Modeling the Effects of Contention on the Performance of Heterogeneous Applications

Silvia M. Figueira* and Francine Berman**
Computer Science and Engineering Department
University of California, San Diego
{silvia,berman}@cs.ucsd.edu

Abstract

Fast networks have made it possible to coordinate distributed heterogeneous CPU, memory, and storage resources to provide a powerful platform for executing high-performance applications. However, the performance of these applications on such systems is highly dependent on the allocation and efficient coordination of application tasks. A key component for a performance-efficient allocation strategy is a predictive model which provides a realistic estimate of application performance under varying resource loads. In this paper, we present a model for predicting the effects of contention on application behavior in heterogeneous systems. In particular, our model calculates the ***slowdown*** *imposed on communication and computation for non-dedicated two-machine heterogeneous platforms. We describe the model for the Sun/CM2 and Sun/Paragon coupled heterogeneous systems. We present experiments on production systems with emulated contention which show the predicted communication and computation costs to be within 15% on average of actual costs.*

1. Introduction

In the last decade, distributed heterogeneous systems have emerged as a powerful platform for executing high-performance applications. Current experience shows that many high-performance distributed applications are composed of a few coarse-grained tasks and execute on a heterogeneous platform formed by two machines [9][13][14][16]. These machines can be MIMD MPPs, workstations, SIMD MPPs, vector computers, etc. For many of these applications, both heterogeneity and parallelism in the code are exploited. For example, in a molecular structure application, an MPP is used for the parallel part of the code whereas a vector computer is used to perform the serial portion [14].

For such applications, the development of a performance-efficient allocation of tasks to machines is dependent upon a realistic prediction of application behavior under changing system load. In particular, additional applications executing on the system can dramatically affect the availability and capability of resources and must be factored into predictions of computation and communication costs for a given application. If an effective predictive model can be developed for coupled non-dedicated heterogeneous systems, heterogeneous application tasks can be allocated in a way that promotes application performance on the system.

To illustrate, consider an application with two tasks A and B mapped to a heterogeneous network with two machines M1 and M2. Tables 1 and 2 contain the times to execute each task of the application on each machine of the platform in dedicated mode, as well as the times to transfer data between the machines (including data conversion) in dedicated mode when the tasks do not execute on the same machine. In this dedicated environment, both tasks should be assigned to machine M1, and the application would execute in 16 time units.

	M1	M2
A	12	18
B	4	30

Table 1: Dedicated Execution Times

	M1 → M2	M2 → M1
A → B	7	8

Table 2: Dedicated Communication Times

However, if M1 is time-shared and there are other applications competing for its CPU, the times in M1's column of Table 1 will change. Table 3 shows the case when the other applications are CPU-bound and slow tasks A and B on M1 by a factor of 3. In this setting, task A should be assigned to machine M2, whereas task B should be assigned to machine M1. Since communication

* Supported by a scholarship from CAPES and UFRJ (Brazil).
** Supported in part by NSF contract number ASC-9301788.

is required, the application would execute in 38 time units, 10 units less than if both tasks were executed on machine M1.

In another setting, the extra applications on M1 could be computing and also transferring data to M2. Tables 3 and 4 show the case where both the computation on M1 and the communication between M1 and M2 are slowed by a factor of 3. In this case both tasks should be assigned to machine M1 even though task A executes faster on M2. This is because the gain obtained by executing task A on machine M2 will be outweighed by the slowed-down communication between the machines.

	M1	M2
A	36	18
B	12	30

Table 3: Non-dedicated Execution Times

	M1 → M2	M2 → M1
A → B	21	24

Table 4: Non-dedicated Communication Times

This simple example illustrates that contention dramatically affects application performance and must be factored into estimates of communication and computation times for efficient allocation. Note that contention effects occur in different ways: In time-shared environments, computation is slowed down by contention for the CPU whereas, in space-shared systems, contention for the network slows down communication. In addition, contention effects depend on workload behavior because different applications utilize different types of resources and the need for these resources varies during the application execution. Therefore, a model to predict contention effects on application performance should reflect both system characteristics and workload behavior.

Despite the fact that contention effects may dramatically affect the performance of applications, and consequently influence the mapping of tasks to machines, there has been little discussion on how to predict these effects. Machine workload has been used to parameterize the allocation of tasks to workstations in a network, however many allocation strategies do not consider load characteristics in the measurement of workload ([2][3][5][6][19], etc.). Load characteristics have been included in performance prediction models for networks of workstations (e.g., [10][20]), however such models assume that each workstation is shared by at most one compute-intensive task and one local task which alternates idle with compute-intensive cycles.

We believe that considering the load on the machines in a network of workstations is important, but may not be enough. In particular, both load characteristics (CPU- versus I/O-bound) and contention on the network should be considered. In heterogeneous systems the problem is even more complex because contention effects must be considered differently on each machine, depending on the architecture and on the local scheduling policy.

In this paper, we develop a contention model for application performance on two-machine heterogeneous platforms based on workload behavior and system characteristics. In order to produce realistic predictions, our contention model utilizes application-dependent workload characteristics that reflect the actual load on the system at run-time. These parameters must be detailed enough to allow good accuracy, but since they are determined at run time they should be easy to obtain or calculate. Our model provides a **slowdown factor** which is used to predict computation and communication costs. Generalization of these results to more than two machines is straightforward.

This paper is organized as follows: Section 2 discusses our approach. In Section 3, we present the models and experiments for the Sun/CM2 and the Sun/Paragon platforms. Section 4 concludes with a summary and future work.

2. Approach

In the following section, we develop predictive models for the Sun/CM2 and the Sun/Paragon heterogeneous platforms. We identify causes for contention for each component (machines and network link) of each platform. The causes and manifestations of contention will vary depending on the local schedulers used by each machine, communication mechanisms employed on the link, and behavior of competing applications. We parameterize such influences as *system-dependent* (information determined statically by a system test suite) or *application-dependent* (based on information provided by the user). These parameters are synthesized into formulas which calculate the slowdown factor and can be used to predict computation and communication costs.

Note that viewing the Sun/CM2 and other Host/MPP systems as a coupled heterogeneous network expands the way in which such systems can be programmed. The host or *front-end* is a machine in its own right which can be shared by a number of applications. The contention model quantifies the slowdown as it is perceived by all applications running on the front-end. In addition, many applications have tasks for which there are efficient codes on both the front-end and the back-end machines. Such codes include commonly used libraries (e.g., LAPACK [1] and ScaLAPACK [8]) and tasks (such as matrix multiplication or sorting) for which different algorithms are used to optimize the running time on different machines. Our model provides a realistic estimate of the costs of computing a task on the front-end machine (with one algorithm) as compared to moving the data across the network link and computing the task (perhaps with a different algorithm) on the back-end machine. Such costs can be factored into the strategy for scheduling the coupled system so as to optimize application performance. We believe that these techniques will prove useful for such systems as the C90/T3D.

In order to validate our models we have used scientific computations and synthetic benchmarks on non-dedicated production systems in which the contention was emulated. In this paper, we demonstrate our model on two benchmarks used in scientific applications: an SOR algorithm, which solves Laplace's equation, and a Gaussian Elimination algorithm. Both are typical of basic tasks used in heterogeneous applications.

Note that the accuracy of our model closely correlates with the accuracy of the application-dependent parameters, i.e., more accurate estimates of the behavior of the current workload will lead to better predictions of the contention effects on application performance. Typical heterogeneous applications (e.g., [9][13][14]) execute for a long period of time, alternating computation with communication cycles. Therefore, contention effects should be considered in the long term, when each application is likely to be affected by both the computation and communication cycles of the other applications executing on the platform. Even in this case the variance in execution time on production systems can be high, which makes it difficult to accurately model contention effects. Our initial experiments have been promising and we describe them here.

In this paper, we make the following assumptions: We assume that access to the coupled heterogeneous system is controlled so that only applications executing on the system share its resources. We also assume that the working set of each application executing on the platform fits in memory, i.e., no delay is imposed by swapping, and that contention is experienced for the entire duration of an application. We assume we know the set of all applications executing on the system. Although this may seem limiting, it is common for system administration to have information about applications executing in supercomputing environments. This information may be provided by the users or obtained from the resource management system.

In the following section, we define a contention measure, the *slowdown factor*, to adjust the computation times and communication costs of an application to accommodate for system load. The adjusted predictions can be used to rank candidate schedules of application tasks to system resources. We assume that computation times have already been calculated for a dedicated environment. However, we calculate the communication costs for dedicated environments.

The slowdown factor reflects the current load of the system and is always calculated at run-time. It can be recalculated every time the system status changes or when new applications arrive. The slowdown factor is used to determine realistic performance predictions for the tasks, and will be used by the scheduler. Since the slowdown factor is always calculated at run-time, it must be efficient to compute relative to how quickly applications enter and leave the system.

3. Contention Models

We have performed experiments on two different platforms: a Sun/CM2 and a Sun/Paragon at the San Diego Supercomputer Center. The Sun/CM2 is formed by a Sun 460 and a Thinking Machines CM2, connected by a dedicated link. The Sun/Paragon platform is formed by a Sun Sparcstation and an Intel Paragon. The fact that they are the only machines on an Ethernet network allows us to consider the link between them to be dedicated. It is important to note that the links between these machines are dedicated to the machines, but shared by the applications executing on the machines.

3.1. Sun/CM2 Experiments

The Sun acts as a front-end for the CM2, an SIMD machine whose processors execute instructions received from the Sun in a synchronous manner. The Sun and the CM2 are tightly coupled in the sense that the CM2 never executes a program by itself; it always depends on the Sun, which sends the instructions and executes, concurrently or not, the serial and scalar parts of the code.

The Sun/CM2 platform is used as a coupled heterogeneous system. The Sun hosts the application and may execute tasks on the CM2 for better performance. Executing a task on the CM2 may involve transferring data to the CM2 before execution and back to the Sun afterwards. Therefore, the decision on where to execute each task will depend on the time to execute the task on each machine (Sun and CM2), and on the communication required if the task is executed on the CM2. In fact, a task should execute on the CM2 only when

$$T_{\text{sun}} > T_{\text{cm2}} + C_{\text{sun} \to \text{cm2}} + C_{\text{cm2} \to \text{sun}}, \qquad (1)$$

where T_{sun} is the elapsed time when executing on the Sun, T_{cm2} is the elapsed time when executing on the CM2, $C_{\text{sun} \to \text{cm2}}$ is the communication cost from the Sun to the CM2, and $C_{\text{cm2} \to \text{sun}}$ is the communication cost from the CM2 to the Sun.

In (1), contention generated by other applications executing on the platform may slow down the computation and communication, influencing the decision on where to execute each task. Since there is only one sequencer in our Sun/CM2 platform, only one process can execute on the CM2 at a time. Therefore, the only contention in this system is generated by multiple applications executing on the Sun. However, contention caused by CPU-bound processes on the Sun affects not only tasks executing on the Sun, but also communication and tasks executing on the CM2. This happens because executing a task on the CM2 means that the parallel instructions are executed on the CM2, but the serial and scalar parts of the code execute on the Sun and can be affected by other applications competing for the Sun's CPU.

3.1.1. Communication

In order to transfer data between the Sun and the CM2, elements are copied from an array in one machine to another array in the other machine. Each element is transferred point-to-point between the Sun and the processor that holds the respective element. Communication between the Sun and the CM2 is an element-by-element operation which is affected by contention on the Sun.

The time to transfer data sets from the Sun to the CM2 in dedicated time is

$$dcomm_{\mathrm{sun} \to \mathrm{cm2}} = \sum_{i \in \{\text{data sets}\}} N_i \times \left(\alpha_{\mathrm{sun}} + \frac{size_i}{\beta_{\mathrm{sun}}} \right),$$

where each *data set* is formed by a group of same-sized messages, N_i is the number of messages used to transfer the ith data set from the Sun to the CM2, $size_i$ is the size of the messages in the ith data set going from the Sun to the CM2, α_{sun} is the startup time on the Sun, and β_{sun} is the effective bandwidth from the Sun to the CM2. We define *effective bandwidth* as the actual rate of transmission achieved, as opposed to the (peak) bandwidth provided by the communication link. The time to transfer data sets from the CM2 to the Sun is calculated analogously and given by $dcomm_{\mathrm{cm2} \to \mathrm{sun}}$.

The values for α_{sun}, β_{sun}, α_{cm2}, and β_{cm2} are system-dependent and can be calculated by benchmarks similar to those described in [11]. One benchmark transfers one array with 10^6 elements from the Sun to the CM2 and then transfers 1 word from the CM2 to the Sun. If this takes C seconds, where

$$C = \alpha_{\mathrm{sun}} + 10^6 / \beta_{\mathrm{sun}} + \alpha_{\mathrm{cm2}} + 1 / \beta_{\mathrm{cm2}}$$

and

$$\alpha_{\mathrm{sun}} + \alpha_{\mathrm{cm2}} + 1 / \beta_{\mathrm{cm2}} \ll 10^6 / \beta_{\mathrm{sun}},$$

then $\beta_{\mathrm{sun}} \approx 10^6 / C$ words per second. The value for β_{cm2} can be calculated in a similar way.

Another benchmark calculates α_{sun} and α_{cm2}. It transfers 10^6 arrays of 1 element each from the Sun to the CM2 and then transfers 10^6 arrays of 1 element each back to the Sun. If this takes C seconds, where

$$C = 10^6 \times (\alpha_{\mathrm{sun}} + 1/\beta_{\mathrm{sun}} + \alpha_{\mathrm{cm2}} + 1/\beta_{\mathrm{cm2}}),$$

β_{sun} and β_{cm2} are known, and we assume $\alpha_{\mathrm{sun}} = \alpha_{\mathrm{cm2}}$, then we can calculate

$$\alpha_{\mathrm{sun}} \approx \alpha_{\mathrm{cm2}} \approx \left(C/10^6 - 1/\beta_{\mathrm{sun}} - 1/\beta_{\mathrm{cm2}} \right) / 2 \text{ seconds.}$$

The values for N_i and $size_i$ are application-dependent and should be provided by the user.

Our experiments show that CPU cycles are split equally among all the processes running on the Sun with the same priority. Therefore, a task executing on the Sun will compute or communicate with the CM2 $p + 1$ times slower when there are p extra (CPU-bound) processes executing on the Sun. The effect of contention on the Sun must be factored into the communication time calculation as follows:

$$C_{\mathrm{sun} \to \mathrm{cm2}} = dcomm_{\mathrm{sun} \to \mathrm{cm2}} \times \text{slowdown}$$

and

$$C_{\mathrm{cm2} \to \mathrm{sun}} = dcomm_{\mathrm{cm2} \to \mathrm{sun}} \times \text{slowdown},$$

where

$$\text{slowdown} = p + 1.$$

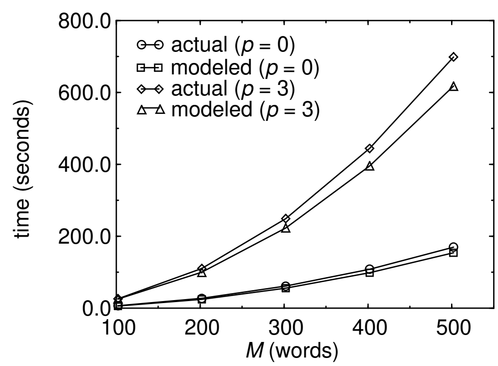

Figure 1: Communication between the Sun and the CM2 in dedicated (p = 0) and non-dedicated (p = 3) modes.

Figure 1 shows the modeled and actual times to transfer a matrix of size $M \times M$ to and from the CM2, where an SOR algorithm is executed, in dedicated mode and non-dedicated mode with 3 extra CPU-bound applications executing on the Sun. The program was written in CM-Fortran. This graph illustrates the fact that contention on the Sun does affect communication between the Sun and the CM2. In this example, our predictions were within an average error of 11% of the actual measurements. In a larger set of experiments, the error was within 15%.

3.1.2. Computation

Since CPU cycles are split equally among all the processes running on the Sun with the same priority, a task executing on the Sun will execute *p + 1* times slower when there are p extra (CPU-bound) processes executing on the Sun. Therefore, the effect of contention on execution time can be calculated as follows:

$$T_{\text{sun}} = dcomp_{\text{sun}} \times \text{slowdown},$$

$$T_{\text{cm2}} = \max\{(dcomp_{\text{cm2}} + didle_{\text{cm2}}), (dserial_{\text{cm2}} \times \text{slowdown})\},$$

and

$$\text{slowdown} = p + 1,$$

where $dcomp_{\text{sun}}$ is the time to execute the task on the Sun in dedicated mode, $dcomp_{\text{cm2}}$ is the time to execute the parallel instructions of a task on the CM2 (dedicated mode), $didle_{\text{cm2}}$ is the idle time on the CM2 (dedicated mode) when waiting for instructions from the Sun, $dserial_{\text{cm2}}$ is the time to execute on the Sun (in dedicated mode) the serial and scalar parts of a task whose parallel instructions are executing on the CM2, and p is the number of extra CPU-bound applications executing on the Sun.

Sun		**CM2**
serial instruction		idle
serial instruction		idle
parallel instruction	instruction →	execute
serial instruction		execute
serial instruction		execute
parallel instruction	instruction →	execute
serial instruction		execute
serial instruction		idle
serial instruction		idle
parallel instruction	instruction →	execute
idle		execute
idle	← result	execute
serial instruction		idle

Figure 2: Example of a task executing on the CM2.

Figure 2 illustrates the execution of a task on the CM2. The serial instructions execute on the Sun whereas the parallel instructions are sent to the CM2, which already holds the data required for the computation. Depending on how long the parallel instructions take to execute, the CM2 may stay idle waiting for the Sun. Note that $didle_{\text{cm2}}$ and $dserial_{\text{cm2}}$ are different values. Actually, $didle_{\text{cm2}}$ never exceeds $dserial_{\text{cm2}}$ because the Sun and the CM2 may execute concurrently, i.e., the Sun may be pre-executing serial code while the CM2 is executing parallel instructions. As shown in Figure 2, the Sun may also stay idle waiting for the result of a parallel instruction (e.g., a reduction operation).

Figure 3 shows the modeled and actual times to execute a Gaussian Elimination program on a matrix of size $M \times M + 1$ on the CM2 in non-dedicated mode with 3 extra CPU-bound applications executing on the Sun. The algorithm was implemented in CM-Fortran. In the same graph we present the time to execute the same algorithm in dedicated mode ($p = 0$) to show that the algorithm actually does take longer to execute on the CM2 when $M < 200$ and $p = 3$. This happens because the product of the time to execute the serial part of the algorithm on the Sun and the slowdown factor outweighs the time spent on the CM2. However, when $M \geq 200$, sufficient computation is executed by the CM2 so the three extra processes on the Sun do not affect the execution substantively. In this case, the times in dedicated and non-dedicated mode are the same.

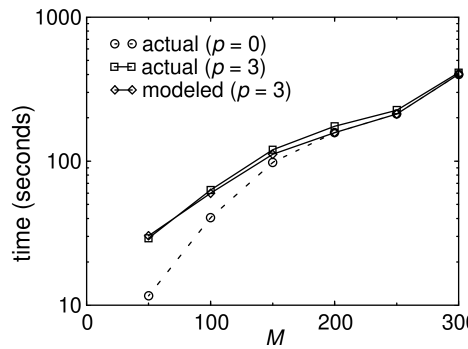

Figure 3: Modeled and actual times for Gaussian Elimination on the CM2.

We have performed a large number of experiments using synthetic benchmarks, which employ a representative subset of the operations provided by the CM2 and used in high-performance programs, in order to verify the generality of the model. The experiments executed so far have shown the error between predicted and actual times to be within 15% for both communication and computation, which is encouraging.

3.2. Sun/Paragon Experiments

The Sun/Paragon platform is very different from the Sun/CM2 because the Sun and the Paragon are totally independent. However, the Sun can work as a "logical" front-end to the Paragon by hosting applications which may have tasks executed serially (on the Sun) or in parallel (on the Paragon). As before, a task should execute on the Paragon only when

$$T_{\text{sun}} > T_{\text{p}} + C_{\text{sun} \to \text{p}} + C_{\text{p} \to \text{sun}},$$

where T_{sun} is the elapsed time when executing on the

Sun, T_{p} is the elapsed time when executing on the Paragon, $C_{\mathrm{sun} \to \mathrm{p}}$ is the communication cost from the Sun to the Paragon, and $C_{\mathrm{p} \to \mathrm{sun}}$ is the communication cost from the Paragon to the Sun.

For this platform we will consider the contention on the Sun and on the network link. Note that, even though the Paragon (a coarse-grained mesh-connected MIMD MPP) is space-shared, traffic on the mesh may affect an application's performance by slowing down its communication. This kind of inter-partition contention is addressed by Liu et al. [12], who show that traffic effects vary with the size of the messages on the network, and by Tron and Plateau [17]. Also, contention for CPU in each node may occur if the nodes are time-shared and gang-scheduling [7] is implemented. These effects can be included in T_{p}.

3.2.1. Communication

Communication costs are affected by contention on the link, which occurs when multiple applications executing on the Sun must move data to computations concurrently executing on the Paragon. (Recall that only one application could communicate with the CM2). However, communication is also delayed by contention for the Sun's CPU, because communication involves some CPU execution for data format conversion.

There are two modes of communication between the Sun and the Paragon. The Sun can either communicate directly to each compute node of the Paragon using TCP/IP, or it can use a service node as a bridge. In the second mode, the Sun uses TCP/IP to communicate with the service node, which communicates with the compute nodes using NX, the communication interface available on the Paragon. We have performed experiments using both modes (which we call 1-HOP and 2-HOPS, respectively). Figure 4 shows communication times (including data format conversion costs) to transfer bursts of 1000 equal-sized messages to and from the Paragon using both modes of communication. Since they present very similar behavior (see Figure 4), we will show the results for the 1-HOP mode only.

Figure 4 and other experiments indicate that communication cost is a piecewise linear function of the message size. We model the communication from the Sun to the Paragon in dedicated mode as

$$dcomm_{\mathrm{sun} \to \mathrm{p}} = \sum_{i \in \{\text{data sets}\}} N_i \times \left(\alpha_{\mathrm{sun}} + \frac{size_i}{\beta_{\mathrm{sun}}} \right),$$

where each *data set* is formed by a group of same-sized messages, N_i is the number of messages used to transfer the *i*th data set from the Sun to the Paragon, $size_i$ is the size of the messages of the *i*th data set traversing the link from the Sun to the Paragon, α_{sun} is the startup time on the Sun, and β_{sun} is the effective bandwidth from the Sun to the Paragon.

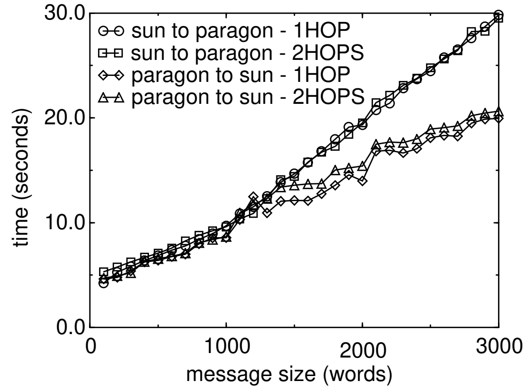

Figure 4: Time to send bursts of 1000 equal-sized messages to and from the Paragon in dedicated mode.

The values for α_{sun} and β_{sun} can be calculated by linear regression on the numbers obtained with a ping-pong benchmark. This benchmark transfers messages from the Sun to the Paragon in bursts containing 1000 messages of the same size. After each burst, one message containing one word is transferred back to the Sun. Although α_{sun} and β_{sun} are intrinsically system-dependent, they must be calculated for each communication interface and buffer size. However, these values do not change dynamically and are calculated fairly efficiently just once, not imposing any overhead on the scheduling process at run-time. Both N_i and $size_i$ are application-dependent and easy for the user to provide. Usually, these values are related to the size of the problem being solved by the task.

Since communication cost is modeled as a piecewise linear function, a better approximation can be obtained by calculating different startup costs and bandwidths for two pieces of the function: one for message size $\le threshold$ words and another for message size $> threshold$ words, where *threshold* is the piecewise boundary. The *threshold* which gives the best accuracy is system-dependent and can be determined by exhaustive search. The number of possible thresholds is small (it is equal to the number of different message sizes used by the ping-pong benchmark), and the threshold value can be calculated statically, just once for each platform. Hence, the overhead to calculate this value is negligible.

In the Sun/Paragon case, *threshold* = 1024 words. Given a threshold value, we model the communication from the Sun to the Paragon as follows:

$$dcomm_{\mathrm{sun} \to \mathrm{p}} = \sum_{i \in \{\text{data sets}\}_1} N_i \times \left(\alpha_{\mathrm{sun}_1} + \frac{size_i}{\beta_{\mathrm{sun}_1}} \right) + \sum_{i \in \{\text{data sets}\}_2} N_i \times \left(\alpha_{\mathrm{sun}_2} + \frac{size_i}{\beta_{\mathrm{sun}_2}} \right),$$

where the first term represents the first piece of the function (message size $\leq$ *threshold*) and the second term represents the second piece (message size $>$ *threshold*). In addition, $\{\text{data sets}\}_1$ contains data sets in which the messages have *threshold* or less words, α_{sun_1} is the startup time on the Sun for messages with *threshold* or less words, β_{sun_1} is the effective bandwidth from the Sun to the Paragon for messages with *threshold* or less words, $\{\text{data sets}\}_2$ contains data sets in which the messages have more than *threshold* words, α_{sun_2} is the startup time on the Sun for messages with more than *threshold* words, and β_{sun_2} is the effective bandwidth from the Sun to the Paragon for messages with more than *threshold* words.

The communication cost from the Paragon to the Sun, given by $dcomm_{\text{p} \to \text{sun}}$, is modeled analogously. Both $dcomm_{\text{sun} \to \text{p}}$ and $dcomm_{\text{p} \to \text{sun}}$ are only calculated once per <application, problem-size, platform> triple. Since they do not vary with load, they do not need to be recalculated at run-time. In particular, calculating communication costs should not impact the efficiency of the scheduling process.

Our experiments show that both the number of applications on the Sun (computing and communicating with the Paragon) and the size of the messages being transferred affect communication costs. However, we have noticed that the effect of different message sizes is limited and communication costs can be approximated by:

$$C_{\text{sun} \to \text{p}} = dcomm_{\text{sun} \to \text{p}} \times \text{slowdown},$$

where

$$\text{slowdown} = 1 + \sum_{i=1}^{p} \left(pcomp_i \times delay^i_{comp} \right) + \sum_{i=1}^{p} \left(pcomm_i \times delay^i_{comm} \right).$$

In the slowdown calculation, the first summation accounts for other applications competing for CPU cycles, whereas the second summation accounts for other applications communicating with the Paragon.

The value $delay^i_{comp}$ is the average delay imposed on the communication by i applications computing on the Sun and can be calculated by measuring the delay imposed by i compute-intensive contention generators on the ping-pong benchmark (executed from the Sun to the Paragon). The value $delay^i_{comm}$ is the average delay imposed on the communication by i applications communicating with the Paragon. It is the average of two delays: the average delay imposed on the ping-pong benchmark (executed from the Sun to the Paragon) by i contention generators that transfer one-word messages from the Sun to the Paragon and the average delay imposed on the same benchmark by i contention generators that transfer one-word messages from the Paragon to the Sun.

In addition, p is the number of extra applications executing on the Sun, $pcomp_i$ is the probability that i applications will compute on the Sun at the same time, and $pcomm_i$ is the probability that i applications will communicate with the Paragon at the same time. Both $pcomp_i$ and $pcomm_i$ are calculated based on the percentages of computation and communication associated with each application executing on the Sun. Suppose $p = 2$ and one application communicates 20% of the time and computes 80% of the time whereas the other communicates 30% of the time and computes 70% of the time. In this case:

$$pcomm_1 = 0.2 \times 0.7 + 0.3 \times 0.8$$
$$pcomm_2 = 0.2 \times 0.3$$
$$pcomp_1 = 0.2 \times 0.7 + 0.3 \times 0.8$$
$$pcomp_2 = 0.7 \times 0.8$$

The percentages of computation and communication associated with each application executing on the Sun can be either directly given by the users or calculated from computation and communication costs (in dedicated mode) provided by the user. Since computation and communication estimates are typical parameters for performance-efficient schedulers, we consider them to be available for the slowdown calculation.

The slowdown factor is calculated at run-time and, therefore, it is important to guarantee that its calculation is efficient so that it does not impose too much overhead on the scheduling process. The values $delay^i_{comp}$ and $delay^i_{comm}$ are system-dependent and do not change dynamically. They are calculated just once for each platform and, therefore, their calculation does not impose any additional overhead on the scheduling process at run-time. The values $pcomp_i$ and $pcomm_i$ change with the load on the system and are obtained at run-time. Using dynamic programming, it is possible to generate all $pcomp_i$ (or $pcomm_i$), for $1 \leq i \leq p$, in $O(p^2)$ time. If a new application is added to the system, the new values for $pcomp_i$ (or $pcomm_i$) can be obtained in $O(p)$ time. If an application finishes, it will take $O(p^2)$ time for the values to be regenerated, but updating the slowdown factor whenever an application finishes is only necessary if task migration is allowed. The slowdown calculation itself takes $O(p)$ time. Since p is small[1] and the overall calculation of the slowdown takes $O(p^2)$ time, the overhead imposed by its calculation is negligible.

Figure 5 illustrates $C_{\text{sun} \to \text{p}}$ by showing the modeled and actual times for transferring bursts of 1000 equal-sized messages to the Paragon when there are two additional applications executing on the Sun which alternate computation and communication cycles. The contending applications communicate 25% and 76% of

[1] The number of applications executing on a computer is generally limited by the computer's resources and typically small, e.g. see [18].

the time, respectively, transferring messages with 200 words. In this experiment, the average error for modeled versus actual communication cost was within 12%.

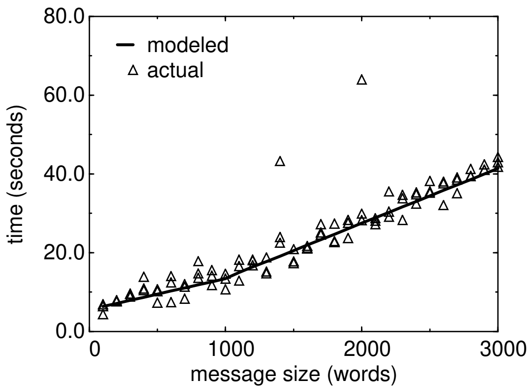

Figure 5: Time to send bursts of 1000 equal-sized messages from the Sun to the Paragon in non-dedicated mode, with two processes executing on the Sun, one of which communicates 25% of the time and the other of which communicates 76% of the time.

The communication from the Paragon to the Sun, given by $C_{\mathrm{p} \rightarrow \mathrm{sun}}$, is modeled analogously. Figure 6 shows the modeled and actual times for transferring bursts of 1000 equal-sized messages from the Paragon to the Sun when there are two additional applications executing on the Sun which communicate 25% and 76% of the time, respectively, transferring messages with 200 words. In this experiment, the average error for modeled versus actual communication cost was within 14%.

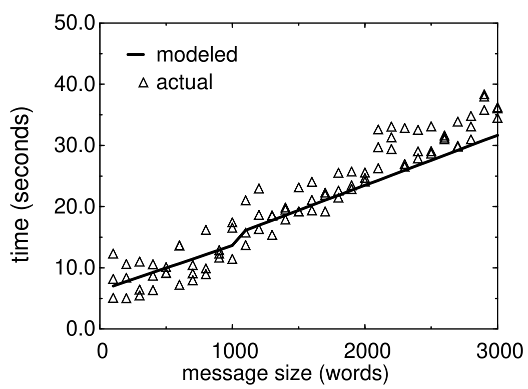

Figure 6: Time to send bursts of 1000 equal-sized messages from the Paragon to the Sun in non-dedicated mode, with two processes executing on the Sun, one of which communicates 25% of the time and the other of which communicates 76% of the time.

We have also performed experiments with different sets of contention generators which use different message sizes, communicate with different frequencies, and have various computation per communication ratios. For these experiments, we have observed a typical average error of 15%. This error reflects both the variance in the actual communication times and the fact that we do not consider the size of the messages (transferred by the competing applications) in the model. In our experiments, we have observed that the maximum error our model exhibits on the average does not exceed 30%. The maximum error occurs when applications competing for the link are communicating intensively. This happens because, in this case, the message size (which is not factored into the calculation of $delay^{i}_{comm}$) has a significant impact on the communication costs.

3.2.2. Computation

Computation on the Sun is affected by other applications computing on the Sun and/or communicating with the Paragon. The delay imposed by other applications computing on the Sun reflects the fact that CPU time is evenly split among the competing processes. The delay imposed by other applications communicating on the link between the Sun and the Paragon depends on the number of applications in the Sun communicating with the Paragon and the size of the messages being transferred. The impact of the message size on computation is significant and cannot be ignored by the model. Therefore, the execution time on the Sun can be approximated by:

$$T_{\mathrm{sun}} = dcomp_{\mathrm{sun}} \times \mathrm{slowdown}$$

and

$$\mathrm{slowdown} = 1 + \sum_{i=1}^{p} (pcomp_i \times i) + \sum_{i=1}^{p} \left(pcomm_i \times delay^{i,j}_{comm} \right).$$

In the slowdown calculation, the first summation accounts for other applications competing for CPU cycles, which are evenly split among all applications on the Sun. The second summation accounts for other applications communicating with the Paragon.

The value $dcomp_{\mathrm{sun}}$ is the time to execute the task on the Sun in dedicated mode, p is the number of extra applications executing on the Sun, $pcomp_i$ is the probability that i applications will compute on the Sun at the same time, $pcomm_i$ is the probability that i applications (on the Sun) will try to communicate with the Paragon at the same time, and $delay^{i,j}_{comm}$ is the delay imposed on the computation by i applications (on the Sun) which transfer messages with j words to and from the Paragon.

The value $delay^{i,j}_{comm}$ is calculated from two delays: the average delay imposed on a CPU-bound application by i contention generators that transfer j-word messages from the Sun to the Paragon, and the average delay imposed on the same CPU-bound application by i contention generators that transfer j-word messages from the Paragon to the Sun, where j is the maximum message size used in the system. Both $pcomp_i$ and $pcomm_i$ are calculated in the same way as for the communication case.

From our experiments, the effect of message size on the slowdown is significant, and we have observed that a "bad" j can cause the error to be as high as 75%. However, we have noticed that above a threshold on the message size the delay imposed is roughly constant. In the Sun/Paragon case, this threshold is around 1000 and, therefore, $delay^{i,1000}_{comm} \approx delay^{i,k}_{comm}$, for $k \geq 1000$. This threshold can be determined by calculating the delay imposed by contention generators transferring messages with increasing sizes.

We have also observed that good accuracy can be obtained by calculating $delay^{i,j}_{comm}$ for three values of j (1, 500, 1000). In this case, to determine $delay^{i,k}_{comm}$, k closest to j = 1, 500 or 1000 is chosen[2]. It is not clear, at this point, if three values will be enough for platforms other than the Sun/Paragon. It is clear, though, that j is an important part of the slowdown calculation; its value should reflect the maximum message size used in the system and can be obtained from the current applications' message sizes as given by the user.

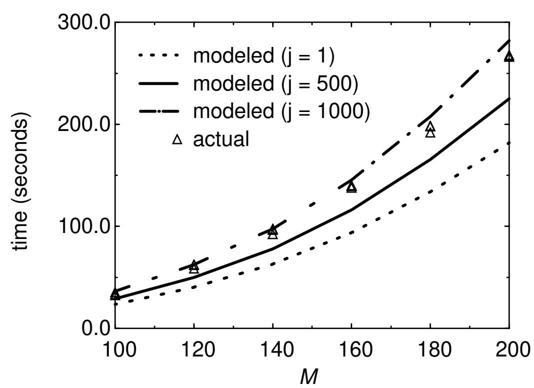

Figure 7: Time to execute the SOR algorithm on the Sun, in non-dedicated mode, with two extra applications executing on the Sun. The contending applications communicate 66% and 33% of the time respectively, and compute the remaining time.

Figure 7 shows the modeled and actual times for executing the SOR program on the Sun in dedicated and non-dedicated modes, parameterized by problem size (which is $M \times M$). In this experiment, two more applications are executing on the Sun. They alternate computation and communication cycles and communicate with the Paragon 66% (using 800-word messages) and 33% (using 1200-word messages) of the time, respectively. In this example, our predictions were within an average error of 4% of the actual measurements when j = 1000. For j = 500, the error was around 16% and, for j = 1, the error was within 32%. This example illustrates the importance of using the appropriate j.

Figure 8 shows the modeled and actual times for executing the SOR program on the Sun in dedicated and non-dedicated modes. However, in this experiment, the two extra applications on the Sun communicate with the Paragon 40% (using 500-word messages) and 76% (using 200-word messages) of the time respectively, and compute the remaining time. In this example, our predictions were within an average error of 5% of the actual measurements when j = 500. For j = 1 and j = 1000, the error was 25%.

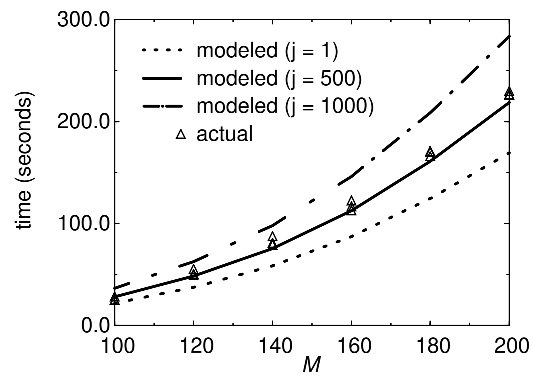

Figure 8: Time to execute the SOR algorithm on the Sun in non-dedicated mode with two extra applications executing on the Sun. The contending applications communicate 40% and 76% of the time respectively, and compute the remaining time.

We have also performed experiments with different sets of contention generators which use different message sizes, communicate with different frequencies, and have various computation per communication ratios. The typical average error was below 15%. According to our observations, competing applications communicating intensively or transferring small bursts can cause the error to be as high as 33%. We have noticed, however, that in these cases the delay imposed by communication corresponds to the delay imposed by contention for CPU in the Sun. We are currently investigating this fact in order to improve our predictions.

4. Summary and Future Work

Most applications share the resources of coupled heterogeneous systems with other applications. Since system load can vary dramatically, allocation strategies which assume that resources have a constant availability and/or capability are unlikely to promote performance-efficient allocations in practice. It is critical to provide a realistic model of the effects of contention on application performance in order to best allocate application tasks to

[2] Note that j = 1 is only used for message size < 95 words.

machines.

In this paper, we have presented a model which predicts contention effects in Host/MPP coupled heterogeneous platforms. The model provides a basis for predicting realistic communication and computation costs, and was shown to correlate with actual times within an average 15% error for a set of scientific benchmarks commonly found in heterogeneous applications. Note that the variance in times to execute applications on production systems is typically high, and therefore makes it difficult for a contention model to be accurate. For this reason, our objective has been to obtain accuracy on an average basis. The experiments executed thus far have shown that we have achieved our goal.

We are continuing to investigate causes for contention. We are currently extending our model to include memory constraints, as well as I/O operations. In addition, we plan to characterize the setting in which contending applications execute for only part of the execution of a given application. Since system load may vary during the execution of an application, the slowdown factors should be recalculated when the job mix changes, and task migration should be considered.

Finally, we believe that the slowdown factors developed for these small platforms can be used for larger heterogeneous systems. An accurate contention model is a fundamental part of a performance-efficient allocation strategy for heterogeneous systems, and ultimately essential to the use of these systems as platforms for parallel distributed applications.

Acknowledgments

We are grateful to Rich Wolski, Jennifer Schopf, and Cosimo Anglano for useful discussions. We would also like to thank Elizabeth Simon, Reagan Moore, Mike Vildibill, George Kremenek, David Lilja, the UCSD Parallel Computation Laboratory and the San Diego Supercomputer Center for their support.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. "LAPACK Users' Guide, Second Edition", SIAM, Philadelphia, PA, 1995.

[2] M. Atallah, C. Black, D. Marinescu, H. Siegel, and T. Casavant, "Models and Algorithms for Coscheduling Compute-Intensive Tasks on a Network of Workstations", *Journal of Parallel and Distributed Computing*, vol. 16, pp. 319-327, 1992.

[3] A. Beguelin, J. Dongarra, G. Geist, R. Manchek, and V. Sunderam, "Graphical Development Tools for Network-Based Concurrent Supercomputing", in *Proceedings of Supercomputing 91*, pp. 435-444.

[4] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks", Technical Report CS96-482, University of California, San Diego, Computer Science and Engineering Department, May 1996.

[5] A. Bricker, M. Litzkow, and M. Livny, "Condor Technical Summary", Technical Report #1069, University of Wisconsin, Computer Science Department, May 1992.

[6] H. Dietz, W. Cohen, and B. Grant, "Would you run it here...or there? (AHS: Automatic Heterogeneous Supercomputing)", in *Proceedings of the International Conference on Parallel Processing*, vol. II, pp.217-221, August 1993.

[7] D. Feitelson, "A Survey of Scheduling in Multiprogrammed Parallel Systems", Research Report, IBM Research Division, T. J. Watson Research Center, October, 1994.

[8] http://www.netlib.org/scalapack/scalapack_home.html, ScaLAPACK Software Library.

[9] A. Kuppermann and M. Wu, "Quantum Reaction Dynamics on a Gigabit/Sec Network", in *Proceedings of the Gigabit Testbed Maxijam*, November 1994.

[10] S. Leutenegger and X. Sun, "Distributed Computing Feasibility in a Non-Dedicated Homogeneous Distributed System", NASA - ICASE Technical Report 93-65, September 1993.

[11] D. Lilja, "Experiments with a Task Partitioning Model for Heterogeneous Computing", in *Proceedings of the Heterogeneous Computing Workshop*, pp. 29-35, April 1993.

[12] W. Liu, V. Lo, K. Windisch, and B. Nitzberg, "Non-contiguous Processor Allocation Algorithms for Distributed Memory Multicomputers", in *Proceedings of Supercomputing 94*, pp. 227-236, 1994.

[13] C. Mechoso, J. Farrara, and J. Spahr, "Running a Climate Model in a Heterogeneous, Distributed Computer Environment", in *Proceedings of the Third IEEE International Symposium on High Performance Distributed Computing*, pp. 79-84, August 1994.

[14] A. Phillips, J. Rosen, and V. Walke, "Molecular Structure Determination by Convex Global Underestimation of Local Energy Minima", University of Minnesota Supercomputer Institute Research Report UMSI 94/126, July 1994.

[15] V. Sunderam, "PVM: A Framework for Parallel Distributed Computing", *Concurrency: Practice and Experience*, vol. 2, n. 4, pp. 315-339, December 1990.

[16] Virtual Environments and Distributed Computing at SC'95. GII Testbed and HPC Challenge Applications on the I-WAY. Edited by Holly Korab and Maxine D. Brown. A publication of ACM/IEEE Supercomputing'95.

[17] C. Tron and B. Plateau, "Modeling of Communication Contention in Multiprocessors", in *Proceedings of Modeling Techniques and Tools for Computers - Performance Evaluation*, LNCS, pp. 406-424, 1994.

[18] M. Wan, R. Moore, G. Kremenek, and K. Steube, "A Batch Scheduler for the Intel Paragon with a Non-contiguous Node Allocation Algorithm", in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 29-40, April 1996.

[19] J. B. Weissman, "The Interference Paradigm for Network Job Scheduling", in *Proceedings of the Heterogeneous Computing Workshop*, pp. 38-45, April 1996.

[20] X. Zhang and Y. Yan, "A Framework of Performance Prediction of Parallel Computing on Non-dedicated Heterogeneous Networks of Workstations", in *Proceedings of 1995 International Conference of Parallel Processing*, vol. I, pp. 163-167, 1995.