Memory Hierarchy Considerations for Cost-Effective Cluster Computing

Xing Du, *Member*, *IEEE Computer Society*,
Xiaodong Zhang, *Senior Member*, *IEEE*, and Zhichun Zhu

Abstract—Using off-the-shelf commodity workstations and PCs to build a cluster for parallel computing has become a common practice. The cost-effectiveness of a cluster computing platform for a given budget and for certain types of applications is mainly determined by its memory hierarchy and the interconnection network configurations of the cluster. Finding such a cost-effective solution from exhaustive simulations would be highly time-consuming and predictions from measurements on existing clusters would be impractical. We present an analytical model for evaluating the performance impact of memory hierarchies and networks on cluster computing. The model covers the memory hierarchy of a single SMP, a cluster of workstations/PCs, or a cluster of SMPs by changing various architectural parameters. Network variations covering both bus and switch networks are also included in the analysis. Different types of applications are characterized by parameterized workloads with different computation and communication requirements. The model has been validated by simulations and measurements. The workloads used for experiments are both scientific applications and commercial workloads. Our study shows that the depth of the memory hierarchy is the most sensitive factor affecting the execution time for many types of workloads. However, the interconnection network cost of a tightly coupled system with a short depth in memory hierarchy, such as an SMP, is significantly more expensive than a normal cluster network connecting independent computer nodes. Thus, the essential issue to be considered is the trade-off between the depth of the memory hierarchy and the system cost. Based on analyses and case studies, we present our quantitative recommendations for building cost-effective clusters for different workloads.

Index Terms—Clusters, cost model, memory hierarchy, performance evaluation, SMP, workstations.

1 Introduction

With the rapid development and advancement of commodity processors and networking technology, parallel computing platforms are shifting from expensive customer-designed MPPs (such as CRAYs and CM-5s) to cheap and commodity-designed symmetric multiprocessors (SMPs) and clusters of workstations, personal computers (PCs), and even of SMPs. Using off-the-shelf hardware and software to construct a parallel system provides a large range of scalability from "desktop-to-teraflop" [13]. Compared with simply buying an expensive MPP box, the cluster approach gives users flexibility in constructing, upgrading, and scaling a parallel system. However, the flexibility also provides multiple system configuration options for a given budget and a given type of workload, raising performance optimization issues which need addressing. We believe the following two questions are fundamental to cluster computing: First, what is an optimal or a nearly optimal cluster platform for cost-effective parallel computing under a given budget and a given type of workload? Second, what is a cost-effective way to upgrade or scale an existing cluster platform for a given budget increase and a given type of workload? There are no existing optimization solutions to help users construct clusters in a cost-effective way. Solutions from exhaustive simulations would be highly time-consuming; and predictions based on measurements from existing clusters would be impractical.

In this paper, we present an analytical model to address the two above questions. The model covers the platforms of a single SMP, a cluster of workstations/PCs, or a cluster of SMPs by changing various architectural parameters. Network models covering two representative networks are also included in the analysis. Different types of applications are characterized by parameterizing workloads with different computation and communication requirements. The model is based on a prediction of the average execution time per instruction for an application. It is derived from the application's locality property and the memory hierarchy of a targeted parallel cluster platform. Using the model, we can quickly determine a nearly optimal platform for a given budget and for a given workload. The model can also be used to guide how to upgrade an existing system in a cost-effective way for a given budget increase. We have also made efforts to simplify the model for practical usage.

The analytical model is verified by simulations and measurements. We verify the accuracy of the model by constructing a set of simulators to simulate different types of clusters and by comparing the modeling results with the simulations. The comparison indicates that, for a single SMP, the modeling results are fairly close to the simulation results (the difference is below 5 percent). For clusters of workstations and clusters of SMPs, after properly adjusting the access rates to remote memory modules, we are able to

- *X. Du is with the Server Technologies Division, Oracle Corp., Redwood Shores, CA 94065. E-mail: xing.du@us.oracle.com.*
- *X. Zhang and Z. Zhu are with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23187-8795. E-mail: {zhang, zzhu}@cs.wm.edu.*

Manuscript received 28 June 1999; revised 9 Mar. 2000; accepted 16 May 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 110132.

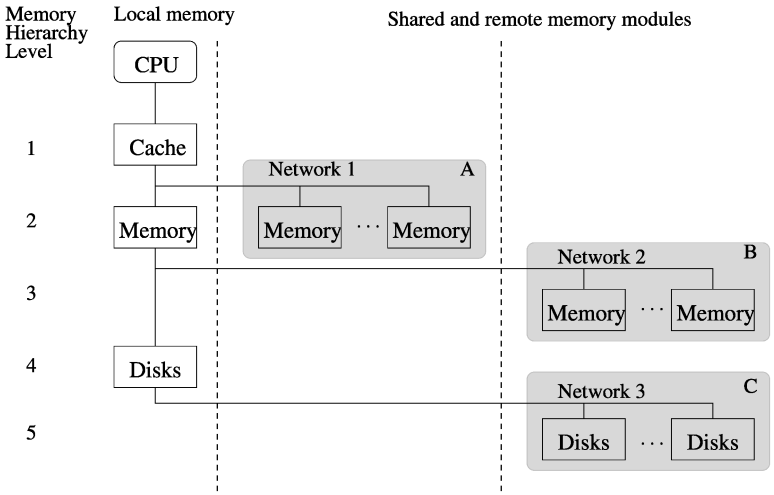

Fig. 1. The memory hierarchy from the standpoint of one processor.

obtain modeling results fairly close to the simulation results (less than 10 percent) as well. The adjustment is necessary to compensate for the shared memory coherence overhead, which is an important part of system activities, but difficult to model. Our modeling results are acceptable for performance prediction and evaluation of parallel computing on clusters. In addition, compared with measured TPC-C commercial workload execution results on SMPs, the error rates of our modeling results are also below 5 percent.

Finally, we present several case studies which use the model to effectively build a cluster and upgrade an existing cluster for different parallel computing applications. By using the cost model to choose a parallel computing platform, we find that an optimal choice mainly depends on the type of workloads to be run on the selected platform. The performance of some applications is sensitive to the total number of processors in the system, while the performance of others is more dependent on the speed of the interconnection networks. The cost model provides a quick and simple way to determine an optimal or a nearly optimal solution without conducting a time-consuming exhaustive simulation.

The paper is organized as follows: Section 2 presents a general memory hierarchical model of a cluster architecture and the parallel programming model and program characterization method we have used for this performance study. We discuss our analytical model and its variations of memory hierarchies and of networks in Section 3. We present the model verification results by simulations using scientific applications in Section 4 and verification results by measurements using TPC-C commercial workload in Section 5. Several case studies using the model are given in Section 6. Finally, we summarize the paper and discuss our current and future work in Section 7.

2 Characterizing Memory Hierarchy and Parallel Applications

We are considering three types of clusters: a single SMP, multiple workstations or PCs, and multiple SMPs. There are five memory access levels for a processor in the hierarchy of a cluster covering the three types:

1. accesses to its own cache,
2. accesses to its own memory or the shared-memory associated with all the processors in an SMP,
3. accesses to a remote memory module associated with another machine in the cluster,
4. accesses to its own disk, and
5. accesses to a remote disk associated with another machine in the cluster (see Fig. 1).

Fig. 1 also shows different interconnection networks in a cluster. For an SMP system, a processor may access the memory modules of other processors (at the same level) with the same latency through Network 1 (usually a memory bus inside an SMP). For a cluster of workstations, the access to a remote memory module goes through Network 2 with a much higher latency than the access to the local memory. Network 2 is the cluster network. Two representative types are bus-based and switch-based networks. Remote disks can also be shared through Network 3, which in most cases uses the same physical networks used for Network 2.

We consider the single-program multiple-data (SPMD) programming model and bulky synchronous scientific applications [7], [8], [14], [19]. Each process independently executes a task defined as a loop and it synchronizes with other parallel processes through the barrier at the end of the loop. We also consider the on-line transaction processing (OLTP) workload in our study. Different from scientific applications, there is no execution order constraint among transactions running on different processors. For example, TPC-C workloads have no barrier operations [1].

A shared-memory implementation has been shown to be a promising and desirable paradigm for exploiting parallel execution. We adopt this paradigm for the bulky synchronous structure and OLTP workloads. The shared-memory is supported by hardware in SMPs. For clusters of workstations or clusters of SMPs, some work [18], [21], [29] has been done on the emulation of shared-memory. We assume

there is a software layer for programmers which emulates the shared-memory in the cluster.

Our execution model of cluster computing is mainly based on the probabilities of references to different levels of the memory hierarchy in Fig. 1. The probability is determined based on *stack distance curves* taken directly from an address stream [6].

The work in [15] uses the same approach for evaluating the performance of memory hierarchies of uniprocessor systems. In general, the stack distance of datum A at one position of the address stream is the number of unique data items between this reference and the next reference to A. The distribution of stack distances can be expressed as a cumulative probability function, denoted $P(x)$, which represents the probability of references within a given stack distance of x. This fits an LRU-managed and fully associative cache hit rate well if x is considered as the cache size. The probability density function, denoted $p(x)$, describes the frequency of references at stack distance x. Similar to other related work [15], [20], [22], we model $P(x)$ in the form of

$$P(x) = 1 - \frac{1}{(x/\beta + 1)^{\alpha-1}}, \tag{2.1}$$

thus, $p(x)$ in the form of

$$p(x) = \frac{\beta^{\alpha-1}(\alpha-1)}{(x+\beta)^{\alpha}}, \tag{2.2}$$

where $\alpha > 1$ and $\beta > 1$ are workload parameters to characterize the locality of a program. The program locality improves with the decrease of β or the increase of α. The memory modules at different levels in a hierarchy of the cluster can be viewed as caches of different sizes and access speeds. Thus, the stack distance model discussed above is suitable for our performance evaluation of the cluster memory hierarchy.

In addition to locality information expressed in terms of α and β, we use another parameter, γ to represent the ratio between the number of instructions which incur memory references (M) and the number of total instructions in an application ($m + M$), where m is the number of instructions which do not incur any memory reference. Parameter γ reflects the memory access variations of application programs. The larger γ is, the more significantly the memory accesses affect the application's performance. Parameters α, β, and γ may be obtained through address stream analysis and instruction counting in the execution of a program on a target cluster or through a simulated execution of application programs.

Detailed descriptions about memory hierarchy and program characteristics can be found in [9].

3 A Cost Model and Memory Access Time Predictions

3.1 A Cost Model

We assume that parallel tasks are evenly distributed among all processors. Based on Amdahl's Law, the average execution time of an application on a parallel system is modeled as the sum of the computation time without network communication (instructions without memory accesses) and the computation time with network communication (instructions with memory accesses):

$$\begin{aligned} E(App) &= \frac{m}{nN}\frac{1}{S} + \frac{M}{nN}\left(\frac{1}{S} + T_{mem}\right) \\ &= \frac{1}{nN}\left(\frac{m+M}{S} + MT_{mem}\right), \end{aligned} \tag{3.3}$$

where App represents an application program, m is the number of instructions without memory accesses, M is the number of instructions with memory accesses, n is the number of processors in a machine, N is the number of machines in the cluster, S is the processor speed as the number of instructions executed per second, and T_{mem} is the average memory access time per reference in the cluster. The total number of instructions in the program is $m + M$. Consequently, we have the average execution time per instruction:

$$E(Instr) = \frac{E(App)}{m+M} = \frac{1}{nN}\left(\frac{1}{S} + \gamma T_{mem}\right), \tag{3.4}$$

where $Instr$ represents a program instruction and $\gamma = \frac{M}{m+M}$. The average memory access time per reference, T_{mem}, is a key factor affecting the execution performance. We adopt the same model as the one used in [5], [15], and [26] in computing the average memory access time:

$$\begin{aligned} T_{mem} &= \sum_{i}^{k} P_i t_i \\ &= t_1 + t_2 \int_{s_1}^{\infty} p(x)dx + t_3 \int_{s_2}^{\infty} p(x)dx + \cdots + \\ &\quad t_k \int_{s_{k-1}}^{\infty} p(x)dx, \end{aligned} \tag{3.5}$$

where P_i and t_i are the access probability and the average access time, respectively, to the memory hierarchy at the ith level, $i = 1, \ldots, k$. Simultaneous accesses to the same level of the memory hierarchy from several processors cause contention and make the average access time to that level significantly higher than that without contention. The average access time varies due to variations of network architectures and of the number of simultaneous accesses.

Table 1 lists all notations used in the model and their descriptions in three different groups: architecture parameters, program parameters, and budget/cost parameters. In the group of architecture parameters, all the parameters except t_i $(i = 1, \ldots, k)$ and T_{mem} are known for a given platform and are architecture dependent. In the group of program parameters, λ_i and P_i $(i = 1, \ldots, k)$ are modeled based on the program dependent parameters α, β, and γ. The budget/cost parameters are user-specified and case dependent.

For given architecture and program parameters, the execution performance in (3.3) and (3.4) can be determined if the average memory access time T_{mem} is known. Therefore, T_{mem} is the key variable to be modeled in this study.

The cost of a cluster is the sum of the cluster machine cost and the cluster network cost. It can be expressed as:

TABLE 1
Notations for the Cost Model

Architecture	Parameter descriptions
N	The number of machines in a cluster.
n	The number of processors in a machine.
S	The processor speed (the number of instructions executed per second).
k	The number of levels in the memory hierarchy.
s_i	The memory size in bytes at the ith level of the memory hierarchy, $i = 1, ..., k$.
τ_i	Access time per reference to the ith level of the memory hierarchy without contention.
t_i	Access time per reference to the ith level of the memory hierarchy with contention.
τ_{bus}	The service time per reference of a cluster bus network.
τ_{switch}	The service time per reference of a cluster switch network.
T_{mem}	Average memory access time per reference in the cluster.
Program	Parameter descriptions
m	The total number of instructions without memory references in a program.
M	The total number of instructions with memory references in a program.
γ	$\gamma = \frac{M}{m+M}$.
λ_i	The access rate to the ith level of the memory hierarchy.
P_i	The probability of accessing to the ith level of the memory hierarchy.
Budget/cost	Parameter descriptions
B	The budget in dollars for building a cluster.
B'	The budget increase in dollars for upgrading a cluster.
$C_{machine}(n)$	The cost in dollars of one machine with n processors.
C_{net}	The cost in dollars for connecting one machine in a cluster.
$C_{cluster}$	The total cost in dollars of a cluster.

$$C_{cluster} = NC_{machine}(n) + NC_{net}, \tag{3.6}$$

where N is the number of machines, $C_{machine}(n)$ is the cost of a machine with n processors, and C_{net} is the network cost to connect one machine in the cluster. We assume that the cluster is a homogeneous platform consisting of identical machines, either SMPs or uniprocessor workstations.

One major goal in our study is to determine n, N, and the types of networks of the cluster by minimizing the average execution time per instruction in (3.4), for a given budget B and the other given architecture, program, and budget/cost parameters. This optimization problem forms our cost model and is expressed as:

$$\begin{cases} minimize \;\; E(Instr) \\ subject \;\; to \;\; C_{cluster} \leq B. \end{cases} \tag{3.7}$$

Another goal in our study is a variation of the above optimization problem, which is to determine an optimal way to scale or upgrade an existing cluster system for a given budget increase. The problem can be defined as follows: For a given existing cluster with all the given architecture parameters, a budget increase B', and new architecture and cost parameters for upgrading, we determine a new cluster configuration by minimizing the execution time per instruction in (3.4).

Because our target solution variables are integer types in (3.6) and (3.7), this is a typical integer programming problem. Fortunately, in the real world, the problem domain is not very large because n is a small number for an SMP and N is also not a large number for a cluster, especially as the power of each machine has rapidly increased. We can determine these integer variables by enumerating solutions and choosing the best as the optimal solution. The quality of our predicted solutions is determined by the correctness and the accuracy of the model in predicting the average memory access time, T_{mem}, for each of these three cluster platforms. We will discuss the models of T_{mem} in detail in the following subsections.

3.2 A Memory Access Time Model for SMPs

A single SMP ($N = 1$, and $n > 1$) is a special cluster platform not requiring a cluster network. Each processor has its own cache and shares the main memory with other processors through a memory bus. The memory hierarchy consists of three levels: a local cache, the shared memory, and disks. Since the cache is dedicated to each processor, the access time to it from its own processor (t_1) can be considered as a constant and equals the cache access time without contention (τ_1). The average access time to the memory (t_2) is determined by two types of accesses to it: accesses to ordinary variables and accesses to variables used for barriers (we call them the barrier variables and hereafter define that the access to a barrier is finished when and only when all the accesses reach the barrier). The access to barriers incurs extra time because of the synchronization

with other processes. We model the average memory access time t_2 as:

$$t_2 = P_o t_2(o) + P_b t_2(b), \tag{3.8}$$

where P_o and P_b are the access probabilities to ordinary and barrier variables, respectively, and $t_2(o)$ and $t_2(b)$ are the average access times to the two types of variables, respectively. Assume the access rate to the memory for ordinary variables from a processor is $\lambda_2(o)$. (Without losing generality, we assume the access rates from different processors are identical.) The access time to the memory without contention is a constant (τ_2). Since each processor executes one process at a time, the system can be viewed as n terminals (processors) submitting requests to one server (the shared memory). The utilization of the memory can be calculated as

$$U = 1 - \frac{1}{n!(\lambda_2(o)\tau_2)^n} \cdot \frac{1}{\sum_{k=0}^{n} \frac{1}{k!(\lambda_2(o)\tau_2)^k}}. \tag{3.9}$$

According to the flow balance principle, the throughput of the memory and the throughput of processors have the following relationship:

$$U \cdot \frac{1}{\tau_2} = (n - Q) \cdot \lambda_2(o), \tag{3.10}$$

where Q is the average number of requests at the memory waiting or receiving service. Using Little's Law, we calculate the average memory access time as

$$t_2(o) = \frac{Q}{U/\tau_2}. \tag{3.11}$$

Combining (3.9), (3.10), and (3.11), we obtain the average memory access time as

$$t_2(o) = \frac{n\tau_2}{1 - \frac{1}{n!(\lambda_2(o)\tau_2)^n} \cdot \frac{1}{\sum_{k=0}^{n} \frac{1}{k!(\lambda_2(o)\tau_2)^k}}} - \frac{1}{\lambda_2(o)}. \tag{3.12}$$

A barrier is used to achieve synchronization among processes for scientific applications. The average time to go through a barrier is derived as follows: We denote the barrier time in process i as X_i and X as the barrier cycle time of the whole system. Then we have

$$X = max\{X_1, X_2, \ldots, X_n\}.$$

Using Order Statistics [17], [24], we have the expectation of X:

$$E[X] = \frac{1}{\lambda_2(b)} \sum_{i=1}^{n} \frac{1}{i},$$

where $\lambda_2(b)$ is the access rate to a barrier in the shared memory. Consequently, the average waiting time for a barrier is:

$$t_2(b) = E[X] - \frac{1}{\lambda_2(b)}.$$

Then, the average access time to barrier variables is expressed as

$$t_2(b) = \begin{cases} 0 & \text{if } n = 1 \\ \frac{1}{\lambda_2(b)}(\frac{1}{2} + \cdots + \frac{1}{n}) & \text{if } n > 1. \end{cases} \tag{3.13}$$

Furthermore, the probabilities and access rates of the two types hold the following relationships:

$$P_o = \frac{\lambda_2(o)}{\lambda_2(o) + \lambda_2(b)},$$

and

$$P_b = \frac{\lambda_2(b)}{\lambda_2(o) + \lambda_2(b)}.$$

Substituting (3.12), (3.13), and the above expressions of P_o and P_b into (3.8) for $n > 1$, we obtain the average access time to the shared-memory as follows:

$$t_2 = \frac{1}{\lambda_2(o) + \lambda_2(b)} \left(\frac{n\lambda_2(o)\tau_2}{1 - \frac{1}{n!(\lambda_2(o)\tau_2)^n} \cdot \frac{1}{\sum_{k=0}^{n} \frac{1}{k!(\lambda_2(o)\tau_2)^k}}} - 1 + \frac{1}{2} + \cdots + \frac{1}{n} \right). \tag{3.14}$$

A uniprocessor system is a special case of an SMP. If we substitute $n = 1$ into (3.14), we obtain a model identical to the one proposed in [15] for a uniprocessor system.

Similarly, if n processors share disks through an I/O bus, the average access time for the disks is

$$t_3 = \frac{n\tau_3}{1 - \frac{1}{n!(\lambda_3\tau_3)^n} \cdot \frac{1}{\sum_{k=0}^{n} \frac{1}{k!(\lambda_3\tau_3)^k}}} - \frac{1}{\lambda_3}, \tag{3.15}$$

where λ_3 is the access rate to disks, τ_3 is the access time to a disk without contention, and we assume that all barrier operations are performed in the main memory.

Substituting (3.14) and (3.15) into (3.5), we obtain the average memory access time T_{mem} for an SMP as follows:

$$\begin{aligned} T_{mem} &= t_1 + t_2 \int_{s_1}^{\infty} p(x)dx + t_3 \int_{s_2}^{\infty} p(x)dx \\ &= \tau_1 + \frac{1}{\lambda_2(o) + \lambda_2(b)} \int_{s_1}^{\infty} p(x)dx \\ &\quad \left(\frac{n\lambda_2(o)\tau_2}{1 - \frac{1}{n!(\lambda_2(o)\tau_2)^n} \cdot \frac{1}{\sum_{k=0}^{n} \frac{1}{k!(\lambda_2(o)\tau_2)^k}}} - 1 + \frac{1}{2} + \cdots + \frac{1}{n} \right) \\ &\quad + \left(\frac{n\tau_3}{1 - \frac{1}{n!(\lambda_3\tau_3)^n} \cdot \frac{1}{\sum_{k=0}^{n} \frac{1}{k!(\lambda_3\tau_3)^k}}} - \frac{1}{\lambda_3} \right) \int_{s_2}^{\infty} p(x)dx. \end{aligned}$$

In a parallel program, $\lambda_2(o) \gg \lambda_2(b)$. Then,

$$\lambda_2 = \lambda_2(o) + \lambda_2(b) \approx \lambda_2(o).$$

Considering this approximation, we further simplify T_{mem} as:

$$T_{mem} = \tau_1 + \frac{1}{\gamma S} \left(\frac{n\lambda_2\tau_2}{1 - \frac{1}{n!(\lambda_2\tau_2)^n} \cdot \frac{1}{\sum_{k=0}^{n} \frac{1}{k!(\lambda_2\tau_2)^k}}} + \frac{n\lambda_3\tau_3}{1 - \frac{1}{n!(\lambda_3\tau_3)^n} \cdot \frac{1}{\sum_{k=0}^{n} \frac{1}{k!(\lambda_3\tau_3)^k}}} - 2 + \frac{1}{2} + \cdots + \frac{1}{n} \right), \tag{3.16}$$

where

$$\lambda_2 = \frac{M}{m+M} \int_{s_1}^{\infty} p(x)dx \times S = \gamma S \int_{s_1}^{\infty} p(x)dx$$

and

$$\lambda_3 = \frac{M}{m+M} \int_{s_2}^{\infty} p(x)dx \times S = \gamma S \int_{s_2}^{\infty} p(x)dx.$$

For given $p(x)$ and γ characterizing a class of parallel programs and, for given n, S, s_1, s_2, τ_1, τ_2, and τ_3 characterizing an SMP, we can determine the average access time T_{mem} in an SMP using (3.16).

3.3 A Memory Access Time Model for a Cluster of Workstations

A cluster of workstations is considered a parallel system with $N > 1$ and $n = 1$, whose memory hierarchy is composed of four levels: the cache, the local memory module, $(N-1)$ remote memory modules, and disks. The interconnection network of workstations is either bus-based (e.g., an Ethernet) or switch-based (e.g., an ATM network). In addition, we assume that the cluster is a homogeneous system where each workstation has an identical architecture. Considering the four levels of the memory hierarchy, we model the average memory access time (T_{mem}) of the cluster of workstations as follows:

$$T_{mem} = t_1 + t_2 \int_{s_1}^{\infty} p(x)dx + t_3 \int_{s2}^{\infty} p(x)dx + t_4 \int_{s_3'}^{\infty} p(x)dx. \tag{3.17}$$

The variables are listed in Table 1. Variable s_3' represents the total size of the shared-memory from each workstation point of view, which is the sum of its own local memory size (s_2) and the total size of all the local memory modules of other workstations ($(N-1)s_2$):

$$s_3' = s_2 + s_3 = s_2 + (N-1)s_2 = Ns_2. \tag{3.18}$$

We will discuss models for two representative types of interconnection networks, bus-based and switch-based networks, in the rest of the subsection.

3.3.1 Bus-Based Clusters

The behavior of remote memory accesses from a workstation in a bus-based cluster is similar to that of memory accesses in an SMP except that the contention occurs in the cluster bus rather than in the memory bus. The average access time varies as access rates and the number of simultaneous accesses vary. In such a system, the cache access time from its own processor in each workstation is identical and constant ($t_1 = \tau_1$). Because the number of disk accesses by local and remote workstations is small in comparison with the number of memory accesses and because the probability of simultaneous accesses by several workstations to a disk is even lower, we also assume the average disk access time is a constant ($t_4 = \tau_4$) in order to simplify the discussion.

The memory is accessed by two sources: the local processor (at rate λ_2), and remote processors (at rate λ_3). Because of the symmetric features of parallel systems and application programs, the memory is accessed by remote processors at the rate of

$$\frac{\lambda_3}{N-1}(N-1) = \lambda_3.$$

However, parallel applications with $\lambda_2 \gg \lambda_3$ are practically meaningful in a bus-based cluster. Thus, we deemphasize the effect of the remote accesses in the model, and consider the average memory access time to the memory as τ_2. Then, T_{mem} can be rewritten as:

$$T_{mem} = \tau_1 + \tau_2 \int_{s_1}^{\infty} p(x)dx + t_3 \int_{s_2}^{\infty} p(x)dx + \tau_4 \int_{s_3'}^{\infty} p(x)dx.$$

Defining the service time in the bus for one remote memory access as τ_{bus}, a derivation similar to that of the average memory access time of SMPs in (3.14) gives the average response time of the bus as follows:

$$\frac{1}{\lambda_3(o) + \lambda_3(b)} \left(\frac{N\lambda_3(o)\tau_{bus}}{1 - \frac{1}{N!(\lambda_3(o)\tau_{bus})^N} \cdot \frac{1}{\sum_{k=0}^{N} \frac{1}{k!(\lambda_3(o)\tau_{bus})^k}}} - 1 + \frac{1}{2} + \cdots + \frac{1}{N} \right).$$

The average access time to a remote memory module is approximated as the response time of the bus plus the local memory access time:

$$t_3 = \frac{1}{\lambda_3(o) + \lambda_3(b)} \left(\frac{N\lambda_3(o)\tau_{bus}}{1 - \frac{1}{N!(\lambda_3(o)\tau_{bus})^N} \cdot \frac{1}{\sum_{k=0}^{N} \frac{1}{k!(\lambda_3(o)\tau_{bus})^k}}} - 1 + \frac{1}{2} + \cdots + \frac{1}{N} \right) + \tau_2.$$

Considering $\lambda_3(o) + \lambda_3(b) = \lambda_3 \approx \lambda_3(o)$ $(\lambda_3(o) \gg \lambda_3(b))$, we have the following average memory access time model of a bus-based workstation cluster:

$$T_{mem} = \tau_1 + \frac{1}{\gamma S} \left((\lambda_2 + \lambda_3)\tau_2 + \lambda_4\tau_4 + \frac{N\lambda_3\tau_{bus}}{1 - \frac{1}{N!(\lambda_3\tau_{bus})^N} \cdot \frac{1}{\sum_{k=0}^{N} \frac{1}{k!(\lambda_3\tau_{bus})^k}}} - 1 + \frac{1}{2} + \cdots + \frac{1}{N} \right), \tag{3.19}$$

where

$$\lambda_2 = \gamma S \int_{s_1}^{\infty} p(x)dx,$$
$$\lambda_3 = \gamma S \int_{s_2}^{\infty} p(x)dx,$$

and

$$\lambda_4 = \gamma S \int_{s_3'}^{\infty} p(x)dx.$$

3.3.2 Switch-Based Clusters

A major difference between a switch-based cluster and a bus-based cluster is the point of contention: The contention occurs in each switch port in the switch-based cluster and it occurs in the bus in the bus-based cluster. A typical switch contention occurs when two workstations simultaneously access the local memory of another workstation. Messages can be passed simultaneously as long as there is no message contention at switch ports. This major difference is reflected in the different models of t_3 between the two clusters. We assume that the remote memory accesses from one workstation are uniformly distributed among all other workstations. A workstation accesses remote memory modules at the rate of λ_3 and its average access rate to the memory of any other workstations is $\frac{\lambda_3}{N-1}$. There are $N-1$ other workstations. Defining the switch service time as τ_{switch}, we have

$$t_3 = \frac{1}{\lambda_3(o)+\lambda_3(b)} \left(\frac{(N-1)\lambda_3(o)\tau_{switch}}{1-\frac{(N-1)^{N-1}}{(N-1)!(\lambda_3(o)\tau_{switch})^{N-1}} \cdot \frac{1}{\sum_{k=0}^{N-1}\frac{(N-1)^k}{k!(\lambda_3(o)\tau_{switch})^k}}} -(N-1)+\frac{1}{2}+\cdots+\frac{1}{N}\right)+\tau_2.$$

Considering the assumptions of $\lambda_3(o)+\lambda_3(b)=\lambda_3 \approx \lambda_3(o)$ ($\lambda_3(o) \gg \lambda_3(b)$), we have the average memory access time for the switch-based cluster as follows:

$$T_{mem} = \tau_1 + \frac{1}{\gamma S} \left((\lambda_2+\lambda_3)\tau_2 + \lambda_4\tau_4 + \frac{(N-1)\lambda_3\tau_{switch}}{1-\frac{(N-1)^{N-1}}{(N-1)!(\lambda_3\tau_{switch})^{N-1}} \cdot \frac{1}{\sum_{k=0}^{N-1}\frac{(N-1)^k}{k!(\lambda_3\tau_{switch})^k}}} -(N-1)+\frac{1}{2}+\cdots+\frac{1}{N}\right), \tag{3.20}$$

where

$$\lambda_2 = \gamma S \int_{s_1}^{\infty} p(x)dx,$$
$$\lambda_3 = \gamma S \int_{s_2}^{\infty} p(x)dx,$$

and

$$\lambda_4 = \gamma S \int_{s_3'}^{\infty} p(x)dx,$$

3.4 A Memory Access Time Model for a Cluster of SMPs

Our target cluster of SMPs is homogeneous, where all SMPs in the cluster are identical. In such a platform, the memory hierarchy consists of the local cache, the shared memory module in each SMP, remote shared memory modules in other SMPs, and disks. Similar to a cluster of workstations, SMPs may be connected by either a bus-based network or a switch-based network. Considering the memory hierarchy, we model the average memory access time as

$$T_{mem} = t_1 + t_2 \int_{s_1}^{\infty} p(x)dx + t_3 \int_{s2}^{\infty} p(x)dx + t_4 \int_{s_3'}^{\infty} p(x)dx. \tag{3.21}$$

The access time to the local cache is a constant ($t_1 = \tau_1$). However, access times to higher levels, t_2, t_3, and t_4 will vary due to the contention occurring in the memory bus in each SMP and in the cluster network. Only remote memory access (t_3) is dependent on the type of a cluster network and the access time to the local shared-memory (t_2) and the access time to the local disk (t_4) are independent of the cluster network. In the next two subsections, we discuss the models of t_2 and t_4 and, then, t_3, first using a bus-based cluster network and then a switch-based one.

The access time to the local memory (t_2) is determined by the contention occurring in the memory bus. Two types of memory requests compete for the memory bus. One is requests from local processors inside an SMP and the other is those from remote processors in other SMPs. Assuming that the accesses from one processor to remote memory modules of other SMPs are uniformly distributed, we compute the average access rate to the memory bus from a local SMP and from remote SMPs as

$$n\lambda_2 + \frac{(N-1)n\lambda_3}{N-1} = n(\lambda_2+\lambda_3).$$

Because $\lambda_2 \gg \lambda_3$ in practice, the average access time can be approximated as:

$$t_2 = \frac{n\tau_2}{1-\frac{1}{n!(\lambda_2\tau_2)^n} \cdot \frac{1}{\sum_{k=0}^{n}\frac{1}{k!(\lambda_2\tau_2)^k}}} - \frac{1}{\lambda_2}. \tag{3.22}$$

Since disk accesses are rare compared with the cache and memory accesses, we simplify the computation of the average disk access time, t_4, by assuming that each SMP machine owns and only accesses its disks through its I/O bus. So, t_4 can be derived as

$$t_4 = \frac{n\tau_4}{1-\frac{1}{n!(\lambda_4\tau_4)^n} \cdot \frac{1}{\sum_{k=0}^{n}\frac{1}{k!(\lambda_4\tau_4)^k}}} - \frac{1}{\lambda_4}. \tag{3.23}$$

The second-level shared-memory space from each SMP's point of view consists of its own local shared-memory module and $N-1$ other shared-memory modules. Thus, the total second-level shared-memory size is defined as $s_3' = Ns_2$.

3.4.1 Bus-Based Clusters

The access to a remote shared-memory module is performed through a sequential access to the cluster bus and then to the memory bus of the remote SMP. Thus, the average access time to the remote shared-memory is the sum of the average access time of the cluster bus and the average access time to a remote memory bus. A total of $n \times N$ processors access the cluster bus at the rate of λ_3. We have the average response time at the bus as

$$\frac{nN\tau_{bus}}{1-\frac{1}{(nN)!(\lambda_3\tau_{bus})^{nN}}\cdot\frac{1}{\sum_{k=0}^{nN}\frac{1}{k!(\lambda_3\tau_{bus})^k}}}-\frac{1}{\lambda_3}.$$

Since the cluster is symmetric, the average access time from any processor to a remote memory bus equals the average local shared-memory access time (t_2). Using (3.22) and a derivation similar to (3.14), we obtain t_3 as follows:

$$\begin{aligned}t_3 &= \frac{n\tau_2}{1-\frac{1}{n!(\lambda_2\tau_2)^n}\cdot\frac{1}{\sum_{k=0}^{n}\frac{1}{k!(\lambda_2\tau_2)^k}}}-\frac{1}{\lambda_2}\\ &\quad+\frac{nN\tau_{bus}}{1-\frac{1}{(nN)!(\lambda_3\tau_{bus})^{nN}}\cdot\frac{1}{\sum_{k=0}^{nN}\frac{1}{k!(\lambda_3\tau_{bus})^k}}}-\frac{1}{\lambda_3}\\ &\quad+\frac{1}{\lambda_3}\left(\frac{1}{2}+\cdots+\frac{1}{nN}\right).\end{aligned} \tag{3.24}$$

Combining (3.22), (3.23), (3.24), we obtain the average memory access time of a bus-based cluster of SMPs as follows:

$$\begin{aligned}T_{mem} &= \tau_1+\frac{1}{\gamma S}\left(\left(\frac{n\tau_2}{1-\frac{1}{n!(\lambda_2\tau_2)^n}\cdot\frac{1}{\sum_{k=0}^{n}\frac{1}{k!(\lambda_2\tau_2)^k}}}-\frac{1}{\lambda_2}\right)(\lambda_2+\lambda_3)\right.\\ &\quad+\left(\frac{nN\tau_{bus}}{1-\frac{1}{(nN)!(\lambda_3\tau_{bus})^{nN}}\cdot\frac{1}{\sum_{k=0}^{nN}\frac{1}{k!(\lambda_3\tau_{bus})^k}}}-\frac{1}{\lambda_3}\right)\lambda_3\\ &\quad\left.+\left(\frac{n\tau_4}{1-\frac{1}{n!(\lambda_4\tau_4)^n}\cdot\frac{1}{\sum_{k=0}^{n}\frac{1}{k!(\lambda_4\tau_4)^k}}}-\frac{1}{\lambda_4}\right)\lambda_4+\frac{1}{2}+\cdots+\frac{1}{nN}\right),\end{aligned} \tag{3.25}$$

where

$$\lambda_2=\gamma S\int_{s_1}^{\infty}p(x)dx,$$
$$\lambda_3=\gamma S\int_{s_2}^{\infty}p(x)dx,$$

and

$$\lambda_4=\gamma S\int_{s_3'}^{\infty}p(x)dx.$$

3.4.2 Switch-Based Clusters

In a switch-based cluster, we assume the communication load is uniformly distributed among all SMPs. Thus, the average access rate from a processor to a remote switch port is $\frac{\lambda_3}{N-1}$. By considering the contention in switch ports, assuming the average service time of a switch-based cluster as τ_{switch}, and applying the model, we obtain the average access time to the switch network:

$$\frac{n(N-1)\tau_{switch}}{1-\frac{(N-1)^{n(N-1)}}{(n(N-1))!(\lambda_3\tau_{switch})^{n(N-1)}}\cdot\frac{1}{\sum_{k=0}^{n(N-1)}\frac{(N-1)^k}{k!(\lambda_3\tau_{switch})^k}}}-\frac{N-1}{\lambda_3}.$$

Combining t_2 and the above formula of the average switch network access time, we have

$$\begin{aligned}t_3 &= \frac{n\tau_2}{1-\frac{1}{n!(\lambda_2\tau_2)^n}\cdot\frac{1}{\sum_{k=0}^{n}\frac{1}{k!(\lambda_2\tau_2)^k}}}-\frac{1}{\lambda_2}\\ &\quad+\frac{n(N-1)\tau_{switch}}{1-\frac{(N-1)^{n(N-1)}}{(n(N-1))!(\lambda_3\tau_{switch})^{n(N-1)}}\cdot\frac{1}{\sum_{k=0}^{n(N-1)}\frac{(N-1)^k}{k!(\lambda_3\tau_{switch})^k}}}\\ &\quad-\frac{N-1}{\lambda_3}+\frac{1}{\lambda_3}\left(\frac{1}{2}+\cdots+\frac{1}{nN}\right).\end{aligned} \tag{3.26}$$

Combining (3.22), (3.23), (3.26), we obtain the average memory access time for a switch-based cluster of SMPs as follows:

$$\begin{aligned}T_{mem} &= \tau_1+\frac{1}{\gamma S}\left(\left(\frac{n\tau_2}{1-\frac{1}{n!(\lambda_2\tau_2)^n}\cdot\frac{1}{\sum_{k=0}^{n}\frac{1}{k!(\lambda_2\tau_2)^k}}}-\frac{1}{\lambda_2}\right)(\lambda_2+\lambda_3)\right.\\ &\quad+\left(\frac{n(N-1)\tau_{switch}}{1-\frac{(N-1)^{n(N-1)}}{(n(N-1))!(\lambda_3\tau_{switch})^{n(N-1)}}\cdot\frac{1}{\sum_{k=0}^{n(N-1)}\frac{(N-1)^k}{k!(\lambda_3\tau_{switch})^k}}}-\frac{N-1}{\lambda_3}\right)\lambda_3\\ &\quad\left.+\left(\frac{n\tau_4}{1-\frac{1}{n!(\lambda_4\tau_4)^n}\cdot\frac{1}{\sum_{k=0}^{n}\frac{1}{k!(\lambda_4\tau_4)^k}}}-\frac{1}{\lambda_4}\right)\lambda_4+\frac{1}{2}+\cdots+\frac{1}{nN}\right),\end{aligned} \tag{3.27}$$

where

$$\lambda_2=\gamma S\int_{s_1}^{\infty}p(x)dx,$$
$$\lambda_3=\gamma S\int_{s_2}^{\infty}p(x)dx,$$

and

$$\lambda_4=\gamma S\int_{s_3'}^{\infty}p(x)dx.$$

3.5 Summary

In summary, for given architectural and application program parameters in these three types of cluster platforms, the average memory access time, T_{mem}, can be modeled and predicted. Consequently, the average execution time per instruction, $E(Instr)$, is determined. By enumerating all practically possible ns (the numbers of processors in each machine), Ns (the number of machines in the cluster), and types of networks, we can determine an optimal cluster platform using the aid of numerical calculations for a given budget and for a given class of parallel applications.

TABLE 2
Parameters and Their Values (in Processor Clock Cycles) in the Simulators

Parameter	Value		
Basic			
One instruction execution	1		
Cache hit	1		
Cache miss to local memory	50		
I/O overhead of a memory miss to local disk[1]	2000		
SMP			
Cache miss to remote cache	15		
Cluster of workstations			
	10M Ethernet	100M Ethernet	155M ATM
Cache miss to remote home	45075	4575	3275
Cache miss to remotely cached data	90150	9150	6550
Cluster of SMPs			
	10M Ethernet	100M Ethernet	155M ATM
Cache miss to local memory	53	53	53
Cache miss to remote cache in an SMP	18	18	18
Cache miss to remote home	45078	4578	3278
Cache miss to remotely cached data	90153	9153	6553

[1] *In practice, a disk access could be easily overlapped with CPU operations. Thus, in our model, we only consider an average I/O overhead of the CPU initiation for the disk access.*

The cost models can also be used for system upgrading purposes. The model variations can be handled by including the budget increase, the parameters of the existing system, and the architectural parameters that may be changed in the upgrade. Similarly, using the models, we enumerate all practically possible changing parameters, such as the new switch networks, and additional numbers of workstations or SMPs. We then determine an optimal upgrading plan of an existing cluster system for a given budget increase.

4 Model Verification by Simulations

We verify the accuracy of the model in this section by comparing the model results with the simulation results.

4.1 Simulators

We used MINT [25] (Mips INTerpretor) as our simulation tool since our interest is primarily in the memory hierarchies of clusters. MINT provides a program-driven simulation environment that emulates multiprocessing execution environments and generates memory reference events which drive a memory system simulator, called the Back-end. We developed five memory hierarchal system simulators, which correspond to the five parallel platforms, to serve as the back-ends of MINT. The simulators were run on an SGI workstation. By varying the configuration parameters, such as the sizes of each level of memory hierarchy, we obtained the simulated execution time for a given application.

The simulated memory hierarchy of parallel systems is the one discussed in Section 2. For an SMP, we assume that a snooping-based protocol is used to maintain the cache coherence. In detail, the cache line size is 64 bytes, the cache is two-way set-associative, and the replacement policy is least-recently-used (LRU). The write invalidation protocol is used as the cache coherence protocol. Two- and four-processor SMPs are simulated because these configurations are used by most SMPs available in the market. Disks are employed as the backup storage.

For a cluster of workstations, a directory-based protocol is employed. The block size is 256 bytes. Each block of the memory has three states: shared, uncached, and exclusive. The states are identical to those in the snooping protocol. The state transition and the transactions are also similar to the snooping protocol, with explicit invalidate and write-back requests replacing the write misses that are formerly broadcast on the bus.

In a cluster of SMPs, the shared memory consists of two parts, the local memory shared by multiple processors of an SMP and the remote memory of other SMPs. To maintain the cache coherence in such a system, we applied a hybrid protocol. A directory-based protocol is used to maintain coherence between SMPs and a snooping protocol is employed to keep the caches in an SMP coherent. We extend the directory in each node (SMP) to include the processor id. The directory entries are shared by the two protocols.

The principal architecture parameters we used in the simulators are given in Table 2. They are represented in the unit of cycles, and are consistent with the values given in [10], [11], and [16].

TABLE 3
Characteristics and Parameters of the Four Programs (the Unit of the Number of Instructions Is Million)

Program	Problem size	Total Instr. $(m+M)$	Read/Write (M)	α	β	γ
FFT	64K points	34.79	6.95	1.21	103.26	0.20
LU	512×512 matrix	494.05	152.00	1.30	90.27	0.31
Radix	1M integers, radix 1024	50.99	19.09	1.14	120.84	0.37
EDGE	128×128 bitmap	88.75	39.84	1.71	85.03	0.45

4.2 Applications

We used three SPLASH-2 computational kernels [27] and one edge detection program as our applications. They are *FFT*, *LU*, *Radix*, and *EDGE*. We selected them as our benchmarks because the SPLASH-2's three kernels are representative components of a variety of computations in scientific and engineering computing and EDGE is a real-world application which detects edges from an image map.

- The **FFT** kernel is a complex 1D six-step FFT algorithm. The data consist of some complex data points to be transformed and another set of data points used as the roots of unity. Both sets of data are partitioned into submatrices so that each processor is assigned a contiguous subset of data which are allocated in its local memory.
- The **LU** kernel factors a dense matrix into the product of a lower triangular and an upper triangular matrices. The dense matrix is divided into blocks and the blocks are assigned to processors using a 2D scatter decomposition to exploit temporal and spatial locality.
- The **Radix** sort kernel sorts integers based on a method proposed by [2]. The algorithm is iterative, performing one iteration for each radix r digit of the keys.
- **Edge detection** (EDGE): The edge detection program we used is based on Bergholm's *edge focusing* algorithm [3]. (Its parallel version is presented in [28].) This program combines high positional accuracy with good noise reduction. The algorithm iterates over four steps: 1) blurring, 2) registering, 3) matching, and 4) repeating or halting. A basic operation in the edge focusing algorithm is image blurring. Let $f(i,j)$ denote the gray level image and $g(i,j)$ the blurred image—then the blurred image is computed from the discrete convolution:

$$g(i,j) = \frac{1}{2\pi\sigma^2} \sum_{x=-\lceil \frac{w}{2} \rceil}^{\lfloor \frac{w}{2} \rfloor} \sum_{y=-\lceil \frac{w}{2} \rceil}^{\lfloor \frac{w}{2} \rfloor} Gauss(i-x, j-y, \sigma) f(x,y),$$

where

$$Gauss(i,j,\sigma) = e^{-(i^2+j^2)/2\sigma^2}$$

is the Gaussian operator and w is the size of the convolution window. Usually, the size of a Gaussian window, w, is determined by the blurring scale: $w = \lceil 8\sigma \rceil$.

We began the edge focusing process with an initial σ_0 of 3.8 and reduced σ by 0.5 at each iteration for a total of eight iterations. The algorithm is parallelized by partitioning the image in rows among multiple processors. A barrier is performed after each iteration.

Using the MINT-based simulator, we first collected the memory access traces on one processor for the four applications. The traces were analyzed to present each program's temporal locality and to produce the stack distance curves. Using the standard least squares techniques, we fit (2.1) and (2.2) to the data and determined the values of α and β for the applications.

We first collected the values of α and β of the four applications on a one-processor system. As we know from the discussion on parallel program structures in Section 2, when an application program is symmetrically distributed and run on n processors, its maximum stack distance reduces approximately by a factor of n and the cumulative access probability at the corresponding reduced distance remains almost unchanged. Thus, if the cumulative probability function for an application running on a one-processor system is

$$P(x) = 1 - \frac{1}{(x/\beta + 1)^{\alpha-1}},$$

then the cumulative probability function for the same application running on an n-processor system can be approximated by

$$P(x) = 1 - \frac{1}{(nx/\beta + 1)^{\alpha-1}}.$$

We use the above revised formula as the approximation in the following model computation when there is more than one processor in the system. The parameter values of the four applications are listed in Table 3.

4.3 Analysis

4.3.1 SMPs

Bus-based SMPs with two or four processors are the most popular in the market. Due to the speed gap increase between the CPU execution and memory access, the maximal number of processors of SMPs is getting smaller. The cache sizes for them are usually 256 or 512 Kbytes and the main memory sizes are 64 or 128 Mbytes.[1] We selected

1. These numbers may become larger for the SMP products by the time this paper is published.

TABLE 4
Selected SMP Configurations

Name	CPU Speed (MHz)	# of CPUs	Cache (KB)	Memory (MB)
C1	200	2	256	64
C2	200	2	512	64
C3	200	2	256	128
C4	200	2	512	128
C5	200	4	256	128
C6	200	4	512	128

TABLE 5
Comparisons of Modeling and Simulated $E(Instr)$ in Nanoseconds on the SMPs

	FFT			LU			Radix			EDGE		
	Model	Sim	%	Model	Sim	%	Model	Sim	%	Model	Sim	%
C1	45.9	47.0	-2.3	23.7	22.9	+3.5	198.8	194.1	+2.4	3.19	3.11	+2.4
C2	45.8	46.6	-1.7	23.5	23.0	+2.4	198.4	195.9	+1.3	3.17	3.13	+1.6
C3	40.3	40.1	+0.2	20.0	20.2	-1.2	181.0	183.4	-2.8	3.13	3.20	-2.2
C4	40.1	41.5	-3.4	19.8	19.9	-0.6	180.7	187.6	-3.7	3.11	3.16	-1.3
C5	20.1	21.1	-4.6	10.0	9.8	+1.5	90.5	94.0	-4.3	1.57	1.55	+0.7
C6	20.1	21.1	-4.8	9.9	9.5	+4.7	90.3	95.0	-4.9	1.55	1.52	+2.5

these commonly used SMP configurations to verify the accuracy of the model. Table 4 lists these configurations.

Table 5 presents the modeling and simulated average execution time per instructions ($E(Instr)$) in nanoseconds and the differences between them in percentage compared with the simulated results. The results show that the differences between the simulated results and modeling results are very small (less than 5 percent), which means that the model is very accurate when modeling the SMPs. The difference comes from the approximation of the probability function $P(x)$ in comparison with the actual reference probabilities.

The overhead of cache coherence is another factor. We do not take into account the effect of cache coherence activities in the modeling. Modeling this process accurately is very difficult and will make the model too complicated to be used. In the simulation, we evaluated the memory bus traffic caused by the cache coherence protocol. It is 6.3 percent, 4.7 percent, 7.2 percent, and 2.1 percent of the total traffic on the bus for applications FFT, LU, Radix, and EDGE, respectively. This indicates that it affects performance slightly. That is the reason why the modeling results are still close to the simulated ones even though we do not model the memory bus traffic caused by the cache coherence activities.

4.3.2 Clusters of Workstations

The programming model of our cluster of workstations is based on a shared-memory system supported by a software distributed shared-memory layer. The memory accesses to the global shared-memory at different levels are nonuniform. For example, from a workstation viewpoint, the access latency to its local memory module is lower than the latency to a remote memory module although both memory memory modules belong to the global shared-memory.

We selected the five cluster configurations (C7 to C11) listed in Table 6. We predicted the execution times using the model and measured the execution times by the simulations. However, our experiments show that the modeling and simulation results are not as close as those on SMPs. The differences are up to 16.3 percent (this comes from the FFT, others are around 12 percent or less). After further investigation, we found that timing differences are caused by shared-memory coherence overheads. Our model does not include the coherence operations and the overhead. The coherence overhead is determined by two factors: 1) locality and data sharing patterns of application workloads and 2) the interconnection network speed between processors and the shared-memory of the cluster. The workstation cluster configurations (C7 to C11) are much more loosely coupled than the SMP configurations and cause higher coherence overhead. This is the reason why the modeling results for clusters of workstations are not as accurate as those for SMPs.

In order to compensate for the coherence overhead and to make the model more accurate for those applications with active coherence activities on clusters of workstations, a proper adjustment of the model is required. To retain the simplicity of our models, there are two ways to address this problem. The first one is to increase the average memory access time accordingly. The second one is to increase the average memory access rate. We have used the second approach.

Our modeling description indicates that we do not consider data sharing effects on locality. For example, even though the distance between two consecutive accesses to a

TABLE 6
Selected Clusters of Workstations Configurations

Name	CPU Speed (MHz)	# of machines	Cache (KB)	Memory (MB)	Network (bits/s)
C7	200	2	256	32	10M bus
C8	200	4	256	64	100M bus
C9	200	4	512	64	100M bus
C10	200	4	256	64	155M switch
C11	200	8	512	64	155M switch

TABLE 7
Comparisons of Modeling and Simulated $E(Instr)$ in Nanoseconds on Clusters of Workstations

	FFT			LU			Radix			EDGE		
	Model	Sim	%	Model	Sim	%	Model	Sim	%	Model	Sim	%
C7	1030.8	958.1	+7.6	491.4	521.4	-5.8	4725.0	4478.4	+5.5	6.60	6.96	-4.9
C8	73.1	70.3	+4.4	33.0	35.4	-8.4	355.8	339.6	+4.7	1.68	1.81	-5.8
C9	72.7	75.0	-3.5	32.4	33.7	-3.2	355.2	381.6	-6.9	1.62	1.57	+3.4
C10	120.1	127.9	-6.3	53.4	52.8	+0.9	583.7	615.1	-5.1	1.75	1.80	-1.7
C11	117.0	128.5	-8.9	51.6	50.4	+2.3	568.8	614.4	-7.4	1.02	0.96	+0.9

datum, D, is less than the cache size, the second access may still miss D if D is shared and modified by another processor between those two accesses. It needs an extra access to the low level memory hierarchy.

How many data accesses are shared data accesses is an important factor. We define a shared data access rate to quantify it:

$$\frac{number\ of\ shared\ data\ accesses}{total\ number\ of\ data\ accesses} \times 100\%.$$

Since not all shared data accesses need extra memory accesses, this rate may be used as an upper bound reference for the increase of average access rate to include the coherence overhead.

Through access trace analysis, we found that FFT has the highest shared data access rate. It ranges from 7.5 percent for two processors to 13.9 percent for eight processors. This is consistent with the results reported in [27]. By averaging several access rates and comparing the execution results, we found that we could obtain reasonably accurate modeling results by adjusting the average memory access rate by a factor of 12.4 percent. This adjustment makes the differences between modeling and simulation results less than 9 percent. Table 7 lists the execution results with such adjustments.

4.3.3 Clusters of SMPs

We used SMPs with two CPUs and four CPUs to build the cluster. The network varies from a traditional 10M bus, through a 100M bus to a 155M ATM. Table 8 lists the platforms we used in the verification.

In a way similar to the arrangement to the cluster of workstations, we adjusted the access rates to the remote memory in order to compensate for the overhead caused by coherence activities, which is not modeled in our formulas. We still adjusted the rate by a factor of 12.4 percent for each application. Table 9 presents the comparisons. The difference is within the range of 8 percent for all applications.

In summary, through verifications, we find that results of our model are reasonably close to the results from the simulation. The prediction accuracy is acceptable. We have also shown that coherence overhead is not negligible for applications having large data sharing. The advantages of using the modeling approach are obvious because the cost to do simulations is significantly higher than that of modeling. For example, the modeling computation for each of the cluster configurations took less than one second with a small memory allocation. In contrast, it usually took more than 20 minutes to conduct one simulation on a high-end workstation, let alone the time spent on developing the simulators.

5 Model Verification by Measurements

To further verify our model, we have also compared the modeling results with measurement results of commercial workload on SMPs. One important usage of SMP systems is the execution of commercial workloads, which represent one of the most rapidly growing market segments. In comparison with scientific, numeric-intensive, and engineering applications, commercial workloads contain more sophisticated system software activities. We have used our analytical model to study the SMP architectural impacts on performance of commercial workloads. We use the TPC Benchmark C (TPC-C) [23], a standard commercial workload benchmark, as the workload. Different from scientific applications, the TPC-C workload contains no barrier operations [1]. Thus, when applying the model on the TPC-C workload, we remove the items due to barrier operations $(\frac{1}{2} + \cdots + \frac{1}{n})$ in (3.16). We have evaluated the accuracy of the model using the published performance

TABLE 8
Selected Clusters of SMPs Configurations

Name	Speed (MHz)	# of CPUs	# of Machines	Cache (KB)	Memory (MB)	Network (bits/s)
C12	200	2	2	256	64	10M bus
C13	200	2	2	256	128	100M bus
C14	200	4	2	256	128	100M bus
C15	200	4	2	256	128	155M switch

TABLE 9
Comparisons of Modeling and Simulated $E(Instr)$ in Nanoseconds on the Clusters of SMPs

	FFT			LU			Radix			EDGE		
	Model	Sim	%	Model	Sim	%	Model	Sim	%	Model	Sim	%
C12	476.4	493.8	+3.7	235.8	231.0	+2.2	2263.8	2196.0	+3.1	3.24	3.18	+1.1
C13	60.6	61.8	+1.9	28.8	29.4	-0.9	290.4	286.2	+1.4	1.68	1.68	-0.4
C14	31.8	30.6	-2.6	14.4	15.0	-2.3	145.3	150.0	-3.3	0.84	0.84	+2.3
C15	26.4	25.3	-4.3	11.5	12.0	-1.7	117.0	126.7	-7.6	0.84	0.84	+1.9

results of TPC-C workload measured by hardware counters on a Pentium Pro-based SMP server.

Using the modeling average execution time per instruction of the workload, $E(Instr)$, and other SMP and TPC-C workload dependent parameters, we can further determine following important performance measures:

- Cycles Per Instruction (CPI): $CPI = E(Instr) \times CR$,
- execution time per transaction:

$$E(Tran) = E(Instr) \times I,$$

  and
- total number of transactions completed per minute ($tpmC$): $tpmC = \frac{n}{E(Tran)}$,

where CR is the CPU clock rate, I is the average number of instructions per transaction, and n is the number of processors in the SMP.

A group of computer professionals from the University of California at Berkeley and Informix Software, Inc. recently evaluated different architectural effects of a 4-processor Pentium Pro-based SMP on a TPC-C-like workload operated by an Informix database [12]. Their approach is purely experimental by using hardware counters and the measured results are considered accurate. Using their measured workload locality data and the Pentium Pro-based SMP architectural parameters, we have validated our model and found the modeling results are fairly close to the measured results of execution time and CPI. Based on the model parameter notations in Table 1, we collect both the workload and SMP parameter values from [12] to be used in our model, and list them in Table 10.

The Pentium Pro-based SMP has four 200 MHz processors where a process is limited to 2 GB of user space in the shared-memory. The average CPU execution time per instruction is 0.97 cycles. There are four levels in the memory hierarchy: L1 cache (on-chip) of 8 KB instructions and 8 KB data, 1 MB unified L2 cache (external cache), 4 GB of 4-way interleaved shared memory, and 90 Quantum 4.55 GB Ultra SCSI-3 disks. The access times to the L1 cache and L2 cache are three cycles and seven cycles, respectively. The access time to the shared memory without contention is 58 cycles. The average access time to the shared-memory with contention (t_3) will be predicted by our model, as will the average memory access time per instruction of the SMP, T_{mem}. The I/O activity was not measured for these experiments. Thus, we are unable to model the average access time per instruction to the disks, t_4.

The workload access probabilities to different levels provided by the experiments are 14.1 percent to the L2 cache and 0.7 percent to the shared-memory. The TPC-C-like workload does not change the structure of the workload program itself, but only changes the way of submitting transactions by using two client machines to simulate thousands of remote terminal emulators, generating requests with no think time between requests. Since the average number of instructions per transaction, I, of the TPC-C and the TPC-C-like are the same, we collected the measurement results of $I = 625,000$ from [1].

In the Pentium Pro, there are three parallel decoders to translate the macro-instructions into triadic μops. In each cycle, multiple μops can be executed. The access time to the L1 cache is likely to be overlapped with other operations not only by the parallel decoders, but also by other pipeline hardware support, such as stream buffers. Both experiments in [4] and [12] show the strong effectiveness of the overlapping on the Pentium Pro processor. Considering this architectural effect, we make t_1 fully overlapped with average CPU execution time per instruction $1/S$ in our model.

Applying the given P_2 and P_3 characterizing the locality of the TPC-C-like workload and the givens S, n, s_1, s_2, s_3, τ_1, τ_2, and τ_3, characterizing the Pentium Pro-based SMP, we use our model to quantitatively determine the following four variables: $E(Instr)$, t_3, λ_3, and T_{mem}. Table 11 gives comparisons between our modeling results and the

TABLE 10
The Measured Locality Data and the Pentium Pro-Based SMP Paramaters

Pentium Pro-based SMP	Parameter descriptions
$n = 4$	4 processors in the SMP.
S= 194 million instructions/second[2]	The processor speed.
$k = 4$	4 levels of the memory hierarchy.
$s_1 = 16$ KB	size of the L1 cache.
$s_2 = 1$ MB	size of the L2 cache.
$s_3 = 2$ GB	size of the shared-memory.
$s_4 = 360$ GB	size of the disks.
$\tau_1 = 3$ cycles/instruction	Access time to the L1 cache without contention.
$\tau_2 = 7$ cycles/instruction	Access time to the L2 cache without contention.
$\tau_3 = 58$ cycles/instruction	Access time to the shared-memory without contention.
$t_1 = \tau_1$	Access time to the L1 cache with contention.
$t_2 = \tau_2$	Access time to the L2 cache with contention.
t_3, to be modeled	Access time to the shared-memory with contention.
T_{mem}, to be modeled	Average memory access time in the SMP.
TPC-C-like workload	Parameter descriptions
$I = 625,000$	Average number of instructions per transaction.
$P_2 = 14.1\%$	The probability of accessing to the L2 cache.
$P_3 = 0.7\%$	The probability of accessing to the shared-memory.
$P_4 = \text{NA}$	Accessing to the disks is not applicable.

[2] *This is calculated based on the clock rate of 200 MHz and CPU execution time per instruction of 0.97 cycles.*

TABLE 11
Comparison between the Modeling and Measured Performance Results of the TPC-C-Like Workload on the Pentium Pro-Based SMP

	Modeling Results	Measured Results	Relative Errors
t_3	87 cycles	95 cycles	8.4%
CPI	2.60	2.52	3.2%

measured results on CPI, and average access time to the shared-memory.

The comparisons between the modeling and measured results show that our model is accurate for evaluating the memory hierarchy effects on the commercial workload. Compared with the measurement approach, this modeling approach has two limits. First, some complex hardware operations, such as branch prediction behavior and μops retirement behavior, are difficult to model. Second, a commercial workload involves many operating system activities which are dynamic and also difficult to model. For more precise and detailed performance information, the models need support from simulation. Dynamic workload and system effects can be obtained through simulation experiments. The combination of analytical and experimental results provides a cost-effective and reliable performance evaluation for computer systems.

2. This price is an estimated price, which is derived from the street price of September 1996.

3. The ATM price is based on FORE's FORERunnerLE155 switch and adaptor prices released in January 1998.

6 Application: Case Studies

In this section, we give three case studies on how to use the cost model to guide the design of a nearly optimal parallel platform for a specific kind of workloads (applications). The first case is for a small budget ($5,000) request, which can only financially cover a cluster of workstations, considering the current market prices for SMPs. In the second case, we consider a budget as high as $20,000. This budget provides the designer with more choices for the configuration, while making the decision harder without the support of any analysis tools. The third case study illustrates how to use the cost model to upgrade an existing system when additional budget funds are available.

Tables 12, 13,[2] and 14[3] list the prices of workstations, SMPs, and networks. They are estimated prices. The actual prices may change by the time this paper is published. Fortunately, the accuracy of the prices is not critical here. We are only using these as examples to show how to use the cost model to design a nearly optimal cluster for an application.

TABLE 12
Workstation (Pentium-Based PCs without Monitors) Prices

Name	CPU Speed (MHz)	Cache (KB)	Memory (MB)	Disk (GB)	Price ($)
WS1	200	256	32	2	1,000
WS2	200	256	64	2	1,300

TABLE 13
SMP (Pentium-Based Compaq Proliant 5000) Prices

Name	Speed (MHz)	# of CPUs	Cache (KB)	Memory (MB)	Price ($)
SMP1	200	4	512	348	10,000

6.1 Case Study 1: $5,000 budget

Based on the prices of computers and networks, we enumerate all possible clusters of workstation platforms as shown in Tables 15 and 16. Since FFT and Radix require the number of processors to be the power of 2, the possible configurations for FFT/Radix are different from those for LU/EDGE. There are five types of configurations, named D1 to D5 for LU and EDGE. Table 15 gives those configurations and the execution time per instruction, measured in nanoseconds when running LU and EDGE on each platform. Similarly, Table 16 lists those for FFT and Radix.

For these applications, the performance, denoted by $\Theta = 1/(\text{execution time})$, is different on different platforms. Specifically, we have:

$$FFT : \Theta(D7) > \Theta(D8) > \Theta(D6) > \Theta(D10) > \Theta(D9),$$

$$LU : \Theta(D3) > \Theta(D1) > \Theta(D2) > \Theta(D4) > \Theta(D5),$$

$$Radix : \Theta(D7) > \Theta(D6) > \Theta(D8) > \Theta(D10) > \Theta(D9),$$

$$EDGE : \Theta(D3) > \Theta(D1) > \Theta(D4) \geq \Theta(D5) > \Theta(D2).$$

Platform D7 yields the best performance for applications FFT and Radix. Platform D3 provides the best performance for LU and EDGE. For a budget of $5,000, if the application is of the type of FFT and Radix (having similar α, β, and γ), we will select the platform which consists of two workstations, each of which has 256KB cache and 64MB main memory, connected by a switch-based ATM network. For the LU and EDGE type applications, a nearly optimal platform would be a 100M bus-based network cluster composed of three computers with 256KB cache and with 64MB of memory.

TABLE 14
Network (ATM: FORE FORERunner LE155) Prices

Name	Port ($)	Adaptor ($)	Total ($)
10M Ethernet	30	30	60
100M Ethernet	110	50	160
155 ATM	399	176	575

In order to match the same performance with a single processor system to what the best $5,000 cluster can offer, we use the same model to make predictions. For program FFT, the single processor system should have a 1100 MHz CPU, for program LU, the CPU speed should be 670 MHz, for program Radix, 530 MHz, and for program EDGE, 510 MHz. We assume that the speeds of all other devices in the system increase accordingly so that the numbers of cycles they use are the same as those listed in Table 2.

For all four applications, increasing only the sizes of cache and memory cannot reach the performance of the best cluster. A large cache or memory can only reduce the memory or disk access times, but cannot decrease the time spent in the CPU execution. In contrast, parallel execution of an application reduces the execution time spent in each CPU.

6.2 Case Study 2: $20,000 Budget

The possible cluster computing platforms meeting the budget requirements can be classified into two major classes: workstation clusters and an SMP cluster. For the same set of applications, they yield completely different performances, as shown below (see Table 17 and Table 18):

$$FFT : \Theta(D18) > \Theta(D16) > \Theta(D17) > \Theta(D15),$$

$$LU : \Theta(D14) > \Theta(D11) > \Theta(D13) > \Theta(D12),$$

$$Radix : \Theta(D18) > \Theta(D16) > \Theta(D15) > \Theta(D17),$$

$$EDGE : \Theta(D13) > \Theta(D11) > \Theta(D12) > \Theta(D14).$$

An SMP cluster does not always offer the best performance. It is suitable for FFT, LU, and Radix, but, for EDGE, it is the worst option. An appropriately configured workstation cluster (D11) yields the best performance for EDGE.

From the above discussions, it is clear that, for a given budget, it is hard to predict which kind of cluster computing platform is good for one specific application. Our cost model gives a quantitative and easy way to determine the optimal or nearly optimal choices for the designer.

TABLE 15
Possible Configurations of Clusters of Workstations with a Budget of $5,000 for LU and EDGE and Their Modeling $E(Instr)$ in Nanoseconds

	Node	# of machines	Network	LU	EDGE
D1	WS1	3	ATM	46.6	2.34
D2	WS2	2	ATM	63.6	3.42
D3	WS2	3	100M	45.4	2.29
D4	WS1	5	10M	191.4	2.58
D5	WS2	4	10M	195.7	2.58

TABLE 16
Possible Configurations of Clusters of Workstations with a Budget of $5,000 for FFT and Radix and Their Modeling $E(Instr)$ in Nanoseconds

	Node	# of machines	Network	FFT	Radix
D6	WS1	2	ATM	151.5	723.9
D7	WS2	2	ATM	137.3	647.8
D8	WS2	2	100M	145.1	730.5
D9	WS1	4	10M	472.6	2376.8
D10	WS2	4	10M	438.4	2119.6

TABLE 17
Possible Configurations of Clusters with a Budget of $20,000 for LU and EDGE and Their Modeling $E(Instr)$ in Nanoseconds

	Node	# of machines	Network	LU	EDGE
D11	WS1	12	ATM	9.48	0.56
D12	WS2	10	ATM	9.61	0.65
D13	WS2	12	100M	9.54	0.54
D14	SMP1	2	10M	8.88	0.77

TABLE 18
Possible Configurations of Clusters with a Budget of $20,000 for FFT and Radix and Their Modeling $E(Instr)$ in Nanoseconds

	Node	# of machines	Network	FFT	Radix
D15	WS1	8	ATM	38.9	183.6
D16	WS2	8	ATM	38.4	174.2
D17	WS2	8	100M	38.6	184.3
D18	SMP1	2	10M	21.8	114.9

6.3 Case Study 3: $5,000 Budget Increase

We assume that the original parallel computing platform consists of four WS1 workstations connected by 100M fast Ethernet. The $5,000 budget increase may be used in

1. increasing the number of nodes,
2. increasing the memory size of each node,
3. replacing 100M Ethernet with a 155M ATM network, and
4. the combination of 1, 2, and 3.

Our model currently only covers homogeneous parallel platforms. This imposes some limits on how to expand the system.

Table 19 and Table 20 list three ways to expand the system for LU/EDGE and FFT/Radix, respectively. The performance of the four applications on the new platform is also given in the tables. For all applications, D20 and D23 (which simply add another four workstations of the same type) give the best performance. Comparing the upgraded configuration of D19 with the original platform, we find that, even though we only increase the number of nodes by one, the combined effect of increasing the memory sizes and reducing the remote memory access latency (through a fast network) makes the performance of application LU increase almost 50 percent. But, the effect on EDGE is not so significant. This is because the locality of EDGE is better than LU. For FFT and Radix, because of their number of processors requirements, we could only increase the size of memory and replace the network (D22). The improvement is not so significant as that of increasing the number of

TABLE 19
Increased Budget Platforms and Their Performance for LU and EDGE

	Node	# of machines	Network	LU	EDGE
Original	WS1	4	100M	39.5	1.80
D19	WS2	5	ATM	21.1	1.33
D20	WS1	8	100M	18.2	0.90
D21	WS2	6	100M	20.6	1.14

TABLE 20
Increased Budget Platforms and Their Performance for FFT and Radix

	Node	# of machines	Network	FFT	Radix
Original	WS1	4	100M	83.9	391.4
D22	WS2	4	ATM	73.8	254.9
D23	WS1	8	100M	39.2	186.5
D24	WS2	4	100M	77.6	286.3

TABLE 21
Recommendations of Cost-Effective Cluster Configurations Based on Workload Characteristizations

Workload parameters	Characteristics	Recommended clusters
small γ $\beta < 100$.	CPU-bound, good locality, example: LU.	large number of high speed workstations with a slow network.
small γ $\beta > 100$.	CPU-bound, poor locality example: FFT.	small number of high speed workstations with a fast network.
large γ, $\beta < 100$.	memory-bound, good locality, example: EDGE.	each workstation has a large memory with a slow network.
large γ, $\beta > 100$.	memory-bound poor locality, example: Radix.	a powerful SMP.
large γ, very large β.	memory-bound, very poor locality, example: TPC-C.	a powerful SMP, or a fast cluster of SMPs.

nodes only (D23). The comparison between D20 and D21 indicates that, for LU and EDGE, and for the given remote memory access latencies, the number of nodes affects the performance more significantly than the size of memory.

For the $5,000 budget increase, the best way to upgrade the system is to add four new identical nodes to the original system if the applications are of the types we discuss here.

7 Conclusions

A major objective of this paper is to find a nearly optimal cluster computing platform for a given budget and for certain types of workloads in a timely and cost-effective manner. We address the problem by proposing a performance model to quantitatively predict the average execution time per instruction based on the locality parameter values obtained by program memory access pattern analysis. By comparing the average execution time per instruction for an application on different cluster computing platforms, we determine an optimal configuration for that specific application. How to upgrade an existing cluster platform in a cost-effective way for a given budget increase can be addressed in the same model. Our analytical model is verified by simulations and measurements. We also present two case studies that use the model to effectively build a cluster for different parallel computing applications and a case study to upgrade an existing system.

Regarding the system upgrading, we have the following recommendations: First, if the applications are CPU-bound and budget is limited, money should be spent on increasing the number of nodes, which will increase the aggregate memory space and the total processing power. Second, if

applications are memory bound with poor locality, money should be first spent on increasing cache/memory capacity to reduce the network usage. If the network activities are more or less independent of the cache/memory capacity, upgrading the cluster network bandwidth should be the first priority.

7.1 Recommendations for Cluster Configurations

Our study shows that the depth of the memory hierarchy is the most sensitive factor for minimizing the execution time for many applications. This factor is playing a more important role as the speed gap between processors and memory hierarchy access continues to widen and as the memory hierarchy depth continues to increase. However, the interconnection network cost of a tightly coupled system with fewer levels of memory hierarchy, such as an SMP, is significantly higher than a normal cluster network connecting independent computer nodes. The essential issue to be considered is the trade-off between the distance of memory hierarchy and system cost. Table 21 gives principles we obtained from the study and recommendations in building a cost-effective cluster system.

7.2 Limitations of This Study

Modern processors exploit instruction level parallelism (ILP) by performing out-of-order executions of instructions. With the ILP processor technology, CPU executions and memory references can be overlapped. The overlapping degree depends on the length of the processor pipeline and other architectural designs, and instruction dependencies of application programs. The average execution time model for an instruction, (3.4), still holds in principle for ILP processors. However, the memory access rate increases for a given application program on an ILP processor and memory reference times can be partially overlapped with CPU operations. Thus, (3.5) should be adjusted to reflect the overlapping, as we did in the TPC-C case study. The formula used for calculating the access rate, λ, (for example, those in (3.16)) also need some revisions to reflect concurrent executions of instructions.

Modeling ILP processor behavior is much more complex than modeling processors with in-order executions. One way to characterize the dynamic ILP behavior using our model is to introduce a new set of weight parameters, which quantifies the ratio between the number of concurrent instructions and the total number of instructions in a workload and quantifies the overlapping between CPU operations and memory accesses. Our model needs some revisions to use these parameters. The accuracy of the revised model is determined by the accuracy of weight parameters which are ILP processor architecture- and application workload-dependent.

7.3 Future Work

We are currently working on four supporting tools and integrating them together:

1. an efficient tool to collect application program memory access traces,
2. a trace analysis tool to compute the application parameters α, β, and γ,
3. a trace tool to measure and quantify ILP overlapping parameters, and
4. a tool to support the generation of all possible cluster configurations meeting the budget requirements.

We believe software that integrates these tools will provide a timely and effective vehicle to support the design of cost effective parallel cluster computing.

Our model currently only covers homogeneous cluster platforms. The model can be extended to evaluate heterogeneous platforms by making variables associate with each individual node. The memory access rate in each node is no longer a constant, which makes the modeling work more complex. We will address this difficult issue by developing approximation methods and by collecting more dynamic data in experiments.

Acknowledgments

We appreciate the discussions with J. Ding of Intel on the Pentium processor architecture. Our colleagues at the Wharton Business School, University of Pennsylvania, and at the Engineering School of MIT provided constructive suggestions when this research was presented to them. We are thankful to Stefan Kubricht and N. Wagner for reading the paper and for their comments. Finally, we wish to thank the anonymous referees for their constructive and insightful comments. The report from Referee B was particularly helpful for us to improve the quality and readability of the paper. This work is supported in part by the US National Science Foundation under grants CCR-9400719, CCR-9812187, and EIA-9977030, by the US Air Force Office of Scientific Research under grant AFOSR-95-1-0215, and by Sun Microsystems under grant EDUE-NAFO-980405.

References

[1] G.A. Abandah and E.S. Davidson, "Configuration Independent Analysis for Characterizing Shared-Memory Applications," *Proc. 12th Int'l Parallel Processing Symp.*, pp. 485-491, Apr. 1998.
[2] D. Bailey et al., "The NAS Parallel Benchmarks," *Int'l J. Supercomputing Applications*, vol. 5, no. 3, pp. 63-73, Fall 1991.
[3] F. Bergholm, "Edge Focusing," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 9, no. 6, pp. 726-741, June 1987.
[4] L. Bhandarkar and J. Ding, "Performance Characterization of the Pentium Pro Processor," *Proc. Third High Performance Computer Architecture*, pp. 288-297, Feb. 1997.
[5] C.K. Chow, "On Optimization of Storage Hierarchies" *IBM J. Research and Development*, pp. 194-203, May 1974.
[6] E.G. Coffman and P.J. Denning, *Operating System Theory*. Englewood Cliffs, N.J.: Prentice Hall, 1973.
[7] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "Architectural Requirements of Parallel Scientific Applications with Explicit Communication," *Proc. 20th Ann. Int'l Symp. Computer Architecture*, pp. 2-13, May 1993.
[8] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart, *LINPACK User's Guide*. Philadelphia: SIAM, Philadelphia, 1979.
[9] X. Du and X. Zhang, "Performance Models and Simulation," *High Performance Cluster Computing*, R. Buyya, ed., vol. 1, chapter 1, pp. 135-153. Upper Saddle River, N.J.: Prentice Hall, 1999.
[10] M. Heinrich et al., "The Performance Impact of Flexibility in the Stanford FLASH Multiprocessor," *Proc. Sixth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 274-285, 1994.
[11] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, second ed. San Francisco: Morgan Kaufmann, 1996.

[12] K. Keeton et al., "Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads," *Proc. 25th Int'l Symp. Computer Architecture,* pp. 15-26, May 1998.
[13] K. Kennedy, C.F. Bender, J.W.D. Connolly, J.L. Hennessy, M.K. Vernon, and L. Smarr, "A National Parallel Computing Environment," *Comm. ACM,* vol. 40, no. 11, pp. 63-72, 1997.
[14] High Performance Fortran Forum, *High Performance Fortran Language Specification Version 1.0, Draft,* Jan. 1993.
[15] B.L. Jacob, P.M. Chen, S.R. Silverman, and T.N. Mudge, "An Analytical Model for Designing Memory Hierarchies," *IEEE Trans. Computers,* vol. 45, no. 10, pp. 1,180-1,194, 1996.
[16] L. McVoy and C. Staelin, "Inbench: Portable Tools for Performance Analysis," *Proc. 1996 USENIX Technical Conf.,* pp. 279-295, Jan. 1996.
[17] S.M. Ross, *Introduction to Probability Models,* sixth ed. San Diego: Academic Press, 1997.
[18] R. Samanta et al., "Home-Based SVM Protocols for SMP Clusters: Design, Simulation, Implementation and Performance," *Proc. Fourth Int'l Symp. High Performance Computer Architecture,* Feb. 1998.
[19] J.P. Singh, W.-D. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared Memory," *Computer Architecture News,* vol. 20, no. 1, pp. 5-44, Mar. 1992.
[20] A.J. Smith, "Cache Memories," *Computing Surveys,* vol. 14, no. 3, pp. 473-530, 1982.
[21] R. Stets et al., "CASHMERE-2L: Software Coherent Shared Memory on a Clustered Remote-Write Network," *Proc. 16th ACM Symp. Operating Systems Principles,* Oct. 1997.
[22] H.S. Stone, *High Performance Computer Architecture.* Addison-Wesley, 1993.
[23] Transaction Processing Performance Council, *TPC Benchmark C, TPC Benchmark C Standard Specification,* Revision 3.3.3, Apr. 1998.
[24] K.S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications.* Englewood Cliffs, N.J.: Prentice Hall, 1982.
[25] J.E. Veenstra and R.J. Fowler, "MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors," *Proc. Second Int'l Workshop Modeling, Analysis, and Simulation of Computer and Telecomm. Systems,* pp. 201-207, 1994.
[26] T.A. Welch, "Memory Hierarchy Configuration Analysis," *IEEE Trans. Computers,* vol. 27, no. 5, pp. 408-415, May 1978.
[27] S.C. Woo et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proc. 22nd Ann. Int'l Symp. Computer Architecture,* pp. 24-36, June 1995.
[28] X. Zhang, S.G. Dykes, and H. Deng, "Distributed Edge Detection: Issues and Implementations," *IEEE Computational Science & Engineering,* pp. 72-82, Spring 1997.
[29] Y. Zhou et al., "Relaxed Consistency and Coherence Granularity in DSM Systems: A Performance Evaluation," *Proc. Sixth ACM Symp. Principles and Practice of Parallel Programming,* June 1997.

Xing Du received his BS and PhD degrees in computer science from Nanjing University, China, in 1986 and 1991, respectively. He has been a senior member of the technical staff at Oracle Corporation since 1999. He was a research associate at the University of Texas at San Antonio and at the College of William and Mary between 1995 to 1998. He worked as a research scientist at the University of Virginia between 1998 to 1999. His research interests are parallel/distributed systems and software engineering. He is a member of the IEEE Computer Society.

Xiaodong Zhang received his BS in electrical engineering from Beijing Polytechnic University, China, in 1982, and his MS and PhD degrees in computer science from the University of Colorado at Boulder, in 1985 and 1989, respectively. He is a professor of computer science at the College of William and Mary. His research interests are parallel and distributed systems, computer memory systems, and scientific computing. He is an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and has chaired the IEEE Computer Society Technical Committee on Supercomputing Applications. He is a senior member of the IEEE.

Zhichun Zhu is a PhD candidate in computer science at the College of William and Mary. She received her BS degree in computer science from Huazhong University of Science and Technology, China, in 1992. Her research interests are computer architecture and parallel systems. She is a student member of the ACM.