Preprint from ICPP'91: International Conference on Parallel Processing, Chicago, Illinois, August 1991.

Efficient execution of homogeneous tasks with unequal run times on the Connection Machine [*]

Azer Bestavros
Harvard University
azer@harvard.edu

Thomas Cheatham
Harvard University
cheatham@harvard.edu

December 1990

Abstract

Many scientific applications require the execution of a large number of identical *tasks*, each on a different data set. Such applications can easily benefit from the power of SIMD architectures (*e.g. the Connection Machine*) by having the array of processing elements (PEs) execute the task in parallel on the different data sets.

It is often the case, however, that the task to be performed involves the repetitive application of the same sequence of steps, *a body*, for a number of times that depend on the input or computed data. If the usual *task-level synchronization* is used, the utilization of the array of PEs degrades substantially. In this paper, we propose a *body-level synchronization* scheme that would boost the utilization of the array of PEs while keeping the required overhead to a minimum. We mathematically analyze the proposed technique and show how to optimize its performance for a given application. Our technique is particularily efficient when the number of tasks to be executed is much larger than the number of physical PEs available.

[*]This work was supported partially by DARPA N00039-88-C-0163

1 Introduction

Lots of scientific applications [Marchuk:80] require the execution of a large number of identical *tasks*, each on a different set of data. For example, estimating the area under a curve using Monte-Carlo simulation [Rubinstein:81, Onion:90] involves deciding whether a point lies on or under a curve. This task is applied to a large number of randomly generated points. The proportion of points under the curve to the total number of points can be used to estimate the area under the curve, and thus the result of the integration. Such applications can easily benefit from the power of SIMD architectures (*e.g.* the Connection Machine) by having the array of Processing Elements (PEs) execute the task in parallel on the different data sets kept on the different PEs.

It is often the case, however, that the task to be performed involves the repetitive application of the same sequence of steps, *a body*, for a number of times that depends on the input or computed data. For example, in simulating the transmission of Neutrons through a plate of Beryllium [Onion:90], a while loop is entered, calculating for each iteration, the next horizontal position of a given Neutron, and testing whether it has escaped, or has been absorbed. If so, it is deactivated. Otherwise, another iteration is executed with possibly different initial conditions due to possible collision with Beryllium atoms. To accurately determine the reflection, absorption, and pass through probabilities, the experiment has to be conducted on millions of Neutrons. Using a SIMD architecture where each Neutron is assigned to a PE becomes a necessity.

The usual technique used to encode such applications uses a coarse grain *task-level synchronization*. The idea is to execute repetitively the body of the loop in a SIMD manner [Bestavros:88]. When a PE is done with its task, it simply deactivates itself so it will not participate in the following iterations. This process continues until no PEs are active. For example, in the Neutron/Beryllium experiment, each Neutron is associated to a PE. The PE keeps executing the body of the while loop until it determines that its Neutron has escaped or has been absorbed. The randomness associated with each Neutron's journey results in a large *variance* of the total number of iterations. Using task-level synchronization, the utilization of the array of PEs degrades substantially. In particular, the average execution time of a task becomes the *maximum* of the execution times of all the PEs (or tasks). The utilization of the array of PEs degrades even more when *virtual processing* is used. Virtual processing [CM-ref-2] allows programmers to scale-up their problems without worrying about the limited number of PEs in the available SIMD architecture.

In this paper, we propose a *body-level synchronization* scheme that would boost the utilization of the array of PEs while keeping the required overhead to a minimum. In particular, we aim at achieving an average task execution time that is the *mean* of the execution times of all the PEs. We mathematically analyze the proposed technique and show how to fine-tune its parameters to optimize its performance for a given application. Contrary to task-level synchronization, our technique becomes even more efficient when virtual processing is used. In this paper, we base our presentation and discussion on the Connection Machine architecture. Our methodology, however, can be easily applied to any other SIMD architecture.

2 Statement of the Problem

The problem that we wish to address is as follows. We assume that there is a set of tasks $T_1, \cdots, T_K$ to perform and that task T_j is accomplished by carrying out the following three steps:

1. Get some input and/or execute some initialization code.
2. Execute a body of computational steps for a number of times.
3. Execute some exit code and/or report results.

We refer to the body of computation in step 2 as the α-cycle. Also, we refer to the initialization and exit code in steps 1 and 3 as the ι-code and ω-code, respectively, and as the γ-code, collectively.

There are certain applications in which supplying actual inputs is not necessary, examples being Monte Carlo processes that compute their inputs as random numbers. For such applications we want to take advantage of the fact that we do not have to deal with explicit inputs. Similarly, there are many applications where we are not interested in reporting the individual results but only in some summarization thereoff. For such applications we want to accomodate doing the summarization as the results are computed so that we do not have to save them, only the summarization.

The basic issue addressed here is how to minimize the computational cost of carrying out the K tasks and disposing of the results, where it is assumed that K is large, relative to the number of available physical processors, P. If the number of steps, n_j, to complete task T_j is not a constant then it is probably a bad idea to simply assign the work to K virtual processors (even if there is sufficient memory to do so), because the cost would be on the order of:

$$\frac{K}{P} \max_{1 \le j \le K} n_j$$

where $\frac{K}{P}$ is the VP ratio (The ratio between Virtual processors and Physical processors [CM-ref-2]). What we hope to achieve is a cost closer to the average:

$$\frac{K}{P} \sum_{j=1}^{K} \frac{n_j}{K}$$

In other words, we are aiming at an effective VP ratio of:

$$\text{Effective VP ratio} \quad = \quad \frac{K}{P} \frac{\sum_{j=1}^{K} \frac{n_j}{K}}{\max_{1 \le j \le K} n_j} \tag{1}$$

3 Proposed Methodology

As we hinted before, the usual technique used to encode such applications on SIMD architectures[1] is to have each virtual processor take on the execution of one of the tasks. The whole procedure follows:

1. The input data (if any) is distributed to all PEs, and all PEs execute the ι-code.

2. The α-cycle is executed repetitively. When a PE is done with its task, it simply deactivates itself so it will not participate in the following α-cycles. This step is repeated until no PEs remain active.

3. All PEs execute the ω-code, and, if necessary, the results are gathered.

As noted before, the problem with the above procedure is that, as tasks are completed and PEs become inactive, the utilization of the PE array degrades dramatically. In particular, the average task execution time becomes the *maximum* of the execution times of all tasks. In this section, we propose a scheme that would boost the utilization of the PE array while keeping the required overhead to a minimum. In particular, we aim at achieving an average task execution time that is the *mean* of the execution times of all the tasks.

The solution that we propose works as follows. Instead of repetitively executing α-cycles until all tasks are finished, we execute α-cycles for a constant number of times m. At the end of these m cycles, tasks that finished their requested α-cycles are allowed to execute the ω-code and report their results. PEs associated with completed tasks become free and are allowed to start new tasks by getting input data and executing the ι-code. Once this is done, another round of m α-cycles is executed, and the same process repeats. We use the term β-cycle to mean the sequence consisting of assigning tasks to free PEs, executing m α-cycles, and freeing PEs assigned to terminated tasks (see figure 1). Thus a β-cycle consists of (see figure 1):

1. The input data (if any) is distributed to free PEs. All such PEs are labeled as busy and start by executing the ι-code.

2. The α-cycle is executed exactly m times. Only busy PEs participate. When a PE is done with its task, it simply deactivates itself so it will not participate in the remaining α-cycles.

3. Deactivated PEs execute the ω-code, and, if necessary, the results are gathered. Such PEs are declared free.

[1] for example the Connection Machine

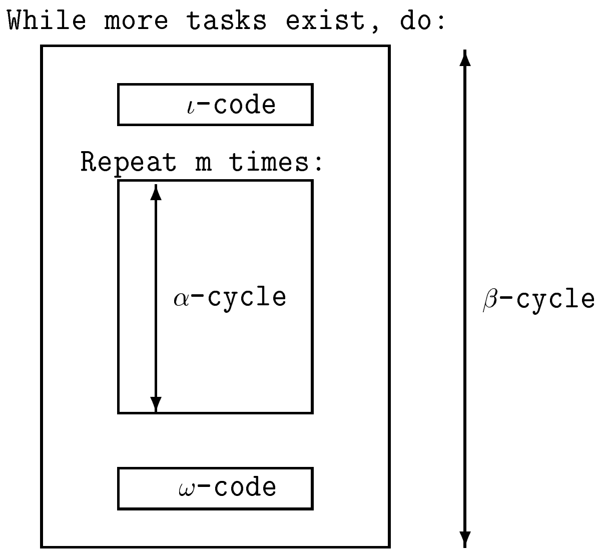

Figure 1: The β-cycle

4 Performance Analysis

In this section, we start by deriving the average task execution time using our proposed technique. Next, we derive an expression for the number of α-cycles to be executed in every β-cycle so as to achieve a minimal average task execution time. Finally, we derive an estimate for the total time required to terminate the execution of any K tasks on a SIMD architecture with P physical processors.

4.1 Average task execution time

Let t_α be the time it takes to execute an α-cycle and t_γ be the overhead time[2] associated with a β-cycle. Furthermore, let m be the number of α-cycles executed in every β-cycle. It follows that the time necessary to execute one β-cycle is given by

$$t_\beta = mt_\alpha + t_\gamma \tag{2}$$

Let n be the discrete random variable denoting the number of α-cycles necessary to execute a task Q. The probability density function of n is problem dependent. For the purpose of this paper, we assume that n follows a uniform distribution $[1, N]$,[3] where $N = rm + s$. The total number of β-cycles necessary to terminate the task Q is $\lceil \frac{n}{m} \rceil$. Using equation 2, it follows that the total time it takes to

[2]Namely, the time to assign tasks to free PEs and to gather results from terminated tasks

[3]The uniform distribution assumption is not necessarily realistic for many applications. We adopt it for the purpose of simplifying the analysis. Our method is applicable to any other distribution.

terminate Q is given by

$$t_Q = \lceil \frac{n}{m} \rceil (mt_\alpha + t_\gamma) \tag{3}$$

To compute the expected value of t_Q, the task execution time,we proceed as follows:[4]

$$\begin{aligned}
T_Q &= E(t_Q) \\
&= \frac{1}{N} \sum_{n=1}^{N} \lceil \frac{n}{m} \rceil (mt_\alpha + t_\gamma) \\
&= \frac{1}{N}(mt_\alpha + t_\gamma) \sum_{n=1}^{N} \lceil \frac{n}{m} \rceil \\
&= \frac{1}{N}(mt_\alpha + t_\gamma) \left[\sum_{n=1}^{m} \lceil \frac{n}{m} \rceil + \sum_{n=m+1}^{2m} \lceil \frac{n}{m} \rceil + \ldots + \sum_{n=rm+1}^{rm+s} \lceil \frac{n}{m} \rceil \right] \\
&= \frac{1}{N}(mt_\alpha + t_\gamma) \left[\sum_{n=1}^{m} 1 + \sum_{n=m+1}^{2m} 2 + \ldots + \sum_{n=rm+1}^{rm+s} (r+1) \right] \\
&= \frac{1}{N}(mt_\alpha + t_\gamma) \left[m(1 + 2 + \ldots + r) + s(r+1) \right] \\
&= \frac{1}{N}(mt_\alpha + t_\gamma) \left[\frac{rm}{2}(r+1) + s(r+1) \right] \\
&= \frac{1}{N}(mt_\alpha + t_\gamma) \left[(\frac{rm}{2} + s)(r+1) \right] \\
&= \frac{1}{rm+s}(mt_\alpha + t_\gamma) \left[\frac{r+1}{2}(rm + 2s) \right] \\
&= \frac{r+1}{2}(mt_\alpha + t_\gamma)(1 + \frac{s}{rm+s}) \\
T_Q &= \frac{r+1}{2}(mt_\alpha + t_\gamma)(1 + \frac{s}{N}) \qquad (4)
\end{aligned}$$

Equation 4 indicates that two factors contribute to T_Q, the average task execution time. The term $(r+1)t_\gamma$ reflects the effect of β-cycles overhead, whereas the term $(1 + \frac{s}{N})$ reflects the effect of synchronization overhead.

For a constant m, and as the value of N (and consequently r) increases, the effect of t_γ, the overhead associated with β-cycles, becomes an issue. This, of course, depends on the ratio between t_γ and t_α.[5] Generally speaking, the smaller the value of r the lesser the effect of β-cycles overhead. On the other hand, as the value of N decreases, the synchronization overhead becomes significant. Obviously, some kind of a balance is needed.

[4]We assume that the variables N, t_α and t_γ are all independent of the random variable n.

[5]For any practical scientific problem the ratio $\frac{t_\gamma}{t_\alpha}$ is expected to be very small.

4.2 Optimizing the Average Task Execution time

If we assume that N is much larger than m, and since s is necessarily smaller than m, we get the following approximations:

$$\begin{aligned} \frac{s}{N} &\approx 0 \\ r &\approx \frac{N}{m} \end{aligned}$$

Using the above approximation in equation 4, we get:

$$\begin{aligned} T_Q &= \frac{\frac{N}{m}+1}{2}(mt_\alpha + t_\gamma) \\ T_Q &= \frac{N+m}{2}(t_\alpha + \frac{1}{m}t_\gamma) \qquad (5) \end{aligned}$$

To find the value of m that would minimize the average task execution time in equation 5, we proceed as follows:

$$\begin{aligned} \frac{\delta}{\delta m}(T_Q) &= 0 \\ t_\alpha - \frac{N}{m^2}t_\gamma &= 0 \\ m &= \sqrt{N\frac{t_\gamma}{t_\alpha}} \qquad (6) \end{aligned}$$

Equation 6 is valid as long as $N \gg m$. This condition, however, can be easily satisfied. For instance, if $N = 100$ and $\frac{t_\gamma}{t_\alpha} = 0.25$ (very conservative), we get $m = 5$, which is indeed much smaller than N.

4.3 Average Cost

Let P be the total number of physical processors and K be the total number of tasks to be executed. From the analysis above, and on the average a total of P tasks can be executed every T_Q unit of time. It follows that the average time (cost) to terminate the execution of the K tasks is given by:

$$C = \frac{K}{P}\frac{N+m}{2}(t_\alpha + \frac{1}{m}t_\gamma) \qquad (7)$$

In the Connection Machine literature, the ratio between the declared number of *virtual processors* and the actual number of *physical processors* is called the *Virtual Processing Ratio* (VP ratio). The VP ratio measures the level of parallelism achievable for a given problem size. A VP ratio of 1 identifies the highest achievable parallelism. A VP ratio greater than 1 indicates that the physical array of PEs is simulating (by multiplexing in time and space) a larger virtual array of PEs. For a VP ratio v, the total simulation time is v-fold longer, and the available memory per PE is v-fold tighter.

In order to compare our proposed technique to other ones (*e.g.* the task-level syncronization approach), we define the *effective VP ratio* to be the ratio between the total time required to terminate all the K tasks and the maximum execution time of any one of the K tasks. Using our proposed technique, the effective VP ratio is reduced to:

$$\text{EffectiveVPratio} \quad = \quad \frac{K}{P} \frac{\frac{N+m}{2}(t_\alpha + \frac{1}{m} t_\gamma)}{(\max_{1 \leq j \leq K}(n_j))t_\alpha + t_\gamma} \tag{8}$$

For a large value of K, and a uniform distribution $[1, N]$ the value of $\max_{1 \leq K}(n_j)$ can be approximated by N.[6] Furthermore, we assume that t_γ is small compared to t_α,[7] and, as justified earlier, N is much larger than m. Applying the above approximations to equation 8, we get:

$$\begin{aligned} \text{Effective VP ratio} \quad &\approx \quad \frac{K}{P} \frac{\frac{N}{2} t_\alpha}{N t_\alpha} \\ &\approx \quad \frac{1}{2} \frac{K}{P} \end{aligned} \tag{9}$$

Equation 9 means that our body-level synchronization scheme achieves an effective VP ratio of one half that achieved using a straightforward task-level synchronization. In other words, it reduces the cost of the computation by 50 percent. This saving is, of course, dependent on the distribution of the task execution times, and is valid only when $K \gg P$, $N \gg m$, and $t\alpha \gg t\gamma$.

4.4 Utilization of the PEs

In this section, we derive a formula for the expected utilization of the available capacity of a SIMD architecture. In particular, we obtain equations for the percentage of useful cycles (used cycles) to the total available cycles. Keeping this percentage as close to 100% as possible is obviously a desired property of any efficient scheme for load balancing. As we shall see, achieving this goal does not necessarily mean achieving the minimum average task execution time.

4.4.1 α-Cycle Utilization

In order to compute the average utilization during α-cycles, we start by calculating the expected number of idle cycles within each group of m α-cycles. The probability of a task terminating at the i^{th} α-cycle, $0 < i \leq m$, is given by:

$$\text{Prob(termination at cycle } i) = \frac{1}{N}$$

The Processing Element associated with such a task will be idle for $m - i$ α-cycles (until the start of the next β-cycle.) Thus, the expected number of waisted α-cycles per PE per β-cycle can be estimated

[6] Actually, it can be easily shown that, for a uniform distribution $[1, N]$, we get: $\lim K \to \infty \max_{1 \leq K}(n_j) = N$.
[7] This is inevitably the case in real scientific problems

by:

$$\sum_{i=1}^{m} \frac{(m-i)}{N} = \frac{m(m-1)}{2N}$$

Thus, the average utilization per PE over the m α-cycles in a β-cycle is given by:

$$\begin{aligned} U_\alpha &= \frac{(m - \frac{m(m-1)}{2N})}{m} \\ &= 1 - \frac{(m-1)}{2N} \qquad (10) \end{aligned}$$

4.4.2 γ-cycle Utilization

For tasks with a number of iterations n following a uniform distribution $[1, N]$, the probability of terminating after executing m cycles is given by $\frac{m}{N}$. Thus, the average number of *finished* tasks amongst a total of P tasks after executing m cycles is given by $P \cdot \frac{m}{N}$. Processing Elements associated with these tasks will be busy reporting the results of the finished tasks and starting new ones during the following β-cycle. Meanwhile, the remaining Processing Elements, $P \cdot (1 - \frac{m}{N})$, will be idle. Thus, the average utilization during β-cycles is given by:

$$U_\gamma = \frac{m}{N} \qquad (11)$$

4.4.3 Overall β-cycle Utilization

For every β-cycle, we have m α-cycles each taking t_α units of time. In addition, t_γ units of time are spent for overhead computation. Using equations 11 and 10, we get:

$$U_\beta = \frac{mt_\alpha(1 - \frac{(m-1)}{2N}) + t_\gamma \frac{m}{N}}{mt_\alpha + t_\gamma} \qquad (12)$$

4.5 Optimizing the Average Utilization

In this section, we obtain an expression for the value of m that maximizes the average overall utilization of a SIMD architecture using the body-level synchronization technique. Equating to 0 the derivative of equation 12 with respect to m, we get:

$$\frac{\delta}{\delta m}(U_\beta) = 0$$

$$(mt_\alpha + t_\gamma)(t_\alpha + \frac{t_\alpha}{2N} - \frac{mt_\alpha}{N} + \frac{t_\gamma}{N}) - t_\alpha(mt_\alpha(1 - \frac{m}{2N} + \frac{1}{2N}) + t_\gamma) = 0$$

Solving the above equation for m, we get the following approximation:

$$m \approx \lceil \frac{t_\gamma}{t_\alpha}(\sqrt{3 + (2N+1)\frac{t_\alpha}{t_\gamma}} - 1) \rceil \qquad (13)$$

Notice that equation 13 gives an optimum value for m different from that given by equation 6. This means that reducing the average task execution time does not necessarily mean getting the maximum utilization of a SIMD architecture. To illustrate that, consider the case where $N = 10$, and $\frac{t_\gamma}{t_\alpha} = 2.5$. For the best task average execution time, we get $m = 5$. The utilization at this choice of m will be 70 percent. If, instead, we pick $m = 6$, we get a utilization of 71 percent.

5 Connection Machine Implementation

In this section, we look at the potentials of our *body-level synchronization* when applied to Connection Machine applications. (We are currently working on different Connection Machine implementations. Results should be available in the final version of the paper.) Here, we show how much gain we should expect, just by looking at the CM timing information [CM-ref-1]. All our calculations are based on a CM-2 with 4K PEs with computation and communication done on 32 bits.

In order to judge the performance of our technique we have to estimate the overhead time, t_γ. The bulk of t_γ will be spent distributing data to the free PEs and collecting results from finished tasks. A straightforward technique for performing these tasks is using a parallel *rendez-vous* algorithm [Hillis:86]. The idea is to keep the data for the tasks to be started in a virtual processor set (the input-VP-set), and, using a Parallel Prefix ranking procedure, each free processor is assigned a unique index. That index should be used to get the data for a new task from the input-VP-set. The time it takes to compute the rendez-vous index is 894 time units (see [CM-ref-1], pages 24-35.) The time it takes to send the data from the input-VP-set to the set of free PEs is 1111. Thus, distributing the data to the free PEs should take approximately 2000 time units. Gathering the results from the finished tasks can be done using the same *rendez-vous* technique, thus bringing the total overhead time to 4000. Now, let's assume an applications where the body of the computation takes 1000 units of time (An addition or a multiplication on the CM takes approximately 100 units.) Moreover, assume that the number of iterations required per task is uniformly distributed and ranges from 1 to 10000, and that the total number of tasks to be executed is 64,000. Therefore we get the following parameters:

$$\begin{aligned} N &= 10000 \\ P &= 64000 \\ t_\alpha &= 1000 \\ t_\gamma &= 4000 \end{aligned}$$

Substituting with the above numbers in Equation 6, we get:

$$m = 200$$

Now substituting in Equation 5, we get an average execution time per task of $5,202,000$.[8] If *task-level* synchronization were used, this number would bounce to almost $10,000,000$,[9] almost double the time we achieve as we already predicted in Equation 9.

The merits of our methodology become even more compelling if we look at the *total* execution time to finish all 64,000 tasks, using the available 4K PEs. Using *task-level* synchronization, it would take approximately $160,000,000$ units of time, compared to only $83,232,000$ using our *body-level* synchronization.

Obviously, our technique achieves a much better utilization of the CM. Substituting in Equation 12, we get $U_\beta = 97\%$, compared to utilization of almost 50% using the *task-level* synchronization.

All of the above gains become even more accentuated when, for an application like the Neutron/Beryllium experiment mentioned earlier, the number of tasks p is orders of magnitude larger and the number of iterations per task is unbounded.

6 Conclusion and Future work

In this paper we have presented a new methodology for the efficient execution of homogeneous tasks with unequal run times on SIMD architectures (with a special emphasis on the Connection Machine.) We showed that, for some applications, our technique cuts the expected execution time by (almost) one half when compared to the usual virtual processing techniques. Moreover, it ensures a higher utilization of the PE array. Currently, we are working on applying our technique to a number of scientific problems. Our technique extends quite well to support arbitrarily *nested* computations. As a matter of fact, it can be easily shown that for a task with k levels of nesting, the achievable speedup is in the order of 2^k.

The analysis in this paper assumes a uniform distribution of run times for the set of homogeneous tasks. This assumption leads to conservative (pessimistic) performance expectation. For more realistic assumptions (*e.g.* normal or exponential distributions), our methodology results in even better performance gains.

Body level synchronization can be easily and efficiently implemented by compilers. In particular, handling virtual processor sets in SIMD architectures (*e.g.* the CM) can be made much more efficient if our approach rather than the usual task level synchronization approach is adopted.

In this paper we assumed that it is possible to fine tune the parameters of our technique so as to optimize its performance. This, however, requires some domain knowledge (for example the probability distribution for the number of iterations required to complete a task). Such information might be

[8]Notice that a lower bound on this average is $5,000,000$, if we didn't have to suffer any overhead.

[9]assuming that among 4K tasks, it's highly probable that one of them will need approximately 10000 iterations. As a matter of fact, it is very easy to prove that statement (proof ommitted for space limitation.)

difficult to obtain, or even not available. In this case, experimentation[10] (sample executions on much smaller scale) might be necessary to estimate these parameters.

More work remains in order to reduce the overhead associated with distributing/gathering data to/from PEs. In particular, techniques to avoid the expensive "send" communication – using "news" communication instead – should greatly reduce the overhead time, and further improve the performance of our methodology. We are experimenting with a number of such alternatives (using pipelining and buffering) [Cheatham:90, Bestavros:90].

References

[Bestavros:90] Azer Bestavros, Thomas Cheatham, and Dan Stefanescu, "Parallel approaches for bin packing on the Connection Machine." *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, Dallas, Texas, December 1990.

[Bestavros:88] Azer Bestavros and William McKeeman, "Parallel bin packing using first-fit and k-delayed best-fit heuristics." *Technical Report TR-16-88, Department of Computer Science, Harvard University.* August 1988. A shorter version appeared in the *Proceedings of the ISMM International Conference on Parallel and Distributed Computing, and Systems,* New York, New York, October 1990.

[Cheatham:90] Thomas Cheatham, "The Workers Model of Computation for the Connection Machine," *Internal Report, Department of Computer Science, Harvard University,* In progress.

[Hillis:86] W. Hillis, G. Steele, "Data Parallel Algorithms," *Communications of the ACM,* December 1986.

[Onion:90] Frederick Onion, "PMCML: A Parallel Monte Carlo Modeling Language", *B.A. Thesis, Department of Computer Science,* Harvard University, May 1990.

[Marchuk:80] G. Marchuk, G, Mikhailov, M. Nazaraliev, M. Darbinjan, R. Kargin, and B. Elepov, *The Monte Carlo methods in atmospheric optics*, Springer-Verlag, Berlin, 1980.

[Rubinstein:81] R. Rubinstein, *Simulation and the Monte Carlo Method*, John Wiley & Sons, Inc., 1981.

[Sabot:86] G. Sabot, "Bulk processing of text on a massively parallel computer", *Proceedings of the 24th annual meeting of the association for computational linguistics,* June 1986.

[Waltz:87] D. Waltz, "Applications of the Connection Machine", *IEEE Computer 20(1),* January 1987.

[CM-ref-1] "The Connection Machine Parallel Instruction Set – Ver 5.2", *Thinking Machines Corporation*, October 1989.

[CM-ref-2] "Introduction to Programming in C/Paris – Ver 5.0", *Thinking Machines Corporation*, June 1989.

[10]whether compiler driven or user driven