The Impact of Memory Hierarchies on Cluster Computing *

XING DU[1] XIAODONG ZHANG[2]

[1]Department of Computer Science
University of Virginia
Charlottesville, VA 22903-2242

[2]Department of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795

Abstract

Using off-the-shelf commodity workstations and PCs to build a cluster for parallel computing has become a common practice. A choice of a cost-effective cluster computing platform for a given budget and for certain types of application workloads is mainly determined by its memory hierarchy and interconnection network of the cluster. Finding such a solution from exhaustive simulations would be highly time-consuming and expensive; and predictions from measurements on existing clusters would be impractical. We present an analytical model for evaluating 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 modeling and architectural parameters. Network variations covering bus and switch networks are also included in the analysis. Applications of different types are characterized by parameterized workloads with different computation and communication requirements. The model has been validated by simulations. Our study shows that the length of memory hierarchy is the most sensitive factor to affect the execution time for many types of workloads. However, the interconnection network cost of a tightly coupled system with a short length of 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 length of memory hierarchy and system cost. Based on analytical and case studies, we present our quantitative recommendations for building cost-effective clusters for different application workloads.

*This work is supported in part by the National Science Foundation under grants CCR-9400719 and CCR-9812187, by the Air Force Office of Scientific Research under grant AFOSR-95-1-0215, and by Sun Microsystems under grant EDUE-NAFO-980405.

1 Introduction

With rapid development and advances of commodity processors and networking technology, parallel computing platforms are shifting from expensive customer-designed MPPs to cheap and commodity-designed symmetric multiprocessors (SMPs) and clusters of workstations, of 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". Compared with simply buying an expensive MPP box, the cluster approach is highly flexible for users to construct, upgrade and scale a parallel system. However, the flexibility also provides multiple system configuration options for a given budget and a given type of workload. This raises performance optimization issues which need addressing. We believe the following two questions are fundamental for 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 few optimization solutions to help users construct clusters in a cost-effective way. Solutions from exhaustive simulations would be highly time-consuming and expensive; and predictions from measurements on existing clusters would be impractical.

In this paper, we present an analytical model to address the above two questions. The model covers platforms of a single SMP, a cluster of workstations/PCs, or a cluster of SMPs by changing various modeling and architectural parameters. Network models covering two representative networks are also included in the analysis. Applications of different types are characterized by parameterized workloads with different computation and communication requirements. The model is based on the computation 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 application workload.

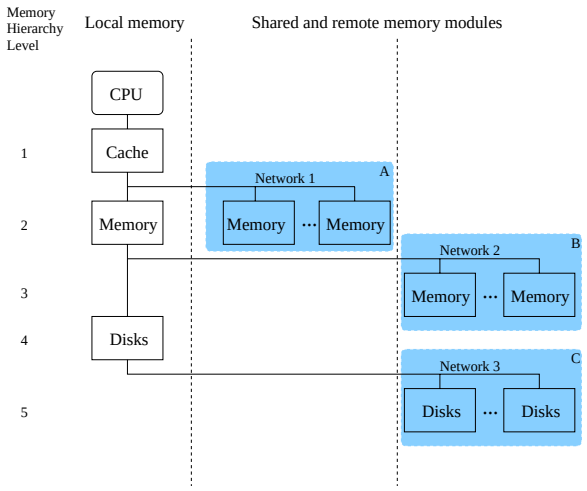

Figure 1: The memory hierarchy from the standpoint of one processor.

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 validated by constructing a set of simulators to simulate different types of clusters, and by comparing the modeled results with the simulated ones. The comparison indicates that for a single SMP, the modeling results are fairly close to the simulated results (the difference is below 5%). For clusters of workstation and clusters of SMPs, after properly adjusting the access rates to remote memory, we are able to obtain modeling results fairly close to the simulated results (less than 10%) 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. Finally we present results and implications of several case studies which use the model to effectively build a cluster and upgrade an existing cluster for different parallel computing applications.

2 Memory Hierarchy Abstraction

Our target cluster consists of the following computing components which form a standard parallel computing platform:

- a single SMP,
- multiple workstations or PCs, or
- multiple SMPs.

Because the capability and technology of workstations and high-end PCs are merging, we use the term “workstation” to refer to both types in the reminder of the paper.

The memory hierarchy of a computing platform plays an important role in determining the performance of an application program running on it. From an architecture point of view, the way a parallel process accesses the memory hierarchy is shown in Figure 1. There are five memory access levels for a processor in a cluster: 1) access to its own cache, 2) access to its own memory or the shared-memory associated with all the processors in an SMP, 3) access to a remote memory module associated with another machine in the cluster, 4) access to its own disk, and 5) access to a remote disk associated with another machine in the cluster. In general, access latencies to a lower level are smaller than the ones to a higher level. The access latency to any memory component at the same level is considered to be identical. Each machine in the cluster has its own cache, memory, and disks.

The memory hierarchy in Figure 1 covers three targeted parallel computing platforms. 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 (see the gray block A in Figure 1). These memory modules can also be viewed as a single shared-memory pool by all the processors in the SMP. Network 1 is 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 (see the gray block B in Figure 1). Network 2 is the cluster network, an important part of off-the-shelf hardware. Two representative types are bus-based and switch-based networks. Remote disks can also be shared through Network 3, which in most cases use the same physical networks used for Network 2 (see the gray block C in Figure 1). Their access latencies are higher than those to local disks which are performed through I/O buses. However, whether the access to a remote memory module has a lower latency than an access to the local disk or not is determined by the speed of Network 2. With the advances of low-latency, high bandwidth networks, it is more likely that the access to a remote memory has a relatively lower access latency. For a cluster of SMPs, uniform memory accesses are performed at level 2, and non-uniform memory accesses are performed at level 3. Remote disks may also be used. Table 1 classifies the three parallel systems by the cluster memory hierarchy of Figure 1.

Parallel systems	Additional memory levels
a single SMP	gray block A
a cluster of workstations	gray blocks B and C
a cluster of SMPs	gray blocks A, B, and C

Table 1: Classifying the three parallel systems by the cluster memory hierarchy.

A single memory image is provided either by hardware like SMPs or by a software layer across Network 2 in Figure 1 to form a CC-NUMA shared-memory system. A small scale SMP usually consists of 2 to 4 processors connected by a bus (Network 1 in Figure 1). We consider such SMPs in this study. For a cluster of workstations or SMPs, Networks 2 and 3 in Figure 1 often use the same physical networks. We discuss two types of commonly used cluster networks in this paper, the bus-based (e.g., Ethernet, Fast Ethernet) and the switch-based (e.g., an ATM switch).

3 Parallel Program Characterizations

We consider the single-program multiple-data (SPMD) programming model, where a parallel application consists of one process per processor on a fixed number of processors throughout the execution. For each processor, it executes the same program but operates on different data sets. Each process is intended to run equal weight computation tasks for load balancing. The structure of applications is bulky-synchronous, where phases of purely local computation alternate with phases of interprocessor communications and synchronizations.

Our execution model of cluster computing is based on the probabilities of references to different levels of the memory hierarchy in Figure 1. The probability is determined based on *stack distance curves* taken directly from an address stream [2].

The work in [6] 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 by $P(x)$, which represents the probability of references within a given stack distance of x. This fits an LRU-managed and fully-associative cache hitting rate well if x is considered as the cache size. The probability density function, denoted by $p(x)$, describes the frequency of references at stack distance x. Similar to other related work [6, 8], we model $P(x)$ in the form of

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

thus $p(x)$ in the form of

$$p(x) = \frac{\beta^{\alpha-1}(\alpha - 1)}{(x + \beta)^{\alpha}}, \tag{2}$$

where α (> 1) and β (> 1) are workload parameters to characterize 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 at different access speed. 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 any 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 memory references. 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.

4 Memory Access Time Prediction

The parameters used in our model are divided into three groups: architecture parameters, such as the number of machines in a cluster (N), the number of processors in a machine (n), the processor speed by the number of instructions per second (S), the length (or the number of levels) of the memory hierarchy (k), and the memory size in each level of the hierarchy, (s_i, $i = 1, ..., k$); program parameters, such as the total number of instructions with or without memory accesses (M or m) in a program, the access rate to the ith level hierarchy (λ_i) and the probability to access the i level hierarchy (P_i); and budget/cost parameters, such as the total budget for building a cluster (B), the budget increase for upgrading a cluster (B') and of costs for different system components.

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 communications (instructions without memory accesses) and the computation time with network communications (instructions with memory accesses):

$$\begin{aligned} E(App) &= \frac{m}{nN}\frac{1}{S} + \frac{M}{nN}(\frac{1}{S} + T) \\ &= \frac{1}{nN}(\frac{m + M}{S} + MT), \end{aligned} \tag{3}$$

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 processors in a machine, N is the number of machines in the cluster, S is the processor speed as the number of instructions per second, and T 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

$$E(Instr) = \frac{E(App)}{m + M} = \frac{1}{nN}(\frac{1}{S} + \gamma T), \tag{4}$$

where $Instr$ represents a program instruction, and $\gamma = \frac{M}{m+M}$.

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

$$C_{cluster} = NC_{machine}(n) + NC_{net}, \tag{5}$$

where $C_{machine}(n)$ 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 (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:

$$\begin{cases} minimize\ E(Instr) \\ subject\ to\ C_{cluster} \leq B. \end{cases} \tag{6}$$

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 (4).

For given architecture and program parameters, the execution performance in (3) and (4) can be determined if the average memory access time T is known. Therefore, T is the key variable to be modeled in this study. We adopt the same model as the one used in [6], in computing the average memory access time T:

$$\begin{aligned} T &= \sum_{i=1}^{k} P_i t_i \\ &= t_1 + t_2 \int_{s_1}^{\infty} p(x)dx + ... + t_k \int_{s_{k-1}}^{\infty} p(x)dx, \quad (7) \end{aligned}$$

where P_i and t_i are the access probability and the average access time respectively to the memory hierarchy at the ith level, $i = 1, ..., k$. 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.

In an SMP, each processor has its own cache. Since the cache is dedicated to each processor, the average access time to it from its own processor (t_1) is a constant, and equals to the cache access time without contention (τ_1). The average access time to the main memory (t_2) is determined by the accesses to ordinary variables and accesses to variables used for barriers, and is modeled as:

$$t_2 = P_o t_2(o) + P_b t_2(b), \quad (8)$$

where P_o and P_b are the access probabilities to ordinary and barrier variables, and $t_2(o)$ and $t_2(b)$ are the average access times to ordinary and barrier variables, respectively. Assume the access rate to the memory for ordinary variables from a processor is $\lambda_2(o)$. The access time to the memory without contention is a constant (τ_2). The accesses to the memory by n processors can be modeled as a memoryless, general, and one-server (M/G/1) queue [7], and the average access time is approximated as:

$$t_2(o) = \frac{\tau_2 - \frac{1}{2}(n-1)\lambda_2(o)\tau_2^2}{1-(n-1)\lambda_2(o)\tau_2}.$$

We denote the barrier time in process i as X_i, and X as the barrier cycle time of the whole system. We have $X = max\{X_1, X_2, ..., X_n\}$. Using Order Statistics [7, 9], 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 barrier variables. Thus, 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

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

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

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

Substituting the above expressions into (8), we obtain:

$$\begin{aligned} t_2 &= \frac{1}{\lambda_2(o)+\lambda_2(b)} \times \\ &\quad (\frac{\lambda_2(o)\tau_2 - \frac{1}{2}(n-1)\lambda_2(o)^2\tau_2^2}{1-(n-1)\lambda_2(o)\tau_2} + (\frac{1}{2} + . + \frac{1}{n})) \quad (9) \end{aligned}$$

A uniprocessor system is a special case of an SMP. If we substitute $n = 1$ into (9), we obtain a result identical to the one presented in [6] for a uniprocessor system.

Similarly, if n processors share disks through an I/O bus, the average access time to the disks can be modeled as:

$$t_3 = \frac{\tau_3 - \frac{1}{2}(n-1)\lambda_3\tau_3^2}{1-(n-1)\lambda_3\tau_3}, \quad (10)$$

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.

Since $\lambda_2(o) \gg \lambda_2(b)$, we have $\lambda_2 = \lambda_2(o)+\lambda_2(b) \approx \lambda_2(o)$. Substitute (9) and (10) into (7), and transform it, we obtain the average memory access time T for an SMP:

$$\begin{aligned} T &= t_1 + t_2 \int_{s_1}^{\infty} p(x)dx + t_3 \int_{s_2}^{\infty} p(x)dx \\ &= \tau_1 + \frac{1}{\gamma S}(\frac{\lambda_2\tau_2 - \frac{1}{2}(n-1)\lambda_2^2\tau_2^2}{1-(n-1)\lambda_2\tau_2} + \\ &\quad \frac{\lambda_3\tau_3 - \frac{1}{2}(n-1)\lambda_3^2\tau_3^2}{1-(n-1)\lambda_3\tau_3} + \frac{1}{2} + \frac{1}{3} + .. + \frac{1}{n}), \quad (11) \end{aligned}$$

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 similarly,

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

The derivations of various timing models of T for the cluster of workstations connected either by bus-based networks or switch-based networks, and the cluster of SMPs connected by bus-based or switch-based networks are presented in [3] due to the page limit.

Because our target solution variables are integer types in (5) and (6), the optimization 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 and solve the optimization problem by enumerating solutions, and choosing the best as the optimal solution. The quality of our predicted solutions are determined by the correctness and the accuracy of the model in predicting the average

memory access time, T for each of the three cluster platforms. For given architectural and application program parameters, the average memory access times, T, in the three types of cluster platforms, can be modeled and predicted. Consequently, the average execution time per instruction, $E(Instr)$ is determined. By enumerating all practically possible n's, the numbers of processors in each machine, N's, the number of machines in the cluster, and types of networks with the aid of numerical calculations, we can determine an optimal cluster platform for a given budget, and for a given class of parallel applications.

5 Simulation Experiments

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

5.1 Simulators

We used MINT [10] (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 hierarchical memory system simulators, which correspond to the five parallel platforms using bus-based and switch-based networks, 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 times for a given application.

The simulated memory hierarchy of a parallel system 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 many 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, but 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 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 among 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 as follows. (They are represented in the unit of cycles, and are consistent with the values given in [4] and [5]).

- **The basic parameters**
  - One instruction execution: 1.
  - Cache hit: 1.
  - Cache miss to local memory: 50.
  - Memory miss to local disk: 2000.
- **The SMP system parameters**
  - Cache miss to remote cache: 15.
- **The cluster of workstations parameters**
  - Cache miss to a remote node: 45075 (10 Mb Ethernet), 4575 (100Mb Ethernet), and 3275 (155 Mb ATM switch).
  - Cache miss to remotely cached data: 90150 (10 Mb Ethernet), 9150 (100Mb Ethernet), and 6550 (155 Mb ATM switch).
- **The cluster of SMPs parameters**
  - Cache miss to local memory: 50.
  - Cache miss to remote cache within an SMP: 15.
  - Cache miss to a remote node: 45078 (10 Mb Ethernet), 4578 (100 Mb Ethernet), and 3278 (155 Mb ATM switch).
  - Cache miss to remotely cached data: 90153 (10 Mb Ethernet), 9153 (100 Mb Ethernet), and 6553 (155 Mb ATM switch).

5.2 Applications

We used three SPLASH-2 computational kernels [11] and one edge detection program [12] as our applications. They are *FFT*, *LU*, *Radix*, and *EDGE*. We selected them as our benchmarks because the three SPLASH-2 kernels are representative components of a variety of computations in scientific and engineering computing, while EDGE is a real-world application which detects edges from an image map.

- The **FFT** kernel is a complex 1-D 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 matrix. The dense matrix is divided into blocks and the blocks are assigned to processors using a 2-D scatter decomposition to exploit temporal and spatial locality.
- The **Radix** sort kernel sorts integers based on a method proposed in [1]. The algorithm is iterative, performing one iteration for each radix r digit of the keys.

- **Edge detection** (EDGE): The parallel edge detection program we used is from [12]. This program combines high positional accuracy with good noise reduction. The algorithm iterates over four steps: (1) blurring (2) registering (3) matching (4) repeating or halting. The algorithm is parallelized by partitioning the image in rows among multiple processors. A barrier is performed after each iteration.

Using the simulators, 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 equations (1) and (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 3, 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 in the form of (1), then the cumulative probability function for the same application running on a 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 2.

The value of γ gives the ratio between the number of instructions involving memory accesses and the total number of instructions of a program. This ratio provides information of memory access frequency, but not of locality, which is determined by α and β. A program with good locality accesses data in a relatively short distance of the memory hierarchy. This good locality feature is characterized mainly by a low value of β (increasing α could also improve the locality). For example, program EDGE has the highest memory access frequency ($\gamma = 0.45$), but it has the best program locality considering its highest α value (1.71) and lowest β value (85.03). On the other hand, program Radix has the worst program locality among the 4 programs because it has the lowest α value (1.14) and the highest β value (120.84). In a separate study, we have also examined the program locality of the TPC-C commercial workload with a relatively small scale data set on an SMP multiprocessor, and found that the value of β is over 10 times higher than that of any scientific programs we presented in this paper: $\alpha = 1.73$, $\beta = 1222.66$ and $\gamma = 0.36$. The β value continues to increase as the size of the workload data set increases.

5.3 Model Validation

5.3.1 SMPs

Bus-based SMPs with 2 or 4 processors are very popular in the market. Due to the speed gap increase between the CPU and the memory access, the maximal number of processors of an SMP is getting smaller. The cache sizes for them are usually 256 or 512 Kbytes, and the main memory sizes are 64 or 128 Mbytes. We selected these commonly used SMP configurations to verify the accuracy of the model. Table 3 lists these configurations.

Program	Problem size	α	β	γ
FFT	64K points	1.21	103.26	0.20
LU	512×512 matrix	1.30	90.27	0.31
Radix	1M integers, 1024	1.14	120.84	0.37
EDGE	128×128 bitmap	1.71	85.03	0.45

Table 2: Characteristics of the 4 programs.

Name	n	Cache	Memory
C1	2	256KB	64MB
C2	2	512KB	64MB
C3	2	256KB	128MB
C4	2	512KB	128MB
C5	4	256KB	128MB
C6	4	512KB	128MB

Table 3: Selected SMPs (the CPU speed is 200 MHz).

Figure 2 presents the modeled, simulated average execution time per instruction ($E(Instr)$). The time unit is 10^{-8}, 10^{-8}, 10^{-7}, and 10^{-9} second for FFT, LU, Radix, and EDGE respectively. The results show that the differences between the simulated results and modeled results are very small (less than 5%), which means that the model is sufficiently 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 is very difficult and will make the model too complicated to use. In the simulation, we evaluated the memory bus traffic caused by the cache coherence protocol. It is 6.3%, 4.7%, 7.2%, and 2.1% of the total traffic on the bus for applications FFT, LU, Radix, and EDGE, respectively. It indicates that it only 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 cache coherence activities.

5.3.2 Clusters of workstations

The cluster of workstations is constructed as a shared-memory system. The shared memory image is assumed to be supported by a software layer. Compared with SMPs, the cluster of workstations has an additional memory level above the shared memory, the local memory. The local memory absorbs most of the

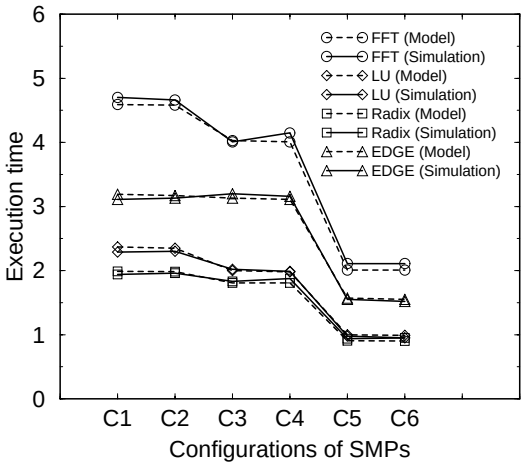

Figure 2: Comparisons of modeled and simulated $E(Instr)$ on SMPs.

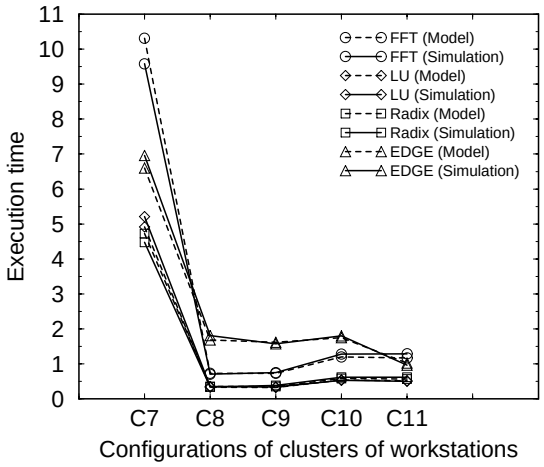

Figure 3: Comparisons of modeled and simulated $E(Instr)$ on clusters of workstations.

references to the higher level (remote memory). So the overall memory traffic on the cluster bus/switch is relatively low. But since the shared memory is composed of the local memory of all workstations, the shared memory coherence overhead is not reduced. In addition, larger block sizes and longer shared memory access latencies incur additional cost. All make the coherence overhead effect on the performance more significant than those in SMPs. This overhead must be considered in the performance evaluation.

Name	N	Cache	Memory	Network
C7	2	256KB	32MB	10Mb bus
C8	4	256KB	64MB	100Mb bus
C9	4	512KB	64MB	100Mb bus
C10	4	256KB	64MB	155Mb switch
C11	8	512KB	64MB	155Mb switch

Table 4: Selected clusters of workstations (the CPU speed is 200 MHz).

There are two general ways to address this problem by adjusting the model parameters: (1) adjusting the average access time to the remote memory, and (2) adjusting the average access rate to the remote memory. The second approach seems more reasonable in our scenario, and we used it.

We selected five cluster platforms (C7 to C11) as shown in Table 4; we then modeled, and simulated the execution of the four applications on them. Through experiments, we find that by adjusting the average remote memory access rate by a factor of 12.4%, the differences between modeled results and simulated results for all applications are below 10%. Figure 3 presents the results with such adjustments. The time unit is 10^{-7}, 10^{-7}, 10^{-6}, and 10^{-9} second for FFT, LU, Radix, and EDGE respectively.

5.3.3 Clusters of SMPs

We used SMPs with two CPUs and four CPUs to build the cluster. The network varies from a traditional 10Mb bus, through a 100Mb bus to a 155M switch ATM. Table 5 lists the platforms we used in the validation.

Name	n	N	Cache	Memory	Network
C12	2	2	256KB	64MB	10Mb bus
C13	2	2	256KB	128MB	100Mb bus
C14	4	2	256KB	128MB	100Mb bus
C15	4	2	256KB	128MB	155Mb switch

Table 5: Configurations of selected clusters of SMPs (the CPU speed is 200 MHz).

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 are not modeled in our formulas. We still adjusted the rate by a factor of 12.4% for each application. Figure 4 presents the comparisons. The time unit is the same as that in Figure 3. The difference is within the range of 8% for all applications.

In summary, through validation, we find that the modeled system is sufficiently close to the simulated system. For the cluster of workstations and cluster of SMPs, an adjustment of 12.4% of the access rates to the remote memory makes the difference between the modeled results and simulated results reduce to less than 10%, which is acceptable in most application environments. However, comparing the computation time needed for simulation and modeling, the difference is extremely high. The modeling computation for each of all the above configurations took between 0.5 and 1 second, and required only about a hundred bytes of mem-

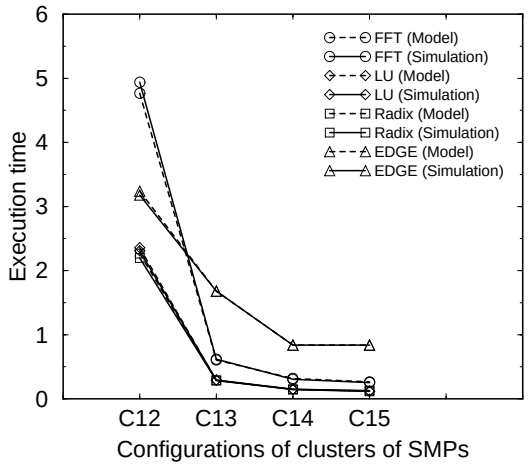

Figure 4: Comparisons of modeled and simulated $E(Instr)$ on clusters of SMPs.

ory. In contrast, it usually took more than 20 minutes to obtain one simulation result, let alone the time spent on developing the simulators.

6 Case Studies

In [3], we give three case studies on how to use the cost model to guide the design of a cost effective parallel platform for a specific kind of workload (applications). The first case is for a small budget ($5,000) request, which can only financially cover a cluster of workstations rather than SMPs, 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, meanwhile making the decision even 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 money is available.

We show that the execution performance of a parallel program could vary significantly on different cluster platforms of the same cost. For example, the execution times of the FFT program were 4 times higher on a slow Ethernet of workstations than that on a fast ATM network of workstations, where the number of workstations in the Ethernet cluster is 4 and each workstation has a 200 MHz CPU and a 64 MB memory, and the number of workstations in the ATM cluster is 3 and each workstation has a 200 MHz CPU and a 32 MB memory.

Our study also shows that the length of the memory hierarchy is the most sensitive factor to minimize 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 length 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 more expensive 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. Here are some principles we obtained from the study, and recommend to follow in building a cost-effective cluster system:

- If the workload is highly CPU bound indicated by a small γ, and has a good program locality ($\beta < 100$), data accesses to higher levels of the memory hierarchy will be rare. Thus, an optimal cluster choice would be a slow network of a large number of high-speed workstations. Program LU is an example.
- If the workload is highly CPU bound indicated by a small γ, and has a relatively poor program locality ($\beta > 100$), data accesses using the network will be frequent in a network of workstations. Thus, an optimal cluster choice would be a fast network of a small number of high-speed workstations. Program FFT is an example.
- If the workload is memory bound indicated by a large γ, but has a good program locality ($\beta < 100$), data accesses are likely kept within a computing node. Thus, an optimal cluster choice would be a slow network of workstations with a large capacity of memories. This platform will take advantage of both parallel computing among CPUs and parallel data accesses among memory modules. Program EDGE is an example.
- If the workload is memory bound indicated by a large γ and has a relatively poor program locality ($\beta > 100$), data accesses to higher levels of the memory hierarchy will be frequent. A choice of SMPs would minimize the execution time of this type of workload although the number of processors could be limited. Program Radix is an example.
- If the workload is both memory and I/O bound indicated by a large γ, and by a very large β, the computation would mainly depend on the performance of data transfer through a network. Thus, an optimal choice would be an SMP or a fast cluster of SMPs. Commercial workload TPC-C is an example.
- Regarding the system upgrading, we recommend that money 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 Conclusions

We have developed an analytical model to help cluster builders and users to quickly determine an optimal platform for a given budget and for certain types of application workloads. We are currently working on three 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, and (3) 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.

Acknowledgement: G. Li and Z. Zhu participated some technical discussions of this work, and made constructive suggestions. We are grateful to N. Wagner for reading the paper and for his comments.

References

[1] D. Bailey, et. al., "The NAS parallel benchmarks", *International Journal on Supercomputing Applications*, Vol. 5, No. 3, Fall 1991, pp. 63-73.

[2] E. G. Coffman and P. J. Denning, *Operating System Theory*, Prentice-Hall Inc., Englewood Cliffs, 1973.

[3] X. Du and X. Zhang, *An analytical model for cost-effective cluster computing*, Technical Report, Department of Computer Science, College of William and Mary, October, 1998.

[4] M. Heinrich et al., "The performance impact of flexibility in the Stanford FLASH multiprocessor", *Proceedings of 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994, pp. 274-285.

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Second Edition, San Francisco, Morgan Kaufmann, 1996.

[6] B. L. Jacob, P. M. Chen, S. R. Silverman, and T. N. Mudge, "An Analytical Model for Designing Memory Hierarchies", *IEEE Transactions on Computers*, Vol. 45, No. 10, 1996, pp. 1180-1194.

[7] S. M. Ross, *Introduction to Probability Models*, Sixth Edition, Academic Press:San Diego, 1997.

[8] A. J. Smith, "Cache Memories", *Computing Survey*, Vol. 14, No. 3, 1982, pp. 473-530.

[9] K. S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall Inc., Englewood, 1982.

[10] J. E. Veenstra and R. J. Fowler, "MINT: a front end for efficient simulation of shared-memory multiprocessors", *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 1994, pp. 201-207.

[11] S. C. Woo, et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations", *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995, pp. 24-36.

[12] X. Zhang, S. G. Dykes, and H. Deng, "Distributed edge detection: issues and implementations", *IEEE Computational Science & Engineering*, Spring Issue, 1997, pp. 72-82.