Appears in Journal of Parallel and Distributed Computing 21:3 (June 1994), 316–322

Program Speedup in a Heterogeneous Computing Network*

Val Donaldson Francine Berman Ramamohan Paturi
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114

Abstract

Program speedup is an important measure of the performance of an algorithm on a parallel machine. Of particular importance is the near linear or superlinear speedup exhibited by the most performance-efficient algorithms for a given system. We describe network and program models for heterogeneous networks, define notions of speedup and superlinear speedup, and observe that speedup consists of both heterogeneous and parallel components. We also consider the case of linear tasks, give a lower bound for the speedup, and show that there is no theoretical upper limit on heterogeneous speedup.

1 Introduction

Program speedup is a widely used measure of the performance of an algorithm on a multiprocessor or multicomputer. Although there are differing notions of the definition of speedup [7], speedup measurements are still almost universally quoted as proof of system efficiency.

A common definition of speedup is that if machine M_1 can solve problem P in time T_1, and machine M_2 solves the same problem in time T_2, then the *speedup* S_P of machine M_2 over machine M_1 on P is the ratio $S_P = T_1/T_2$. In choosing two machines for comparison, M_1 is often viewed as a base machine against which other machines are compared. M_2 typically solves P in less time than M_1, so $T_2 < T_1$, and $S_P > 1$.

This definition of program speedup is especially prevalent in comparing the performance of sequential processors with that of parallel configurations of the same base processor. If M_2 is composed of m instances of the base processor of M_1 (M_2 is a homogeneous parallel machine), the ratio is expressed as $S_P = T_1/T_m$. In this case, since there is a relation between the cost and performance of the single processor of M_1 and the m identical processors of M_2, a question arises as to the relation between the values of m and S_P. If $S_P = m$, we say that machine M_2 achieves *linear* speedup over machine M_1, whereas if $S_P < m$, the speedup

*This research was supported in part by NSF Grants ASC-9106465 and ASC-9301788.

is *sublinear*. *Superlinear* speedup, where $S_P > m$, is especially interesting and somewhat controversial. Quinn [15] and Gustafson [7] discuss arguments for and against the possibility of superlinear speedup.

A heterogeneous network is one in which multiple, possibly dissimilar, machines cooperate in solving a problem. The intuitive advantage of a heterogeneous network is the ability to match segments of code with machines in a manner which best utilizes the capabilities of each machine. As with homogeneous parallel computers, speedup results have been reported in the literature, but heterogeneous speedup has not been consistently defined. Moreover, the role of superlinear speedup has not been explored. We proceed in Section 2 to outline a model of computation for a general heterogeneous network. In Section 3, we use this model to discuss program speedup and superlinear speedup. In Section 4, we discuss some of the practical aspects of speedup in heterogeneous networks. We conclude in Section 5. Our goal is to provide consistent and reasonable definitions for these terms and to discuss their potential as a measurement of the performance of heterogeneous networks.

2 Model Definitions

Before we define the execution of a program on a heterogeneous network, we must first delineate our network and program models. Our heterogeneous network and program models are similar to the models defined in [1, 4, 8, 14, 18] etc. with some modifications.

2.1 Heterogeneous Network Model

Our heterogeneous network model is a set $H = \{M_1, M_2, \ldots, M_m\}$ of m distinct machines. Each machine can communicate with any other machine. The machines in H may include any combination of sequential, VLIW, dataflow, vector, parallel or other processors. A parallel machine such as a hypercube, for example, may be viewed as either a single uniform parallel processor, a set of sequential processors, or even a collection of smaller hypercubes, but the designation is fixed and may not vary. Our primary model of program execution on H assumes that each machine M_j may execute at most one program task at a time. In this case, we say the network is *nondecomposable*. If multiple tasks can be executed concurrently on an individual machine, we call the network *decomposable*. In this paper, we assume that H is nondecomposable.

2.2 Program Model

The basis of our program model is the well-known *task graph*. A task graph for a program P is a directed acyclic graph $G_P = (V, E)$. $V = \{t_1, t_2, \ldots, t_n\}$ is a partition of P consisting of n tasks. Tasks are constrained to be *atomic*—each task is executed to completion by exactly one machine without intermediate communication or synchronization with other tasks. $E \subseteq V \times V$, and $(t_i, t_k) \in E$ iff t_i must execute before t_k due to data dependence or task synchronization requirements of P. Since G_P is acyclic, E may be viewed as imposing a partial order on the tasks in V. Without loss of generality, we assume that G_P has unique

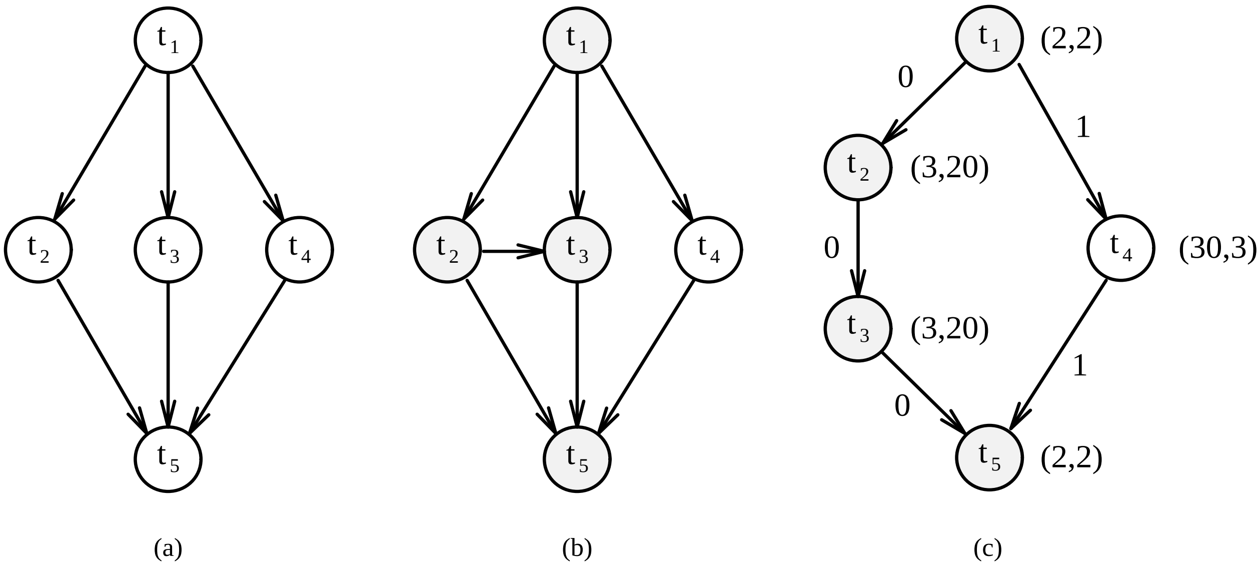

Figure 1: (a) Task graph. (b) Scheduled task graph on two machines—shaded tasks are mapped to one machine. (c) Scheduled task graph without redundant edges. Timing values in parentheses for task t_i are ($T_{t_i}^{M_1}$,$T_{t_i}^{M_2}$). $S_P = 40/10 = 4$.

entry and exit nodes identified with tasks t_1 and t_n, and for any task t_i, there is a path from t_1 to t_i, and a path from t_i to t_n. Figure 1a is an example of a task graph.

For a program consisting of n tasks and a network of m machines, there are m^n possible mappings of tasks to machines. The problem of choosing an optimal mapping is NP-complete [6], even with a number of simplifying restrictions. Stone [16] solves the heterogeneous network scheduling problem for the case $m = 2$ machines using a network flow algorithm. Bokhari [2] solves the problem for arbitrary m and n when the task graph is a tree. Related results are derived in [3, 13, 17].

A (nonpreemptive) mapping of tasks to machines is a function $\Pi : V \to H$. If Π maps all n tasks to the same machine, the mapping is sequential. If the codomain of Π contains more than one machine, the mapping may permit concurrent execution of tasks on distinct machines. Except when considering sequential mappings, we assume that the mapping Π is onto—all m machines execute at least one task. If $n > m$, any mapping of tasks to machines must map multiple tasks to at least one machine.

For nondecomposable heterogeneous networks, the requirement that each machine may execute at most one task at a time can be enforced by augmenting the task graph. A *scheduled task graph* is a task graph with additional edges defining a total order on all tasks mapped to the same machine. If similar edges already exist in E, the additional scheduling edges are redundant, and are not added (the scheduled task graph is *not* a multigraph). Edges which are transitively redundant may also be deleted if desired, although when data communication times are considered, otherwise redundant edges may need to be retained. Figures 1b and 1c are examples of scheduled task graphs.

In the rest of the paper, unless otherwise specified, we assume that G_P is a scheduled task graph, and that the mapping Π on G_P is fixed. Many of the quantities defined below depend on Π, although this dependence is not always explicitly stated.

Timing values are associated with the components of G_P. For each of the n tasks and each of the m machines, $T_{t_i}^{M_j}$ is the time required to execute task t_i on machine M_j. $T_{t_i}^{H}$ is defined to be the value $T_{t_i}^{M_j}$ such that $\Pi(t_i) = M_j$. We also define timing values for communication between tasks. For each edge $(t_i, t_k) \in E$, N_{t_i,t_k} is the time required for data communication (including data representation conversion if required) and synchronization between task t_i on machine $\Pi(t_i)$ and task t_k on machine $\Pi(t_k)$. All task and network times are assumed to be fixed, finite, and nonnegative. It is often assumed that if tasks t_i and t_k are mapped to the same machine, $N_{t_i,t_k} = 0$. This simplification can be justified by noting that relative to network times, communication times internal to individual machines in the network are typically negligible. *In this paper, we assume zero communication times for intramachine communication.*

The preceding model contains an additional simplification. In practice, both network times and task times will vary, depending on the state of the machines and the network. We make the common simplifying assumption that the computation times of individual tasks and the communication times between tasks are constant.

With these assumptions, we can define T_P^H, the time to execute program P on a heterogeneous network H, using a macro dataflow approach. Let G_P be a scheduled task graph. Define $\mathrm{est}(t_1)$, the *earliest starting time* of entry task t_1 to be 0, and for all remaining tasks $t_i, i = 2, 3, \ldots, n$, define

$$\mathrm{est}(t_i) = \max_{(t_k,t_i)\in E} \{\mathrm{est}(t_k) + T_{t_k}^H + N_{t_k,t_i}\}$$

Then

$$T_P^H = \mathrm{est}(t_n) + T_{t_n}^H$$

We are also interested in $T_P^{M_j}$, the time required for each individual machine $M_j \in H$ to execute P. Since network H is nondecomposable and intramachine communication times are assumed to be negligible, for $j = 1, 2, \ldots, m$, we define

$$T_P^{M_j} = \sum_{i=1}^{n} T_{t_i}^{M_j}$$

For the remainder of the paper, we assume that the program P, the network of machines H, the scheduled task graph G_P with timing values, and the mapping Π are fixed.

3 Program Speedup for Heterogeneous Networks

In this section we give a definition of speedup for heterogeneous networks, and consider some of the consequences of this definition.

Definition 1 The *speedup* S_P of heterogeneous network H on problem P under mapping Π is

$$S_P = \frac{\min\{T_P^{M_1}, T_P^{M_2}, \ldots, T_P^{M_m}\}}{T_P^H}$$

□

For notational convenience, we assume that the minimum in Definition 1 is achieved by machine M_1. In other words, machine M_1 executes program P as fast or faster than the other machines. We then have $S_P = T_P^{M_1}/T_P^H$. If $S_P > 1$ then the heterogeneous network performs better than the best individual machine.

Program speedup for heterogeneous networks is discussed in several papers. Freund et al. [5] discuss how speedups for code segments on individual machines may vary. Wang et al. [18] define a notion of "actual speedup" as the product of optimal speedup, available parallelism and a measure of how well a code segment matches its machine. Moore [11] defines a similar notion of speedup for the particular instance of a heterogeneous network of a SIMD and a MIMD machine. Our goal is to establish a definition of program speedup for general heterogeneous networks which is consistent with the usual definition of program speedup for a uniform set of processors. We remark that an interesting feature of this definition of heterogeneous speedup is that it measures both parallelism and heterogeneity. We discuss this observation further in the following subsection.

There are two values of S_P which are especially interesting. The first is the boundary between program speedup and program "slowdown." Clearly, S_P should be greater than one, since otherwise the use of H to solve P is counterproductive. The second interesting value of S_P is m.

Definition 2 The speedup $S_P = T_P^{M_1}/T_P^H$ of heterogeneous network H on problem P under mapping Π is *superlinear* iff $S_P > m$, where m is the number of component machines in H. □

In the homogeneous case, parallel algorithms which exhibit linear or superlinear speedup are considered particularly efficient. In the heterogeneous case, particularly efficient algorithms should execute faster on the heterogeneous network of m machines than they would on m copies of the fastest machine in H, i.e. $T_P^H < T_P^{M_1}/m$. Hence we define the speedup of a heterogeneous network on a problem to be superlinear when the speedup exceeds $|H| = m$.

In the rest of this section we analyze program speedup in terms of component system times. In Section 3.1 we consider general task graphs, and relate program speedup to processor, communication, and synchronization times. In Section 3.2, we discuss the special case of linear task graphs, and consider bounds on speedup in terms of three system parameters.

3.1 General Task Graphs

In this subsection, we consider the general case where G_P is an arbitrary directed acyclic graph. Let $\rho = (t_1, ..., t_n)$ be any path from the entry node to the exit node in G_P; ρ need not be a critical path. For $j = 1, 2, \ldots, m$, define

$$\Delta_\rho^{M_j} = \begin{cases} \sum_{\{i|t_i \in \rho \,\wedge\, \Pi(t_i)=M_j\}} (T_{t_i}^{M_1} - T_{t_i}^{H}) & \text{if } \{i|t_i \in \rho \,\wedge\, \Pi(t_i) = M_j\} \neq \phi \\ 0 & \text{otherwise} \end{cases}$$

$\Delta_\rho^{M_j}$ is the difference between the execution time of M_1 on the tasks on ρ assigned to M_j by Π, and the execution time of M_j on these tasks. This is the heterogeneous reduction in execution time along ρ due to M_j. $\Delta_\rho^{M_1} = 0$ is immediate from the definition. For a "good" mapping Π, we would expect that $\Delta_\rho^{M_j}$ would be nonnegative for most M_j on most paths ρ.

We define two additional quantities:

$$T_\rho^{M_1} = \sum_{t_i \in \rho} T_{t_i}^{M_1} \qquad \text{and} \qquad O_\rho = T_P^H - \sum_{t_i \in \rho} T_{t_i}^H$$

$T_\rho^{M_1}$ is the time base machine M_1 takes to execute all tasks along ρ. O_ρ is the aggregate overhead required by network H for data communication and synchronization along ρ. If ρ is a critical path in G_P, then O_ρ is just the sum of the corresponding network times along ρ. Otherwise, O_ρ also includes time spent waiting for task precedence constraints to be satisfied.

Proposition 1 Let H be a nondecomposable heterogeneous network with scheduled task graph G_P. Let ρ be a path in G_P from entry task t_1 to exit task t_n. Let M_1 be the fastest machine in H on P, executing P in time $T_P^{M_1} = \sum_{i=1}^n T_{t_i}^{M_1}$. Then

$$S_P \;=\; \frac{T_P^{M_1}}{T_\rho^{M_1} - \sum_{j=2}^m \Delta_\rho^{M_j} + O_\rho}$$

Proof Follows from the definitions. □

Proposition 1 provides a decomposition of T_P^H into the execution time that reference machine M_1 spends on tasks in a path ρ ($T_\rho^{M_1}$), the "extra" time M_1 takes to compute these tasks over the time network H takes to compute the same tasks ($\sum_{j=2}^m \Delta_\rho^{M_j}$), and the overhead of communication and synchronization in the network on the chosen path (O_ρ). When H is a homogeneous collection of machines, the middle term in the denominator is zero, and Proposition 1 defaults to $S_P = T_P^{M_1}/(T_\rho^{M_1} + O_\rho)$, which is the usual notion of parallel speedup.

Speedup is derived from two distinct components of H: heterogeneity and parallelism. Proposition 1 can be rewritten as:

$$S_P \;=\; \frac{T_P^{M_1}}{T_P^{M_1} - (T_P^{M_1} - T_\rho^{M_1}) - \sum_{j=2}^m \Delta_\rho^{M_j} + O_\rho}$$

Starting from the best sequential time $T_P^{M_1}$, speedup in H is increased if either parallelism $(T_P^{M_1} - T_\rho^{M_1})$ or heterogeneity $(\sum_{j=2}^m \Delta_\rho^{M_j})$ is increased, or the overhead O_ρ is decreased.

An interesting question arises as to what limits speedup S_P may have. Although we could address this issue for general task graphs, we get a more interesting result in the next subsection. As an example of the general case, however, Figure 1*c* shows an example task graph exhibiting superlinear speedup: a speedup of 4 with 2 machines.

3.2 Linear Task Graphs

In this subsection, we consider the special case of a linear task graph $G_P = \{t_1, ..., t_n\}$. Each task t_i must complete execution before task t_{i+1} is allowed to begin. The linear graph assumption forces the heterogeneous network H (as well as machine M_1) to execute tasks serially. Since there is only one path from t_1 to t_n, we drop the subscript ρ of the previous subsection, and let $O = \sum_{i=1}^{n-1} N_{t_i,t_{i+1}}$. Linear task graphs have been considered by [5, 11, 18]. Heterogeneous applications like the MaxSegs algorithm [12] are modeled by linear task graphs.

The case of linear tasks is especially interesting since the best assignment of tasks to machines can be easily determined for an arbitrary set of heterogeneous machines and arbitrary communication times between machines executing adjacent tasks. Communication times between successive tasks are a function of the machines assigned to the tasks as well as the tasks themselves. Bokhari [2] describes an $O(m^2n)$ shortest path algorithm which can be used to derive the optimal assignment.

A particularly appealing assignment of linear tasks to machines is one which assigns each task t_i to a machine for which $T_{t_i}^{M_j}$ is minimized with respect to j, i.e. $T_{t_i}^H = \min_j T_{t_i}^{M_j}$ (if more than one mapping satisfies this requirement, any one may be chosen). Call this mapping Π_0. Note that if communication times are relatively high, Π_0 may not be the most performance-efficient mapping. We use Π_0 to derive bounds for S_P. Note that an analogous mapping is less appealing for general task graphs, since it may conflict with concurrent task execution.

We proceed to give a lower bound on the speedup achieved by assignment Π_0. For $j = 1, 2, \ldots, m$, define

$$\alpha_j = \frac{\sum_{\{i|\Pi(t_i)=M_j\}} T_{t_i}^H}{\sum_{i=1}^n T_{t_i}^H}$$

For each j, $0 \le \alpha_j \le 1$, and α_j is the fraction of the heterogeneous computation time of P spent on machine M_j. Define a summary quantity

$$\alpha = \max\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$$

Since average$\{\alpha_j\} = 1/m$, $1/m \le \alpha \le 1$. α measures how well Π_0 is balanced with respect to utilization of machines in H. If α is close to $1/m$, Π_0 is well balanced.

Let "second-min" select the smallest element after the minimum (if the minimum value is repeated, the second minimum equals the minimum), and for $i = 1, 2, \ldots, n$, define

$$\beta_i = \frac{\text{second-min}\{T_{t_i}^{M_1}, T_{t_i}^{M_2}, \ldots, T_{t_i}^{M_m}\}}{\min\{T_{t_i}^{M_1}, T_{t_i}^{M_2}, \ldots, T_{t_i}^{M_m}\}}$$

Each $\beta_i \ge 1$, and measures how specialized task t_i is, where a more specialized task requires a more specific machine architecture for fast execution, rather than a general purpose machine. Indirectly, β_i measures how "indispensable" some machine in H is. Define

$$\beta = \min\{\beta_1, \beta_2, \ldots, \beta_n\}$$

$\beta \ge 1$, and summarizes how useful or indispensable the machines in H are for executing P under partition G_P and mapping Π_0. A larger value of β suggests that all machines contribute significantly to improving the execution time of P. There is no theoretical upper limit on the value of β.

Finally, define $\gamma = O / \sum_{i=1}^n T_{t_i}^H$ as the ratio of communication time to computation time for execution of P on H. A lower bound for S_P can then be given in terms of α, β and γ.

	M_1	M_2	M_3
t_1	1	k	k
t_2	k	1	k
t_3	k	k	1

$\alpha = \alpha_1 = \alpha_2 = \alpha_3 = 1/3$

$\beta = \beta_1 = \beta_2 = \beta_3 = k/1 = k$

$\gamma = c/3$

$$S_P = \frac{2k+1}{c+3} \qquad \frac{(1-\alpha)\beta + \alpha}{1+\gamma} = \frac{2k+1}{c+3}$$

Table I: Task times and speedup parameters for three machines executing a linear sequence of three tasks. Task time $k \geq 1$, but is otherwise arbitrary; communication time $O = c$.

Proposition 2 For linear task graphs G_P and mapping Π_0, we have

$$S_P \;\; \geq \;\; \frac{(1-\alpha)\beta + \alpha}{1+\gamma}$$

where α, β and γ are as defined above.

Proof Let $I_1 = \{i | \Pi_0(t_i) = M_1\}$ be the set of task indices assigned to M_1 by Π_0. Then

$$\begin{aligned}
S_P &= \frac{T_P^{M_1}}{\sum_{i=1}^n T_{t_i}^H + O} \\
&= \frac{\sum_{i \notin I_1} T_{t_i}^{M_1} + \sum_{i \in I_1} T_{t_i}^{M_1}}{\sum_{i=1}^n T_{t_i}^H + O} \\
&\geq \frac{\beta \sum_{i \notin I_1} T_{t_i}^H + \sum_{i \in I_1} T_{t_i}^{M_1}}{\sum_{i=1}^n T_{t_i}^H + O} && \text{(using definition of } \beta) \\
&= \frac{\beta \sum_{i \notin I_1} T_{t_i}^H + (\beta \sum_{i \in I_1} T_{t_i}^{M_1} - \beta \sum_{i \in I_1} T_{t_i}^{M_1}) + \sum_{i \in I_1} T_{t_i}^{M_1}}{\sum_{i=1}^n T_{t_i}^H + O} \\
&= \frac{\beta \sum_{i=1}^n T_{t_i}^H - (\beta - 1) \sum_{i \in I_1} T_{t_i}^{M_1}}{(\sum_{i=1}^n T_{t_i}^H)(1+\gamma)} && \text{(using definition of } \gamma) \\
&\geq \frac{\beta - (\beta - 1)\alpha}{1+\gamma} && \text{(using definition of } \alpha) \\
&= \frac{(1-\alpha)\beta + \alpha}{1+\gamma}
\end{aligned}$$

□

Table I is a simple example of the application of Proposition 2. This example can be used to show a somewhat surprising result.

Proposition 3 Based on the definition of speedup in Definition 1 and the network and program models defined above, there is no theoretical upper limit on potential speedup in a heterogeneous network. This result holds even without exploiting parallelism in G_P. In particular, superlinear speedup is possible.

Proof A simple proof can be based on the example in Table I. The speedup S_P of this linear task sequence is $(2k+1)/(c+3)$, for communication cost c and arbitrary $k \geq 1$. For fixed c, as k increases without bound, so does S_P. $\Box$

We remark that in practice, larger speedup values may be attained by choosing one or more (homogeneous) parallel machines as component machines of H. In this case, the value m specified in Definition 2 is the number of machines in network H, including multiprocessors/multicomputers, and not the number of individual processing elements from which the parallel machines are built.

4 Heterogeneous Applications

The results of the previous section show that based on Definition 1, speedup in a general heterogeneous network is due to two different sources: the heterogeneity of machines in H and the parallel execution of tasks in G_P.

Each machine in H should contribute to improving the execution time of P by being the best machine for one or more tasks in G_P, subject to the additional constraints imposed by varying data communication times between different pairs of machines in H. If this does not hold for some machine in H, the utility of that machine is limited to providing concurrent computing resources, based on the parallelism available in G_P. In the restricted situation where G_P is a linear task graph, in many cases each machine *must* be fastest on at least one task, since otherwise there is no advantage to incurring additional intermachine data communication delays to execute a task on such a machine. Exceptions to this statement occur only when communication times are significant. If a machine which is not fastest on any task nevertheless is capable of fast data communication with the machines executing preceding and succeeding tasks, then the machine may still be the best choice for some task.

Although these results show that heterogeneity provides added potential for speedup over the corresponding speedup of a homogeneous parallel processor, the important question remains as to whether or not heterogeneous networks are useful for improving execution performance on actual application programs. Due to the difficulties of programming in a heterogeneous environment there are as yet few applications which have exploited heterogeneity at the network level for single programs, although recently developed systems such as HeNCE [1] may change this. Kung et al [9], describe three different heterogeneous applications, but only report execution results for running the applications on a homogeneous network of Sun 4 workstations. Mechoso et al. [10] have designed a climate modeling application to run on several machines, but only report execution times for executing the different tasks on separate processors of a multiprocessor Cray Y-MP.

Nicholas et al. [12] report on a heterogeneous application which runs as a sequence of two tasks on a Thinking Machines CM-2 and Cray Y-MP. Each machine implements a version of the MaxSegs dynamic programming algorithm, which compares a DNA or protein query sequence against a large set of library sequences. The CM-2 version is a quick version used to filter out library sequences which fall below a user-specified similarity threshold

	CM-2	Y-MP	Heterogeneous Network
Filter Task Time	6.0	74.3	6.0
Communication Time	—	—	0.5
Backtrack Task Time	?	0.9	0.9
Total Time	?	75.2	7.4

Table II: Task execution times (CPU times in minutes) for the heterogeneous MaxSegs algorithm as reported in [12].

value. Similarity scores for the remaining library sequences are then recalculated on the Y-MP, as the first step in calculating actual sequence alignments using a backtracking scheme. Additionally, the Y-MP version is capable of finding multiple alignments for the same library sequence. For comparison, the filtering version was also implemented on the Y-MP. The results of running the algorithm for several input data sets on both the heterogeneous CM-2/Y-MP network, as well as a comparable Y-MP only version are reported. No CM-2 only results are reported. Precise network communication times are not given, but the researchers suggest that 0.5 minutes is a conservative estimate of the actual data communication times between machines. The results most favorable to the heterogeneous network are summarized in Table II.

The time to execute MaxSegs on the heterogeneous network is 7.4 minutes, and the time to execute the algorithm on the Y-MP alone is 75.2 minutes, so the speedup of the heterogeneous network over the Y-MP is $S_P = 75.2/7.4 = 10.2$. Although results for a CM-2 only execution are not available, note that in order to obtain a heterogeneous speedup greater than 2, the CM-2 must require more than 8.8 minutes to execute the backtracking task.

5 Conclusions

The technology for executing programs on a heterogeneous network of machines has advanced to a point where it can become an increasingly useful alternative for solving large problems. We have considered a theoretical basis for calculating speedup in a heterogeneous environment, and given definitions for speedup and superlinear speedup in a heterogeneous network. Our definition of speedup defaults to the usual notion of speedup when the parallel system is homogeneous.

Based on these definitions, we have observed that speedup for an arbitrary task graph can be viewed as having both a heterogeneous and a parallel component. Additional analysis of the special case of linear task graphs shows that in a heterogeneous network, not only is superlinear speedup possible, but unbounded speedup is possible even without exploiting parallelism across machines. We also obtained a lower bound on the speedup derived from several summary parameters for a mapping Π_0, which assigns each task to its "best" machine.

As a practical example of the use of heterogeneity, the MaxSegs algorithm suggests that heterogeneous computing networks may provide significant speedups over homogeneous systems. We have presented a framework in which to discuss performance improvements due to heterogeneity. We hope that further progress in developing a theoretical framework and realistic performance measures will shed more light on the task of identifying applications whose performance can be significantly improved on heterogeneous networks.

Acknowledgments

We are grateful to Vaidy Sunderam and the referees for their helpful comments and suggestions.

References

[1] Beguelin, A., Dongarra, J.J., Geist, G.A., Manchek, R., and Sunderam, V.S. Graphical development tools for network-based concurrent supercomputing. *Proc. Supercomputing '91.* Albuquerque, NM, Nov. 1991, pp. 435–444.

[2] Bokhari, S.H. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Trans. Software Engrg.* SE-7, 6 (Nov. 1981), 583–589.

[3] Bokhari, S.H. Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Trans. Comput.* 37, 1 (Jan. 1988), 48–57.

[4] Eshagian, M.M., and Freund, R.F. Cluster-M paradigms for high-order heterogeneous procedural specification computing. *Proc. Workshop on Heterogeneous Processing.* Beverly Hills, CA, Mar. 1992, pp. 47–49.

[5] Freund, R.F. Optimal selection theory for superconcurrency. *Proc. Supercomputing '89.* Reno, NV, Nov. 1989, pp. 699–703.

[6] Garey, M.R., and Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[7] Gustafson, J.L. The consequences of fixed time performance measurement. *Proc. 25th Hawaii International Conference on System Sciences.* Jan. 1992, pp. 113–124.

[8] Khokhar, A., Prasanna, V., Shaaban, M., and Wang, C.-L. Heterogeneous supercomputing: problems and issues. *Proc. Workshop on Heterogeneous Processing.* Beverly Hills, CA, Mar. 1992, pp. 3–12.

[9] Kung, H.T., Steenkiste, P., Gubitoso, M., and Khaira, M. Parallelizing a new class of large applications over high-speed networks. *Symposium on Principles and Practice of Parallel Programming.* Williamsburg, VA, Apr. 1991, pp. 167–177.

[10] Mechoso, C.R., Ma, C.-C., Farrara, J.D., Spahr, J.A., and Moore, R.W. Distribution of a climate model across high-speed networks. *Proc. Supercomputing '91.* Albuquerque, NM, Nov. 1991, pp. 253–260.

[11] Moore, R.W. Distributing applications across wide area networks. *Proc. Los Alamos Gigabit Testbed Workshop.* Santa Fe, NM, Jan. 1990.

[12] Nicholas, H., Giras, G., Hartonas-Garmhausen, V., Kopko M., Maher, C., and Ropelewski, A. Distributing the comparison of DNA and protein sequences across heterogeneous supercomputers. *Proc. Supercomputing '91.* Albuquerque, NM, Nov. 1991, pp. 139–146.

[13] Nicol, D.M., and O'Hallaron, D.R. Improved algorithms for mapping pipelined and parallel computations. *IEEE Trans. Comput.* 40, 3 (Mar. 1991), 295–306.

[14] Prakash, S., and Parker, A.C. A design method for optimal synthesis of application-specific heterogeneous multiprocessor systems. *Proc. Workshop on Heterogeneous Processing.* Beverly Hills, CA, Mar. 1992, pp. 75–80.

[15] Quinn, M.J. *Designing Efficient Algorithms for Parallel Computers.* McGraw-Hill, New York, 1987.

[16] Stone, H.S. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Software Engrg.* SE-3, 1 (Jan. 1977), 85–93.

[17] Towsley, D. Allocating programs containing branches and loops within a multiple processor system. *IEEE Trans. Software Engrg.* SE-12, 10 (Oct. 1986), 1018–1024.

[18] Wang, M.-C., Kim, S.-D., Nichols, M.A., Freund, R.F., Siegel, H.J., and Nation, W.G. Augmenting the optimal selection theory for superconcurrency. *Proc. Workshop on Heterogeneous Processing.* Beverly Hills, CA, Mar. 1992, pp. 13–22.