Modeling the Cache Effects of Interprocessor Communication*

Ibrahim Hur Calvin Lin
The University of Texas at Austin
Austin, Texas 78712

Abstract

This paper presents a new communication cost model that incorporates cache effects by including a term that is a function of the number of unique *memory lines touched* (MLT). This term improves accuracy because it is sensitive to data layout and the order in which data is touched.

We use experiments on three parallel computers—the Cray T3E, the IBM SP-2, and the Sun Enterprise 5000—to show that our model is significantly more accurate than previous models. We also show that our approach can be used to guide program development, as our model can predict the relative performance of two parallel programs, accurately identifying their crossover point.

1 Introduction

Parallel performance modeling and prediction have many uses in algorithm development, compiler optimization, and performance tuning. Unfortunately, such predictions are complicated by the existence of caches, which are notoriously difficult to model because they have complex behavior that depend on many variables. In addition to hardware properties such as line size, cache size, set associativity, and cache miss times, program characteristics such as the relative alignment of data and the order in which data is touched affect the number of cache misses and, ultimately, the application's performance. While many researchers have attempted to model particular aspects of cache behavior for specific purposes—for example, to analyze conflict misses of blocked algorithms [1]—no one has successfully incorporated cache effects into a general model of communication performance.

This paper presents a simple method of modeling the cache effects of interprocessor communication in parallel programs. The key idea is to capture cache effects by understanding how data is laid out in memory and how it is accessed. In particular, we estimate the number of unique memory lines accessed when marshalling data. Therefore, in contrast to previous models, which typically include a term for message startup cost, α, and a term for per-byte transmission cost, β, our model includes a third term that is proportional to the number of memory lines accessed, where a *memory line* is defined to be a cache-line-sized block of memory. Other models attempt to include marshalling costs [2, 3], but our model is unique in its ability to capture the significant cache effects of marshalling costs.

*This work was supported in part by an NSF Research Infrastructure Award CDA-9624082 and ONR grant N00014-99-1-0402.

The first step towards building our model is to derive a closed form formula for MLT, the number of memory lines accessed when marshalling a message. We assume that this message corresponds to a rectangular portion of an array. Once MLT is known, our model can be stated as $\alpha + \beta n + \gamma MLT$. We can imagine that the third term, which captures cache effects, is a more involved function of MLT. We explore this possibility and show that a linear term is sufficient. To complete our model, we use a training set to measure actual costs for various message sizes and layouts. From these measured values we use curve fitting techniques to determine coefficients for α, β, and γ. Thus calibrated, our model can be used to predict communication costs by providing specific values of n and MLT.

This paper makes the following contributions: (1) We introduce a new communication model that includes cache effects; (2) we present equations to compute MLT for rectangular blocks of data; and (3) we demonstrate that the addition of the MLT feature leads to a significantly more accurate model.

We demonstrate the superiority of our model by first applying it to point-to-point communication operations, since these are the basis upon which all other communication models can be built. To show that our approach works for different machines, we apply it to three very different parallel computers: the Cray T3E, the IBM SP-2 and the Sun Enterprise 5000. To show that our approach works for different languages, we apply it to benchmarks written in MPI and in the ZPL parallel programming language [4, 5]. Finally, to show that our model can be used to guide program design, we show that it can accurately predict the relative performance of two ZPL programs that solve the same problem. In particular, we show that our model accurately predicts the crossover point between these two programs.

2 Related Work

Most models of communication performance do not consider cache effects. For example, Ivory [3] creates a model that considers the effects of data copying, but ignores the cache, so the prediction results are inaccurate for non-contiguous data accesses. Fahringer has done extensive work [6, 7], which includes a static model of communication costs for HPF [7], but his method does not consider cache miss effects, either.

Others have modeled cache behavior, but they have not related their models to runtime performance. For example, Temam et al. [8] develop an analytical model that focuses

primarily on conflict misses of direct mapped caches, and Ferrante et al. [9] derive an upper bound for the number of unique memory lines accessed in a loop.

Ghosh et al. [10, 11] estimate cache misses in loop-oriented scientific codes for uniprocessors by extending the notion of reuse analysis to summarize the memory behavior of loops. This work is developed for virtually addressed one level caches, which is no longer a realistic configuration. In addition, speed is a drawback of this approach, as it is reported to be "potentially faster" than simulation.

Amato et al [12] use a training set approach that is similar to ours, but their method of modeling cache behavior is significantly different. They assume different costs for read and write operations, which they use to generate upper and lower bounds, respectively, for their cost function. This method produces very large performance bounds, and it is insensitive to differences in data layout.

3 Our Solution

Analytic modeling of communication performance is almost impossible because such performance is often non-linear due to the effects of caches and other prefetching devices. For example, the Cray T3E's stream buffers can make it faster to send larger amounts of data than smaller amounts of data. We thus use a "benchmarking model" [6] that first identifies a number of significant features and then runs a training set on the target machine. By fitting curves to the measured data, we can determine machine-specific coefficients to our model. An advantage of this approach is that it captures the actual behavior of the system, including all the complexities of the hardware.

The key issue is the determination of the model features. We use three features: the number of messages sent, the message size, and MLT. In particular, we model execution time, t, as follows:

$$t = \alpha + \beta n + f(MLT) \tag{1}$$

We conjecture that $f(MLT)$ represents cache miss behavior as a simple function of the number of lines touched [9], most likely of the form γMLT, but we explore other possibilities as well (see Section 4).

To understand why the MLT term is important, consider the transmission of two n-byte messages, one contiguous in memory (such as a row of a matrix) and the other not (such as a column of a matrix). The conventional model ($\alpha + \beta n$) does not distinguish between these two messages, yet the row-transfer only brings useful data into the cache while the column-transfer only uses one word of useful data per cache line, assuming that the width of the matrix is larger than one cache line. For example, our measurements on the IBM SP-2 show that for a 4000 $\times$ 4000 matrix of integers, column tranfers incur 16 times more cache misses than row transfers, leading to a factor of 8 slower performance [13]. Our solution is to compute MLT, the number of distinct cache lines to which the message would be mapped, as this approximates the number of cache misses.

Our approach should capture the effects of both compulsory misses and capacity misses, since both are proportional to the number of unique memory lines touched. However, our model ignores other features, such as conflict misses, that might improve the accuracy of our model at the expense of increased model complexity. We choose to focus on compulsory and capacity misses to see if a simple model will suffice, and because it can be difficult to estimate the number of conflict misses.

3.1 Calculating the Model Parameters

Our model makes the following assumptions.

The processor mesh is assumed to be $P_r \times P_c$, where $P_r \geq 1$ and $P_c \geq 1$.

p: total number of processors ($p = P_r \times P_c$)

b_r: size of the subblocks in bytes in the row dimension

b_c: size of the subblocks in bytes in the column dimension

l**:** cache line size

We assume that each communication operation is performed for a matrix of size $L_R \times L_C$ bytes, where L_R is the number of rows and L_C the number of columns.

3.1.1 Number of Bytes Sent

The number of bytes (n) to be transmitted from one processor to another is given by two formulae, depending on whether rows are transferred or columns are transferred. We give formulas for the transfer of d rows. A similar formula for column transfers can be obtained by replacing L_C with L_R, P_c with P_r, and b_c with b_r.

Each processor transmits d rows, and the maximum row length is $\lceil L_C/P_c \rceil$, so the maximum number of bytes sent by one processor is

$$n = \left\lceil \frac{L_C}{P_c} \right\rceil d \tag{2}$$

3.1.2 Number of Unique Memory Lines Touched

This section provides formulae to compute MLT for accessing d rows or d columns of an array. We assume that all data blocks have the same size, and that for column transfers, $b_c > 2l$.

For row transfers MLT is trivial to calculate for one block. We simply divide the number of bytes transferred by the cache line size, l. However, since there is no way to know if the data to transfer starts on a cache line boundary, we give lower and upper bounds, LB_R and UB_R.

$$LB_R = \left\lfloor \frac{b_c d}{l} \right\rfloor \tag{3}$$

$$UB_R = \left\lceil \frac{b_c d}{l} \right\rceil + 1 \tag{4}$$

We can then use $T_R = (LB_R + UB_R)/2$ to approximate MLT for row transfers.

Calculating MLT for column transfers is more complicated. The upper and lower bounds, LB_C and UB_C, for MLT are given by the following theorem, which we prove elsewhere [13].

Theorem 1 *For a rectangular array with b_r rows and b_c columns, which is stored in row-major order in memory, the lower and upper bounds, LB_C and UB_C, for the number of distinct lines accessed, MLT, when transferring d columns of the array are*

$$LB_C = \left\lfloor \frac{b_r \gcd(b_c, l)}{l} \right\rfloor \left(\frac{l}{\gcd(b_c, l)} + \left\lceil \frac{d}{\gcd(b_c, l)} \right\rceil - 1 \right) \quad (5)$$

$$UB_C = \left\lceil \frac{b_r \gcd(b_c, l)}{l} \right\rceil \left(\frac{l}{\gcd(b_c, l)} + \left\lceil \frac{d}{\gcd(b_c, l)} \right\rceil \right) \quad (6)$$

where l is the number of bytes in a cache line, and $\gcd(b_c, l)$ is the greatest common divisor of b_c and l.

4 Background and Methodology

This section describes the architectures used in our experiments, and then explains how we develop linear regression models for specific computers. Although the discussion here focuses on the modeling of point-to-point communication costs, the same principles apply to other communication operations.

4.1 Architectures

Our experiments use three parallel machines, the IBM SP-2, the Cray T3E, and the SUN E5000. The IBM SP-2 consists of 16 RS-6000 nodes each running at 67MHz. Each node has two levels of caches, and the communication channel has a peak rate of 40 MB/second. The Cray T3E employs 68 nodes which are embedded in a 3D toroidal interconnect. Each node has 8KB data and 8KB instruction caches, and a 96KB on chip L2 cache. The communication channel has a peak bandwidth of 300 MB/second. The Sun E5000 is an 8 processor bus-based machine with superscalar processors, 16KB data caches, 16KB instruction caches, and a L2 cache.

4.2 Regression Models

Linear regression can be used to develop performance models by setting up a system of equations where the known values are measured communication times and calculated values of the model features, and the unknowns are the model coefficients. Solving this system gives us the values of the model coefficients that we are looking for.

The data used to determine unknown coefficients in regression analysis will be referred to as the *training set*, and the data used for testing the performance of models is known as the *test set*.

Linear regression models for the communication cost can be defined as

$$\mathbf{y} = \mathbf{\Phi}\beta \quad (7)$$

The elements of the $\mathbf{\Phi}$ matrix are known. Each column of this matrix represents one feature of the model. For example, for our model (eq. 1) the first column represents the number of messages sent, the second column the number of bytes transferred, and the third column the number of memory lines touched. The values of $\mathbf{y}$ are the measured communication times from our training set. To find the value of the β vector, the coefficients of our model, we use a least squares method, which is defined as

$$\beta = \mathbf{\Phi}^{+}\mathbf{y} \quad (8)$$

where $\mathbf{\Phi}^{+}$ is the pseudo-inverse of $\mathbf{\Phi}$ [15].

Our model features have different scales, so we normalize the columns of the Φ matrix to improve the numerical properties of the system. The models we have discussed thus far are called first-order regression models. Alternatively we can define second-order models which include quadratic, Φ_j^2, and cross-product, $\Phi_j\Phi_k$, terms.

Model Name	Regression Function
M1 (Standard Model)	$\alpha + \beta\mathbf{n}$
M2 (Our Model)	$\alpha + \beta\mathbf{n} + \gamma\mathbf{MLT}$
M3	$\alpha + \beta n + \beta_1 n^2$
M4	$\alpha + \beta n + \beta_1 n^2 + \beta_2 n^3$
M5	$\alpha + \beta n + \gamma MLT + \gamma_1 nMLT$
M6	$\alpha + \beta n + \gamma MLT + \gamma_1 nMLT + \beta_1 n^2 + \gamma_2 MLT^2$

Table 1: Regression models for point-to-point comm. cost.

4.3 Statistical Analysis

We use two statistical measures to assess the adequacy of our fitted models, *estimated error standard deviation*, $s_e = MS_E$, and the *coefficient of determination*, R^2. Small values of MS_E and R^2 indicate a good fit between predicted and measured results, with acceptable values of R^2 typically being less than 0.01.

5 Modeling Point-to-Point Communication

Given our three model features described in the previous section, we define six different regression models (See Table 1). M1 corresponds to models discussed in the literature, which only use the number of messages and message length as model features [16, 17, 6]. We refer this as the *standard model.* M2 is our proposed model, the simplest model that includes MLT as a feature. The other models are higher order models that we test for completeness.

5.1 Experimental Results

Figures 1 and 2 illustrate the difference between model predictions for M1, M2, and M6 on the SP-2 using the ZPL and C+MPI benchmarks. In these figures bold lines (dots) represent measured values and thin lines represent predicted values. We see that M1 predicts badly, particularly for column transfers, while M2 is quite accurate, and M6 is slightly better yet. (The other models that do not include MLT (M3, and M4) produce results that are comparable to M1, and those that include MLT (M5) produce results that are comparable to M2.)

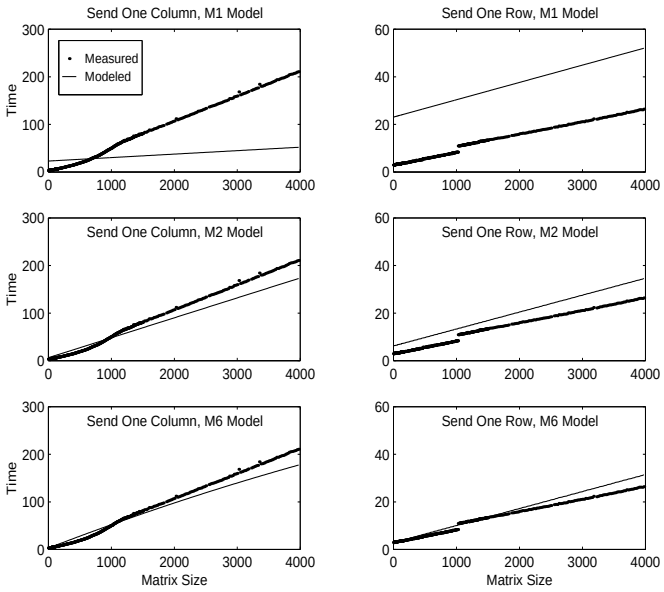

Figure 1: Measured vs. predicted results for ZPL on the IBM SP-2. Our models, M2 and M6, give much better predictions than the standard model, M1.

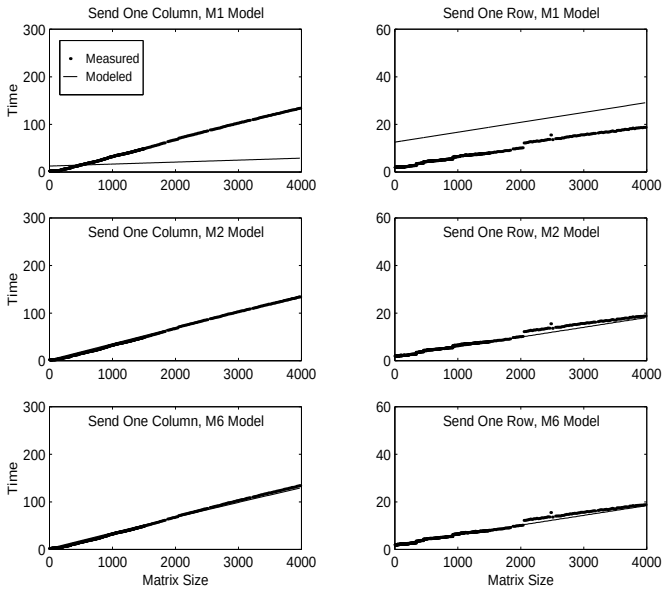

Figure 2: Measured vs. predicted results for C+MPI on the IBM SP-2. Our models, M2 and M6, give much better predictions than the standard model, M1.

These graphs show results for transferring one column or one row of a matrix, where cache effects are most prominent, so we are showing the worst case for model M1. However, an informal survey of ZPL programs shows that $d = 1$ represents the most common type of communication. Results for the other machines are similar. Figure 3 shows how model M2 predicts performance on the SP-2 and T3E. While our model cannot predict all of the non-linearities of the observed behavior, the model does capture overall performance quite well.

5.2 Statistical Assessment

We have gathered six sets of measured data, namely C+MPI and ZPL data on three different architectures to compare the regression models defined in Table 1. For these results we set the training set size to 100, which we determined to be sufficient [13], and use the rest of the measurements as the test set. Table 2 shows the accuracy of our models. Each entry in the table statistically summarizes an entire set of measured data points. All of the models that include MLT are superior to the standard model. For example, the R^2 or MS_E values of M1 model are 3-8 times larger than corresponding M2 values. For M2, the R^2 is always smaller than the desired 0.01 value, and M6, a second order model which includes MLT, gives the best results in all cases.

6 Modeling Whole Programs

To test the practical benefits of our approach, we see if our model can predict the performance of two different programs for computing the uniform convolution [18]. This task is challenging because the two programs exhibit complex tradeoffs that depend upon both machine-specific and problem-specific parameters.

The uniform convolution accepts as inputs an $L \times L$ image and a block size, b. The output is an $L \times L$ image where each pixel is the weighted sum of the $b \times b$ block of image values for which that pixel is the lower right corner. The Conv-Shift algorithm computes these sums by repeatedly shifting the image to the right (requiring point-to-point communication) to accumulate the sums along each row, and then repeatedly shifting the values down each column to compute the sum of the entire block. The Conv-Scan algorithm computes the same result using two parallel prefix operations and three shift operations. In general, it's unclear which of the two programs will run faster. (Additional details, including the actual ZPL codes that we used, are given by Snyder [5].) To conduct this experiment, we extend both the standard model and our model to include computation costs, such as the addition of two arrays and the copying of one array to another, and to model the cost of scans (parallel prefix operations). These are natural extensions of the point-to-point models that add a feature for *the number of arithmetic operations.*

Figure 4 compares our model against the standard model on the Cray T3E. The y-axis is the execution time of the Conv-Shift algorithm divided by that of the Conv-Scan algorithm, so a value smaller than 1.0 means that Conv-Shift runs faster. The x-axis represents different block sizes and matrix sizes. We performed experiments for block sizes ranging from 1 to 10 and matrix sizes from 100×100 to 1000×1000. These results use a 2×2 processor mesh. Results for other processor configurations are similar. For a block size of 4, the standard model incorrectly predicts that Conv-Shift is faster. Our model does not make this mistake and chooses the correct algorithm for all block sizes.

Figure 5 shows the details for $b = 4$. The top graphs

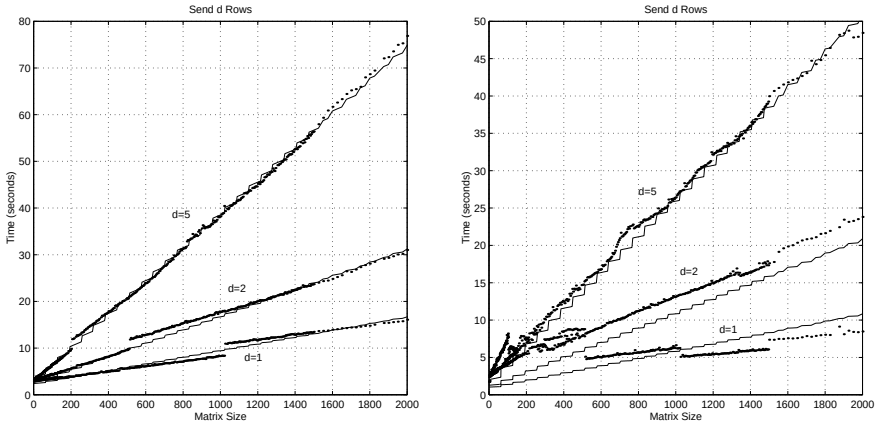

Figure 3: Predicted cost (using M2) on the IBM SP-2 (left) and the Cray T3E (right) for sending d rows of a 2000 $\times$ L Matrix, as L varies from 1 to 2000.

Model Name	IBM SP-2				CRAY T3E				SUN E5000			
	ZPL		C+MPI		ZPL		C+MPI		ZPL		C+MPI	
	R^2 ($\times 10^3$)	MS_E ($/10^3$)	R^2 ($\times 10^3$)	MS_E ($/10^3$)	R^2 ($\times 10^3$)	MS_E ($/10^3$)	R^2 ($\times 10^3$)	MS_E ($/10^3$)	R^2 ($\times 10^3$)	MS_E ($/10^3$)	R^2 ($\times 10^3$)	MS_E ($/10^3$)
M1	**3.54**	**1.84**	**2.81**	**0.48**	**9.05**	**2.23**	**10.12**	**0.59**	**25.87**	**3.21**	**12.02**	**0.52**
M2	**0.60**	**0.31**	**0.29**	**0.05**	**3.29**	**0.81**	**2.34**	**0.14**	**13.41**	**1.66**	**9.60**	**0.42**
M3	3.45	1.79	2.75	0.47	8.89	2.19	9.71	0.57	25.74	3.19	7.45	0.32
M4	3.39	1.76	2.72	0.46	8.85	2.18	9.56	0.56	25.54	3.17	6.02	0.26
M5	0.59	0.30	0.29	0.05	2.87	0.71	2.21	0.13	12.83	1.59	4.24	0.18
M6	0.44	0.23	0.26	0.04	2.87	0.71	2.01	0.12	8.12	1.01	3.76	0.16

Table 2: Our model, M2, and its derivatives, M5 and M6, give better prediction accuracies than the standard model, M1, and its derivatives, M3 and M4. Smaller values indicate better accuracy.

show that the standard model overpredicts the running times of both programs. The bottom graphs show that our model is more accurate. Here the crossover occurs at $b = 4$, but for the Sun E5000 the crossover occurs at $b = 8$, with the standard model again choosing the wrong algorithm.

7 Conclusions

This paper has described how cache effects can be incorporated into a communication comst model by including a term that is a function of MLT, the number of unique memory lines touched. This term is significant because it is sensitive to data layout, effectively modeling the number of compulsory and capacity cache misses. The benchmarking approach that we use is also crucial, because as it customizes the model to individual machines, it implicitly captures details such as cache access times.

We have presented formulae for computing MLT for rectangular subarrays and have shown that our model is significantly more accurate than the standard model for point-to-point communication operations. We have also conducted an experiment to predict the performance of two ZPL programs that compute the uniform convolution using different algorithms. Our model correctly predicts the crossover point between these two programs, whereas the standard model fails in this regard.

We are currently studying ways to model cache conflict misses, in an attempt to understand the importance of this as a model feature. Ultimately, our goal is to use our model to guide compiler optimizations, for example, to trade off load balance for extra communication and to choose optimal block sizes for block-cyclic data distributions.

References

[1] M. Lam, E. Rothberg, and M. Wolf. The cache performance and optimizations of blocked algorithms. In *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, Apr. 1991.

[2] J. Gunnels, C. Lin, G. Morrow, and R. van de Geijn. A flexible class of parallel matrix multiplication algorithms. In *12th Int'l Parallel Processing Symposium and 9th Symp. on Parallel and Distributed Processing*, Mar. 1998.

[3] M. Ivory. Modeling communication layers in portable parallel applications for distributed-memory multiprocessors. Master's thesis, UC Berkeley, 1996.

[4] B. Chamberlain, S. Choi, E. Lewis, C. Lin, L. Snyder, and W. Weathersby. The case for high level parallel

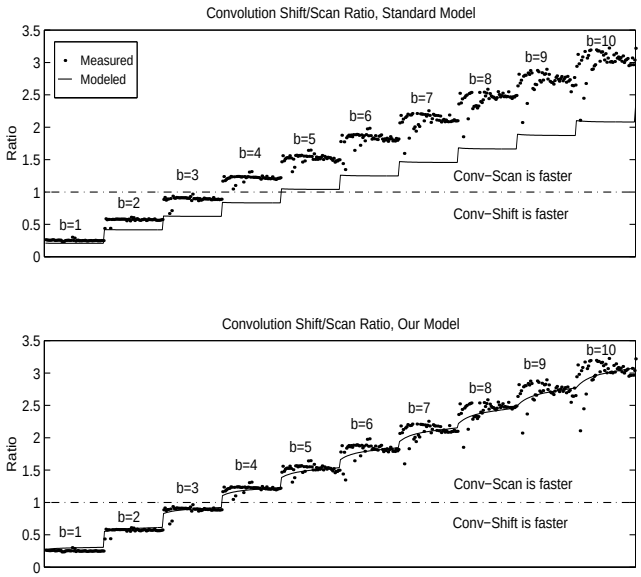

Figure 4: Modeling the relative performance of two uniform convolution programs on the Cray T3E. The x-axis represents different block sizes (b) and different matrix sizes. The y-axis gives the ratio of execution times for the Conv-Shift and Conv-Scan programs, so ratios above 1 indicate that Conv-Scan is faster, while ratios below 1 indicate that Conv-Shift is faster. Note that for $b = 4$ the standard model incorrectly predicts that Conv-Shift is faster.

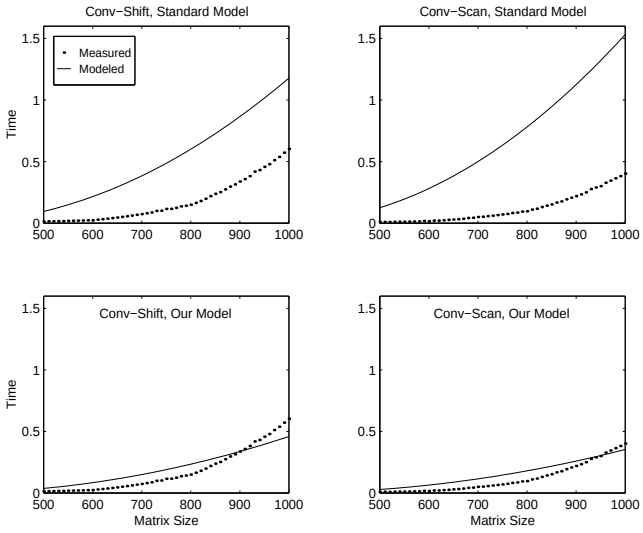

Figure 5: Standard model vs. our model for the two uniform convolution algorithms. Measurements are done on the Cray T3E with a 2×2 processor mesh. The standard model always overpredicts the running times of both of the algorithms, whereas our model gives accurate results.

programming in zpl. *IEEE Computational Science and Engineering*, 5(3):76–86, July-Sept. 1998.

[5] L. Snyder. *A Programmer's Guide to ZPL*. MIT Press.

[6] T. Fahringer. *Automatic Performance Prediction of Parallel Programs*. Kluwer Academic, 1996.

[7] T. Fahringer. Compile-time estimation of communication costs for data parallel programs. *J. of Parallel and Distributed Computing*, 39(1):46–65, Nov. 1996.

[8] O. Temam, C. Fricker, and W. Jalby. Cache interference phenomena. In *SIGMETRICS Conf. on Measurement and Modeling Computer Systems*, 1994.

[9] J. Ferrante, V. Sarkar, and W. Thrash. On estimating and enhancing cache effectiveness. In *Workshop on Languages and Compilers for Parallel Computing*, pp. 328–343, Aug. 1991.

[10] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: An analytical representation of cache misses. In *ICS'97, Vienna, Austria*, pp. 317–324, 1997.

[11] S. Ghosh, M. Martonosi, and S. Malik. Precise miss analysis for program transformations with caches of arbitrary associativity. In *8th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 1998.

[12] N. Amato, A. Pietracaprina, G. Pucci, L. Dale, and J. Perdue. A cost model for communication on a symmetric multiprocessor. TR98-004, Dept. of CS, Texas A&M Univ., January 1998.

[13] I. Hur and C. Lin. Modeling cache effects in interprocessor communication. Tech. Report 99-18, Dept. of CS, UT-Austin, June 1999.

[14] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. McGraw Hill, 1990.

[15] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford Univ. Press, 1995.

[16] V. Balasundaram, G. Fox, K. Kennedy, and U. Kremer. A static performance estimator to guide data partitioning decisions. COMP TR90-136, Dept. of CS, Rice Univ., October 1990.

[17] K. Kennedy, N. McIntosh, and K. McKinley. Static performance estimation in a parallelizing computer. CRPC Tech. Rep. TR92294-S, Rice Univ., Apr. 1992.

[18] C. Burrus and T. Parks. *DFT/FFT and Convolution Algorithms, Theory and Implementation*. John Wiley and Sons, 1985.