Report Number: WUCS-94-3

1994-01-01

Speculative Computation: Overcoming Communication Delays in Parallel Algorithms

Vasudha Govindan and Mark A. Franklin

Communication latencies and delays are a major source of performance degradation in parallel computing systems. It is importnat to "mask" these communication delays by overlapping them with useful computation in order to obtain good parallel performance. This paper proposes speculative computation as a technique to mask communication latencies. Speculative computation is discussed in the context of synchronous iterative algorithms. Processors speculate the contents of messages that are not hyet received and perform computation based on the speculated values. When the messages are received, they are compared with the speculated values and, if the error is unacceptable, the resulitng computation... **Read complete abstract on page 2.**

Speculative Computation: Overcoming Communication Delays in Parallel Algorithms

Vasudha Govindan and Mark A. Franklin

Complete Abstract:

Communication latencies and delays are a major source of performance degradation in parallel computing systems. It is importnat to "mask" these communication delays by overlapping them with useful computation in order to obtain good parallel performance. This paper proposes speculative computation as a technique to mask communication latencies. Speculative computation is discussed in the context of synchronous iterative algorithms. Processors speculate the contents of messages that are not hyet received and perform computation based on the speculated values. When the messages are received, they are compared with the speculated values and, if the error is unacceptable, the resulitng computation is corrected or recomputed. If the error is small, the speculated value is accepted and the processor has masked the communication delay. The technique is discussed in detail and is incorporated on to a N-body simulation example; the techniques result in a performance improvement of up to 34%. An empirical performance model is developed to estimate the performance of speculative computing. Model and measured values are compared and shown to be in good agreement.

Speculative Computation: Overcoming Communication Delays in Parallel Algorithms

Vasudha Govindan and Mark A. Franklin

WUCS-94-03

January 1994

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

This research has been sponsored in part by funding from the NSF grant CCR-9021041 and ARPA contract DABT-93-C0057.

Speculative Computation: Overcoming Communication Delays in Parallel Algorithms*

Vasudha Govindan
vasu@wuccrc.wustl.edu

Mark A. Franklin
jbf@random.wustl.edu

Computer & Communications Research Center
Washington University
Campus Box 1115, One Brookings Drive
St. Louis, Missouri 63130

Abstract

Communication latencies and delays are a major source of performance degradation in parallel computing systems. It is important to "mask" these communication delays by overlapping them with useful computation in order to obtain good parallel performance. This paper proposes speculative computation as a technique to mask communication latencies. Speculative computation is discussed in the context of synchronous iterative algorithms. Processors speculate the contents of messages that are not yet received and perform computation based on the speculated values. When the messages are received, they are compared with the speculated values and, if the error is unacceptable, the resulting computation is corrected or recomputed. If the error is small, the speculated value is accepted and the processor has masked the communication delay. The technique is discussed in detail and is incorporated on to a N-body simulation example; the techniques result in a performance improvement of up to 34%. An empirical performance model is developed to estimate the performance of speculative computing. Model and measured values are compared and shown to be in good agreement.

Keywords: Speculative computation, communication latency masking, performance model, synchronous iterative algorithms, N-body simulation.

1 Introduction

Several compute intensive problems in science and engineering have found in parallel processing an attractive solution to their high computing needs[6]. The problem is appropriately partitioned and mapped on to the available processors and the processors cooperate to solve the problem. One of the major sources of performance degradation in parallel computing is the communication delay. The data and control dependency between different sections of the application often result in communication between the processors assigned to the various sections. If the associated delays are

*This research has been sponsored in part by funding from the NSF grant CCR-9021041 and ARPA contract DABT-93-C0057.

large, the processors are poorly utilized, resulting in poor parallel performance. This paper proposes speculative computation and associated communications "masking" as an approach to overcoming these delays. The techniques are implemented on a parallel N-body simulation example to illustrate the potential performance benefits of this approach.

A recent trend in computing is to use networks of high performance workstations as a concurrent computing resource. The workstations are typically connected by standard interconnects like ethernet, FDDI or ATM networks and operate under programming environments such as PVM(Parallel Virtual Machine)[9]. The communication delays in such systems are significant and often subject to large variations due to non-deterministic network traffic. Overcoming communication delays in parallel algorithms has become increasingly important in such distributed computing systems.

A number of algorithmic improvements have been proposed to overcome communication delays in parallel algorithms. Saltz et.al[7, 8] suggest several techniques to develop robust algorithms for solving partial differential equations on hypercube machines. They propose partitioning strategies to reduce communication traffic, efficient convergence check mechanisms and concurrent execution of multiple time-steps. While their techniques are shown to be effective, they are limited to hypercube machines and to a small class of algorithms. Techniques such as asynchronous algorithms[1, 11] and rollback synchronization[1] mask communication delays but are applicable only to a very small set of applications. Overcoming communication delays in message passing systems is analogous to the problem of overcoming memory access delays in shared memory systems. There has been considerable work on developing and analyzing techniques for overlapping remote memory access delays in medium to large scale shared memory systems[5]. Coherent caches, data prefetching and multithreading have been proposed and implemented in some systems.

Speculative computing overcomes communication delays by effectively overlapping the communication time with useful computation. While waiting for a message, the processor speculates the contents of the message and uses the speculated values in its computation. Consider an example where two processors $P1$ and $P2$ cooperate to solve a problem. They exchange messages comprised of certain partial results of their computation. Suppose $P1$ needs some data from $P2$ and cannot proceed with its computation until the data is received. $P1$ expects a message from $P2$ and blocks (or waits) till the message is received. If the communication delays are large, this dependency between $P1$ and $P2$ results in significant performance degradation. If instead, $P1$ speculates the value of the data it needs and proceeds with its computation using the speculated data, then communication and computation are overlapped. When the message from $P2$ arrives, the speculated and actual values are compared. If the error in speculation is large, the resulting computation is corrected or recomputed. If the error is "small", the resulting computation is accepted, and $P1$ has effectively "masked" the communication delay.

Speculative computation has been proposed[2] and implemented[10] to incorporate parallelism into inherently serial algorithms. Work is performed before it is known whether it is needed or not. If the work is needed, it has already been performed, thus speeding up the computation time. Some work may never be needed and thus is wasted. To gain from speculative computation it is critical to identify future work which is likely to be needed. Burton[2] proposes a simple functional language to incorporate speculative computation in inherently serial algorithms. Witte et.al[10] apply speculative computing to the simulated annealing algorithm. The resultant parallel implementation, based on a tree of speculative computations, yields a speedup of about $\log p$ on p processors.

The concept of speculative computation to overcome communication delays can be applied to a host of parallel algorithms. However, in this paper, we focus on applying the technique to a

class of synchronous iterative algorithms[1, 3]. Section 2 describes the synchronous iterative algorithm model employed. Section 3 discusses the speculative computing technique and associated performance benefits. In section 4, we develop an empirical performance model to estimate the performance of speculative computation. Section 5 describes the parallel N-body simulation example which is used to illustrate the speculative computing technique. Our example, implemented on a network of SUN/Sparc workstations, showed a significant performance improvement (up to 34% on 16 processors) with speculative computation. The model and measured performance are then compared.

2 Synchronous Iterative Algorithms

Synchronous iterative algorithms[1, 3] include most of the compute intensive numerical methods used in science and engineering applications. Iterative techniques to solve linear and non-linear equations, solution of partial differential equations, numerical integration, particle simulation, etc., are some examples. These algorithms require frequent communication between processors and their performance is highly sensitive to communication delays. Techniques that reduce or mask communication delay effects potentially yield significant performance improvement for this class of applications.

Synchronous iterative algorithms proceed in synchronous steps or iterations. The work associated with each iteration is distributed over some or all of the available processors. The processors are synchronized either by means of a barrier operation or by message exchanges at the end of each iteration. The next iteration begins only after all the processors complete the current iteration.

In this study, a general and widely applicable model for synchronous iterative algorithms[1] is used. The model is used for ease of illustration and does not limit the applicability of the principal ideas which apply broadly to a host of other algorithms.

An application problem is defined as consisting of a set of n variables, $\mathbf{X} = \{x_1, x_2, x_3....x_n\}$ that are evaluated each iteration according to some vector function $\mathbf{F} = \{f_1(\mathbf{X}), f_2(\mathbf{X}) \ldots f_n(\mathbf{X})\}$ of their values in previous iterations. Each variable may represent a single value or a complex data structure. This can be represented as follows:

$$\mathbf{X}(t+1) = \mathbf{F}(\mathbf{X}(t), \mathbf{X}(t-1), \mathbf{X}(t-2)\ldots) \tag{1}$$

Each of the variables can be evaluated independently if the values of all the variables for previous iterations are known.

$$x_i(t+1) = f_i(\mathbf{X}(t), \mathbf{X}(t-1), \ldots) \tag{2}$$

If there are p processors available, the set $\mathbf{X}$ is partitioned into p subsets and each subset is allocated to a processor. The p subsets may not have equal cardinality since, in general, the functions f_i may be of unequal complexity and the processors may be of unequal power. Load balancing considerations will thus lead to unequal partitions. Let $\mathbf{X}_j$ be the set of variables allocated to processor j. The sets $\mathbf{X}_j$ are disjoint subsets of $\mathbf{X}$ ($\bigcup_j \mathbf{X}_j = \mathbf{X}$ and $\mathbf{X}_j \cap \mathbf{X}_k = \emptyset, j \neq k$). Processor j is now responsible for evaluating all elements of $\mathbf{X}_j$ using equation 2. Note that each variable x_i can potentially be a function of all other variables. This implies that all computation associated with iteration t should be completed before iteration $t+1$ can begin. Also, in a parallel implementation, each processor sends the newly evaluated values of all its variables to all the other processors and has to wait till it receives such messages containing the variable values from all the other processors before it can proceed to the next iteration. The parallel algorithm for a general synchronous iterative algorithm

begin
Read $x_i(0)\ \forall i$
Partition and Distribute X
$\mathbf{X}_j$ = set of variables allocated to processor j
for $t = 0$ to MAX
    Each processor j, **Do:**
    begin
    send $\mathbf{X}_j(t)$ to all processors
    receive $\mathbf{X}_k(t) \quad \forall k \neq j$
    compute: $x_i(t+1) = f_i(\mathbf{X}(t), \mathbf{X}(t-1)\ldots),\ \ \forall x_i \in X_j$
    end
end

Figure 1: Synchronous Iterative Algorithms

is given in pseudocode form in Figure 1. In this formulation, it is assumed that each variable is a function of every other variable.

The rate at which the algorithm executes therefore depends on the computation speeds of all the processors, the partitioning, and the communication time to exchange messages between processors. Figure 2a shows the execution of a synchronous iterative algorithm on two processors where ideal load balancing has been achieved and the complexity of F is not a function of t. The figure shows the effect of a slow communication channel. The execution rate is slowed down by communication delays which are large and often unpredictable in distributed workstation based networks.

3 Speculative Computation

3.1 The Concept

In this development, we apply speculative computation in the context of synchronous iterative algorithms. In synchronous iterative algorithms, each processor has to wait for messages from every other processor (in the general case) before it can proceed with its computation and therefore, a slow communication channel can introduce large communication delays degrading the performance. With speculative computation, in each iteration, each processor j sends a copy of its variables to all other processors. Next, it checks its message queue and incorporates any messages that have arrived. It then speculates the values of all variables that are not yet received based on some speculation function. For example, say that in iteration t, $\mathbf{X}_k(t)$ has not been received. The speculation function for $\mathbf{X}_k(t)$ might be a weighted sum of its past values, $\mathbf{X}_k(t-1), \mathbf{X}_k(t-2), \ldots$, which have been stored on processor j from previous messages(*i.e.*, $x_i^*(t) = w_1 x_i(t-1) + w_2 x_i(t-2)\ldots, \forall x_i \in \mathbf{X}_k$, where $w_1, w_2, \ldots$ are the weighting factors). Using the received values and the speculated values, it computes the next values of its variables (the set $\mathbf{X}_j$). It then waits for messages that have not yet been received. When a message is received, it compares the received values of the variables with the speculated values. If the error is beyond a certain predefined threshold, it calls a correction function to correct its computation, or in some cases, recomputes its variables. The synchronous iterative algorithm (Figure 1) modified to incorporate speculative computation is presented in Figure 3.

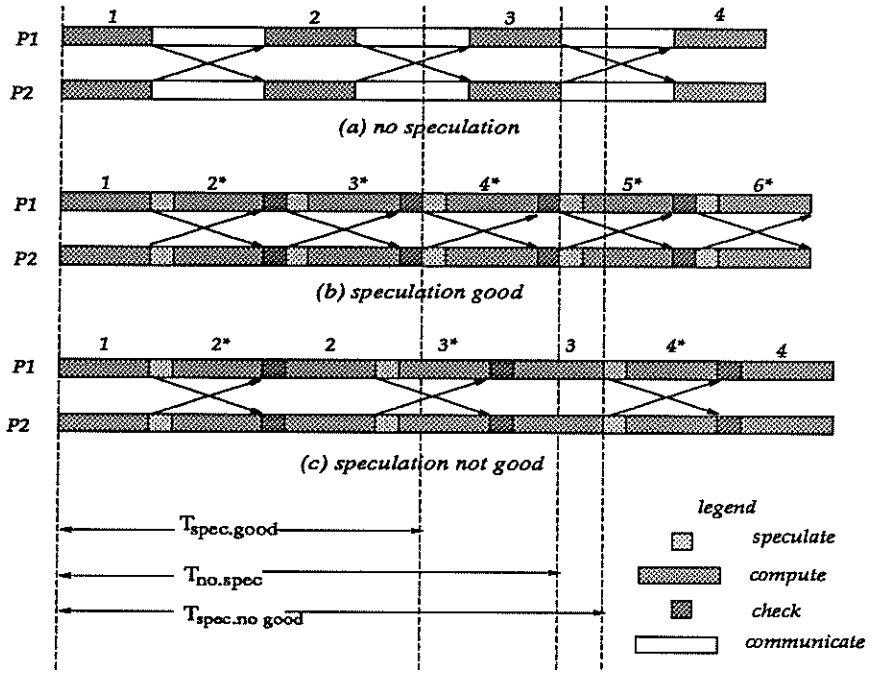

Figure 2: Speculative Computation: An example
Computation based on speculated values are marked with a $*$

Figure 2 shows speculative computation applied a two processor synchronous iterative algorithm example where a slow communication channel results in poor performance. The time taken to execute the first three iterations are shown for each case. For case (a), the processors wait for messages each iteration resulting in poor utilization. For cases (b) and (c), the processors use speculative computing to mask the communication delay. In (b), every speculated value is good and acceptable. The execution time for this case ($T_{spec.good}$) is, therefore, much smaller than the time for the no speculation case ($T_{no.spec}$). In (c), the speculated values are each found unacceptable and the variables have to be recomputed. This incurs a small penalty and as a result, the execution time for($T_{spec.nogood}$) is greater than the time for the no speculation case. When speculated values are acceptable, speculative computation overlaps communication delays but incurs the overhead of speculation, error checking and correction. In section 4 we develop a performance model that takes into account the computation/communication overlap, the speculation and checking overhead and the penalty for erroneous speculations.

3.2 Speculation Windows

If the communication times are greater than the computation time per iteration, speculative computation for a single time value of $\mathbf{X}_j$ (*i.e.*, $\mathbf{X}_j(t+1)$) can only partially mask the communication delay. The remaining idle time may be used to speculate and compute subsequent iterations (*i.e.*, $\mathbf{X}_j(t+2), \mathbf{X}_j(t+3)\ldots$). We define a "forward window" (FW) as the maximum number of iterations into the future that may be speculated. In the example in Figure 2, the processors speculate one iteration into the future and the forward window, $FW = 1$. Speculating for and computing more

```
begin
Read x_i(0), ∀i
Partition and distribute X
(X_j = set of variables allocated to processor j )
for t = 0 to MAX
  Each processor j, Do:
  begin
  send X_j(t) to all processors
  for each k ≠ j
    if (msg from k arrived)
      receive X_k(t)
    else
      speculate X*_k(t)
  compute: x_i(t+1) = f_i(X(t), X(t-1)...),  ∀x_i ∈ X_j
  for each k ≠ j
    if (msg from k not yet received)
      receive X_k(t)
      error = compare(X_k(t), X*_k(t))
      if (error > threshold)
        correct(X_j(t+1))
  end
end
```

Figure 3: Synchronous Iterative Algorithms: With Speculative Computing
(Speculated values are shown with a * superscript)

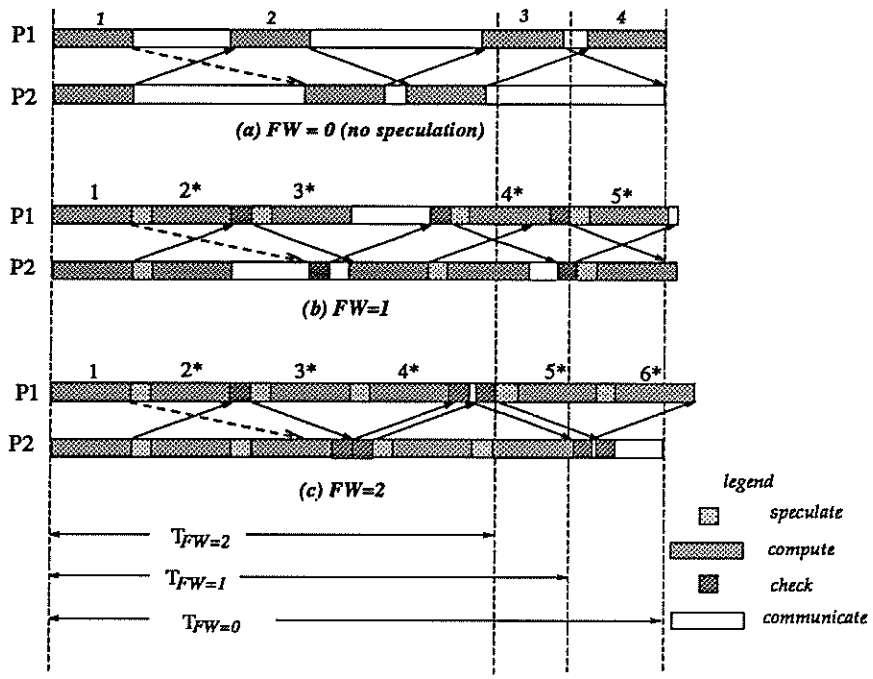

Figure 4: Speculative Computation with $FW = 2$

than one iteration is particularly beneficial in situations where the computation and communication times are likely to vary with each iteration. For example, in a workstation network environment, messages may occasionally experience excessive delays due to network traffic or the background load on timeshared processors may slow down the computation phase. These delays, though transient, bring about significant performance degradation. In situations like this, a larger FW helps in overcoming the transient delays. However, speculating values too far into the future introduces inaccuracies. Speculated values farther into the future are more likely to exceed the error bounds and may have to be discarded. The FW is chosen based on an estimate of the communication and computation times and the accuracy of the speculation function.

Figure 4 illustrates the effect of FW when there is an excessive but transient delay along one communication path. The example is similar to the example Figure 2, but the first message from $P1$ to $P2$ is delayed in transit (shown by dashed line). Figure 4a shows the execution with no speculation ($FW = 0$). The communication delay is greater than the time to speculate for and compute one iteration. Due to the extra long $P1 - P2$ delay path, the $FW = 1$ case (Figure 4b) can only partially mask the communication delay. In the $FW = 2$ case (Figure 4c), the idle time is used to speculate and compute two iterations into the future. If the speculated values are found accurate, this yields additional performance improvement.

Defining an appropriate speculation function for an application is important in realizing the performance benefits of speculative computation. Speculation is most useful in applications where the variables generally follow a relatively slow changing trend that can be detected. In some applications, each variable is associated with some information revealing its trend. For example, in particle simulations, each variable (a particle) has an associated mass, position and velocity. The velocity indicates how the position of the particle is likely to change in the near future. The trend can also

Table 1: Model Definitions

N	Total number of variables
N_i	No. variables allocated to processor i
M_i	No. operations per second on processor i (a measure of processor capacity)
f_{comp}	No. operations to compute a variable
f_{spec}	No. operations to speculate a variable
f_{check}	No. operations to check a variable
$t_{comm}(p)$	Communication time on a p processor system (assumed constant over all processors)
$t_{total}(p)$	Total time per iteration on p processors (no speculation)
$\hat{t}_{total}(p)$	Total time per iteration on p processors (with speculation, $FW = 1$)
k	% recomputations due to speculation error

be detected by examining the history of the variable.

In this context, we define a "backward window"(BW), as the maximum number of past values of the variables used in the speculation function. The speculated value of a variable is an extrapolation of its present value and previous BW values. A larger BW generally implies a more accurate speculated value. It also implies a more complex speculation function and requires more memory to store the past values. The selection of an optimal value for the BW reflects the tradeoff between accuracy and complexity of speculation. FW and BW are tuned for a given algorithm and computing platform to maximize performance.

4 Performance Model

In this section, an empirical performance model is developed which quantifies the benefits of speculative computation. The model estimates the execution time of a synchronous iterative algorithm with and without speculation and examines the effect of speculation accuracy on the performance. The model is used to predict the execution times for the N-body simulation example of section 5 and the predictions are compared with measured values.

Consider a synchronous iterative algorithm that involves the evaluation of N variables every iteration. The computing platform consists of p processors that may be of unequal computing abilities. The N variables are distributed over the p processors such that each processor is allocated workload (number of variables) proportional to its computing abilities. The computation on each processor is therefore balanced and takes equal amount of time. It is assumed that the communication times are equal over all processors and that computation and communication times remain constant over all iterations.

Table 1 defines the model parameters. N_i is the number of application variables allocated to processor i. f_{comp}, f_{spec} and f_{check}, respectively, are the number of operations required to compute, speculate and check for each variable. $t_{comm}(p)$ is the communication time between processors in a p processor implementation. $\hat{t}_{total}(p)$ and $t_{total}(p)$ are the total time per iteration with and without

speculation, respectively, for a p processor execution.

Let $\mathbf{P} = \{P1,\ P2,\ \ldots\}$ denote the set of available processors, ordered with respect to their computing abilities. If the computing abilities of processor Pi, expressed as the number of operations performed per unit time, is given by M_i, the set $\mathbf{P}$ is ordered such that $M_1 \geq M_2 \geq M_3 \geq \ldots$. A $p-$processor execution of the application implies that the application is run on the fastest p processors available, (*i.e.,* on the first p processors $\{P1,\ P2,\ \ldots\ Pp\}$ of the set $\mathbf{P}$).

When the algorithm is run on a single processor ($P1$), the total time per iteration is equal to the time to evaluate the N variables and is given by,

$$t_{total}(p) = N \times (f_{comp}/M_1) \tag{3}$$

For a p processor implementation, the N variables are distributed such that the number of variables allocated to each processor, N_i, is proportional to its computing ability, M_i. With ideal load balancing, (*i.e.,* equal computation time per iteration on each processor), the following two conditions are satisfied:

$$\frac{N_i}{M_i} = \frac{N_j}{M_j}, \quad \forall i, j \tag{4}$$

$$\sum_{i=1}^{p} N_i = N \tag{5}$$

The total time per iteration (no speculation case) is the same on all p processors and is given by the sum of computation and communication times:

$$t_{total}(p) = N_i \times (f_{comp}/M_i) + t_{comm}(p), \quad p > 1 \tag{6}$$

With speculative computation, the computation and communication times are overlapped and the time per iteration has the general form:

$$\begin{aligned} \textit{total time} = \quad & \max[\ \textit{(speculation time + computation time), communication time}\ ] \\ & + \textit{error checking time} \ + \ \%\textit{error} \times \textit{recomputation time.} \end{aligned} \tag{7}$$

The maximum term reflects the potential overlap between computation and communication. The final two terms deal with the time to check whether the speculated variable values were correctly predicted, and for the instances where error limits are exceeded, to perform a recomputation.

Equation 7 can be rewritten by first assuming that processor i speculates and check all variables not allocated to it (*i.e.,* $N - N_i$ variables). The processors are allocated variables such that the computation phase (the most time consuming phase) is equal on all processors. Speculation and checking phases are, however, not equal and this leads to a small load imbalance. The iteration time on processor i is given by:

$$\begin{aligned} \hat{t}^{(i)}_{total}(p) = \quad & \max\ [\ (N - N_i) \times (f_{spec}/M_i) \ + \ N_i \times (f_{comp}/M_i),\ t_{comm}(p)] \\ & + \ (N - N_i) \times (f_{check}/M_i) \ + \ k \times N_i \times (f_{comp}/M_i), \end{aligned} \tag{8}$$

where, k is the percentage of variables that need to be recomputed due to speculation errors. The total time per iteration on p processors with speculative computation, $\hat{t}_{total}(p)$, is the maximum iteration time over all processors:

$$\hat{t}_{total}(p) = \max_{i=1}^{p} \quad \hat{t}^{(i)}_{total}(p) \tag{9}$$

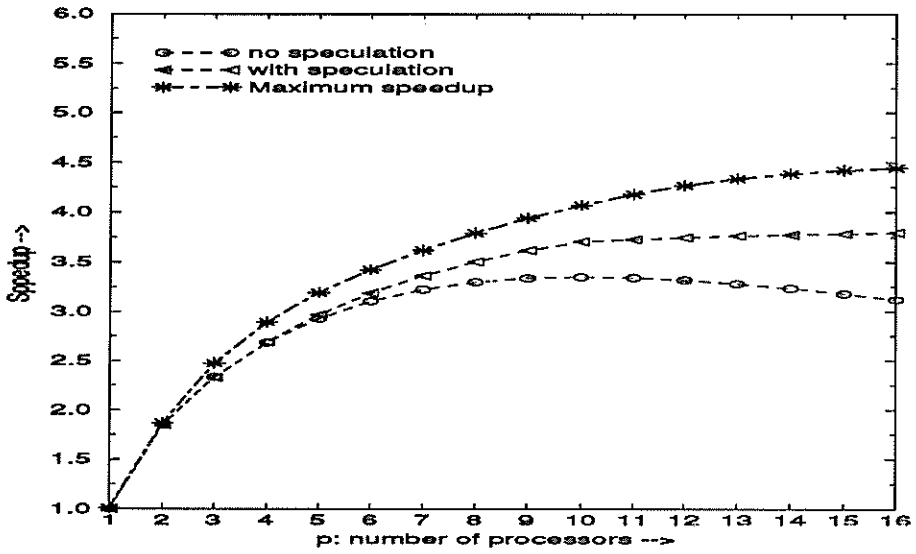

Figure 5: Speculative Computation: Performance Model

Speedups are relative to the fastest single processor case($P1$). Maximum speedup reflects computing power of the p-processor set relative to $P1$.

The speedup obtained through parallel execution is a good measure of the parallel algorithm and system performance. In the model and experimental results presented in this paper, speedup for a p-processor implementation refers to the ratio of the execution time on processor $P1$ to the execution time on the first p processors in the set $\mathbf{P}$.

$$speedup(p) = \frac{\text{Execution time on } P1}{\text{Execution time on the set } \{P1, P2 \ldots Pp\}}.$$

Since the processors do not have equal computational power, and the parallel execution times are compared against the fastest processor, it is not possible to obtain linear speedup through parallel execution. The maximum speedup that can be obtained on a p-processor execution, $speedup_{max}(p)$ is given by the ratio of the computing abilities of the p-processor set to that of the fastest processor, $P1$:

$$speedup_{max}(p) = \frac{\sum_{i=1}^{p} M_i}{M_1}.$$

The performance models developed above are now applied to a synchronous iterative algorithm example with $N = 1000$. Consider a computing platform consisting of up to 16 processors. The processor computing abilities vary linearly with the fastest processor, $P1$, being 10 times faster than the slowest processor, $P16$ (*i.e.*, $M_1 = 10 \times M_{16}$). In a p processor implementation, the N variables are distributed over the first p processors of set $\mathbf{P}$ in accordance with the conditions in equations 4 and 5. The speculation and checking functions are assumed to be small compared to computation ($f_{comp} = 100 \times f_{spec} = 50 \times f_{check}$). It is also assumed that the communication time per iteration, t_{comm} increases linearly with the number of processors used in the implementation. t_{comm} is set equal to the computation time per iteration in the 16-processor case. The parameters chosen are reasonable and are close to the measured values for the N-body simulation example presented in section 5.

Figure 5 shows the speedup versus number of processors predicted by the model with and without speculation for the case where the percentage of recomputations, $k = 2\%$. Speculative computation

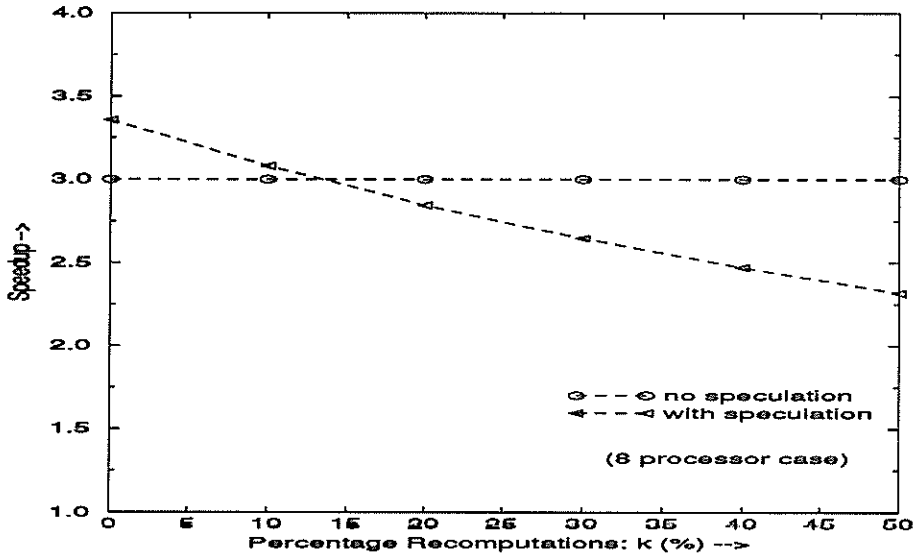

Figure 6: Speculative Computation: Effect of speculation error

has very little impact for small processor systems (2 to 5 processors). This is because the communication delays are relatively small and do not degrade performance significantly. For larger systems, speculative computation yields significant performance benefits, up to 25% on 16 processors. Note that in the "no speculation" case, performance begins to decrease after about 10 processors. This is because the communication costs increases with the number of processors and results in lower performance.

Figure 6 shows the effect of the speculation error on the performance for an 8-processor system. The figure shows the speedup on 8 processors versus the percentage of recomputations, k. Speculation yields performance gain over the no speculation case for errors less than 10%. A large error percentage increases the overhead of recomputing the variables and results in poorer performance. Thus, unless variables can be predicted reasonably well, there is no gain with this method. In many practical applications, however, predicting within a 10% error margin is not difficult. The N-body application discussed next illustrates this.

5 Case Study: Parallel N-body simulation

To illustrate the ideas and performance benefits of speculative computation, the technique was implemented on a simple $O(N^2)$ N-body simulation example[1]. The parallel algorithm was written under the PVM programming environment using the message passing paradigm. The algorithm runs on SUN/Sparc workstation network using up to 16 workstations.

The N-body problem involves a system of N particles where each particle exerts a force on every other particle. The evolution of such a system can be solved only by numerical simulation methods(for $N \geq 3$). The simulation proceeds in timesteps, each timestep computing the force between every pair of particles, the resultant force on each particle and updating its position and velocity based on the resultant force. The force between any two particles is given by Newton's universal law of gravitation. The resultant force on a particle is the vector sum of all its pairwise forces. The

[1] A more efficient $O(N \log N)$ is possible and has been implemented in the past[4]. Our objective here, however, is to illustrate the effectiveness of speculative computation, and the simpler $O(N^2)$ implementation is employed.

```
begin
Distribute particles to processors
for t = 1 to MAX
  On each processor, j, Do:
    begin
    send X_j to all processors
    num_recvd = 0
    while num_recvd < (p − 1)
        begin
        receive a message
        k = sender processor id
        compute force due to X_k
        num_recvd = num_recvd + 1
        end
    update velocity, position,  ∀x_i ∈ X_j
    end
end
```

Figure 7: The N-body simulation (no speculation case)

change in position and velocity of a particle due to the resultant force is given by Newton's laws of motion. Since all pairwise forces are computed, simulation of N particles results in a computational complexity of $O(N^2)$.

The experiment utilized a network of up to 16 SUN/Sparc workstations connected by a standard ethernet. The processors differ widely in their computing abilities. The fastest workstation(SparcStation 10/1) is rated 120 MIPS and the slowest(SUN 4/10) is 10 MIPS. To determine a processor's capacity, a small sequence of several operations was executed on each processor and the runtime on the processor was measured. The runtime divided by the number of operations is a measure of the processor's capacity, M_i.

The N particles simulated are distributed over the p processors such that each processor is allocated workload (*i.e.,* number of particles) proportional to its computing ability. Each processor is responsible for computing the resultant force and updating the velocity and positions of the particles allocated to it. At the start of each iteration (or timestep), each processor sends the current position and velocity of all its particles to all other processors. It then waits to receive such messages from other processors and, when the particle information is received, computes the forces on its particles due to all the particles in the system. Once the resultant forces are computed, it updates the velocity and position of its particles based on the forces and proceeds to the next iteration.

Figure 7 describes the algorithm in pseudocode form using the same notation as in section 2 for synchronous iterative algorithms. p is the number of processors and x_i refers to a particle (*i.e.,* its mass, position, velocity and other attributes), $\mathbf{X}_j$ refers to the set of particles allocated to processor j.

The algorithm requires each processor to wait for messages and then compute the resultant forces. A considerable amount of time (e.g., 40% for a 1000 particle simulation on 16 processors) is spent in communication and most of this time is spent waiting for messages. Speculative computation was

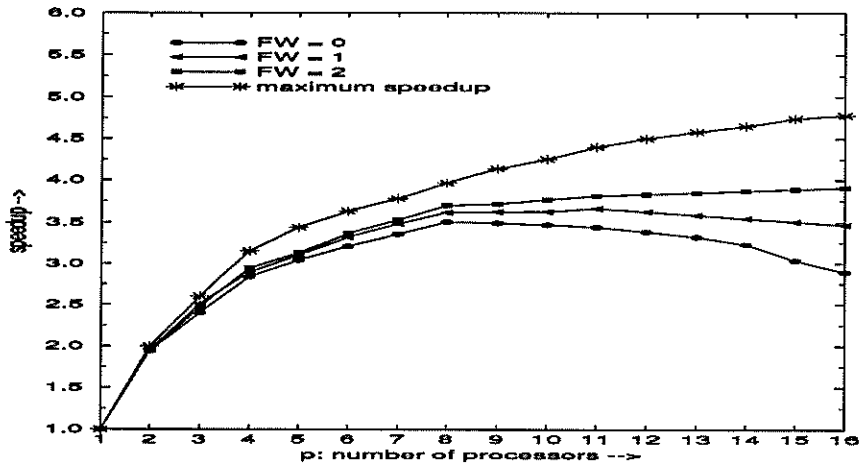

Figure 8: N-body simulations with speculative computation

Speedups are relative to the fastest single processor case($P1$). Maximum speedup reflects computing power of the p-processor set relative to $P1$.

incorporated into the algorithm of figure 7. While a processor is waiting for a message containing particle information from another processor, it speculates the remote particle positions and computes forces based on the speculated values. When the message is received, the speculated and real values are compared; if the error is unacceptable, the resultant force is recomputed.

The position of a remote particle is speculated based on its previous position and velocity, assuming that the velocity remains constant for one timestep. This introduces a small error since the resultant forces on the particle may have altered its velocity. If the position of the remote particle a is given by the vector $\vec{r}_a$ and the velocity by $\vec{v}_a$, the speculated position at iteration t is given by

$$\vec{r}_a{}^*(t) = \vec{r}_a(t-1) + \vec{v}_a(t-1) \times \Delta t, \tag{10}$$

where, Δt is the timestep size. The force between a local particle b and the particle a at iteration t is computed using the speculated position of a, $\vec{r}_a{}^*(t)$.

When the message containing the actual position of a is received, the speculation error and the effect of this error on the force computations are calculated. The speculation error for particle a is given by the difference between the speculated and the actual values. The effect of this error on the force exerted on particle b is (approximately) proportional to the ratio of this error and the distance between particles a and b.

$$error_{a,b} = \frac{\|\vec{r}_a{}^*(t) - \vec{r}_a(t)\|}{\|\vec{r}_a(t) - \vec{r}_b(t)\|}. \tag{11}$$

The computation using the speculated value is considered acceptable if this ratio is less than a predefined threshold, θ.

Figure 8 shows the speedup (based on measured execution times) versus the number of processors for the N-body simulation example (of 1000 particles) for various (forward) speculation windows with the error threshold, $\theta = 0.01$. For small number of processors (2 to 4 processors), speculative computation has very little performance impact. This is because the communication delays are small and do not affect performance to a great extent. As the number of processors increase, the effect of

Table 2: Measured times for a 16-processor, 1000-particle simulation

FW	computation time(sec)	communication time(sec)	speculation time(sec)	check time(sec)	total time(sec)
0	5.83	4.73	0	0	10.56
1	5.85	1.43	0.2	1.02	8.52
2	5.82	0.22	0.3	1.5	7.79

Table 3: Effect of error bounds on recomputations and computation error

θ	Incorrect speculations	Max. error in force
0.1	< 1%	20 %
0.05	< 1%	10 %
0.01	2 %	2 %
0.005	5 %	1 %
0.001	20 %	0.2 %

communication delays are more significant and speculative computation makes a sizable performance impact. For 16-processors, 34% performance gain over the no speculation case is observed. The speedup is within 20% of the maximum speedup on 16 processors. A window size of 0 corresponds to the no speculation case.

The performance of an application with speculative computation is influenced by several factors. Among them are the complexity of the speculation, error checking and correction functions, accuracy of the speculation function and the error bounds defined by the user. In the current implementation of N-body simulation, computing the force between a pair of particles involves about 70 floating point operations, speculating the position of a particle takes 12 floating point operations, error checking involves 24 operations. Table 2 shows the average time spent per iteration in the various phases of the execution for a eight processor, 1000 particle simulation. The force computation, communication, speculation, error checking and correction times are shown for speculation windows of 0, 1 and 2. In the above example, the speculation and error checking time is small compared to the computation times and the communication delay. Since the overheads for speculative computing are small, significant performance improvement is observed.

The accuracy of the speculation function and the user defined error bounds also affect the performance of speculative computation. Force computations using the speculated value are accepted if the effect of the speculation error is less than a certain error threshold, θ. Table 3 shows the effect of θ on the percentage of recomputations. As expected, lower bounds on θ directly result in a lower error in the particle force calculations and in an increased number of erroneous speculations. The table also shows the maximum error in the force computations for a given value of θ. From the table, a θ of 0.01 seems reasonable. In our implementation, the speculation function makes use of the velocity information (first derivative) of each particle to speculate the next position of the particle. Using higher order derivatives may increase the accuracy of speculation but make the speculation function more complex. This tradeoff has not yet been studied in our current implementation.

The performance model developed in section 4 was parameterized to represent the N-body sim-

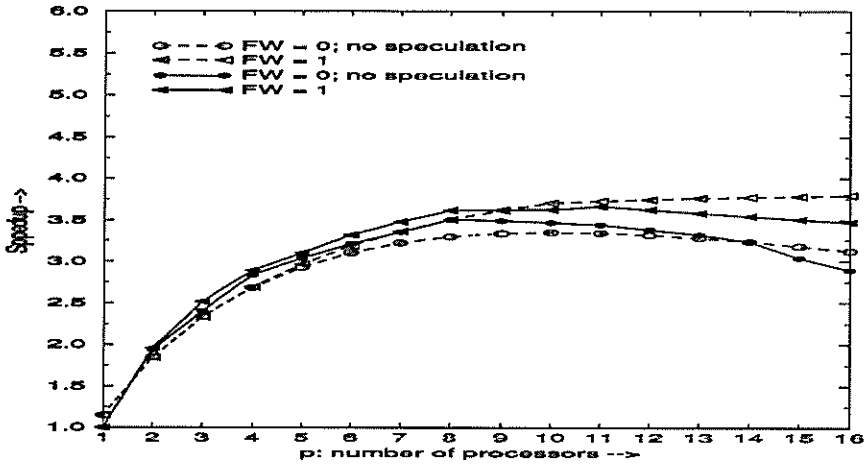

Figure 9: Comparing model and measured performance
Solid lines indicate measured values. Dashed lines indicate model predictions

ulation example. In Figure 9, we compare the model predictions with the measured values. The model predictions are within 10% of the measured values for 8 or fewer processors and within 25% for 8 to 16 processors. The difference in model and measured values are mainly due to some of the simplifying assumptions in the model. The communication time was assumed to remain constant over all iterations and on all processors. However, network traffic and processor loads cause significant variations in the communication times. In large processor systems (> 8 processors), network contention (not accounted for in the model) causes additional communication delay and results in significant performance degradation. Similarly, background processor loads cause the computation times on processors to vary slightly with time.

6 Conclusions

Speculative computation has been proposed as a technique to mask communication latencies and delays by overlapping them with useful computation. The technique potentially can yield significant performance improvements particularly in distributed systems such as workstation networks where communication delays are large and subject to considerable variation. The paper presented speculative computation in the context of synchronous iterative algorithms. The performance benefits of speculative computation are illustrated using an N-body simulation application. Up to 34% improvement in performance over the no speculation case was measured . The speedups obtained using the technique were within 25% of the maximum attainable speedup for the set of processors used. The experiment was run on a network of SUN/Sparc workstations operating under the PVM environment. This shows that speculative computing is an effective technique to overcome communication delays. The technique is likely to yield similar performance benefits for other applications.

An empirical performance model was developed to estimate the potential the performance benefits of speculative computing. The model also illustrates the effect of speculation error on performance. Comparison of the model and measured performance shows less than 10% error for small systems(< 8 processors) and about 25% error for larger systems.

Future work on speculative computation includes developing a more sophisticated performance model that accounts for variations in computation and communication times of processors and different forward and backward window sizes for speculation. The model can be used to estimate performance of the technique for a given application and computing system and in making design decisions with respect to the various tradeoffs involved in its implementation. With the recent trend in parallel computing favoring cost-effective networked computing systems, overcoming communication delays in parallel algorithms is crucial in obtaining good parallel performance.

References

[1] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentics Hall, Englewood Cliffs, New Jersey 07632, 1989.

[2] F.W. Burton. Speculative computation, parallelism, and functional programming. *IEEE Trans. Comput.*, C(34):1190–1193, Dec 1985.

[3] M. Dubios and F.A. Briggs. Performance of synchronized iterative processes in multiprocessor systems. *IEEE Trans. Software Eng.*, SE-8(4), July 1982.

[4] Mark Franklin and Vasudha Govindan. The N-body Problem: Distributed System Load Balancing and Performance Evaluation. In *Proceedings of the 6th International Conference on Parallel and Distributed Computing Systems.* ISCA, October 1993. Louisville, Kentucky, USA.

[5] Anoop Gupta, John Hennessey, Kourosh Gharachorloo, Todd Mowry, and Wolf-Dietrich Weber. Comparitive evaluation of latency reducing and tolerating techniques. In *Proceedings of International Symposium on Computer Architecture*, pages 254–263, 1991.

[6] Gary M. Johnson. Exploiting parallelism in computational science. *Future Generation computer Systems*, 5:319–337, 1989.

[7] Joel Saltz and Vijay Naik. Towards developing robust algorithms for solving partial differential equations on mimd machines. *Parallel Computing*, 6:19–44, 1988.

[8] Joel Saltz, Vijay Naik, and David Nicol. Reduction of the effects of communication delays in scientific algorithms on message passing mimd architectures. *SIAM J. SCI. STAT. COMPUT.*, 8(1), Jan 1987.

[9] Vaidy S. Sunderam. PVM: A Framework for Parallel Distributed Computing. *Concurrency: Practice and Experience*, 2, december 1990.

[10] Ellen Witte, Roger Chamberlian, and Mark Franklin. Parallel simulated annealing using speculative computation. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):483–494, october 1991.

[11] David E. Womble. The performance of asynchronous algorithms on hypercubes. Technical Report SAND88-2714, Sandia National Laboratories, Albuquerque, New Mexico, Dec 1988.