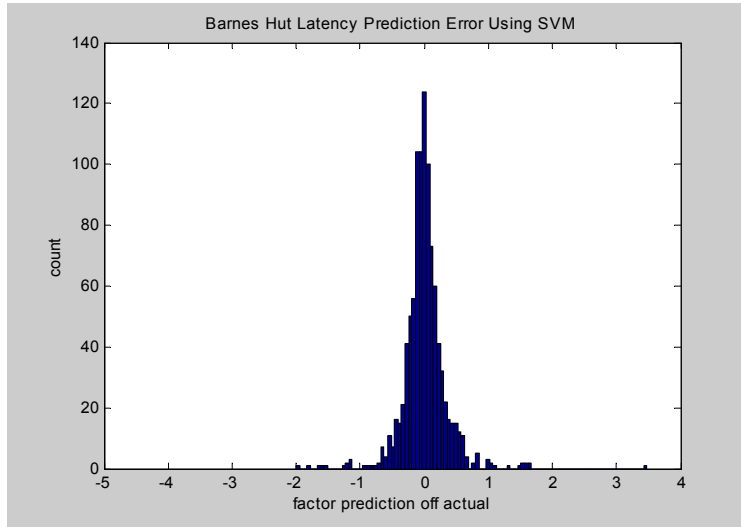
Runtimes indicate that the regression methods require a significant amount of time for training. This is because FlexSim simulations must be performed in order to generate the training vectors. Once the model is created, however, *exhaustive exploration of the parameter space requires less than one hour*. If less accurate predications can be tolerated, then the Johnson model should be used. Using a learning method makes sense only if the number of testing examples grossly outweighs the number of training examples and the most accurate prediction is required.

11. References

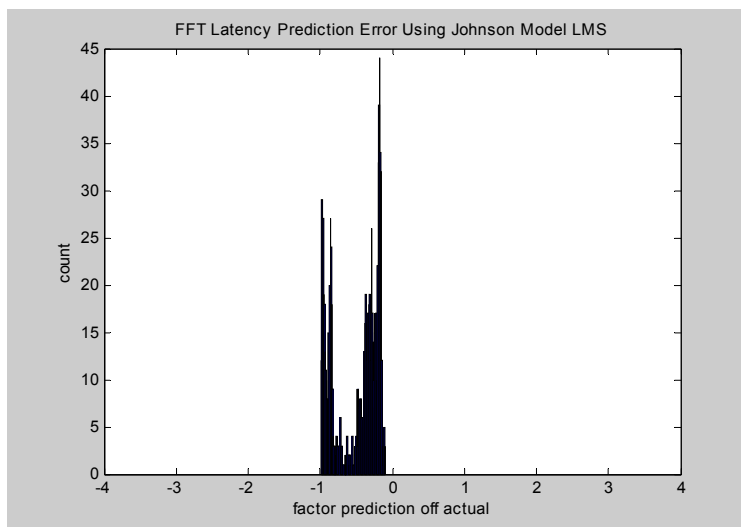
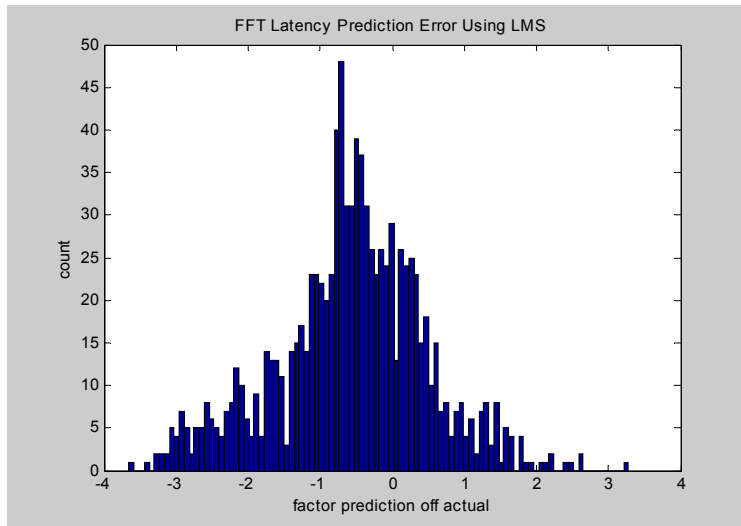
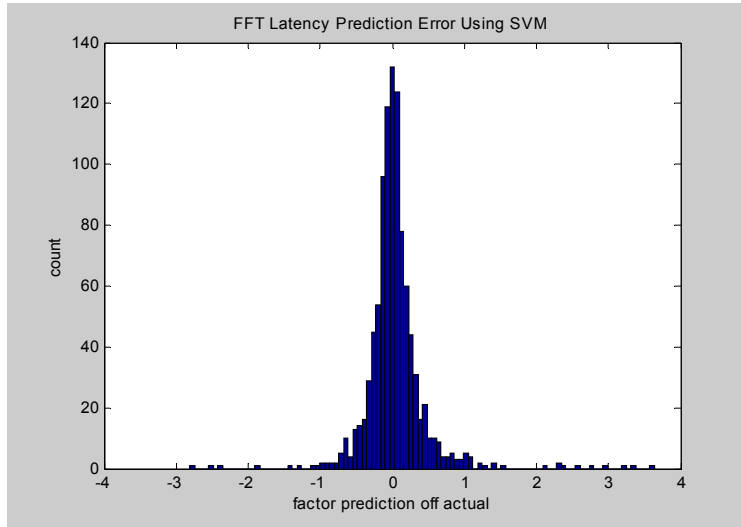
- [1] IBM Corporation. "Ancient Game, Modern Masters," IBM Deep Blue Press Release, May 1997.
- [2] NEC Corporation. "The World's Fastest Supercomputer System for Resolving Global Environmental Problems Completed," Press Release, March 2002.
- [3] J. Makino. "Next-Generation Massively Parallel Computers – Massively Parallel Computer for Particle-based Simulations," Slideshow presentation at the 5th JSPS Symposium on Computational Science and Engineering, 2002.
- [4] W. Dally. "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, Vol. 39, No 6, July 1990.
- [5] "Nonlinear Programming FAQ," Northwestern University and Argonne National Laboratory, <http://www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming-faq.html>, May 2002.
- [6] SMART Interconnects Group. "Flexsim 1.2 User Guide," University of Southern California, 2002.
- [7] W. Dally, C. Seitz. "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, C-36(5):547-553, 1987.
- [8] J. Duato. "Deadlock-Free Adaptive Routing Algorithms Multicomputers: Evaluation of a New Algorithm," *Proceedings of the Third IEEE Symposium in Parallel and Distributed Processing*, pp. 840-847, December 1991.
- [9] W. Dally, H. Aoki. "Deadlock-free adaptive routing in multicomputer networks using virtual channels." *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466-475, April 1993.
- [10] K. V. Anjan, T. M. Pinston. "DISHA: A Deadlock Recovery Scheme for Fully Adaptive Routing," *Proceedings of The 9th International Parallel Processing Symposium*, Santa Barbara, CA, April 1995.
- [11] A. Smola, B. Schölkopf, "A Tutorial on Support Vector Regression", *NeuroCOLT Technical Report NC-TR-98-030*, Royal Holloway College, University of London, UK, 1998. Statistics and Computing, 2001.
- [12] N. Cristianini, J. Shawe-Taylor. "An Introduction to Support Vector Machines (and other kernel-based learning methods)", ISBN: 0 521 78019 5, Cambridge University Press, 2000.
- [13] M. Jordan, C. Bishop. "An Introduction to Graphical Models", *Unpublished*.
- [14] Ronan Collobert, Samy Bengio, "SVMTool: Support Vector Machines for Large-Scale Regression Problems", *Journal of Machine Learning Research*, vol 1, pages 143-160, 2001.
- [15] S. Woo, M. Ohara, E. Torrie, J. Singh, A. Gupta. "The SPLASH-2 programs: Characterization and methodological considerations." *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995.
- [16] H. Shan, J Feng. "Programming FFT on DSM Multiprocessors", *Unpublished*.
- [17] R. Cypher, A. Ho, S. Konstantinidou, P. Messina. "Architectural Requirements of Parallel Scientific Applications with Explicit Communication." *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May 1993.
- [18] Advanced Micro-Devices, "HyperTransport™ Technology I/O Link: A High-Bandwidth I/O Architecture," *White Paper*, July 2001.
- [19] K. Johnson. "The Impact of Communication Locality on Large-Scale Multiprocessor Performance." *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp 392-402, May 1992.
- [20] M. Sakr, S. Levitan, D. Chiarulli, B. Horne, C. Giles. "Predicting Multiprocessor Memory Access Patterns with Learning Models," *Proceedings of the 14th International Conference on Machine Learning*, pp. 305-312, 1997.
- [21] M. Goudreau, C. Giles. "Using Recurrent Neural Networks to Learn the Structure of Interconnection Networks," *Neural Networks*, 8(5), p. 793, 1995.
- [22] Y. Jiang, T. Lee, T. Hwang, Y. Lin. "Performance Driven Interconnection Optimization for Microarchitecture Synthesis," *IEEE Transactions on CAD*, Vol. 13, No. 2, pp. 137-149, February 1994.

- [23] T. Meeuwen, A. Vandecappelle, A. Zelst, F. Catthoor, D. Verkest. "System-level Interconnect Architecture Exploration for Custom Memory Organizations," *Proceedings of the 2001 International Symposium on Systems Synthesis*, pp. 13-18, September 2001.
- [24] A. Silvasubramaniam, A. Singla, U. Ramachandran, H. Venkateswaran. "Synthesizing Network Requirements Using Parallel Scientific Applications," *Technical Report GIT-CC-94/31*, July 1994.
- [25] M. Crovella, T. LeBlanc. "Parallel Performance Prediction Using Lost Cycles Analysis," *Proceedings of Supercomputing 1994*, pp. 600-610.
- [26] W. Feng, K. Shin. "The Effect of Virtual Channels on the Performance of Wormhole Algorithms in Multicomputer Networks," *University of Michigan Directed Study Report*, May 1994.
- [27] W. Feng, K. Shin. "Impact of Selection Functions on Routing Algorithm Performance in Multicomputer Networks," *Proceedings of the 11th International Conference on Supercomputing*, July 1997.
- [28] W. Feng, J. Rexford, S. Daniel, A. Mehra, K. Shin, "Tailoring Routing and Switching Schemes to Application Workloads in Multicomputer Networks," *University of Michigan CSE-TR-239-95*, May 1995.
- [29] A. Agarwal. "Limits on Interconnection Network Performance," *IEEE Transactions on Parallel and Distributed Systems*, 1991.
- [30] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Erik Schauer, E. Santos, R. Subramonian, T. von Eicken. "LogP: Towards a Realistic Model of Parallel Computation," *Proceedings of PPOPP*, May 1993.

Appendix A.1 Barnes Hut Latency Comparison



Appendix A.2 FFT Latency Comparison



Appendix A.3 Latency Comparison

