Modeling Parallel Computation

Lorie M. Liebrock
Ken Kennedy

CRPC TR94499
May 1994

Rice University
Center for Research on Parallel Computation - MS 41
6100 South Main Street
Houston, TX 77005-1892

This research was supported by the Center for Research on Parallel Computation, a National Science Foundation Science and Technology Center, through Cooperative Agreement No. CCR-9120008, NSF/NASA Agreement No. ASC-9213821, and ONR Agreement No. N00014-93-1-0158. Use of the Intel i860 was provided under a Texas CER Grant No. CISE 8619893. Contact Lorie M. Liebrock via e-mail at *lorie@cs.rice.edu* and Ken Kennedy at *ken@cs.rice.edu*.

1 Introduction

We attempt to model parallel computations that are composed of one or more irregularly coupled regular meshes (ICRMs). The model will be used to determine whether a distribution of an ICRM problem is a good one. Further, it will be used to evaluate and improve algorithms for finding distributions of ICRM problems.

2 Assumptions

Assumptions for the model are as follows. First, the simulation problem in the computation must be static. The amount of computation is the same for each element of each DECOMPOSITION. The computation does not have to be uniform across all DECOMPOSITIONs, just internally for each individual DECOMPOSITION. We also assume that all of the various operations can be grouped into at most three groups with all components of each group having approximately the same runtime. Each pair of "neighbors" in each dimension of a DECOMPOSITION performs the same set of communications. This does not imply that the communication is the same in rows and columns, but that the amount of communication in each row is the same as that in every other row and similarly for the columns. Further, each pair of "neighbors" in a given coupling between any fixed pair of DECOMPOSITIONs perform the same set of communications. Since we are talking about mapping DECOMPOSITIONs, which represent the entire set of data structures for a physical component of a system, index arrays will always be mapped the same way as the arrays they are used to index if they are used to index intra-DECOMPOSITION variables. This does not imply that communication will not be necessary for intra-DECOMPOSITION index arrays. If index arrays are used to index inter-DECOMPOSITION variables, they may be mapped differently than the variables they index and some communication will definitely be required.

3 Notation

Given a set of coupled DECOMPOSITIONs, $D = \{D_1, D_2, ..., D_N\}$, and a set of processors, $P = \{P_1, P_2, ..., P_M\}$, the following notation is used in the model.

For each DECOMPOSITION, D_i, we have the following quantities. The elements of D_i are e_{ij} for $1 \le j \le |D_i|$. The number of computations performed on each element, e_{ij}, of D_i is $add_i + function_i + divide_i$, where add_i is the number of computations in the fastest group, $function_i$ is the number of computations in the medium speed group, $divide_i$ is the number of computations in the slowest group. The number of memory cells needed for storing variables associated with element e_{ij} is $|e_{ij}|$. The set of communications between neighbors e_{ij} and e_{ik} in D_i is denoted $comm_{id}$, where d is the dimension for communication. The set of communications between neighbors e_{ij} in D_i and e_{jk} in D_j is denoted $comm_{ij}$.

Next we define some terms that will simplify the modeling discussion. Two DECOMPOSITIONs that are neighbors in the simulation are defined to be "coupled". When neighboring elements in coupled DECOMPOSITIONs are located on different processors, communication will result. The set of elements for which communication is necessary in such a mapping is defined as follows

$$Coupling(D_i, D_j) = \{(e_{ik}, e_{jl}) \mid e_{ik} \in D_i \;\&\; e_{jl} \in D_j \;\&\; e_{ik} \textit{ is coupled to } e_{jl}\}.$$

For each element of each DECOMPOSITION there is one processor, its home processor, on which it is permanently stored. The home processor of an element performs all computations for that element. The home processor or owner of an element e_{ij} is defined by:

$$Map(e_{ij}) = P_k \equiv e'_{ij}s \textit{ home processor.}$$

Note that we follow the owner-computes rule.

4 The Model

4.1 The Complete Model

The components being modeled are communication, computation, indirection, and the inspector. In all of the components we are interested in the maximum over all processors so that we get a worst-case upper bound on the total time of execution. This will allow us to compare different distributions of the same problem and select the better one.

Communication represents the cost associated with a series of "gets" with communication grouped to reduce overhead where possible. Note that communication occurs when subblocks of a DECOMPOSITION are mapped to different processors and when coupled DECOMPOSITIONs (or coupled subblocks thereof) are mapped to different processors.

Computation represents the cost of performing all of the computations associated with the elements of decompositions mapped to a given processor. This is simplified to use only three types of computations with only three associated execution times.

Indirection represents the cost of extra communication associated with the use of any index arrays.

The inspector represents the cost of running an inspector to determine what communication is necessary when compile-time techniques are not sufficient for such determination. When indirection is not used the inspector should not be needed.

For the complete model, we do a simple sum of the component terms:

$$\begin{aligned} Cost_{Model}(Map) &= Computation(Map) + Communication(Map) + Indirection(Map) \\ &\quad + Inspector(Map). \end{aligned}$$

4.2 Communication

Modeling of communication is based on the use of the "get" as opposed to the use of send/receive pairs.

In order to effectively model communication we have a two level approach. At the bottom, fine granularity, level we model the cost of individual communications. At the top, large granularity, level we model the cost of the set of communications imposed by a particular distribution.

For communication modeling we will use the following definitions.

- Let *hops* be the distance, in number of hops, between the source and sink for a communication.
- Let $cost_{neighbor}$ be the cost for communicating between nearest neighbor processors.
- Let $hops_{general}$ be the number of hops used in general communication between any pair of processors.
- Let $cost_{byte}$ be the cost for communication of one byte between nearest neighbor processors. This is strictly greater than zero.
- Let *bytes* be the number of bytes being communicated.
- Let $cost_{startup\ constant}$ be the startup constant that is associated with calling the "get" command independent of what is being gotten or where it is located.
- Let $constant_{transfer}$ be the constant associated with buffering data on intermediate processors between the source and destination. This is used, for example, to model the pipelining effect in communication.

All of the constants, including the number of hops between any two processors, are a function of the parallel processor being used and are greater than or equal to zero.

4.2.1 Modeling Single Communications

A single communication on a given architecture may have a variety of costs associated with it depending on the size of the value being communicated, the distance between the processors that are the source and sink of the communication, and the startup cost of communication.

First, the number of hops associated with the distance between the source and sink of the communication is

$$H(hops) = \min(hops, hops_{general}).$$

On most architectures $hops_{general}$ will be the maximum distance between any pair of processors. The exceptions to this rule are machines like the Connection Machine, which have separate networks for reducing this maximum communication distance.

The total cost for a single communication is modeled as

$$\begin{aligned} C_T(hops, bytes) &= cost_{startup\ constant} + (H(hops) - 1) * cost_{neighbor} + bytes * cost_{byte} && (1) \\ &\quad + bytes * H(hops) * constant_{buffering}. && (2) \end{aligned}$$

where the current mapping in effect, *Map*, determines the location of the source and sink for communication. This is composed of the cost of the communication instruction, the cost of the setting up the route for wormhole routing, the cost of buffering the message if necessary, and the cost of buffering in intermediate processors between the source and sink.

4.2.2 Modeling for a Fixed Distribution of Decompositions

Once a set of DECOMPOSITIONs has been mapped to processors, the communication requirements for the simulation are fixed. Each communication fits into one of two categories: (1) between two neighboring elements of a single DECOMPOSITION which are assigned in the distribution to different processors or (2) between coupled elements of two neighboring DECOMPOSITIONs when the elements are assigned to different processors. These two categories correspond, respectively, to the two terms of the total simulation communication model, which is a function of the mapping:

$$Communication(Map) \quad = \quad \max_{P_x} \sum_{P_y} \left\{ C_T \left(\#hops(P_x, P_y), \#bytes \left(\sum_{(e_{ij}, e_{ik})} comm_i + \sum_{(e_{ij}, e_{mn})} comm_{im} \right) \right) \right\}$$

where $Map(e_{ij}) = P_x,\ Map(e_{ik}) = P_y, Map(e_{mn}) = P_y,\ and\ (e_{ij}, e_{mn}) \in Coupling(D_j, D_m)$. Here we use $\#hops$ and $\#bytes$ as the obvious functions. This assumes that all communication across a boundary that is of the same type can and will be grouped into a single vector communication. Such an assumption will give a slightly unfair advantage to random distributions as the extra work to group communications for random distributions will be ignored.

As an example, the communication timings on the Intel i860 resulted in the following approximate values for the timing variables:

Variable	*Timing (sec.)*
CostNeighbor	0.04
HopsGeneral	2
CostStartupConstant	0.04
ConstantBuffering	0.2
CostByte	0.00077

4.3 Computation

At the instruction level we model computations as having one of three fixed costs; so the model for computation is very simple:

$$Computation(Map) = \max_{P_m \in P} \left\{ \sum_{e_{ij} \ \mid \ Map(e_{ij}) = P_m} add_i * cost_{add} + function_i * cost_{function} + divide_i * cost_{divide} \right\}$$

where add_i is the number of computations, taking the same time (approximately) as an addition, per element in DECOMPOSITION D_i, and similarly for $function_i$ and $divide_i$. This three-cost component approach is used as some processors, such as the Intel i860, have costs of different orders of magnitude. For an example, on the i860, the following approximate timings were measured:

Operation	*Timing (sec.)*
multiply	5.3e-04
add	7.4e-04
function	4.2e-03
divide	9.4e-02

Since adds and multiplies take the same order of magnitude of time, adds and multiplies are grouped, for this machine, and classified as "adds" in the model.

4.4 Indirection

We model indirection as the cost of doing the "get" when the index array element does not reside on the processor performing the operation. Hence, for $X(IX(I))$ the cost of the indirection is

$$Indirection(Map) = C_T(\#hops(Map(IX(I)), Map(X(IX(I)))), \#bytes(IX(I)))$$

when $Map(IX(I)) \neq Map(X(IX(I)))$.

4.5 Inspector

The inspector's only variation in this model is when index arrays are used. Since this component is handled above via the indirection cost, the inspector overhead is fairly simple:

$$Inspector(Map) = \max_{P_m} \left\{ constant_{inspector} * \sum_{e_{ij} \mid Map(e_{ij})=P_m} |e_{ij}| * |program| \right\}$$

which is the data for all elements on P_m times the program size. This is actually a rather large upper bound on the cost for using the inspector. If there are no index arrays in the program, then communication should be analyzed at compile time and the inspector is not needed.

4.6 Machine 1: Intel i860

On the Intel i860, the following approximate computational timings were measured:

Operation	*Timing (sec.)*
multiply	5.3e-04
add	7.4e-04
function	4.2e-03
divide	9.4e-02

Since adds and multiplies take the same order of magnitude of time, adds and multiplies are grouped, for this machine, and classified as "adds" in the model.

Communication timings resulted in the following approximate values for the timing variables:

Variable	*Timing (sec.)*
CostNeighbor	0.04
HopsGeneral	2
CostStartupConstant	0.04
ConstantBuffering	0.2
CostByte	0.00077

5 Model Validation

5.1 Test Problem: 1-Dimensional Explicit Material Dynamics

1-dimensional explicit material dynamics provides an excellent test problem as I have run this problem on a variety of machines with many different distributions. This experience allows me to predict the relationships between the runtimes of various distributions and verify that the model produces valid results. In this test problem there are 32 "adds", 3 "functions", and 7 "divides" for each element in the mesh on each timestep. There is no indirection and there are 56 bytes of communication between neighbors on each timestep.

5.1.1 Single Decomposition Mapping

Here we simulate the flow of fluid in a single section of pipe. This is arguably the simplest type of problem for distribution as each element of the decomposition has the same computation and communication pattern (with the exception of minor variations at the boundaries or ends of the pipe).

In this first test case, we will just run the model on a single mesh with different distributions. The variations on distribution that are modeled include: load balanced distribution with best block map, load balanced with bad block map, load balanced with cyclic map, and not load balanced with best block map. These types of distributions are illustrated in Figure 1 for a twelve element mesh mapped onto a four processor (hypercube) machine. In the figure each element is numbered with the id of the processor who owns it. A one hundred element problem was used with four processors in the model evaluation. The predicted runtimes for the four different distributions are:

Distribution	*Predicted Time (sec.)*
load balanced with best block map	17.52
load balanced with cyclic map	19.59
load balanced with bad block map	18.68
not load balanced with best block map	18.21

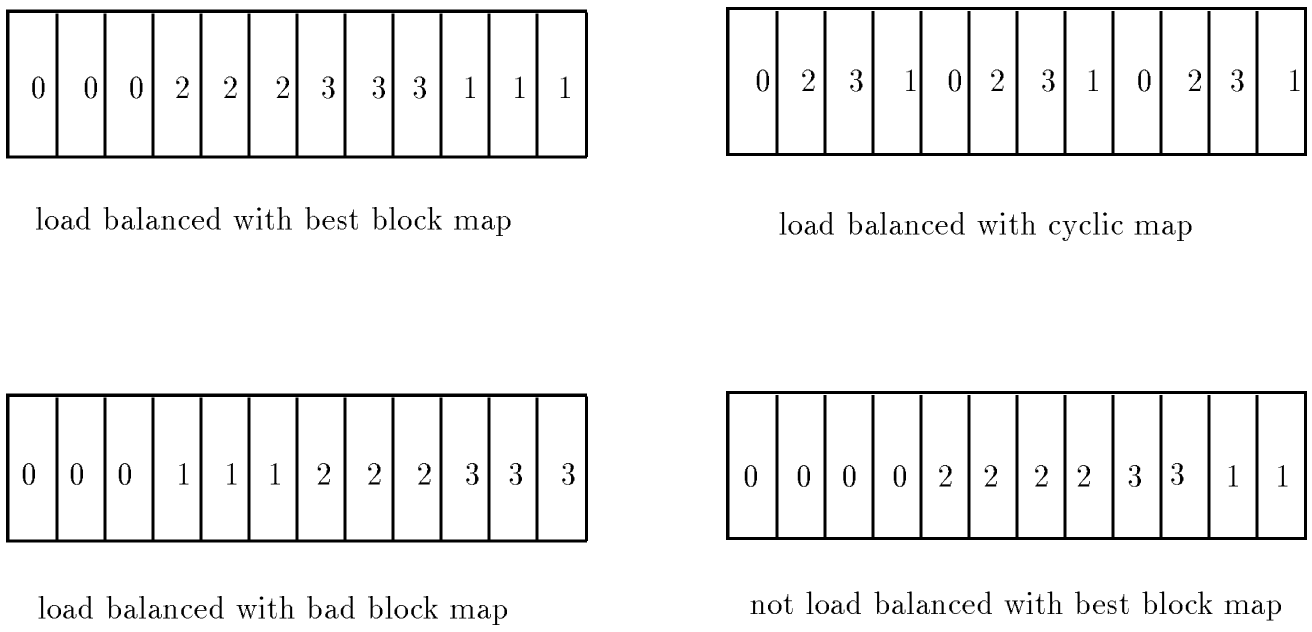

0 0 0 2 2 2 3 3 3 1 1 1
load balanced with best block map
0 2 3 1 0 2 3 1 0 2 3 1
load balanced with cyclic map
0 0 0 1 1 1 2 2 2 3 3 3
load balanced with bad block map
0 0 0 0 2 2 2 2 3 3 1 1
not load balanced with best block map

Figure 1: Single Mesh Sample Distributions.

The ordering of the predicted runtimes is correct according to previous experiments on the Intel (and other distributed memory machines as well).

5.1.2 Multiple Decomposition Mapping

Here we simulate the flow of fluid in two pipes joined as a "Tee". This is similar to the type of computation that actually occurs on a larger scale in the simulation of water-cooled nuclear reactors.

In this second test case, we ran the model on two coupled meshes with four distributions. The variations on distribution that are modeled include split block map, block map, bad block map, and very bad block map. These types of distributions are illustrated in Figure 2 for two twelve-element meshes mapped onto a four processor (hypercube) machine. In addition to numbering elements with the processor number, coupling between meshes is shown with a dashed line. A one hundred element (per mesh) problem was used with four processors in the model evaluation. The predicted runtimes for the four different distributions were:

Distribution	*Predicted Time*
split block map	34.77
block map	34.89
bad block map	35.30
very bad block map	37.46

Note that in this test problem there were 56 bytes of communication between nearest neighbors, but only 16 bytes across the coupling. When there are more communication bytes across the coupling than between neighbors, which does not often happen, the block map will outperform the split block map.

6 Theorems

Here we present a few theorems about distribution in this model of parallel computation and extensions of the model. These theorems provide direction for the design and comparison of distribution algorithms. Unless otherwise stated, elements are to be equally distributed across $p > 0$ processors and we deal only with up to 3-dimensional decompositions.

6.1 Standard Model Theorems

The basic setting for all of these theorems is physical simulation applications for static problems. This applies to all theorems unless otherwise stated.

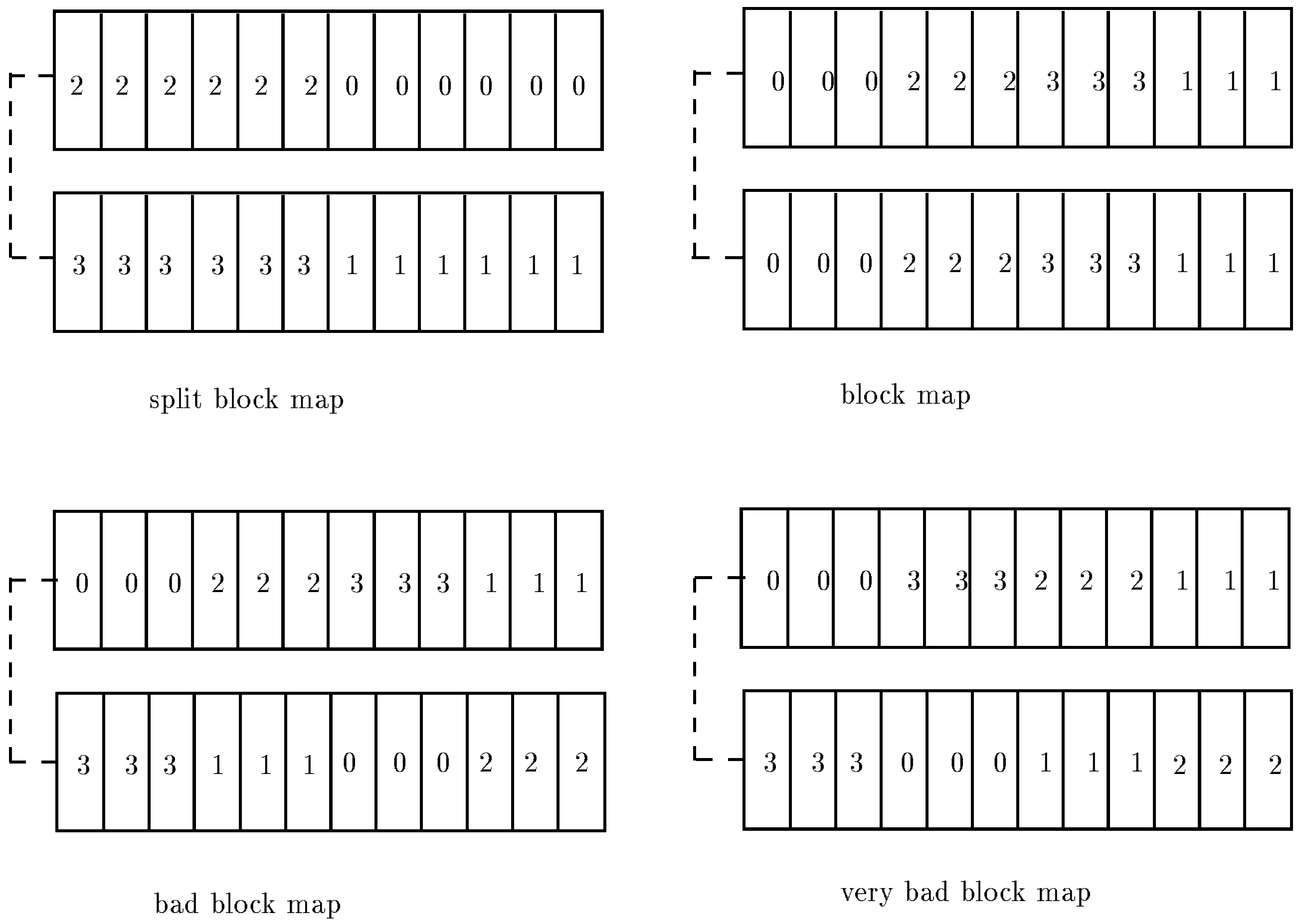


Figure 2: Two Mesh Sample Distributions.

Theorem 1 *Given two distributions with the same overheads due to load balance, context switching, indirection, and the inspector, if one distribution has a random placement of decomposition elements and the other has a neighborhood placement (e.g., based on a domain decomposition) the neighborhood-based distribution will provide better performance. More precisely, for large enough number of* DECOMPOSITION *elements per processor, the worst case communication performance of a topology-based distribution will be better than the expected communication performance for a random distribution.*

Intuitive Argument: In our target physical simulation applications, the communication necessary for each element is in its neighborhood. In the random distribution, no advantage is taken of this knowledge, every element is equally likely to be on any processor. If, instead, we preserve the neighborhood property as much as possible on each processor then there will be minimal communication across processors.

Proof: For 1-dimensional DECOMPOSITIONs, each (interior) element has 2 neighbors. Let there be m DECOMPOSITION elements per processor.

With a random distribution, for each element the probability of needing to communicate to reference a neighbor value is $2 * \frac{p-1}{p}$. Hence, for any processor, the expected number of communications is $2 * m * \frac{p-1}{p}$. For all processors, the expected number of communications is $2 * m * (p - 1)$.

With a topology-based distribution, the worst case is for a processor storing an interior section of the distribution. On such a processor, there are 2 elements that must access 1 neighbor each on a different processor. Hence, the communication for a 1-dimensional DECOMPOSITION distributed in this manner is $2 * (p - 1)$.

Therefore, with $m > 1$, the topology-based distribution has fewer communications than expected with random distributions.

For 2-dimensional DECOMPOSITIONs, each (interior) element has 8 neighbors. Let there be $m*n$ DECOMPOSITION elements per processor.

With a random distribution, for each element the probability of needing to communicate to reference a neighbor value is $8 * \frac{p-1}{p}$. Hence, for any processor, the expected number of communications is $8 * m * n * \frac{p-1}{p}$. For all processors, the expected number of communications is $8 * m * n * (p-1)$.

With a topology-based distribution, the worst case is for a processor storing an interior section of the distribution. On such a processor, there are $2 * (m+n) - 4$ elements that must access 3 neighbors each on other processors and 4 elements that access 5 elements on other processors for an upper bound of $6*(m+n)+8$. Hence, the number of communications for a 2-dimensional DECOMPOSITION distributed in this manner is bounded by $p * [6 * (m+n) + 8]$.

Therefore, with $m, n > 3$, the topology-based distribution has fewer communications than expected with a random distributions.

For 3-dimensional DECOMPOSITIONs, each (interior) element has 26 neighbors. Let there be $l * m * n$ DECOMPOSITION elements per processor.

With a random distribution, for each element the probability of needing to communicate to reference a neighbor value is $26 * \frac{p-1}{p}$. Hence, for any processor, the expected number of communications is $26 * l * m * n * \frac{p-1}{p}$. For all processors, the expected number of communications is $26 * l * m * n * (p-1)$.

With a topology-based distribution, the worst case is for a processor storing an interior section of the distribution. On such a processor, there are $2 * [(l-2)*(m-2)+(l-2)*(n-2)+(m-2)*(n-2)]$ elements that must access 9 neighbors each on other processors, $4 * [(l-2)+(m-2)+(n-2)]$ elements that must access 15 neighbors each on other processors, and 8 elements that access 19 elements on other processors for an upper bound of $18 * (lm + ln + mn) - 12 * (l+m+n) + 8$. Hence, the number of communications for a 3-dimensional DECOMPOSITION distributed in this manner is bounded by $p * 18 * (lm + ln + mn) - 12 * (l+m+n) + 8$.

Therefore, with $l, m, n > 3$, the topology-based distribution has fewer communications than expected with random distributions.

Throughout this proof it was only required that p be greater than one, but the larger p is the larger the gap is between the number of communications for a topology-based distribution and the expected number for a random distribution.

Theorem 2 *Given two distributions with the same overheads due to load balance, communication, indirection and the inspector, if one distribution has a single block of elements from one decomposition and the other has a number of blocks of elements from various decompositions, then the distribution with one block per processor will provide better performance. This is easy to see as no context switching is performed in the single block case whereas it is necessary in the multiblock case.*

Theorem 3 *Given two distributions with the same overheads due to context switching, communication, indirection and the inspector, the distribution with the better load balance will outperform the other distribution.*

Proof: This is obvious as the runtime is based, in part, on the maximum of the individual processor's computational time, which increases with decreasing load balance, by definition.

Theorem 4 *Given two distributions with the same overhead due to context switching, load balance, indirection, the inspector, and the same number of communications, with the same blocking capabilities, a distribution with communication between close together processors will outperform a distribution where communication takes place between distant processors (up to, but not past the point where general communication takes over). This implies that maintaining the neighborhoods when mapping blocks of decomposition elements to processors is important.*

Proof: Let h_1 be strictly less than $hops_{general} - 1$, then consider the communication cost, C_T, for b bytes. For h_1,

$$C_T(h_1, b) = constant + h_1 * cost_{neighbor} + b * h_1 * constant_{buffering},$$

while for $h_1 + 1$,

$$C_T(h_1+1, b) = constant + (h_1+1) * cost_{neighbor} + b * (h_1+1) * constant_{buffering},$$

or

$$C_T(h_1+1, b) = constant + h_1 * cost_{neighbor} + b * h_1 * constant_{buffering} + cost_{neighbor} + b * constant_{buffering}.$$

Hence,

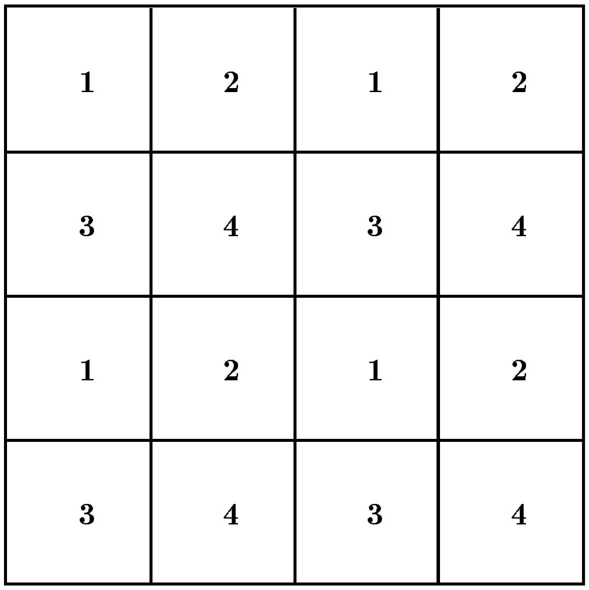
$$C_T(h_1, b) - C_T(h_1 + 1, b) = cost_{neighbor} + b * constant_{buffering}.$$

Since $cost_{neighbor} \geq 0$ and b, $constant_{buffering} > 0$, this shows that, in this model, more distant communication implies longer runtime.

6.2 Extended Model Theorems

Now we shall consider some problems that require extension of our original modeling assumptions.

Consider the case of dynamic physical simulation problems where the computational cost per element in a decomposition is not constant and may change over time. If we attempt a static distribution, then we must accept that we will not achieve perfect load balance over time. Since the computation per element can change at each iteration of the simulation, we may need to sacrifice communication overhead to achieve a reasonable load balance. To do this we could consider breaking blocks into sub-blocks and mapping the sub-blocks onto the processor mesh in a wrapped fashion, e.g., see Figure 3.



1	**2**	**1**	**2**
3	**4**	**3**	**4**
1	**2**	**1**	**2**
3	**4**	**3**	**4**

Figure 3: Sub-block distribution mapping

Using this approach we increase the communication overhead somewhat but we may also improve the load balance. This approach applies particularly well to those applications where the difference in computation between elements can be very large and the expensive type of computation occurs in large clusters of elements. In this case we want to make the sub-blocks some fraction of the normal cluster size (maybe 1/4).

When the expensive type of computation is spread randomly (not clustered) over the elements, then you may as well optimize the communication and just balance the number of elements per processor. This is because at any step each element is just as likely as any other to be doing an expensive computation. Therefore it does not matter how the elements are grouped in a static partition; each partitioning can be as bad as any other.

7 Summary

A model of parallel computation has been presented that can be used in comparing different distributions of meshes in ICRM problems. This model and the theorems proven in relation to it provide insight into the nature of good distributions for these problems. Two small problems were used to verify that the model reflects the problem characteristics that we wish to use in automatic distribution of ICRM problems. For further results on the use of this model see [2, 1].

References

[1] L.M. Liebrock. Using problem topology in parallelization. Technical Report 94-477-S, Rice University, Center for Research in Parallel Computation, Houston, TX, September 1994.

[2] L.M. Liebrock and K. Kennedy. Automatic data distribution of large meshes in coupled grid applications. Technical Report 94-395, Rice University, Center for Research in Parallel Computation, Houston, TX, April 1994.