Optimizing Performance of Parallel Architectures Through Processor Configuration and Data Distribution*

C. D. Wong, E. W. Davis, R. E. Funderlic, and F. A. Smith
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206

April 8, 1999

Abstract

Efficient performance of highly parallel systems is a crucial problem. This paper presents a method for optimizing performance through effective logical configuration of processors and distribution of data. The paper extends a parallel system performance model for total execution time and a method to find an optimal solution for the model. Applications for the method are quite general. Properties of a given application and the target physical computing system are included in the method. The method has been validated with an air quality simulation testbed running on a Cray T3D parallel processor. Empirical results for this application are given.

Keywords: parallel performance, performance modeling, data distribution, data decomposition, multi-processor, grid structured problems.

1 Introduction

Optimizing performance of highly parallel systems is a critical and often difficult problem. One paper reports that delivered performance falls under 5% of theoretical peak performance with the use of currently available optimizing compilers [4]. Industry's ability to package computer systems with high performance potential exceeds the ability of users to extract that performance from the systems on many real applications. This paper directly addresses the issue of performance of an important class of applications on parallel systems, and is instructive with regard to other types of application.

Our work is focused on the problem of selecting an optimal data distribution and logical processor configuration for grid structured problems. Specifically, we extend an existing performance model and present a method for finding a solution that minimizes total execution time. The performance model is sufficiently general that it could be adapted to a variety of applications that are not included in the class of grid structured problems we consider. Our approach is simple enough to be accessible to users from various disciplines, and experimental results suggest a simple model can accurately predict the optimal data distribution and logical processor configuration. We do not require that users be familiar with intricacies of either the compiler or the machine, facilitating the

*This research has been funded in part by the U.S. Environmental Protection Agency under cooperative agreement CR822076-01-2 to North Carolina State University.

user's ability to port an existing application to a different platform.

Many models of performance have appeared in the literature, such as those in [1, 2, 5, 10, 13, 16, 25]. Indurkhya, Stone and Xi-Cheng [16] present a family of performance models for bus based multiprocessors where communication patterns fit a random distribution. A simplified version of this model is presented in widely used textbooks by Stone [24] and Shiva [21]. Hockney and Jesshope [13, 14, 15] introduce a performance model for vector operations and show how that model can be adapted to other aspects of parallel performance. Fox, et al. [10] analyze the performance of several scientific and other applications. Gallivan, et al. [12] consider the impact of memory hierarchy on performance. Culler, et al. [2, 3] and Alexandrov, et al. [1] consider communication performance in distributed memory machines. Toledo [25] describes an automatic prediction system based on analysis of runtime system calls. Fahringer [8] describes a comprehensive performance modeling system intended to guide compiler optimization.

Some of the models in the literature that employ a large number of parameters to accurately model performance provide a profiling tool so that a user can participate interactively in the optimization process. In the absence of an existing implementation, the complexity of these models is likely to render them inaccessible to typical users of parallel machines. Other models focus on a particular aspect of performance. These models, no matter how accurate, are likely to be insufficient for users. Other models are oriented toward a particular type of problem or a particular machine. The specificity of these models may be quite valuable to certain users but of little value to others.

Much effort has been expended on the development of data parallel languages intended to free the user of responsibility of managing communication primitives, e.g. [9, 20, 30]. However, these languages still require that the user choose the number of processors and specify how to distribute the data among the processors, or accept default distributions. Because of architectural differences among various parallel machines and differences in compilers, a program optimized for one machine may not have an optimal execution time on a different machine. When applications are to be run many times using different data sets, their lifespans may exceed that of a given machine. For such applications, the ability to tune performance for different machines is particularly important.

Our model extends that of Indurkhya, et al. [16], adapting it to grid structured problems executed on distributed memory machines. The model is based on the concept of separating those parts of a performance equation that are problem specific from those parts that depend on the compiler and the machine. Although the interdependence is such that a complete separation is difficult, we show that the use of justifiable simplifying assumptions permit the user to optimize a given problem for any number of machines.

We assume the target machine is a distributed memory machine. Our development is based on a two dimensional logical processor grid with an interconnection network connecting adjacent processors. We assume the physical processor configuration is either a two or three dimensional grid or torus, or one which permits the embedding of such a grid. However, this assumption is not essential to the model, and nothing in the model precludes its use for different physical configurations or higher dimensional logical configuations. Our work is motivated by a problem defined on a three dimensional grid, although we project the three dimensional grid onto two dimensions before distributing data elements over the two dimensional processor grid. Our development is thus directly applicable to multidimensional grid problems executed on machines where the processors can be logically configured in a two dimensional grid.

In addition, we have developed a method to find an optimal solution to the problem of selecting a data distribution and logical processor configuration. This method predicts the *relative* performance of candidate solutions. We have implemented the method and tested it in experiments that validate its utility. Our method correctly predicted an optimal solution in all of our experiments, and was reasonably accurate in predicting the relative performance of suboptimal solutions, as well.

The remainder of this paper is organized as follows. We provide an overview of data distribution schemes and terminology in Section 2. We describe the problem to be solved in Section 3 and introduce notation used thereafter. Our model formulation is detailed in Section 4. Section 5 includes our method for finding an optimal solution and experimental results for a simple test code. We also comment on the use of the method for porting applications. The scientific application that motivated this work is outlined in Section 6, along with experimental results for finding an optimal solution for it. Section 7 contains concluding remarks.

2 Data Distribution and Processor Configuration

A parallel version of a sequential program must partition the work to be performed among available processors if parallel execution time is to be less than the time required when using one processor. For grid structured problems, this partitioning typically is equivalent to an assignment of grid cells and their associated data to processors. Parallel languages provide mechanisms, typically compiler directives, by which the programmer may distribute data. For specificity, we use the terminology and data distribution model of High Performance Fortran (HPF) in the work below, although other parallel languages such as Vienna Fortran [30], Fortran D [9] and CRAFT (Cray Adaptive Fortran) provide similar functionality.

The HPF data distribution model is illustrated in Figure 1, adapted from [17]. The `ALIGN` and `REALIGN` directives are provided to specify that the distribution of one array should depend on the distribution of another array. The `DISTRIBUTE` and `REDISTRIBUTE` directives are used to specify how array elements are to be assigned to a logical, or abstract, processor grid. (The dimensions of a logical processor grid are specified using the `PROCESSORS` directive.) Finally, a mapping from a logical processor grid to the physical processors by the operating system completes the distribution of data to processors.

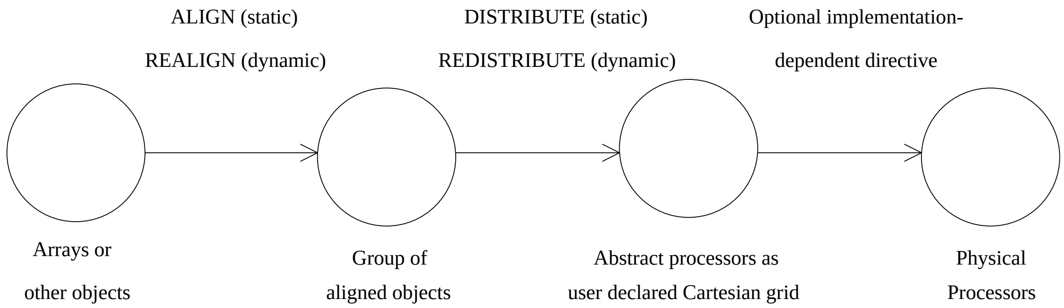

Figure 1: The HPF data mapping model.

Several authors, e.g. Dongarra, et al. [5] and Koelbel, et al. [17], classify the most widely used data partitioning schemes into three major types, i.e. *block*, *cyclic*, and *block cyclic*. (Some languages also support indirect mappings.) These partitioning schemes are oriented toward the arrays of data that are pervasive in scientific and engineering applications. The classification of these data partitioning schemes is in fact a convenience; the block and cyclic schemes are simply special cases of the block cyclic scheme. In the block cyclic scheme, blocks of data are assigned to processors in cyclic fashion, as described in [5] and [17]. The data partition is specified independently for each dimension of a multidimensional array.

We illustrate the variants of the block cyclic partitioning scheme in Figure 2 below. A one dimensional array of 17 elements is to be distributed among 4 processors. A block distribution assigns $\lceil 17/4 \rceil$ elements to as many processors as possible. A cyclic distribution assigns one element to each processor in turn, treating the processor array as cyclic. A block cyclic distribution assigns blocks of elements to each processor in turn.

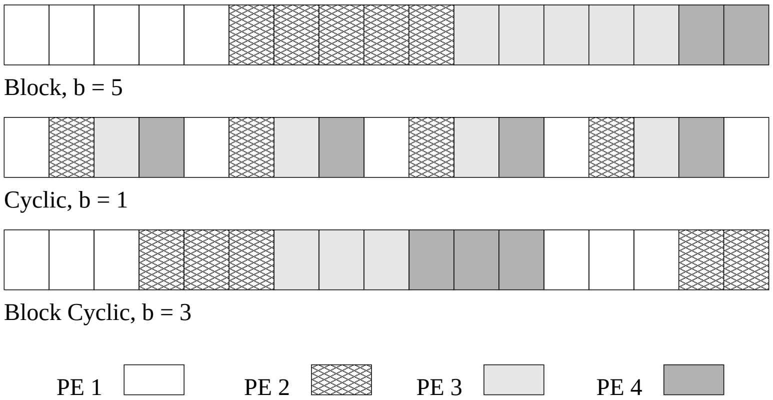

Figure 2: Block, cyclic, and block cyclic data partitions for a logical array of 4 processors and a data array with 17 elements, with block sizes 5, 1 and 3, respectively.

3 Problem Description

We consider the problem of how to choose a data distribution and the dimensions of a logical processor grid so as to minimize execution time. We assume that the sizes of data arrays are fixed, and that the number of physical processors is fixed. We also assume that data alignment has been handled, which is usually straightforward for grid structured problems.

Note that for a given logical processor grid, choosing a data distribution is equivalent to choosing a block size for each data dimension. Thus, our problem can be stated as follows. Given N processors and a data domain W, choose the dimensions of a logical processor grid and a block size in each data dimension so that the total execution time is minimized. We use the term *candidate solution*, or *solution*, to refer to a tuple consisting of the dimensions of a logical processor configuration and a block size for each data dimension.

The choices a user makes regarding logical processor configuration and data distribution can have a significant impact on program performance. Even on simple problems, a suboptimal solution can require twice as much time as an optimal solution, or worse. The situation is complicated when the machine imposes constraints on these choices. (Although some languages permit the user to omit specifying a distribution, the default distribution provided by the compiler is not constrained by the language specifications, and may be virtually anything.)

Consider the impact of the choice of block sizes on performance. The choice of block size affects the amount of loop overhead, the amount of communication required, and the extent to which the workload is balanced, or evenly partitioned, among processors.

We illustrate the impact on loop overhead of the choice of block size with an example. Consider modifying the sequential code on the left in Figure 3 so that the work is partitioned for parallel execution. Given a logical processor grid with dimensions $n_r \times n_c$, a data array with dimensions $w_r \times w_c$, and a block cyclic distribution using blocks with dimensions $b_r \times b_c$, every processor would need to execute code similar to that on the right in Figure 3. The code on the right steps from block to block in the outermost loops, and through elements within a block in the innermost loops. The deeper nesting implies additional overhead for integer operations for loop control. While constructs such as Fortran 95's FORALL may obviate the need for the user to explicitly write such nested loops, the compiler must still produce something similar, and time is required for the additional overhead.

```
DO I = 1, WR
  DO J = 1, WC
    ...
    ...
    ...
    ...
    ...
    ...
    ...
  ENDDO
ENDDO
```

```
DO I = 1, WR, NR*BR
  DO J = 1, WC, NC*BC
    DO II = 1, BR
      DO JJ = 1, BC
        ...
        ...
        ...
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

Figure 3: Loop structure for equivalent sequential and parallel codes

Observe that if a block distribution is used, the outer loops in the code on the right in Figure 3 are unnecessary. If a cyclic distribution is used, the inner loops are not needed.

Block size also affects load balancing, as the following one dimensional example illustrates. Given a data array of 30 elements distributed in block cyclic fashion onto a logical grid with 5 processors, Table 1 shows the number of elements assigned to each processor for various block sizes. Also, the ratio of the number of elements assigned to the first and last processor is indicated. Another measure of load imbalance is given as well, the ratio of the number of elements assigned to the first processor to the mean number of elements.

Block	Number of elements						
Size	PE1	PE2	PE3	PE4	PE5	Max/Min	Max/Avg
1	6	6	6	6	6	1.000	1.000
2	6	6	6	6	6	1.000	1.000
3	6	6	6	6	6	1.000	1.000
4	8	8	6	4	4	2.000	1.333
5	10	5	5	5	5	2.000	1.667
6	6	6	6	6	6	1.000	1.000
7	7	7	7	7	2	3.500	1.167
8	8	8	8	6	0	∞	1.333
9	9	9	9	0	0	∞	1.500
10	10	10	10	0	0	∞	1.667
12	12	12	6	0	0	∞	2.000
15	15	15	0	0	0	∞	2.500
30	30	0	0	0	0	∞	5.000

Table 1: Variation in load balancing due to block size

When the size of the data array w is a multiple of the number of processors n, perfect load balancing is achieved by choosing a block size $b = w/n$ or any divisor of w/n. In our example, $w = 30$ and $n = 5$, so the load is perfectly balanced when the block size is 1, 2, 3 or 6. Significant load imbalance is seen if other block sizes are chosen in the example above. The effect of block size on load imbalance also is illustrated in Figure 4, which conveys graphically some of the data in Table 1. Note that the maximum number of elements for each block size is exactly the number assigned to PE1 in Table 1.

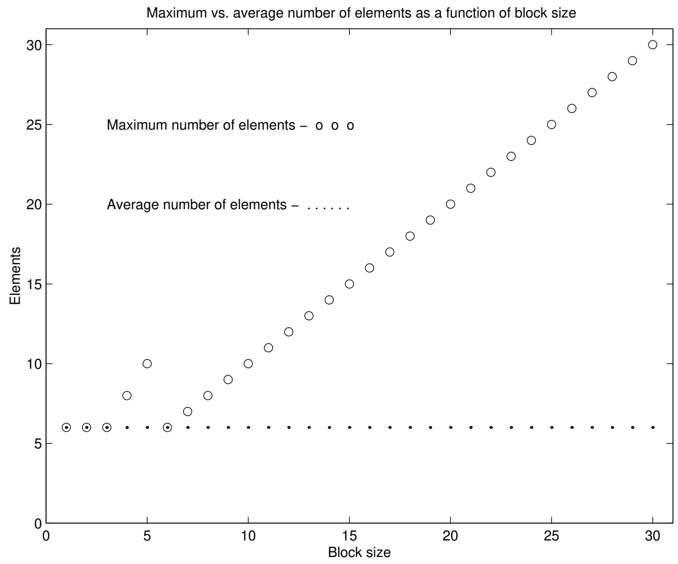

Figure 4: The effect of block size on load imbalance. $w = 30, n = 5$.

The amount of communication is also affected by the choice of block size. Consider a nearest neighbor communication pattern, where all computations for a grid cell depend only on data associated with that cell and cells adjacent to it. When adjacent grid cells are assigned to the same processor, no communication is required; when adjacent cells are assigned to distinct processors, data must be communicated. Given such a communication pattern, the amount of communication is proportional to the surface area of the blocks.

We illustrate the effect of block size on the amount of communication in a simple case in Figure 5. Assuming a 2×2 processor grid and a 4×4 data domain, Figure 5 indicates the amount of communication required for nearest neighbor communication when block sizes are 2×2 (on the left) and 1×1 (on the right).

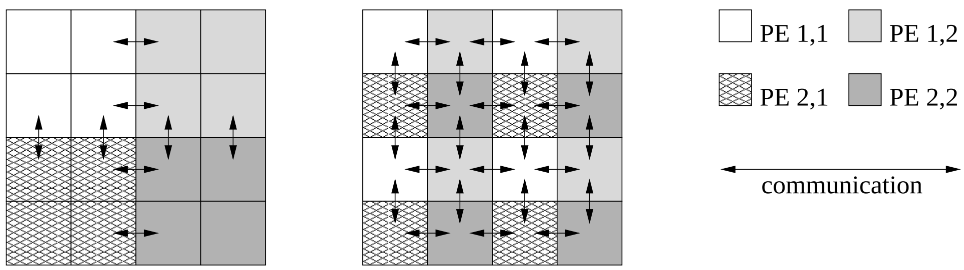

Figure 5: The effect of block size on communication. The number of communications is proportional to the perimeter of the blocks.

When communication cannot be overlapped with computation, the time required for communication represents an undesirable overhead. Thus, communication overhead drives the optimal data distribution toward larger block sizes. If there are constraints on block sizes, such as the power of two constraints imposed by the Cray T3D, large block sizes may result in significant load imbalance. In such cases, consideration of load balancing may drive the optimal data distribution toward smaller block sizes.

4 Model Formulation

Our performance model extends that of Indurkya, et al. [16] and assumes that communication is not overlapped with computation. Our representation of their model is given in equation (1) below.

$$T = R\Phi + C\Psi \tag{1}$$

where T is total execution time to be minimized,
R is the time required for one unit of computation,
Φ is the maximum number of units of computation assigned to any processor,
C is the mean time for one unit of communication between two processors, and
Ψ is maximum number of inter-processor communications assigned to one processor.

An optimal choice of data distribution and logical processor grid minimizes T. Non-optimal solutions increase either the computation time or the communication time, or both, in a way that increases T. Variations of this model have been studied analytically in order to gain understanding of parallel processing systems [24]. This study specifically considers the extension of the model to grid structured problems executed on distributed memory machines. However, the model is not limited to such problems. Our work illustrates how it is possible to apply a very general model to a particular type of problem.

Observe that Φ and Ψ depend only on the problem and the choice of data distribution and processor configuration. In contrast, R is largely machine dependent. The unit of computation is often chosen as one *flop*, or floating point operation. If flops are used and integer operations such as loop overhead are ignored, then R is essentially a constant. R may vary due to the effects of cache (see, e.g., [12, 28]) and, as noted previously, data distribution affects loop overhead. However, for many parallel applications, treating R as constant is reasonable, whether the unit of computation is one flop or a larger unit, such as all computations for a single grid cell.

In the discussion of grid structured problems that follows, we use the computation for a single grid cell as our unit of computation. Similarly, our unit of communication in the definition of C above is the communication between adjacent processors associated with a single grid cell.

The C term in equation (1) depends not only on the machine, but also on the data distribution, logical processor configuration, and the compiler (see, e.g. [1, 2, 7, 14]). In particular, these factors can affect the number of words in a particular message. Since messages generally require significant startup time [1, 2, 7], the mean time to communicate a word of data varies as a function of the number of words in a message. In general, it is advantageous to choose a data distribution and logical configuration that facilitates longer messages. However, the compiler plays an important role in determining the length of messages, and should not be overlooked. Compiler optimizations may prefetch data, combine messages, or use other means to hide latency. Although the user may believe that larger block sizes correspond to larger (and thus fewer) messages, compiler optimizations may produce longer messages than expected for smaller block sizes.

Although data parallel languages relieve the user of the tedium of managing communication primitives, the use of compiler optimizations for such languages may make it more difficult to intuitively grasp the actual communication patterns when a program executes. We believe that most users do not know, and are unlikely to learn, the intricate communication details that are specific to a particular compiler on a given machine. This is especially true when users work with various machines. Our work shows that it is possible to find an optimal solution absent that knowledge.

Additional notation is needed to apply the model in equation (1) to grid structured problems. Recall that our problem is defined in terms of the number of processors to be used, N, and the dimensions of the application data domain, W. Although the model in equation (1) can be applied to a variety of problems by appropriate definitions of Φ and Ψ, the particular mathematical formulation to follow has been developed for two dimensional processor grids and data structures. We investigate block cyclic data distributions with various data block sizes. We also assume data block sizes are chosen so that all processors are utilized. Recall that our choices of units for computation and communication are based on the computation and communication associated with grid cells. Thus we use the following notation:

n_r and n_c:	the number of rows and columns of the logical processor configuration. $N = n_r \times n_c$
w_r and w_c:	the number of rows and columns of the data domain. $W = w_r \times w_c$
b_r and b_c:	the number of rows and columns of each data partition block.
ϕ_r and ϕ_c:	the maximum number of grid cells assigned to a single processor in row and column dimensions. $\Phi = \phi_r \times \phi_c$
ψ_v and ψ_h:	the maximum number of grid cell communications involving any one processor in vertical and horizontal directions. Note that $\Psi = \psi_v + \psi_h$, rather than the product.

Also, t_r is defined as $t_r = w_r - \lfloor w_r/(b_r n_r) \rfloor b_r n_r$, which is the number of grid cells remaining in a block cyclic distribution after blocks of size b_r are allocated n_r at a time, i.e. b_r cells are assigned to each of n_r processors. The concept is illustrated in Figure 6 for t_c, which is similarly defined.

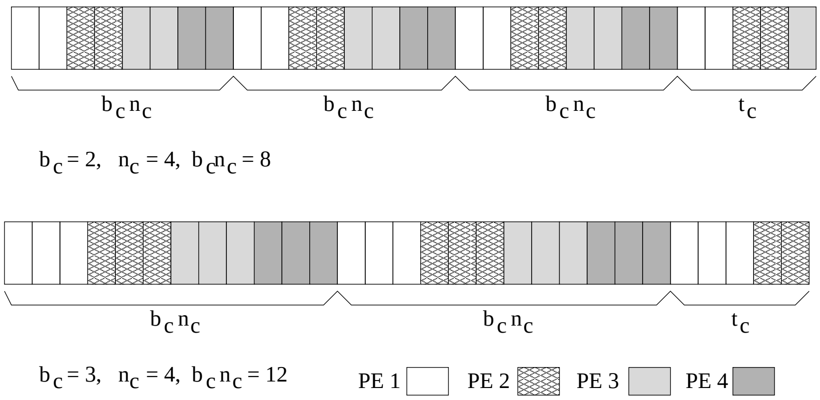

Figure 6: A graphical view of the definition of t_c.

Communication patterns in applications can be quite complex. For tractability we develop formulas for block cyclic data distributions and nearest neighbor communications. Even this comparatively simple situation has sufficient complexity as to produce model equations which are not readily solvable by analytic means.

Expressions for computation and communication in two dimensions are presented in equations (2) through (6). These equations are followed by a simple example that demonstrates how our notation is used.

We assume that data arrays are derived from an underlying grid and that array elements correspond to individual grid cells. It follows that a data distribution of aligned arrays is equivalent to an assignment of grid cells to processors. Assuming independent block cyclic distributions in each dimension, it is clear that the first processor in any row or column is assigned as many grid cells as any other processor in that row or column. Thus, given our decision to define a unit of computation as all computations for a single grid cell, ϕ_r is equal to the number of rows of grid cells assigned to the first row of processors. Thus we have the following equation.

$$\phi_r = \begin{cases} \lfloor w_r/(n_r b_r)\rfloor b_r + t_r & \text{if } 0 < \underbrace{w_r - \lfloor w_r/(b_r n_r)\rfloor b_r n_r}_{t_r} < b_r \\ \lceil w_r/(n_r b_r)\rceil b_r & \text{otherwise.} \end{cases} \tag{2}$$

Due to symmetry, ϕ_r and ϕ_c have the same form and thus ϕ_c can be represented by the same equation with the subscripts changed from r to c.

Equations for communication also differ only in subscripts for row and column directions. Vertical communication is defined in equation (4); the equation for horizontal communication is omitted since it has the same form. The number of cases in equation (4) is greater than for computation in equation (2). Cases are distinguished by the number of rows of processors and by the goodness of fit of data blocks as the blocks are interleaved onto processors. The goodness of fit criteria is given by

$$f_r = \lceil w_r/b_r \rceil \text{ modulo } n_r. \tag{3}$$

For an integral fit, $f_r = 0$ since each mapping iteration uses the same number of rows. When f_r assumes values of $1, 2, \ldots$, it represents the number of rows of processors that are needed in the final, non-integral, mapping iteration. Thus, we find that

$$\psi_v = \begin{cases} 0 & \text{if } n_r = 1 \\ \left(2\frac{\lceil w_r/b_r\rceil}{n_r} - 1\right)\phi_c & \text{if} f_r = 0 \text{ and } n_r = 2 \\ \left(2\left\lfloor\frac{\lceil w_r/b_r\rceil}{n_r}\right\rfloor\right)\phi_c & \text{if} f_r = 0 \text{ and } n_r > 2, \text{ or} f_r = 1 \\ \left(2\left\lfloor\frac{\lceil w_r/b_r\rceil}{n_r}\right\rfloor + 1\right)\phi_c & \text{if} f_r = 2 \\ \left(2\left\lfloor\frac{\lceil w_r/b_r\rceil}{n_r}\right\rfloor + 2\right)\phi_c & \text{otherwise.} \end{cases} \tag{4}$$

Equations (2) and (4) simplify if the product $b_r n_r$ is a divisor of w_r (or if $b_c n_c$ divides w_c), a situation that can be called a "matched" case. In such a case, floor and ceiling functions are unnecessary. Since we assume all processors are used, the computation parameter ϕ is independent of the choice of block size. Thus, in the matched case, the equation for ϕ_r simplifies to

$$\phi_r = w_r/n_r. \tag{5}$$

In the matched case, $f_r = 0$ by definition, so the equation for ψ_v simplifies to

$$\psi_v = \begin{cases} 0 & \text{if } n_r = 1 \\ (2w_r/(n_r b_r) - 1)\phi_c & \text{if } n_r = 2 \\ (2w_r/(n_r b_r))\phi_c & \text{otherwise.} \end{cases} \tag{6}$$

Given our choice associating units of computation and communication with grid cells, it is clear that Figure 4 illustrates the relation of ϕ to block size b for the example in Table 1. If we again consider a one dimensional array of 30 elements distributed over 5 processors, Figure 7 illustrates the relation of ψ to block size b. In contrast to ϕ, Figure 7 illustrates that ψ is a monotone decreasing function of b. As these figures and the form of the equations above suggest, Φ and Ψ are not continuously differentiable over the reals. Thus, we do not expect to minimize T in equation (1) by analytic means. Instead, various candidate solutions must be examined.

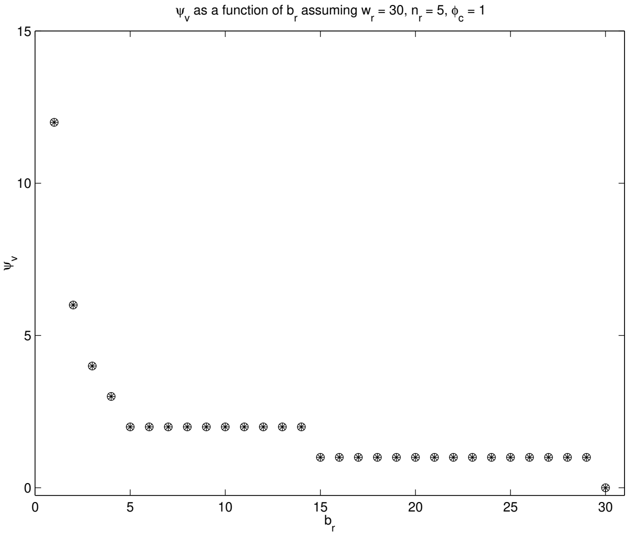

Figure 7: Variation in ψ for a one dimensional array of 30 elements.

Consider a simple two dimensional case where four processors are to be used. Assume that the dimensions of the data domain, data blocks, and processor grid are restricted to be powers of two, as is the case on the Cray T3D. Three possible processor configurations exist, namely 1 row by 4 columns, 2 by 2, and 4 by 1. Assume the data domain is 8 by 4, i.e. $w_r = 8$, $w_c = 4$.

For these values of the parameters, equations (5) and (6) can be applied, provided that block sizes are chosen so that all processors are used. In this example, each of the four processors will be assigned eight grid cells, as will be seen. Here, Φ is constant, although ϕ_r and ϕ_c depend on which logical configuration is chosen.

If the logical grid is 1 by 4, and all processors are used, then $b_r = 8$ and $b_c = 1$. The situation is illustrated in Figure 8a, where communication is indicated by arrows. If the logical grid is 4 by 1, and all processors are used, then $b_c = 4$ and b_r is either 1 or 2. These situations are illustrated in Figures 8b and 8c. Again, communication is shown by arrows. If the logical grid is 2 by 2, and all processors are to be used, then b_r is either 1, 2 or 4, and b_c is either 1 or 2. Of the six possible block sizes, the situations for 2 by 2 and 4 by 2 block sizes are shown in Figures 8d and 8e, respectively.

Now, Figure 8a illustrates the first possibility in equation (6). Since $n_r = 1$ there is no communication in the vertical direction. The second possibility in equation (6) is illustrated in Figures 8d and 8e. Since $n_r = 2$, all processsors are assigned blocks which require vertical communication in only one direction, i.e. the blocks on the top and bottom edges of the data domain. Any other blocks require communication in both directions vertically, as indicated in Figure 8d.

Finally, Figures 8b and 8c illustrate the third possibility in equation (6). When $n_r > 2$, all blocks assigned to the second processor require communication in both vertical directions.

The general situation, described in equation (4), is similar to that illustrated in Figure 8. The first three cases in equation (4) are much like those in (6), except the last block in a given dimension may have fewer grid cells than other blocks. The last two cases in (4) cannot occur in the matched case. However, the situation is similar to those shown in Figures 8b and 8d, except that the last block in a given dimension is not assigned to the last processor in that dimension.

Essentially, equations (4) and (6) provide a measure of the horizontal portion of the perimeters of blocks that correspond to vertical communication. Using the corresponding equation for ψ_h

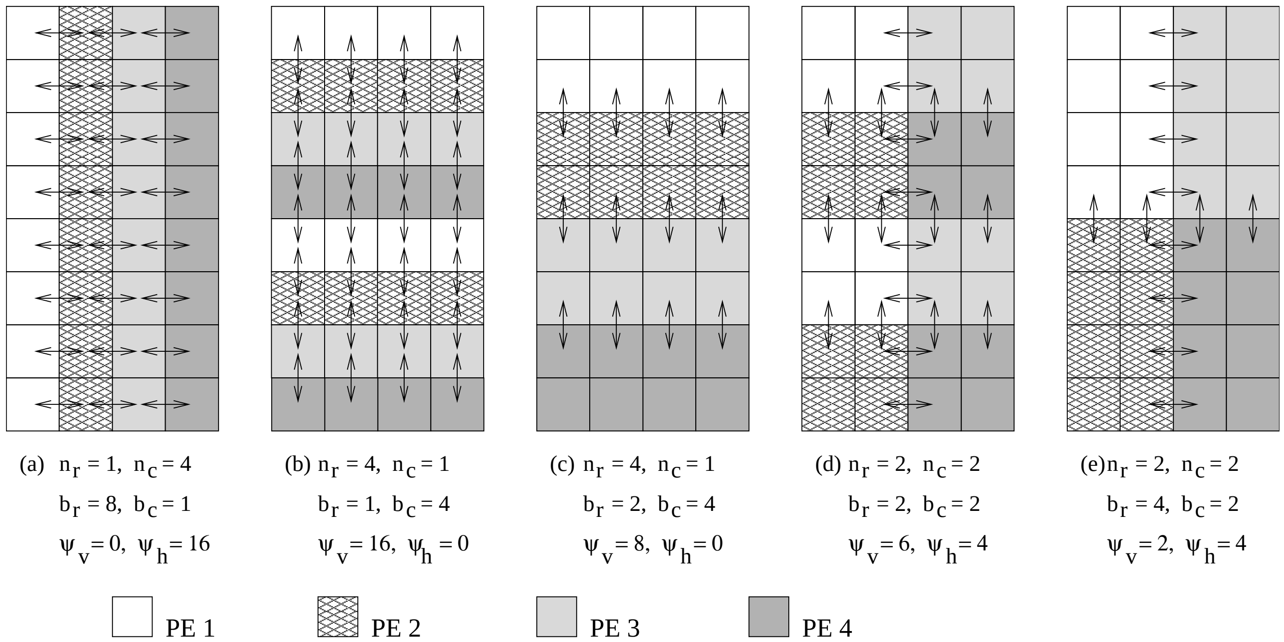

Figure 8: Illustration of various cases for ψ, from equation (6).

completes the calculation of $\Psi = \psi_v + \psi_h$.

The model in equation (1), as expanded in equations (2) through (6), along with the corresponding equations for ϕ_c and ψ_h, is the basis for determining the optimal processor configuration and data distribution.

5 Algorithm Description

Recall that the goal is to minimize total execution time as expressed in equations (2) through (4) by determining an optimal processor configuration and data distribution. Due to the piecewise and non-differentiable nature of the Φ and Ψ functions, a closed-form analytic solution appears infeasible. Thus a programmed solution approach was used with an algorithm that searches for the optimal solution within the solution space.

Clearly, the size of the solution space grows exponentially as the size of the data domain and the number of processors increase. Even when the size of the solution space is restricted, as in the case of the Cray T3D's power of two restrictions, the number of candidate solutions can be quite large.

The reader is likely to have observed in the example in the preceding section that some candidate solutions can be easily dismissed as suboptimal when the product $b_r n_r$ divides w_r, or when $b_c n_c$ divides w_c. However, since there may not be complete freedom in choosing these parameters, the general case is more difficult.

Consider scaling equation (1) by dividing both sides by C. This yields the scaled equation

$$T/C = \Phi R/C + \Psi, \tag{7}$$

which is a linear equation in $\gamma = R/C$. Observe that for each candidate solution (n_r, n_c, b_r, b_c), we can calculate Φ and Ψ.

The remaining term in equation (7), R/C, is strongly tied to the physical system in use and the application being run. R and C, or their ratio, must be determined by measurements on particular systems. Direct measurement of R and C may be difficult, regardless of how they are defined, and certainly the peak or sustained performance figures claimed by a manufacturer for a particular machine may not be applicable to a given application.

Our approach is to experimentally determine a value for the ratio R/C. For many parallel applications, it is straightforward to calculate the time required for initialization and I/O because all input is performed at the outset and all output at the end of a run. The application that motivated this work iterates over an arbitrary number of time steps, as do many similarly structured applications. By limiting the number of timesteps, it is possible to make several runs and use the results to estimate R/C. By choosing several candidate solutions, one can determine a value of R/C for each run. Then a statistical fit, such as least squares, can be used to determine a representative value of R/C for the application and system. As would be expected, optimal configurations change with R/C. Thus it is important to have a good value for this ratio.

Compile time analysis coupled with knowledge of system hardware could also be used to arrive at an estimate of R/C. Hockney [15] advocates the use of benchmarks to derive system measurements.

The essential steps of the method to arrive at an optimal processor configuration and data distribution are:

1. Estimate R/C for your application on your system.

2. Supply parameters (N, w_r, w_c) to the model and search the model solution space, via a program, for candidate solutions (n_r, n_c, b_r, b_c).

3. Select an optimal solution (n_r, n_c, b_r, b_c) from the set of candidate solutions using the estimate of R/C found above.

5.1 Experimental results

The model and method have been validated using both a simple experimental test code and a significant computational application in air quality modeling (see section 6). The test code, using a four-neighbor communication pattern was:

```
DO 100 I = 1,M
   DO 100 J = 1,M
      DATA(I,J) = EXP(-DATA(I,J)) + (EXP(-DATA(I-1,J))
                  + EXP(-DATA(I+1,J)) + EXP(-DATA(I,J-1))
                  + EXP(-DATA(I,J+1))) / 4.0
100 CONTINUE
```

For this code running on a T3D, with the loop index $M = 78$, R/C was experimentally determined to be 16.8. Using this value, the program search of the model solution space was conducted, computing a predicted execution time for each processor configuration and data distribution. To limit the search, we restricted viable solutions to powers of two, consistent with T3D usage. The validation step was then to select the best configurations as predicted, and run the same configurations on the actual system. This was done for processor sets from 4 to 128 processors. Table 2 shows the results for the case of 32 processors. Cases with other numbers of processors resulted in different processor and data configurations, but all provided confirmation that the method produced the optimal result. In table 2 we limit the length to the 14 configurations whose predicted execution times were least and tabulate them in predicted order. (An example of a non-tabulated configuration is 4 x 8 processors with 8 x 8 data.) The configurations are listed along with predicted and measured execution times, and the rank of the measured execution time. Predicted execution

Table 2: The top 14 predicted configurations, in order, and their corresponding actual execution times and rank. The predicted best is also best in measured execution time.

Configurations in predicted time order		Predicted execution time	Measured execution time	
32 processors	data block sizes	normalized[1]	in milliseconds	rank
4 x 8	4 x 2	2.260	2.260	1
8 x 4	2 x 4	2.260	2.285	2
4 x 8	2 x 2	2.322	2.374	5
8 x 4	2 x 2	2.322	2.372	4
2 x 16	4 x 1	2.323	2.301	3
16 x 2	1 x 4	2.323	2.398	6
1 x 32	128 x 1	2.647	2.528	7
32 x 1	1 x 128	2.647	2.663	8
1 x 32	128 x 4	3.247	3.174	9
32 x 1	4 x 128	3.247	3.288	11
2 x 16	8 x 8	3.329	3.312	12
16 x 2	8 x 8	3.329	3.312	12
1 x 32	128 x 2	3.341	3.216	10
32 x 1	2 x 128	3.341	3.351	14

times are normalized using the actual execution time of the optimal configuration, since predicted times were in artificial units due to working with the ratio R/C.

Although there are some entries that are not in the predicted order, the configuration with the best predicted time has the best measured execution time. We have investigated the cause of the out-of-order entries and have concluded that they are due to the use of a fixed value for R/C for all configurations when in fact this ratio varies somewhat with the configuration on real hardware.

These results offer evidence that our approach is effective in finding optimal configurations. In Section 6 we present the motivating application and the results of applying this optimization technique. It will be seen that the method produced optimal results for the real application, thus strengthening the notion of validation.

5.2 Implications Regarding Porting Applications

A significant benefit of the model described above and the method used to determine an optimal solution is revealed when an application is ported to another machine. Assuming the space of candidate solutions is the same, some of the work done to find an optimal solution on one machine is applicable on another machine.

As noted above, $T/C = \Phi R/C + \Psi$ is a linear equation in $\gamma = R/C$. Letting $T' = T/C$, each candidate solution (n_r, n_c, b_r, b_c) corresponds to a line in the γ-T' plane. Figure 9 illustrates the concepts for a simple example where $N = 6$, $w_r = 8$, and $w_c = 4$.

As Figure 9 indicates, the optimal configuration depends on the value of R/C. However, Figure 9 is applicable to this specific problem regardless of the platform. The slope and intercept of each line in the figure are precisely the values of Φ and Ψ computed for the particular configuration

[1] Execution times were predicted using artificial units which have been normalized to the minimum measured time.

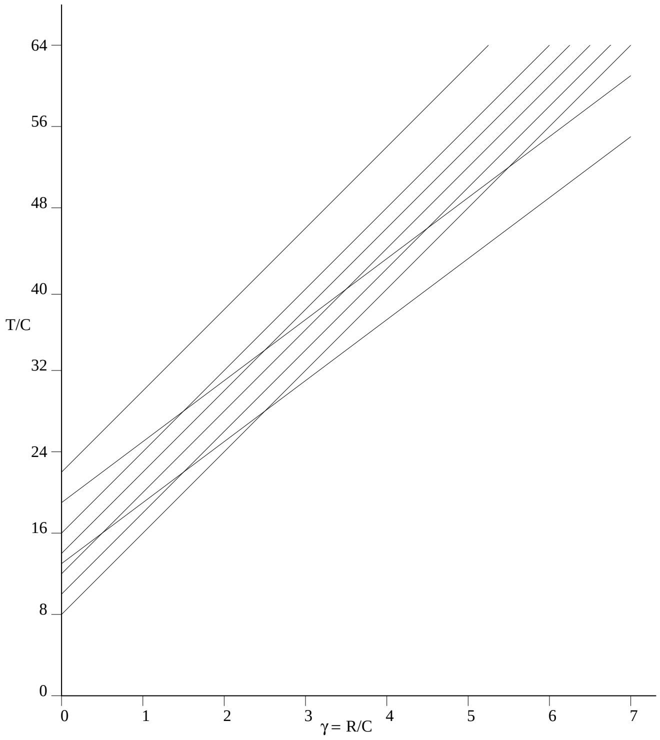

Figure 9: A graphical perspective of the relation of T/C to R/C for various candidate solutions.

corresponding to the line. Once the information represented in a figure similar to Figure 9 is computed, i.e. the value of Φ and Ψ for each configuration, our method requires only that R/C be estimated for a different machine; we then choose the solution corresponding to the minimal line for that value of R/C. In short, one need not repeat the search of the model solution space for potential solutions (n_r, n_c, b_r, b_c).

Table 3 lists all candidate solutions for the problem represented by Figure 9 where $N = 6$, $w_r = 8$, and $w_c = 4$, assuming block sizes are powers of two. (We assume that all processors are used except when the logical processor grid is 1 by 6, since $w_c = 4$.) For each candidate solution, Φ and Ψ are also listed. Each candidate solution corresponds to a line in Figure 9. As illustrated in Figure 9, the optimal solution (n_r, n_c, b_r, b_c) is (3, 2, 2, 2) when $R/C < 2.5$. If $R/C > 2.5$, the optimal solution is (3, 2, 1, 2).

The problem represented in Table 5.2 and in Figure 9 is much simpler than that in most applications, since there are far fewer candidate solutions. However, our experience suggests that Figure 9 is representative of similar figures for real applications in two important respects. First, there tend to be parallel lines in any such figure, implying that some candidate solutions can be immediately dismissed. Second, the number of solutions which are optimal for some value of R/C is generally small relative to the number of candidate solutions. It should be clear that identifying those candidate solutions which could be optimal is beneficial in choosing which configurations to use in finding an experimental value of R/C.

Figure 9 also provides insight into how the model may successfully identify the optimal solution while producing discrepancies between the predicted and actual relative performance of suboptimal solutions, as in Table 2. For any fixed value of γ, say γ_0, we can generate a sequence of solutions based on the order in which the lines corresponding to the solutions intercept the vertical line $\gamma = \gamma_0$. Thus, if our estimate of γ varies even slightly from the true value, the predicted order of suboptimal solutions may differ from the order generated by actual measurement. Since our

n_r	n_c	b_r	b_c	ϕ_r	ϕ_c	Φ	ψ_v	ψ_h	Ψ
1	6	8	1	8	1	8	0	16	16
2	3	1	1	4	2	8	14	8	22
2	3	2	1	4	2	8	6	8	14
2	3	4	1	4	2	8	2	8	10
3	2	1	1	3	2	6	10	9	19
3	2	1	2	3	2	6	4	9	13
3	2	2	1	4	2	8	4	12	16
3	2	2	2	4	2	8	4	4	8
6	1	1	4	2	4	8	12	0	12

Table 3: Φ and Ψ for various candidate solutions (n_r, n_c, b_r, b_c) when $N = 6$, $w_r = 8$, and $w_c = 4$.

goal is to find an optimal solution, we do not regard any disorder among suboptimal solutions as a shortcoming.

6 Applications in Air Quality Modeling

Highly parallel computers are becoming increasingly necessary for the implementation of air quality models. Over many years of development, model complexity has grown to accommodate increases in the number of chemical species and chemical reactions considered, to include additional physical phenomena, to extend the geographical region covered by the models, to increase time and spatial resolution, and to lengthen the duration of model scenario runs. This increased complexity places very high computational demands on computer systems running the models. Future air quality models are expected to have further growth in complexity as well as increased functionality. It is also desirable that they run fast enough that multiple scenarios can be studied. These factors will demand computing resources which significantly exceed those in use at the present. Thus efficient use of high performance parallel computers is important to future air quality studies.

Computational aspects of air quality have two distinct parts: (1) the chemical reactions that can occur in the atmosphere and (2) the movement of chemical concentrations in the air. We have been investigating both the gas-phase chemistry and the transport of these chemicals following the Regional Oxidant Model [19]. An implementation has been completed on the Cray T3D [26, 27]. In this section we describe the application and use it for validation of the processor and data distribution method.

6.1 Chemistry and Transport Equations

The concentration of any chemical species, α, is altered either by production or consumption through chemical reactions. This idea can be depicted mathematically as:

$$\frac{\partial C(\alpha)}{\partial t} = \sum\sum \kappa_{\alpha_{ij}} C(i)C(j) + \sum \hat{\kappa}_{\alpha_i} C(i) \ [22] \qquad (8)$$

where $C(\alpha)$ = the concentration of species α

$\kappa_{\alpha_{ij}}$ = the bi-molecular rate constant for reactions that produce or consume species α by interaction with species i and j

$\hat{\kappa}_{\alpha_i}$ = the mono-molecular rate constant.

In the current problem formulation there are 58 equations with 158 chemical reactions, each one corresponding to a chemical species. These 58 equations form a system of differential equations associated with an initial condition, C_0 , such that:

$$\frac{\partial C}{\partial t} = f(C, t), \qquad C(t_0) = C_0 \tag{9}$$

where C is a species concentration vector.

Each chemical species behaves differently, according to its own physical characteristics. Various elementary chemical reactions typically have velocities (time-scales) that differ substantially. Such large differences result in rate constants $\kappa_{\alpha_{ij}}$ and coefficients of the differential equations which differ by orders of magnitude. This causes the system of differential equations to be stiff. (See, e.g. [23].) Stiffness eliminates the applicability of explicit numerical methods due to instability. Generally, methods such as the Backward Differentiation Formula (BDF), implicit Runge-Kutta, and other methods are used to cope with the stiffness of the problem. In our implementation, the stiff system of differential equations is solved using the quasi-steady-state approximation (QSSA) algorithm [29].

The advection and transport process of atmospheric models is typically driven by the following equation:

$$\frac{\partial C}{\partial t} + \frac{\partial (uC)}{\partial x} + \frac{\partial (uC)}{\partial y} = 0 \tag{10}$$

where C represents the concentration of a given chemical species and u the velocity field. In execution, the gas-phase chemistry is spatially independent. That is, new concentrations are computed in a volume of air space based on initial concentrations and other effects, such as sunlight. No communication is required and optimization reduces to minimizing computation time via load balancing. However, communication is a strong component of the transport process. Following the chemical species concentration updates, new initial conditions at a grid cell are determined by a backtrack process that computes the source and concentration of chemicals transported to the cell. Without fully describing this process, we note that transport calculations are not limited to nearest-neighbor communication, but involve interpolation over a region of grid cells. We applied our optimization method to the transport phase with the following results.

For the transport code running on a T3D, R/C was experimentally determined to be 95. Following the optimization method, this value was used in a program driven search of the model solution space. Predicted execution times for each processor configuration and data distribution were gathered. Again, we limited the search space to configurations which were powers of two, as is consistent with T3D usage. Tests were run where the number of processors was a power of two between 8 and 128, inclusive.

6.2 Optimization Results

Table 4, similar in construction to Table 2, shows the results of experimental validation of our method for the air quality application in the case of using 128 processors. Recall that the table entries are in order of the optimization method's predicted time performance. The first five rows of the table show direct correspondence with the best five empirical results. Experiments for all other numbers of processors in the set also predicted an optimal configuration and distribution, providing strong validation of the method. Full results of this research are in [26].

[2] Execution times were predicted using artificial units which have been normalized to the minimum measured time.

Table 4: Validation results from an air quality application for 128 processors.

Configurations in predicted time order		Predicted execution time	Measured execution time	
128 processors	data block sizes	normalized[2]	in milliseconds	rank
64 x 2	1 x 1	0.504	0.504	1
64 x 2	1 x 2	0.506	0.504	1
32 x 4	2 x 1	0.508	0.509	3
32 x 4	1 x 1	0.512	0.513	4
64 x 2	1 x 8	0.534	0.531	5
32 x 4	2 x 2	0.537	0.543	8
64 x 2	1 x 4	0.537	0.537	7
16 x 8	4 x 1	0.540	0.531	5
8 x 16	8 x 4	0.546	0.584	11
16 x 8	4 x 8	0.546	0.576	9
8 x 16	4 x 4	0.566	0.578	10
16 x 8	4 x 4	0.566	0.585	12

7 Conclusions

Starting with a simple description of a problem, using N processors for an application with data domain size of w_r by w_c, we have developed an approach to finding an optimal logical configuration of processors and block sizes for block-cyclic data distributions in parallel systems. The method has been validated on a test code and on a real application. It is a general method, which can be used on many applications having 2-, 3-, and higher dimensions and on many computer systems. Specific formulations and details have been presented for two-dimensional configurations.

Our current work involves further investigation into this model and method with emphasis on the sensitivity of the approach to the R/C ratio. An "insensitive" application would be one in which an optimal processor configuration and data distribution hold over a large range of values of the ratio. A highly sensitive application would be one in which a small change in the ratio necessitates a change in processor configuration or data distribution needed for optimality.

8 Acknowledgments

We are pleased to recognize the technical assistance of Dr. Jeffrey O. Young of the Atmospheric Sciences Modeling Division, Air Resources Laboratory, National Oceanic and Atmospheric Administration, on loan to the National Exposure Research Laboratory, U. S. Environmental Protection Agency, and of Vance L. Shaffer of SGI – Cray Research. We also acknowledge The North Carolina Supercomputing Center, Cray Research, and the U.S. EPA for providing access to Cray T3D computing resources.

References

[1] A. Alexandrov, M. Ionescu, K. Schauser, C. Scheiman, "LogGP: Incorporating long messages into the LogP model - One step closer towards a realistic model for parallel computation," *7th Annual Symposium on Parallel Algorithms and Architecture* (SPAA '95), pp. 95-105, July, 1995.

[2] D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, T. von Eicken, "LogP: A practical model of parallel computation," *Communications of the ACM*, v. 39, n. 11, pp. 78-85, Nov. 1996.

[3] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, T. von Eicken, "LogP: Towards a realistic model of parallel computation," *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, May 1993.

[4] G. Cybenko and D. J. Kuck, "Revolution or evolution?," *IEEE Spectrum*, September 1992, pp. 39-41.

[5] J. J. Dongarra, I. S. Duff, D. C. Sorenson, and H. A. van der Horst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, 1991.

[6] J. J. Dongarra and J. W. Walker, "Software libraries for linear algebra computations on high performance computers," *SIAM Review*, Vol. 37, No. 2, June 1995, pp. 151-180.

[7] T. Dunigan, "Communication performance of the Intel Touchstone DELTA Mesh," Tech. Report TM-11983, Oak Ridge National Lab, Oak Ridge, TN, October 1994.

[8] T. Fahringer, *Automatic Performance Prediction of Parallel Programs*, Kluwer Academic Publishers, Norwell, MA, 1996.

[9] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu, "Fortran D language specification," Department of Computer Science Rice COMP TR90079, Rice University, March 1991.

[10] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker, *Solving Problems on Concurrent Processors, Vol. I*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[11] B. Furht, "Parallel computing: glory and collapse," *Computer*, November 1994, pp. 74-75.

[12] K. Gallivan, W. Jalby, A. Malony, and H. Wijshoff, "Performance prediction for parallel numerical algorithms," International Journal of High Speed Computing, v. 3, n. 1, pp. 31-62, 1991.

[13] R. Hockney and C. Jesshope, *Parallel Computers: Architecture, Programming and Algorithms*, Adam Hilger, Ltd., Bristol, England, 1981.

[14] R. Hockney and C. Jesshope, *Parallel Computers 2*, Adam Hilger, Ltd., Bristol, England, 1988.

[15] R. Hockney, *The Science of Computer Benchmarking*, SIAM, Philadelphia, PA, 1996.

[16] B. Indurkhya, H. S. Stone and L. Xi-Cheng, "Optimal partitioning of randomly generated distributed programs," *IEEE Transactions on Software Engineering*, SE-12, No. 3, March 1986, pp. 483-495.

[17] C. Koelbel, D. Loveman, R. Schrieber, G. Steele Jr. and M. Zosel, *The High Performance Fortran Handbook*, The MIT Press, Cambridge, MA, 1994.

[18] D. J. Kuck, *High Performance Computing*, Oxford University Press, 1996.

[19] R. G. Lamb, *A Regional Scale (1000km) Model of Photochemical Air Pollution, Part I - Theoretical Formulation*, EPA-600/3-83/035, U. S. Environmental Protection Agency, Research Triangle Park, NC, 1983.

[20] D. B. Loveman, "High Performance Fortran," *IEEE Parallel and Distributed Technology*, v. 1, n. 1, pp. , February 1993.

[21] S. G. Shiva, *Pipelined and Parallel Computer Architectures*, Harper Collins, New York, 1996.

[22] W. R. Stockwell, P. Middleton, and J. S. Chang, "The Second Generation Regional Acid Deposition Model Chemical Mechanism for Regional Air Quality Modeling," *J. Geophys. Res.*, 1990 95, pp. 16343-16367.

[23] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis, 2d. ed.*, Springer-Verlag, New York, 1993.

[24] H. S. Stone, *High Performance Computer Architecture*, Third Edition, Addison-Wesley, 1993.

[25] S. Toledo, "Performance Prediction With Benchmaps," *Proc. of the 10th Int. Parallel Processing Symposium*, IEEE Computer Society Press, Los Alamitos, CA, pp. 479-484.

[26] C. D. Wong, *Performance Enhancement Techniques and Methodologies for Highly Parallel Processing Systems in Scientific Computational Applications*, Ph.D. Dissertation, North Carolina State University, May 1996.

[27] C. D. Wong, J. O. Young and E. W. Davis, "Optimizing a Photochemical Air quality Model Gas-Phase Chemistry Solver for Parallel Processing," *Proc. of the Conf. on Computing in Environmental Resource Management, Research Triangle Park, NC, Dec. 2-4, 1996*, pp. 12345-67899.

[28] C. D. Wong, E. W. Davis, and J. O. Young, "A Software Approach to Avoiding Spatial Cache Collisions in Parallel Processor Systems," *IEEE Transactions on Parallel and Distributed Systems*, v. 9, n. 6, pp. 601-608.

[29] J. O. Young, E. D. Sills and D. A. Jorge, *Optimization of the Regional Oxidant Model for the Cray Y-MP*, IPA/600/R-94/065, U. S. Environmental Protection Agency, Research Triangle Park, NC, January, 1993.

[30] H. Zima, P. Brezny, B. Chapman, P. Mehrotra, and A. Schwald, "Vienna Fortran - a language specification," Internal Report 21, ICASE, Hampton, VA, March 1992.