Task Scheduling in Heterogeneous Systems in the Presence of Channel Contention

Ravi Prakash & Dhabaleswar K. Panda
Department of Computer and Information Science
The Ohio State University, Columbus, OH 43210.
e-mail: {prakash, panda}@cis.ohio-state.edu

Contact Author: Prof. Dhabaleswar K. Panda

Abstract

This paper investigates some of the communication and scheduling issues in heterogeneous systems. A representative heterogeneous system consisting of a set of computers, each architecturally different and exploiting different kinds of parallelism is considered. These computers are connected by a high speed, passive star-coupled photonic network that employs wavelength division multiplexing (WDM). A model scientific computation, exhibiting meta-parallelism, has been studied on this system. Two scheduling strategies have been considered. One strategy involves both subtask data and code migration to the subtask execution site, while the other (the *no-code-migration* strategy) requires only the data to be transmitted. Simulations of the execution of this computation on our heterogeneous system provide the following insights into the design of such a system: A master-slave model of execution can reduce the amount of communication, and is therefore scalable. Contention for communication channels makes a significant contribution towards the overall task completion time. Channel contention increases with an increase in the degree of parallelism. In the presence of fast disk I/O, the *no-code-migration* strategy performs better than the *subtask-code-migration* strategy. A hierarchical communication topology implemented on the passive star-coupled network leads to a considerable reduction in contention and task completion time. These results can be used as guidelines for developing scalable heterogeneous systems.

Keywords: channel contention, heterogeneous system, hierarchical interconnection, meta-parallelism, task scheduling, wavelength division multiplexing.

Contents

1 Introduction

When a task is to be executed on a parallel computer, it is broken down into smaller subtasks. Precedence relations determine the number of subtasks that can be executed in parallel. All the subtasks that can be executed concurrently are assigned to different groups of identical processors of the parallel computer, given that the computer under consideration has sufficient number of processors. However, this scheme is not very efficient for tasks exhibiting meta-parallelism. Such tasks exhibit coarse-grained heterogeneity [12], i.e., the task can be divided into subtasks exhibiting different kinds of parallelism. For example, if a task is broken into two subtasks – one exhibiting MIMD parallelism and the other having vectorizable operations, then executing both the tasks on a machine that exploits MIMD parallelism will not yield the best performance.

Heterogeneous parallel systems are especially suited for such tasks containing meta-parallelism. A heterogeneous parallel computing environment consists of a connected set of heterogeneous computer systems [2, 13, 19]. These computers have different architectures and exploit different types of parallelism. They are capable of communicating using a high speed interconnection network to cooperate in solving an application problem [12]. The high speed network is required to match the high computing power of the computers and to minimize the I/O overheads by providing low latency data transfer. A subtask may be executed on a computer that provides the most suitable mode of parallelism for it. Several independent subtasks may be executed concurrently. A subtask may have homogeneous parallelism embedded in it. This parallelism gets exploited by the numerous identical processors of the computer to which the subtask is assigned.

Multicomputers with heterogeneous processors provide the benefits of conventional multicomputers as well as the flexibility of a general network. The configuration of heterogeneous parallel systems may change with time, as new computers join the network and the old ones leave, either temporarily for repairs or permanently. Research in the area of heterogeneous systems is motivated by the belief that there is not, and will not be a single all-encompassing architecture, suitable for diverse problems [8].

In this paper we evaluate some of the factors that influence the performance of a heterogeneous system. These factors should be considered while evolving the strategies of subtask allocation to a suite of loosely coupled heterogeneous computers.

Section 2 presents a brief overview of some of the earlier work done towards scheduling tasks in heterogeneous systems. Section 3 describes a model of the scientific computation whose execution is studied. Section 4 discusses two strategies for scheduling tasks in a heterogeneous system. In Section 5 the processor, memory, and communication specifications of our abstract heterogeneous system have been presented. Section 6 presents the results obtained by simulating the execution of the model scientific computation on the abstract heterogeneous system. Finally, the conclusion is presented in Section 7.

2 Preliminaries

In heterogeneous systems it has been implicitly assumed that all the tasks are submitted at one particular node which acts as a front-end for the entire system. The operating system has to analyze the nature of the task and find the most suitable selection of component computers on which the various segments of the problem can be executed. This involves *system level scheduling* [12]. As the configuration of the system may change with time, tasks may have to be reassigned from one computer to another.

Several scientific applications can be divided into subtasks for which library routines are present at the component computers. If a subtask is to be executed at a particular computer then the corresponding library routine can be invoked at that site and only the input data and the result need to be passed as parameters of the remote procedure call.

Sometimes a task cannot be split into subtasks that correspond to routines in the software library. In such a situation the code and the data corresponding to a subtask have to be migrated from the front-end computer to the site where it is to be executed. The relative merits of the *subtask-code-migration* and *no-code-migration* approaches will be discussed in the later sections.

In the past, one of the ways to model the problem has been to assume that a master computer spawns a set of slave processes, corresponding to the subtasks, at different computers. All the slaves execute their respective subtasks concurrently and finish around the same time. Then their results can be merged by the master to obtain the final result. The model is illustrated in Figure 1. This model disregards the fact that transferring data and initiating subtasks at the nodes takes time. Besides, the master node can possibly communicate with only one slave at a time.

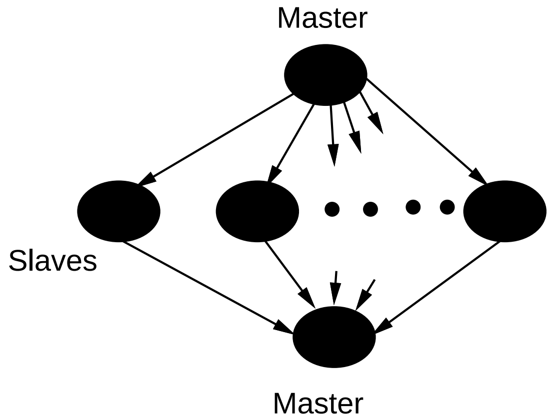

Figure 1: All the subtasks start and finish at the same time in a master-slave model of execution.

Another simplification, in the past, has been to divide the task into equal sized subtasks and get them processed at the slaves. But, different slave processors may have different speeds. So, intuitively, faster processors should get larger subtasks to execute. Similarly, for some applications, it may not be possible to divide tasks into equal sizes. This will lead to a variation in subtask sizes in a realistic heterogeneous environment.

In [14], a heuristic algorithm for piece-wise scheduling has been presented for distributed memory parallel computers. But it does not guarantee the best global schedule. In [16], Sih and Lee present a compile-time heuristic for scheduling tasks, on heterogeneous systems, with irregular interconnection structures. They assume that the code for the subtask already resides at the computer to which it is mapped. But, the overheads involved in ensuring that the code is resident at that computer have been ignored. The amount of data to be exchanged between subtasks has also been assumed to be fixed and known apriori. But, in several situations it is difficult to predict the data-size for a subtask at compile time. Besides, the data-size may vary significantly and affect the overall performance of the system. The following sections shall show that the amount of data to be exchanged between subtasks plays a significant role in determining the scheduling strategy that leads to a lower task completion time.

3 A Model Scientific Computation

In this study we analyze the issues involved in solving large scientific computations in diverse areas like medical imaging, fluid dynamics, finite-element analysis etc. Our problem model is based on two bioelectric field problems presented in [10]. A large task can be broken down into a number of subtasks. Based on the precedence relations some of the subtasks can be executed concurrently on different computers in the heterogeneous system. When subtask A comes to an end it produces some results. These results may be used by some other subtask B as input data. If the two subtasks are mapped onto different computers, A's result will have to be communicated to B's site. Let there be n_1 subtasks that execute concurrently and produce their respective results. Once these subtasks finish, let there be n_2 subtasks that use the results produced by all these n_1 subtasks as their input data. If one-to-one message communication between the subtasks is employed, the total number of messages needed will be $n_1 \times n_2$. The message complexity can be significantly reduced by using a multiple-phase master-slave model as shown in Figure 2.

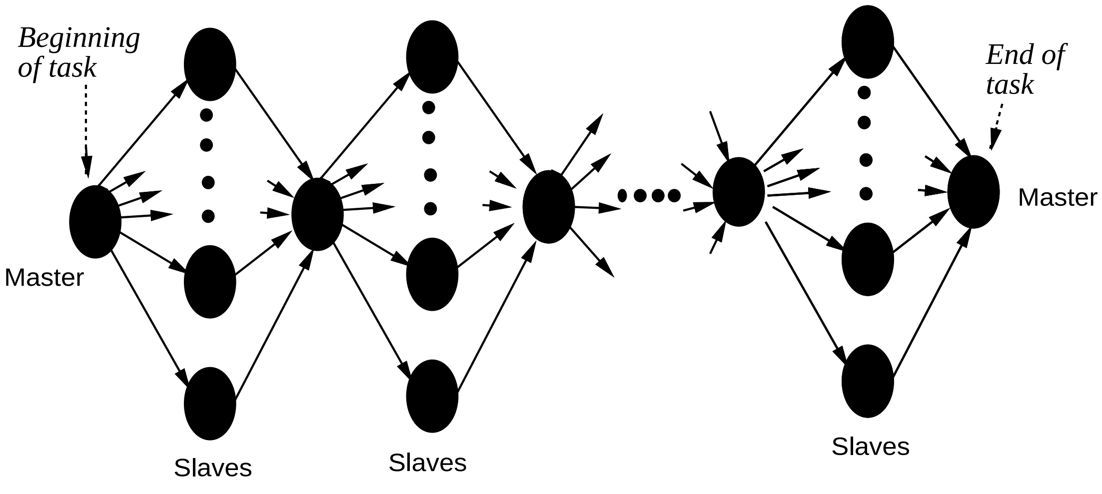

Figure 2: A multiphase master-slave model of task execution.

In the beginning of a phase the master computer spawns n_1 subtasks on different slave computers. Each slave sends its result to the master on completion. Once the master has received results from all the n_1 subtasks the phase comes to an end. In the next phase the master spawns n_2 subtasks on different slaves and sends the appropriate input data to each of them. In this case the number of messages required to distribute the results of the n_1 subtasks to the n_2 subtasks succeeding them is $n_1 + n_2$.

The reduction in message complexity brought about by the master-slave model can be further illustrated by the following example. In developing a computational model for electroencephalography, it should be noted that the electrical currents do not travel from one neuron to all the other neurons. Instead, impulses from a neuron travel along the axons to only some of the other neurons [10]. The influences coming along the axons determine the behaviour of the recipient neurons. If there are N neurons and we model direct message exchange between them there can possibly be $O(N^2)$ messages at each stage of signal transmission. Now, let all the neurons send their signals to a master node, The master gathers the data and redistributes them among the slaves corresponding to the recipient nodes. The message complexity for such a model is $O(N)$ per phase. A model to study the temporal behaviour of the brain would consist of multiple message exchange phases. The lower message complexity of the master–slave model will lead to reduced congestion and delays in the communication network.

In electrocardiography, boundary value problems in Poisson's equation or Laplace's equation for electrical conduction have to be solved in terms of the heart potentials. The problem can be made computationally tractable by "approximating it on a finite dimensional subspace and reformulating the partial differential equations as a set of linear matrix equations" [10]. So, each subtask in our model problem consists of matrix operations. We have used matrix operations just as a tool for modeling purposes. Subtasks may consist of a variety of different operations, depending on the nature of parallelism embedded in them. Our simulation results will hold as long as the ratios between the computational complexity of the actual subtasks and the processing speed of the machines on which they are executed are comparable to the corresponding values for our example problem and the abstract heterogeneous system described later.

4 Task Scheduling Strategies

In Section 2, two possible scheduling strategies for exploiting heterogeneous parallelism were mentioned. One of them involved subtask code migration. The other strategy used software libraries at the nodes. In this Section we describe the two strategies. First *code profiling* is done to identify the different kinds of parallelism in a task. Based on the information gathered from code profiling the task is broken down into several subtasks, each exhibiting a different kind of parallelism. Then *analytical benchmarking* is done to determine the suitability of a particular computer for a given type of parallelism [19]. Once code profiling and analytical benchmarking have been done, the subtasks need to be scheduled on the different computers in the heterogeneous system.

We assume that there are always a sufficient number of computers available and each subtask can be allocated to the computer which best exploits the nature of parallelism in that subtask. Description of such a system, called Optimal Selection Theory, can be found in [7, 5, 12, 19]. We also assume that subtasks corresponding to only one task are being executed by the system at any given time. So, there is no external interference from any other task.

4.1 Subtask Code Migration

In this strategy a subtask's executable code and input data are migrated by the master to the front-end of the slave computer where it is to be executed. The *subtask-code-migration* approach involves multiple phases of computation and communication. First the number of subtasks to be executed in a phase, and their respective executable codes, are determined. Then the subtasks are migrated one-by-one from the master computer to the slave computers. In order to optimize the execution, the subtasks involving higher computation and communication requirements (large code size and input data) should be migrated to their respective computers before the subtasks with lower communication requirements. As soon as a subtask has been migrated to its slave computer it starts its computation at the slave. So, there is an overlap between the computation at some slaves and communication between the master and some other slaves. Also, the larger subtasks get initiated before the smaller ones. This reduces the time spent by the master at the end of a phase waiting for results to arrive from some subtasks. On the completion of a subtask the slave sends the results to the master. If results from several subtasks arrive at the same time, they are queued up at the master. The phase ends when all the results have been accepted by the master.

If i subtasks have been migrated by the master, before a particular subtask ($i+1^{th}$ subtask), then the expected time at which the result from this subtask reaches the master's queue is given by the following expression :

$$Time = \sum_{j=1}^{i+1}[t_{setup} + t_{transfer} \times (\mid code_j \mid + \mid data_j \mid)] + t_{execute} + t_{setup} + t_{transfer} \times \mid result_{i+1} \mid$$

where $t_{execute}$ is the time to execute the $i+1^{st}$ subtask at the corresponding slave. Figure 3 shows the execution of a phase consisting of n subtasks in the *subtask-code-migration* model. The darker ovals represent actions taken by the master. So, *transmit* t_i indicates the transmission of the executable code and data for the i^{th} task to the appropriate slave.

For *subtask-code-migration* to be unconstrained by the nature of the computers in the system, machine-independent and portable parallel programming languages and tools would be needed. This would require *cross-parallel compilers* for all the machines and appropriate debuggers [12]. The *subtask-code-migration* approach also has a high volume of communication.

4.2 No Code Migration

In this strategy, if a subtask has to be executed in a phase, the master computer sends only the data (not the executable code) for the subtask to the appropriate slave computer. Along with the data, the name of the subtask is also sent. Each slave computer has the executable

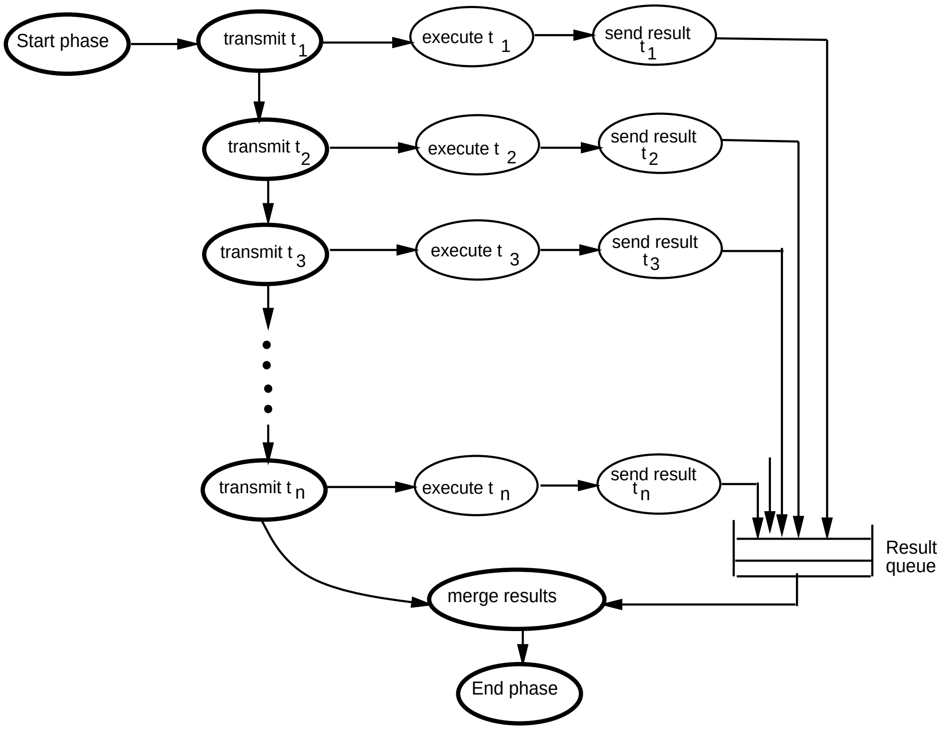

Figure 3: Temporal representation of subtask migration and execution activities in a phase (t_i: subtask i).

code for all the subtasks in its library. The slave invokes the routine corresponding to its subtask from its library on getting the data. The actual implementation of the subtask at a slave employs techniques to provide the best possible performance for the subtask at that computer. So, the implementation of a subtask may be different for different computers.

As soon as a slave finishes its execution, it sends its result back to the master. If results from several subtasks arrive at the same time, they are queued up at the master. When the results from all the subtasks corresponding to a phase have been received by the master the phase comes to an end. The next phase can be started after that. This sequence continues until all the subtasks have been executed. As in the *subtask-code-migration* strategy, Figure 3 represents the execution of a phase consisting of n subtasks. It is to be noted that in the *no-code-migration* strategy, *transmit* t_i corresponds to the transmission of only the data for the i^{th} task from the master to the slave.

Thus the completion time of a phase has a lower bound equal to $t^{max}_{comp} + t^{max}_{comm} + t^{max}_{paging}$. Here t^{max}_{comp} and t^{max}_{comm} correspond to the time taken for computation and communication, respectively, by the longest subtask in the phase. The communication time, t_{comm}^{max} can be expressed as :

$$t_{comm}^{max} = 2 \times t_{setup} \ + t_{transfer} \ \times (\mid data \mid + \mid result \mid)$$

where t_{setup} is the communication set-up time at a node and $t_{transfer}$ is the time required to transfer one unit of data between the master and the front-end of the slave. For the sake of simplicity, we shall assume that $t_{transfer}$ is the same for all the slaves.

Before a slave begins executing this subtask it has to fetch the corresponding executable code from its library on the disk. The parameter t_{paging}^{max} is the time spent fetching pages for the subtask into the main memory of the slave's front-end computer from auxiliary storage. We assume that the main memory at a slave computer is big enough to store the executable code of any subtask and the data that it works with. So, if the code size of the longest subtask is S, there will be $\lceil S/P \rceil$ page faults, where P is the size of a page. If the time to fetch a page from auxiliary memory is equal to t_{disk}, t_{paging}^{max} is equal to $\frac{S}{P}t_{disk}$. A smaller main memory may lead to additional page-faults, whose number would depend on the relative size of the memory and the subtask code, and the locality of reference. Considering the rate at which the size of the main memory of computers is increasing we can safely assume that the executable code, data, and result for a subtask will fit in the main memory. So, we do not consider the additional page-faults mentioned above.

4.3 Qualitative Comparison

Both the strategies have their own strengths and weaknesses. When the executable code is not migrated, great demands are placed on the auxiliary storage of the slaves as the software libraries have to maintained at all the sites. But the only communication required would be to pass the input data and results. Thus, the traffic will be low.

When *subtask-code-migration* is employed, the front-end of the slave computer needs to have just enough main memory to hold the code, input data and result for the subtask. As software libraries do not have to be maintained, a large auxiliary storage is not required. But, the volume of communication is high. Subtask code and input data have to be transferred from the master to the slave and the result has to be communicated in the opposite direction. The communication time contributes to the overall execution time of the task. On the other hand, when the code is not migrated the appropriate routines have to be copied from the software library, which is in the auxiliary memory, to the main memory. The time needed to fetch the corresponding pages from auxiliary memory will contribute to the task completion time.

Disk I/O bandwidth and network communication speed have not grown at the same rate as the speed of processors in the past. Hence, as the processing speed increases, the fraction of the task completion time corresponding to subtask migration and page fault penalty grows. Amdahl's law states that the speedup obtainable is limited by the fraction of the non-parallelizable part of the program. Similarly, communication and disk I/O costs too will limit the possible speedup. So, these costs need to be minimized.

5 System Architecture

In this Section we describe the characteristics of a heterogeneous system composed of a suite of abstract machines connected by a high speed network. The specifications of the abstract machines have been chosen to reflect the performance of the current supercomputers and of those expected to be available in the next few years. These machine specifications have been derived based on the values reported in [4, 11, 17]. The performance specifications of the communication network have been selected based on the information obtained from [3, 6].

5.1 Communication Architecture and Bandwidth

Local area network communication rate has not increased at the same rate as the processing power and memory size. The high performance parallel interface (HIPPI) physical layer standard defines full duplex parallel interfaces that can run at speeds up to 200 megabytes/second [17, 18].

Wavelength-division multiplexing (WDM) protocols are being developed to transmit several 1-gigabit data streams through a single optic-fiber cable [3, 6]. These WDM protocols assume passive optical star-coupled networks with *preassigned* channels. Such protocols have low complexity which is especially important since there is still a great mismatch between the speeds of the optical network interfaces and the bandwidth of the fiber. "Advances in the fast tunable transmitters exceed those in fast tunable receivers; the tuning times for receivers are relatively long as compared to the packet transmission times and their tuning range is small" [6]. We present brief descriptions of two WDM protocols, and their relative merits and demerits.

5.1.1 Communication Using Tunable Transmitters

In this approach each node gets assigned a *home frequency* at which it can receive data. The senders have to tune their transmitters to the frequency of the receiver's home frequency. In the beginning of each phase the master sends messages carrying data, one at a time, to the slaves along different channels corresponding to the home frequencies of the different slaves. Each slave sends its result to the master on the master's channel. If two or more slaves send their results to the master at the same time there is a collision. Since the slaves' receivers are not tuned to the master's channel, a slave can know about a successful transmission to the master only on receiving an acknowledgment from the master. When a slave realizes that there has been a collision, it waits for a random period of time before retrying to send the result. Hence, the communication protocol resembles the ALOHA protocol for communication on a bus [1]. The impact of collisions on the performance of the system depends on the number of slaves contending for the master's channel.

5.1.2 Communication with Tunable Receivers

For the master-slave communication a protocol with tunable receivers and fixed frequency transmitters will have no contention. The master can send data (and code for the *subtask-code-migration* strategy) to the slaves one after the other, once the slaves have tuned to the master's transmission frequency. A slave tries to send its result to the master on finishing its computation. In order to do so, the master needs to be requested to tune to the slave's transmission frequency. If the master is tuned to some other frequency the slave waits until it gets a positive acknowledgment from the master. But this protocol suffers from several problems. It would require a separate control channel dedicated to the task of carrying tuning requests, from a node that wants to transmit data, to the intended receiver and acknowledgments in the other direction. The transmitters should be able to transmit at two frequencies — their own home frequency and the frequency of the control channel. Besides, tuning requests can collide on the control channel unless a time division multiplexed protocol is used for the control channel. Moreover, as already mentioned, tuning a transmitter is much faster than tuning a receiver. Thus a tunable receiver protocol will be very complicated with significant overheads that may offset the advantages due to the absence of contention. Hence, we use the protocol that uses tunable transmitters.

5.1.3 Hierarchical Interconnection

We propose a communication topology, using WDM, that provides hierarchical connectivity. It assumes that communication is being carried out with tunable transmitters. Besides the master and the slaves, there are two other computers called sub-masters. The master directly sends the input data for all the subtasks to the corresponding slaves without the involvement of the sub-masters. At the end of their computations half the slaves send their results to one sub-master and the other half to the other sub-master. The assignments of slaves to sub-masters is done apriori. Once a sub-master gets the results corresponding to all its slaves, it combines and sends them to the master. The phase ends when the master has received results from both the sub-masters. As a result only half the slaves contend for the channel of each sub-master, and only two sub-masters contend for the master's channel. This hierarchical communication topology is shown in Figure 4, with the channels for which contention takes place represented like buses.

In a heterogeneous system where the component computers may be geographically separated, the propagation time of messages cannot be ignored. The speed of light in a medium

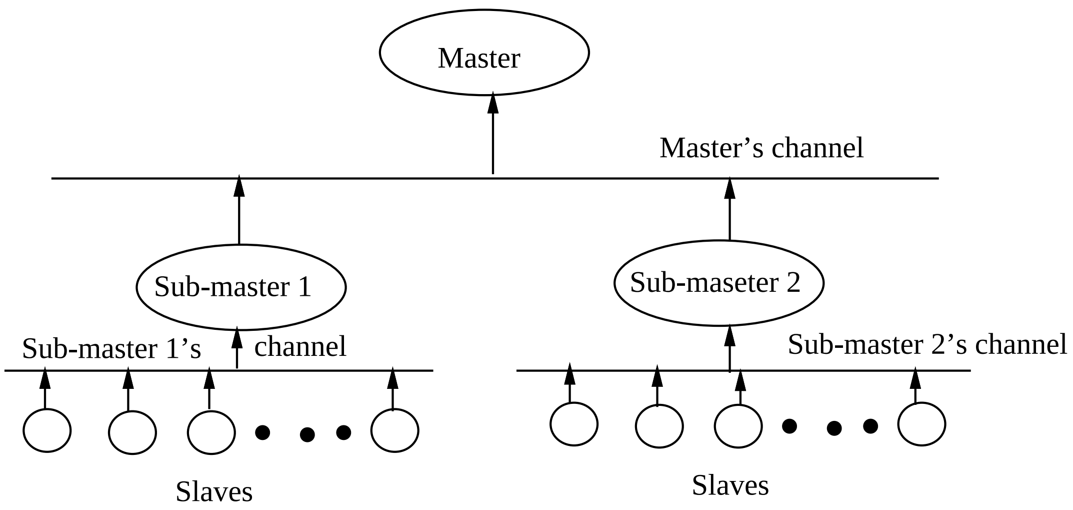

Figure 4: A hierarchical representation of channel contention during the result-receive period using two additional sub-masters.

is inversely proportional to its refractive index. If the mean distance between a master and a slave is d and the speed of light in the optic fiber is s, the propagation time is equal to d/s.

5.2 Processing Speed

Each slave computer is assumed to have a computational rating comparable to a 1024 node CM-5, with peak performance of 131 gigaflops. During normal operation the sustainable performance is assumed to be 10% of the peak. So, the time taken to execute the subtask at a slave is obtained by dividing the computational complexity of the subtask (in gigaflops) by 13.1. There is an implicit assumption that different slaves exploit different kinds of parallelism (SIMD, MIMD, vector etc.) and a subtask gets assigned to the most suitable slave.

The time required to copy the code and data from the front-end of the slave to each of the constituent processors of the slave before the computation starts and the time to gather the result at the front-end also contribute to the total execution time at the slave. For this purpose we assume that the subtask code is broadcast to each constituent processor while a small and distinct fraction of the data is copied from the front-end to individual processors. Emerging high-speed DRAMs like RamBus and RamLink can provide memory bandwidth of up to 500 megabytes/second [11]. The Scalable Coherent Interface (SCI) standard with a potential speed of 1 gigabyte/second has also been proposed [4]. With these technological advances, we assume a transfer rate of 1 gigabyte/second between the front-end and the computational processors at a slave.

5.3 I/O Bandwidth

The I/O bandwidth provided by the disk storage of the current supercomputers is low as compared to their data processing rate. For example, the basic disk storage node and the data vault of CM-5 provide peak transfer rates of 17 megabytes/second and 25 megabytes/second, respectively. The Intel Paragon's disk provides a peak transfer rate of 100 megabytes/second [9]. Disk transfer rates have been rising with technological advancements. They can be further increased through architectural innovations such as connecting an array of disks to the high speed network mentioned above and employing interleaved disk access. Considering these arguments, it should be possible to achieve a peak disk transfer rate of the order of 1 gigabyte/second in the near future. As mentioned earlier, in the *no-code-migration* strategy the code for a subtask is initially fetched from the disk. We assume that the sustainable transfer rate from the disk is 100 megabytes/second in such a situation.

Over the past years, disk access times have not decreased at the same rate as the transfer rates have increased. But this overhead can be hidden in our environment in the following manner: In the first few bytes of the message sent by the master containing the data for a subtask, there is information about the library routine to be invoked at the slave. As soon as the slave gets this information it initiates a request to fetch the code from the disk. So, the disk access time gets overlapped with the data transfer time from the master to the slave. As soon as this data transfer finishes, the slave can start receiving the subtask code from the disk at the rate of 100 megabytes/second.

6 Simulation Experiments and Results

We simulated several executions of tasks that has been broken down into 1000 subtasks using CSIM, a process-oriented discrete-event simulation package [15]. The number of subtasks that can be executed concurrently (degree of parallelism) was uniformly distributed in the range 7–13.

We assumed the executable code size for a subtask to be normally distributed with mean 8 megabytes and a standard deviation of 0.8 megabytes. Usually the object code for a subtask has to be linked with several software libraries that make a significant contribution to the size of the executable code. Hence, an 8 megabyte executable code size was assumed to be quite realistic. Both the input data and result of a subtask were assumed to be dense matrices of equal size. The matrix sizes were assumed to be normally distributed. If the mean number of rows/columns of the matrices is N, then the standard deviation was assumed to be $0.1N$. We evaluated the performance of the two strategies for different data sizes, with N varying from 50 to 1200 and each element of the matrix being a 64-bit floating-point number.

A significant fraction of the computations using $n \times n$ matrices involve LU-decomposition, finding the inverse, multiplication, or some iterative operations (for convergence to a result). The computational complexities of such operations are between $O(N^2)$ and $O(N^3)$. Hence, we modeled the time complexity of a subtask to be either $c_1 n^3$ or $c_2 n^2$, where c_1 and c_2 are integers, selected from a uniform distribution with ranges 5–10 and 20–100, respectively. The computational complexity of a subtask (n^2 or n^3) was also selected randomly.

In our system, the mean distance between a master and a slave was assumed to be about 10 kilometres. The speed of light in the fiber was taken to be 2×10^8 metres/second, giving a mean propagation time of 50 microseconds. We added to that about 10 microseconds as the message set-up time. We assumed the availability of 1 gigabyte/second communication channels between the master and slave computers.

6.1 Effect of Channel Contention

Figure 5 shows the performance of the *subtask-code-migration* and *no-code-migration* strategies when we consider the effect of contention for the master's *home frequency*. All the slaves send their results directly to the master. The unbroken curves represent the hypothetical configuration where each slave sends its result to the master on a separate channel. No contention will take place in such a configuration. These unbroken curves have been shown to contrast the effect of channel contention for different data sizes.

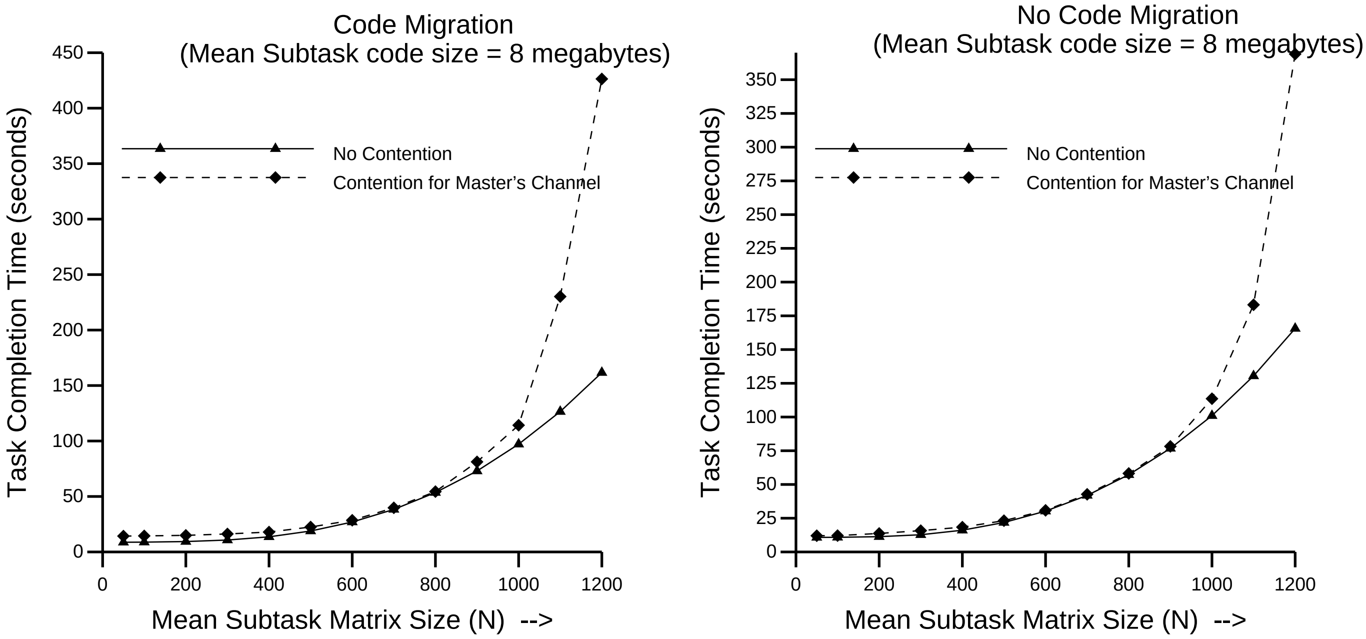

Figure 5: Effect of channel contention on task completion time for two task scheduling strategies with varying data size.

When the size of the subtask result (assumed to be of the same size as the subtasks' input data) is small the communication time is also small. As a result, the collision probability is low and channel contention does not have a significant impact on the task completion time. But for larger result sizes (N = 900–1100) there are many more collisions. Consequently, the task completion time becomes higher. It should be noted that the result size, the number of elements in the matrix, is proportional to the N^2. Hence, a sharp rise in the task completion time is observed for higher values of N.

In Subsection 4.1 it has been suggested that the performance can be improved by migrating the codes for subtasks with higher communication overheads first. The idea was to ensure that the master does not have to wait unnecessarily long for the results of larger subtasks to arrive. But, if such an optimization were to be done, there is also a likelihood that several slaves may finish their subtasks around the same time and be contending for the master's channel. The time saved by migrating the larger tasks first may not offset the delays caused by increased contention. So, this optimization does not work in all situations. A look at Figure 5 supports this conclusion. In the no contention case the *subtask-code-migration* strategy performs slightly better than the one where only the data is communicated. But when contention for the master's channel among the slaves is considered, the *no-code-migration* strategy performs significantly better (has lower task completion time). This is because the slaves starting later had smaller execution times and finished around the same time that the ones spawned earlier did.

6.2 Effect of Degree of Parallelism

The result presented in Subsection 6.1 has assumed that the degree of parallelism (number of slaves spawned in each phase) varies uniformly from 7 to 13. But there may be a sizable number of applications that exhibit a much higher degree of parallelism. If the heterogeneous system has adequate number of computers to run these mutually independent subtasks concurrently, then we would hope that the task completion time would be reduced. But, with increased parallelism there will be a greater demand on the communication network bandwidth at the end of each phase, when the slaves try to send their results to the master. Figure 6 shows the effect of the degree of parallelism on channel contention.

In Figure 6 we see that initially, keeping the mean subtask data size (and consequently, the mean subtask execution time) fixed, an increase in the degree of parallelism leads to a decrease in the task completion time. But subsequent increases in parallelism lead to an increase in the completion time. This is because of increased contention for the master's *home frequency* at the end of a phase. The reduction in time due to increased parallelism is more than offset by the increase in contention. For higher degrees of parallelism the

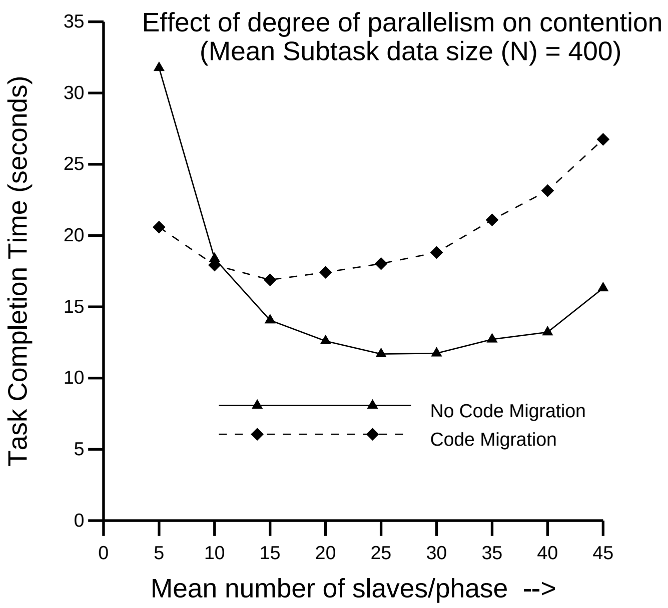

Figure 6: Degree of parallelism and channel contention.

subtask-code-migration strategy has a higher completion time among the two strategies and the opposite is true for lower degrees of parallelism.

As the number of subtasks per phase increases, the start times of the slaves get more and more skewed in the *subtask-code-migration* strategy, because a slave can start only after the code as well as the data for itself and all the preceding slaves in its phase have been communicated by the master. The finish times, too, of the slaves are highly skewed as a result and the finish time of each phase is increased. Hence, the task completion time increases. This increase is more due to the skewed start and finish times of the subtasks than due to increase in contention. Therefore, the *subtask-code-migration* strategy is not able to fully exploit the higher degree of parallelism.

In the strategy with *no-code-migration* the skew is much less for higher degrees of parallelism as only the data has to be migrated. Hence, contention for the master's channel plays a greater role in determining the task completion time. It is apparent from the fact that for degrees of parallelism between 20 and 40 there is not much difference in the task completion time. This is due to increased contention at higher degrees of parallelism.

6.3 Effect of Hierarchical Interconnection

In the *no-code-migration* strategy if there is little variation in the size of the data in a phase, there is not much variance in the execution times of the subtasks. So, several subtasks will be completed around the same time. As all the slaves use a common channel to communicate with the master, there will be contention among the slaves for the channel, while sending the results of their subtasks.

Figure 7 shows the simulation results obtained for the hierarchical interconnection, with *no-code-migration*, when the mean size of a subtask matrix (N) is 400 and 500. For lower degrees of parallelism (number of slaves = 5–20) the hierarchical interconnection performs almost as well as the master-slave topology. This is because the number of slaves is small and the contention is low. Besides, the gains due to less channel-contention in the hierarchical interconnection are offset by the fact that the results of subtasks have to be communicated twice — once from the slaves to the sub-masters and then from the sub-masters to the master. But, this hierarchical interconnection performs much better than the master-slave for higher degrees of parallelism as it leads to a substantial reduction in channel-contention. The improvement in performance is more pronounced for the case when $N = 500$ than when $N = 400$. This is because the larger size of data/result can lead to more contention for the master's channel. Thus, the hierarchical interconnection makes the heterogeneous system scalable.

For larger tasks with many more subtasks, larger size of data and result, and much higher degrees of parallelism, additional levels in the hierarchy between the master and the slaves may be needed. The interconnection may look like a tree with the number of children of interior node increasing as one goes further away from the master. But unlike fat-trees, the communication bandwidth required by the channels at all the levels would be the same. This is because at higher levels of the hierarchy, the increase in the volume of results to be transferred would be compensated for by the reduced contention as fewer nodes will probably be contending for their parent's channel.

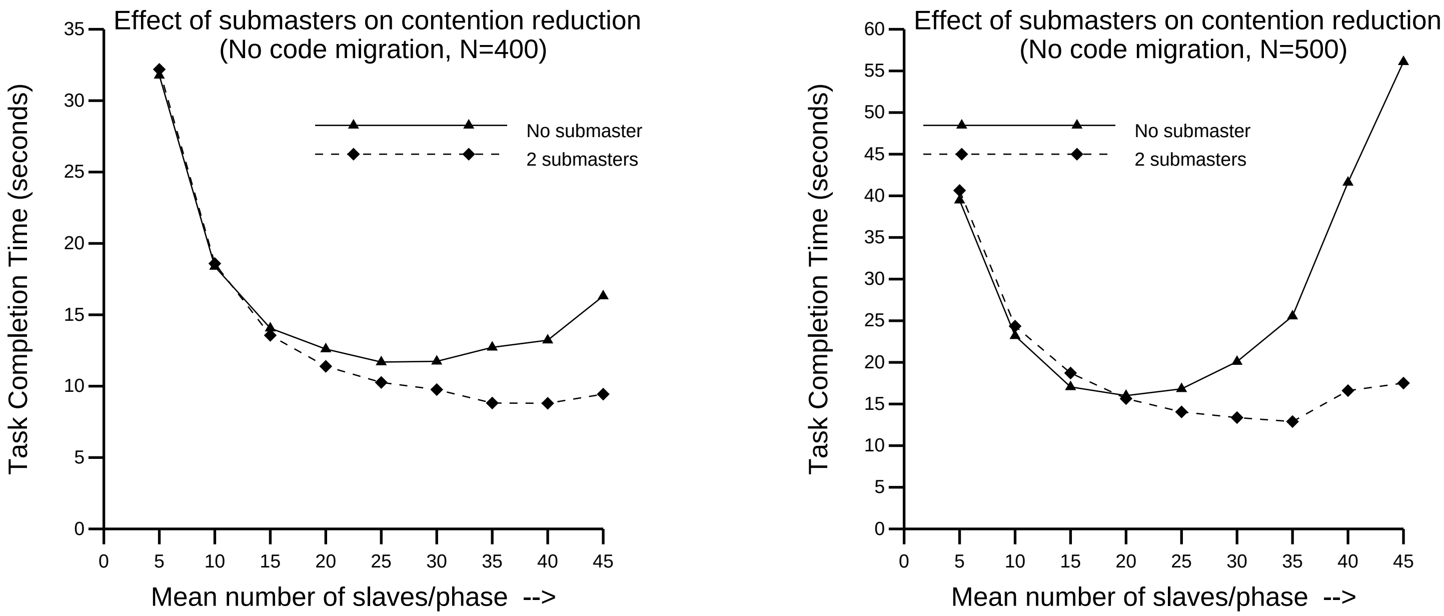

Figure 7: Reduced contention with hierarchical interconnection.

7 Conclusion

We have presented multiphase execution strategies for tasks exhibiting meta-parallelism, on heterogeneous systems. The *no-code-migration* strategy performs better than the *subtask-code-migration* strategy. This shows that increases in the capacity and the I/O bandwidth of the local disks and memory have a great impact on reducing the task completion time. Thus, if a problem with large subtask code sizes is being solved on a system with slow disk I/O, the *subtask-code-migration* strategy may perform better than the other strategy due to the heavy penalty imposed by page-faults. But, on a system with a fast disk I/O the page-fault penalty can be more than offset by the time saved by not having to transmit the code from the master to the slaves.

The communication channels become a bottleneck due to contention, especially for higher degrees of parallelism. The hierarchical communication interconnection on a passive star-coupled optical network leads to a significant reduction in the contention for the master's channel and makes the system scalable. In low contention situations the reduction in contention due to the hierarchical interconnection may be offset by the need to transmit the results twice — from the slaves to the submasters and then from the submasters to the master. Thus, while trying to execute large tasks composed of several subtasks and a high degree of parallelism amongst them, the number of levels in the hierarchy has to be chosen carefully. Fewer levels than the optimum will not be able to reduce contention to the minimum, while more levels than the optimum will increase the execution time due to repeated copying of the results as they are propagated up the hierarchy towards the master.

Laboratory prototypes of several key components required for building the kind of heterogeneous system mentioned in this paper are available. Considering the projected advancements in optic-fiber communication technology and the rate at which processor speeds, and memory and disk I/O bandwidths are increasing it should be feasible to build such systems in the near future. The results obtained from our work can serve as guidelines in the design of such systems.

References

[1] N. Abrahamson. Development of the ALOHANET. *IEEE Transactions on Information Theory*, IT-31:119–123, March 1985.

[2] Emmanuel A. Arnould, Francois J. Bitz, Eric C. Cooper, H. T. Kung, Robert D. Sansom, and Peter A. Steenkiste. The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers. In *Proceedings of the* 3^{rd} *International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS III)*, pages 205–216. ACM/IEEE, April 1989.

[3] Kalyani Bogineni, Krishna M. Sivalingam, and Patrick W. Dowd. Low-Complexity Multiple Access Protocols for Wavelength-Division Multiplexed Photonic Networks. *IEEE Journal on Selected Areas in Communication*, 11(4):590–604, May 1993.

[4] Charles E. Catlett. Balancing Resources. *IEEE Spectrum*, pages 48–55, September 1992.

[5] Song Chen, Mary M. Eshaghian, Ashfaq Khokhar, and Muhammad E. Shaaban. A Selection Theory and Methodology for Heterogeneous Supercomputing. In *Proceedings of the 2^{nd} Workshop on Heterogeneous Processing*, pages 15–22. IEEE, 1993.

[6] Renu Chipalkatti, Zhensheng Zhang, and Anthony S. Acampora. Protocols for Optical Star-Coupled Network Using WDM: Performance and Complexity Study. *IEEE Journal on Selected Areas in Communication*, 11(4):579–589, May 1993.

[7] Mary M. Eshaghian and Richard F. Freund. Cluster-M Paradigm for High-Order Heterogeneous Procedural Specification Computing. In *Proceedings of the Workshop on Heterogeneous Processing*, pages 47–49. IEEE, March 1992.

[8] Richard F. Freund and Howard Jay Siegel. Heterogeneous Processing. *IEEE Computer*, pages 13–17, June 1993.

[9] Paragon XP/S Product Overview. Intel Corporation, 1991.

[10] Christopher R. Johnson, Robert S. MacLeod, and Michael A. Matheson. Computational Medicine: Bioelectric Field Problems. *IEEE Computer*, pages 59–67, October 1993.

[11] Fred Jones. A New Era of Fast Dynamic RAMs. *IEEE Spectrum*, pages 43–49, October 1992.

[12] Ashfaq Khokhar, Viktor K. Prasanna, Muhammad Shaaban, and Cho-Li Wang. Heterogeneous Supercomputing : Problems and Issues. In *Proceedings of the Workshop on Heterogeneous Processing*, pages 3–12. IEEE, March 1992.

[13] H.T. Kung, Robert Sansom, Steven Schlick, Peter Steenkiste, Matthieu Arnould, Francois J. Bitz, Fred Christianson, Eric C. Cooper, Onat Menzilcioglu, Denise Ombres, and Brian Zill. Network-Based Multicomputers : An Emerging Parallel Architecture. In *Supercomputing*, pages 664–673, 1991.

[14] T.G. Lewis. Piece-Wise Scheduling of Composite Task Graphs onto Distributed Memory Parallel Computers. Technical report, Computer Science Department, Oregon State University, 1992.

[15] Herb Schwetman. *CSIM Revision 16*. Microelectronics and Computer Technology Corporation, 3500 West Balcones Center Drive, Austin TX 78759, U.S.A., June 1992.

[16] Gilbert C. Sih and Edward A. Lee. A Compile-Time Scheduling Heuristic for Interconnection- Constrained Heterogeneous Processor Architectures. *IEEE Transactions on Parallel and Distributed Systems*, pages 175–187, February 1993.

[17] J.E. Smith and W.-C. Hsu C. Hsiung. Future General Purpose Supercomputer Architectures. In *Supercomputing*, pages 796–804. IEEE, November 1990.

[18] Ronald J. Vetter, David H.C. Du, and Alan E. Klietz. Network Supercomputing : Experiments with a CRAY-2 to CM-2 HIPPI Connection. In *Proceedings of the Workshop on Heterogeneous Processing*, pages 87–92. IEEE, March 1992.

[19] Mu-Cheng Wang, Shin-Dug Kim, Mark A. Nichols, Richard F. Freund, Howard Jay Siegel, and Wayne G. Nation. Augmenting the Optimal Selection Theory for Supercomputing. In *Proceedings of the Workshop on Heterogeneous Processing*, pages 13–21. IEEE, March 1992.