Average-Case Analysis of Isospeed Scalability of Parallel Computations on Multiprocessors

Keqin Li*
Dept. of Mathematics and Computer Sci.
State University of New York
New Paltz, New York 12561-2499
Email: li@mcs.newpaltz.edu

Xian-He Sun†
Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana 70803-4020
Email: sun@bit.csc.lsu.edu

Abstract

We investigate the average-case speed and scalability of parallel algorithms executing on multiprocessors. Our performance metrics are average-speed and isospeed scalability. By modeling parallel algorithms on multiprocessors using task precedence graphs, we are mainly interested in the effects of synchronization overhead and load imbalance on the performance of parallel computations. Thus, we focus on the structures of parallel computations, whose inherent sequential parts are limitations to high performance. For several typical classes of task graphs, including iterative computations, search trees, partitioning algorithms, and diamond dags, we derive the growth rate of the number of tasks as well as isospeed scalability in keeping constant average-speed.

1 Introduction

The high computational power of a parallel computer may not be realized in solving a given application, because the achievable efficiency of a parallel computation can drop quickly as system size increases. To evaluate the ability of maintaining performance, scalability has been recognized as an important property of algorithm-machine combinations. As parallel computing becomes widely available, the issue of scalability of parallel computing systems has become increasingly important. Though scalability has been defined in different ways for different applications and considerations [2, 5, 9, 12, 14], the scalability of a parallel system is essentially a measure of its capability to increase performance in proportion to the number of processors, where a parallel system consists of a parallel algorithm and a parallel machine that supports the implementation of the algorithm. Loosely speaking, scalability measures the overall effect of parallel processing overheads on performance when system and problem sizes scale up. Scalability has found its important role in parallel algorithm and architecture development in recent years [3, 6]. The importance of scalability analysis is best reflected by the fact that one can predict the performance of a parallel system with a large number of processors from the known performance of a system with a few processors. The scalability of many algorithm-machine combinations have been studied extensively, and a rich literature exists [1, 4, 6, 7, 8, 10, 11].

It is well known that it is the overhead of parallel processing that reduces the scalability of parallel systems. There are essentially three sources of overhead in parallel computations, namely, communication and synchronization time, load imbalance, and extra computation [6]. Such overheads come from inherent sequential parts of a problem, poor design of a parallel algorithm, architectural limitations of a parallel machine, and inefficient system support. Scalability analysis and prediction provide a powerful methodology to reveal the impact of these overheads on the performance of parallel systems.

We observe that current studies of scalability have their limitations. Most of current studies are focused on the scalability of a given algorithm-machine combination [5], or on the peak performance of a parallel system [9]. While both approaches have their practical importance, the former tends to be unique for individual algorithm or architecture development, and the latter tends to be overly optimistic. Therefore, we take a different approach, namely, conducting average-case analysis of the scalability of a class of algorithm-machine combinations using a well defined and general methodology.

2 Isospeed Scalability

A parallel computation can be specified by an algorithm-machine combination $C = (A, M)$, i.e., a parallel algorithm A implemented on a parallel computer system M. Two main driving forces for parallel processing are faster execution time and solving larger problems. Performance analysis, in particular, scalability analysis, is based on the parallel execution time and the problem size. Though there is still no unified definition, the *problem size* is usually defined as the amount of basic operations (e.g., the number of floating-point operations in a scientific computation) performed in running algorithm A on machine M. This is actually a measure of the amount of useful work done to solve a problem, and hence, is also called the *work* of a parallel computation C, denoted by W. We use $T(W, P)$ to represent the *execution time* of algorithm A when the problem size is W and there are P processors in machine M. The parameter P is the *size* of the parallel machine M. To combine the two considerations of problem size and execution time together, the *speed* of the parallel computation C is defined as the ratio of the problem size W divided by the execution time $T(W, P)$, i.e., $S(W, P) = W/T(W, P)$. The *average-speed* is the above speed divided by the machine size P, i.e., $\bar{S}(W, P) = S(W, P)/P = W/(T(W, P)P)$.

The parallel execution time $T(W, P)$ can be divided into two parts, namely, computation time and overhead of parallel processing. That is, we can write $T(W, P) = (W + T_o(W, P))/P$, where $T_o(W, P)$ is the overhead of implementing algorithm A on machine M. The above equation gives rises to $\bar{S}(W, P) = W/(W + T_o(W, P))$. Usually, if the system size P is increased while the problem size W is kept constant, the average-speed decreases because the overhead $T_o(W, P)$ increases with P. On

*Supported by National Aeronautics and Space Administration and the Research Foundation of State University of New York through NASA/University Joint Venture in Space Science Program under Grant NAG8-1313.

†Supported in part by NSF under Grant ASC-9720215, by LSU 1998 COR award, and by Louisiana Education Quality Support Fund.

the other hand, the average-speed increases with W if P is fixed, because increasing the problem size typically reduces the overhead/computation ratio $T_o(W,P)/W$. Thus, it is possible to maintain the average-speed $\bar{S}(W,P)$ at certain constant (or, to keep speed $S(W,P)$ linearly proportional to P) if W is allowed to increase with P, and this is essentially what a scalable parallel computation means.

The *isospeed scalability* of an algorithm-machine combination $C = (A, M)$ when the machine size is scaled from P to P' and the problem size is allowed to increase from W to W' is defined as [13] $\psi(P,P') = (W/P)/(W'/P') = (P'W)/(PW')$. Such a ratio essentially measures how the amount of work per processor should be increased when the machine size is scaled from P to P' so that the same average-speed is still achievable. The way in which the problem size W grows with machine size P so that the average-speed $\bar{S}(W,P)$ is maintained a constant is determined by the nature of the algorithm-machine combination $C = (A, M)$, and is the center of scalability analysis. Such analysis essentially is to find a function $W = f(P)$ so that $\bar{S}(W,P) = f(P)/(T(f(P),P)P) = \Theta(1)$, or at least to find the growth rate of $W = f(P)$ required to keep a constant average-speed as P increases. A small growth rate of $f(P)$ implies high scalability; while a large growth rate of $f(P)$ implies poor scalability.

3 A Probabilistic Model

A parallel algorithm $A = (G, t)$ executing on a multiprocessor can be represented by a task precedence graph G and task execution times t. $G = (V, E)$ is a directed acyclic graph (dag) where, nodes denote tasks, and arcs stand for precedence constraints among the tasks. $V = \{v_1, v_2, ..., v_N\}$ is a set of tasks. Each task executes on one processor. E is a set of precedence constraints such that if $(v_i, v_j) \in E$, then task v_j cannot start its execution until task v_i finishes. The task execution times are given by a function $t : V \to (0, +\infty)$, where $t(v_i)$ is the running time of task v_i, $1 \le i \le N$. It is clear that due to different input data that cause different execution paths and competition and contention for shared resources, task execution times are unlikely to be deterministic. Assume that $t(v_1)$, $t(v_2)$, ..., $t(v_N)$ are independent and identically distributed (i.i.d.) exponential random variables with mean $1/\lambda$. Let $\mathbb{E}(\cdot)$ denote the expectation of a random variable. Then, the problem size has the following expectation, $\mathbb{E}(W) = N/\lambda$.

A multiprocessor system M consists of P identical processors M_1, M_2, ..., M_P. Tasks executed on the P processors can communicate with one another via a shared memory. It is assumed that task preemption and migration are disallowed, that is, once a task is assigned to a processor, the task should be executed on that processor from the beginning to the end. Scheduling overhead and costs due to intertask communication and shared resource conflicts are either negligible, or are included into the task execution times.

The framework of our analysis of the average-case performance of parallel computations is as follows. Let T_j be the execution time of processor M_j, $1 \le j \le P$, and $T(W,P) = \max(T_1, T_2, ..., T_P)$ be the total execution time of A on M. We can represent the expected parallel execution time $\mathbb{E}(T(W,P))$ as

$$\mathbb{E}(T(W,P)) = \frac{\mathbb{E}(W)}{P}\left(1 + \phi(N,P,...)\right), \qquad (1)$$

where $\phi(N,P,...)$ is a function of N, P, and other parameters. We define

$$\tilde{S}(W,P) = \frac{\mathbb{E}(W)}{\mathbb{E}(T(W,P))P} = \frac{1}{1+\phi(N,P,...)} \qquad (2)$$

to be the *average-case average-speed*. Since $\mathbb{E}(W)$ is a linear function of N, finding the growth rate of $\mathbb{E}(W) = f(P)$ is equivalent to finding the growth rate of $N = g(P)$ such that $\phi(N,P,...)$ is a constant. Also, we define

$$\tilde{\psi}(P,P') = \frac{P'\mathbb{E}(W)}{P\mathbb{E}(W')} = \frac{P'N}{PN'} = \frac{P'g(P)}{Pg(P')} = \frac{g(P)/P}{g(P')/P'} \qquad (3)$$

to be the *average-case isospeed scalability*.

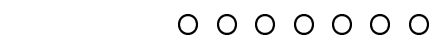

Figure 1. Independent tasks ($N = 7$).

Table 1a: $\tilde{S}(W,P)$ for independent tasks.

P	2	4	8	16	32	64
$k = 2.0$	0.6667	0.7869	0.8397	0.8705	0.8910	0.9058
$k = 1.5$	0.6667	0.7231	0.7515	0.7707	0.7852	0.7970
$k = 1.0$	0.6667	0.6486	0.6359	0.6269	0.6205	0.6158
$k = 0.5$	0.6667	0.5662	0.5021	0.4565	0.4223	0.3955
$k = 0.0$	0.6667	0.4800	0.3679	0.2958	0.2464	0.2108

Table 1b: $\tilde{\psi}(P,P')$ for independent tasks.

P'	2	4	8	16	32	64
$P = 2$	1.0000	0.4615	0.2911	0.2100	0.1635	0.1336
$P = 4$	–	1.0000	0.6306	0.4550	0.3542	0.2894
$P = 8$	–	–	1.0000	0.7216	0.5617	0.4588
$P = 16$	–	–	–	1.0000	0.7784	0.6359
$P = 32$	–	–	–	–	1.0000	0.8169
$P = 64$	–	–	–	–	–	1.0000

4 Independent Tasks

We first consider a parallel computation A consisting of N independent tasks, that is, $E = \emptyset$ (see Figure 1). The execution time $T(W,P)$ of A is determined by a schedule of the N tasks on the P processors. We consider the following analytically tractable scheduling strategy, called *list scheduling*. Initially, each processor is given one task for execution. Whenever a processor M_j completes a task, M_j is given a new task for execution. Such a process is repeated until all the N tasks are finished. Since we are only interested in the expectation of the parallel execution time $T(W,P)$, the order in which tasks are executed seems immaterial.

The following theorem gives $\mathbb{E}(T(W,P))$ for independent tasks.

Theorem 1. *The expectation of the parallel execution time of N independent tasks on P processors under a list schedule is*

$$\mathbb{E}(T(W,P)) = \begin{cases} \dfrac{H_N}{\lambda}, & \text{if } N < P; \\ \left(\dfrac{N}{P} + H_P - 1\right)\dfrac{1}{\lambda}, & \text{if } N \ge P; \end{cases}$$

where the task execution times are i.i.d. exponential random variables with mean $1/\lambda$.

Notice that though there is no communication and synchronization cost for independent tasks, there is still overhead for parallel processing. In particular, when $N \ge P$, the effect of load imbalance on the parallel execution time is represented by $T_o(W,P) = P(H_P - 1)/\lambda$. Such overhead does affect the scalability of parallel processing of independent tasks, even though such parallelism is trivial. By Theorem 1 and the representations in Equations (1) and (2), we know that when $N \ge P$, the average-case average-speed is $\tilde{S}(W,P) = (1 + \phi(N,P))^{-1}$, where $\phi(N,P) = P(H_P - 1)/N$. The above equation implies that to keep a constant average-speed, it is required that $N =$

$g(P) = \Theta(P(H_P - 1)) = \Theta(P \log P)$. In Table 1a, we show $\tilde{S}(W, P)$ where N is selected as $P(\log_2 P)^k$. When $k > 1$, $\tilde{S}(W, P)$ is an increasing function of P; when $k < 1$, $\tilde{S}(W, P)$ is a decreasing function of P. However, when $k = 1$, $\tilde{S}(W, P)$ is quite stable, where the slight change of $\tilde{S}(W, P)$ is due to the inaccuracy of the Θ-notation. As a matter of fact, if we choose $N = cP(H_P - 1)$ for some constant $c > 0$, then the average-speed is maintained at the constant $c/(c + 1)$.

Using the definition in Equation (3), the average-case isospeed scalability is simply $\tilde{\psi}(P, P') = \Theta(\log P / \log P')$, if we set $N = \Theta(P \log P)$ to keep a constant average-speed. By choosing $N = P(H_P - 1)$, we show in Table 1b the value $\tilde{\psi}(P, P') = (P'N)/(PN') = (H_P - 1)/(H_{P'} - 1)$. For instance, the value $\tilde{\psi}(16, 1024) = 0.3657$ means that compared with a system of size 16, the amount of work performed by a processor is roughly tripled when the system size is scaled to 1024, if the same average-speed is to be achieved.

5 Dependent Tasks

Tasks with no incoming arcs are called initial tasks, and tasks with no outgoing arcs are called final tasks. A dag $G = (V, E)$ can be decomposed into levels, denoted by $V_1, V_2, ..., V_L$, where L is the number of levels, i.e., the length of the longest path from an initial task to a final task. A task v_i belongs to level V_l if the longest path from an initial task to v_i is of length l, where the length of a path is the number of nodes on that path. Let $N_l = |V_l|$ be the number of tasks in V_l, where $1 \le l \le L$.

Note that tasks on the same level are independent of each other and can be executed in parallel and in any order. Thus, one simple scheduling algorithm is *level-by-level*, i.e., scheduling tasks in the order V_1, V_2, ..., V_L, and scheduling tasks in V_l using the list scheduling strategy. The following theorem is straightforward.

Theorem 2. *The level-by-level scheduling algorithm results in* $I\!E(T(W, P)) = \sum_{l=1}^{L} I\!E(T(W_l, P))$, *where* $W_l = N_l/\lambda$ *is the amount of work on level* l.

In the next few sections, we analyze several typical classes of dags by using Theorems 1 and 2.

6 Iterative Computations

An iterative computation (see Figure 2) has $L = 2r - 1$ levels. There are r parallel phases interleaved with $r - 1$ serial phases. A parallel phase consists of m tasks, and $N = mr + (r - 1) = (m + 1)r - 1$. Under a level-by-level schedule, and assuming that $m \ge P$, we obtain

$$I\!E(T(W, P)) = \left(\frac{N}{P} + rH_P - \frac{P + r - 1}{P} \right) \frac{1}{\lambda}.$$

The average-case average-speed is $\tilde{S}(W, P) = (1 + \phi(N, P, m, r))^{-1}$, where

$$\phi(N, P, m, r) = \frac{rPH_P - (P + r - 1)}{(m + 1)r - 1}.$$

It is clear that if m is fixed, there will be no way to keep $\phi(N, P, m, r)$ as a constant by increasing r with P. The parameter m indicates the amount of parallelism of an iterative computation. If we fix m and increase r, the problem size will increase; however, this does not make the computation scalable. This observation implies that while a parallel computation can be made scalable by increasing the problem size together with the system size, it is actually the amount of parallelism that should scale up with the system size. For an iterative computation, it is clear that only m can scale against P. No matter whether r is fixed or not, we need $m = \Theta(PH_P)$. When r is fixed, we have $N = \Theta(P \log P)$, so that a constant average-speed can be kept.

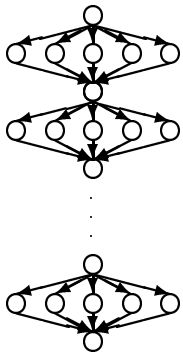

Figure 2. An iterative computation ($m = 5$).

Table 2a: $\tilde{S}(W, P)$ for iterative computations.

P	2	4	8	16	32	64
$k = 2.0$	0.6020	0.7023	0.7815	0.8308	0.8628	0.8850
$k = 1.5$	0.6020	0.6306	0.6759	0.7113	0.7380	0.7587
$k = 1.0$	0.6020	0.5548	0.5503	0.5537	0.5583	0.5625
$k = 0.5$	0.6020	0.4791	0.4206	0.3863	0.3628	0.3450
$k = 0.0$	0.6020	0.4080	0.3050	0.2446	0.2056	0.1782

Table 2b: $\tilde{\psi}(P, P')$ for iterative computations.

P'	2	4	8	16	32	64
$P = 2$	1.0000	0.6592	0.4729	0.3634	0.2933	0.2452
$P = 4$	–	1.0000	0.7174	0.5512	0.4449	0.3720
$P = 8$	–	–	1.0000	0.7683	0.6201	0.5185
$P = 16$	–	–	–	1.0000	0.8071	0.6749
$P = 32$	–	–	–	–	1.0000	0.8362
$P = 64$	–	–	–	–	–	1.0000

Table 2a gives the average-case average-speed $\tilde{S}(W, P)$ for iterative computations with $m = P(\log_2 P)^k$ and $r = 20$. When $k > 1$, $\tilde{S}(W, P)$ is an increasing function of P; when $k < 1$, $\tilde{S}(W, P)$ is a decreasing function of P. To keep a constant average-speed, it is required that $m = \Theta(P \log P)$, i.e., $N = \Theta(P \log P)$. Table 2b further demonstrates the average-case isospeed scalability $\tilde{\psi}(P, P')$ for iterative computations with $m = P \log_2 P$, that is,

$$\tilde{\psi}(P, P') = \frac{P'N}{PN'} = \frac{P'((P \log_2 P + 1)r - 1)}{P((P' \log_2 P' + 1)r - 1)} = \Theta\left(\frac{\log P}{\log P'} \right).$$

The values in Table 2b show that iterative computations have comparable scalability with independent tasks if the parallelism m in parallel phases can be increased together with P.

7 Search Trees

Let us consider a complete b-ary search tree of height h (see Figure 3). This task precedence graph has $L = h + 1$ levels numbered with 0, 1, 2, ..., h. There are $N_l = b^l$ tasks on level l, where $0 \le l \le h$. The total number of tasks is $N = (b^{h+1} - 1)/(b - 1)$. Let $b^p \le P < b^{p+1}$ for some $0 \le p \le h$, i.e., $p = \lfloor \log_b P \rfloor$.

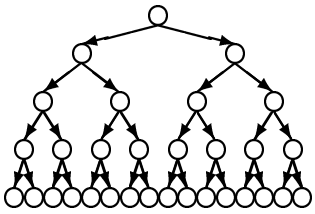

Figure 3. A complete binary search tree with height $h = 4$.

Table 3a: $\tilde{S}(W, P)$ for search trees.

P	2	4	8	16	32	64
$k = 3.0$	0.5000	0.6078	0.7216	0.7559	0.8044	0.8542
$k = 2.5$	0.5000	0.6078	0.6001	0.6361	0.6951	0.7620
$k = 2.0$	0.5000	0.4891	0.6001	0.5008	0.5611	0.6385
$k = 1.5$	0.5000	0.4891	0.4705	0.3702	0.4212	0.3575
$k = 1.0$	0.5000	0.3818	0.3517	0.2614	0.2967	0.2422

Table 3b: $\tilde{\psi}(P, P')$ for search trees.

P'	2	4	8	16	32	64
$P = 2$	1.0000	0.1333	0.0315	0.0314	0.0156	0.0078
$P = 4$	–	1.0000	0.2362	0.2353	0.1173	0.0586
$P = 8$	–	–	1.0000	0.9961	0.4966	0.2481
$P = 16$	–	–	–	1.0000	0.4985	0.2491
$P = 32$	–	–	–	–	1.0000	0.4996
$P = 64$	–	–	–	–	–	1.0000

Then, under a level-by-level schedule, we have

$$I\!E(T(W,P)) \approx \left(\frac{N}{P} + \frac{\ln b}{2}(\log_b P)^2 + \log_b \frac{N}{P} \ln P\right)\frac{1}{\lambda}.$$

The last equation gives rises to the average-case average-speed $\tilde{S}(W,P) \approx (1 + \phi(N,P,b))^{-1}$, where

$$\phi(N,P,b) = \frac{P}{N}\left(\frac{\ln b}{2}(\log_b P)^2 + \log_b \frac{N}{P} \ln P\right).$$

It is clear that to maintain a constant average-speed, we need $N = \Theta(P(\log P)^2)$. In Table 3a, we show $\tilde{S}(W,P)$ for complete binary search trees (i.e., $b = 2$) when $N = 2^{h+1} - 1 \approx P(\log_2 P)^k$. In general, $\tilde{S}(W,P)$ is an increasing function of P when $k > 2$, and $\tilde{S}(W,P)$ is a decreasing function of P when $k < 2$. However, when $k = 2$, $\tilde{S}(W,P)$ is quite stable. The slight fluctuation of $\tilde{S}(W,P)$ is due to the inaccuracy of the Θ-notation and the constraint that $N = 2^{h+1} - 1$ for some integer $h \geq 0$, which makes $\tilde{S}(W,P)$ non-monotonous. To keep a constant average-speed, it is required that $N = \Theta(P(\log P)^2)$. By fixing $k = 2$, Table 3b displays the average-case isospeed scalability

$$\tilde{\psi}(P,P') = \frac{P'N}{PN'} = \Theta\left(\frac{\log P}{\log P'}\right)^2.$$

It is clear that search trees exhibit lower isospeed scalability than independent tasks. For instance, the value $\tilde{\psi}(16, 1024) = 0.1245$ means that compared with a system of size 16, the amount of work performed by a processor is eight times more when the system size is scaled to 1024, if the same average-speed is to be achieved.

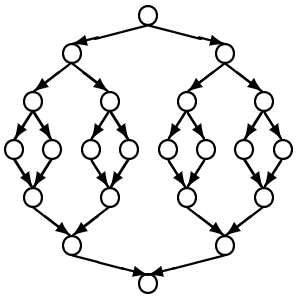

Figure 4. A partitioning algorithm with $b = 2$ and $h = 3$.

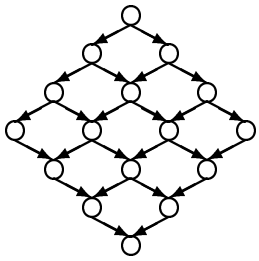

Figure 5. A diamond dag with $d = 4$.

8 Partitioning Algorithms

A partitioning algorithm (see Figure 4) with branching factor b and height h has $L = 2h + 1$ levels numbered with 0, 1, 2, ..., $h-1$, h, $h+1$, ..., $2h$. There are b^l tasks on level l and $2h-l$, i.e., $N_l = N_{2h-l} = b^l$, where $0 \leq l \leq h$. The total number of tasks is $N = (b^{h+1} + b^h - 2)/(b-1)$. The analysis for partitioning algorithms is similar to that of search trees. Let $b^p \leq P < b^{p+1}$ for some $0 \leq p \leq h$, i.e., $p = \lfloor \log_b P \rfloor$. Then, under a level-by-level schedule, we have

$$I\!E(T(W,P)) \approx \left(\frac{N}{P} + \ln b(\log_b P)^2 + 2\log_b \frac{N}{P} \ln P\right)\frac{1}{\lambda}.$$

The last equation gives rises to the average-case average-speed $\tilde{S}(W,P) \approx (1 + \phi(N,P,b))^{-1}$, where

$$\phi(N,P,b) = \frac{P}{N}\left(\frac{\ln b}{2}(\log_b P)^2 + \log_b \frac{N^2}{P} \ln P\right),$$

which implies that to maintain a constant average-speed, we need $N = \Theta(P(\log P)^2)$.

Table 4a gives numerical data of $\tilde{S}(W,P)$ for partitioning algorithms with $b = 2$ and $N = 2^{h+1} + 2^h - 2 \approx P(\log_2 P)^k$. Letting $k = 2$, we show in Table 4b the average-case isospeed scalability

$$\tilde{\psi}(P,P') = \Theta\left(\frac{\log P}{\log P'}\right)^2.$$

Both the average-case average-speed and isospeed scalability of partitioning algorithms exhibit similar properties to those of search trees.

9 Diamond Dags

A diamond dag (see Figure 5) has $L = 2d - 1$ levels and $N = d^2$ tasks, with $N_l = N_{2d-l} = l$, for $1 \leq l \leq d$. The

Table 4a: $\tilde{S}(W, P)$ for partitioning algorithms.

P	2	4	8	16	32	64
$k = 3.0$	0.5000	0.5633	0.6761	0.7113	0.7643	0.8211
$k = 2.5$	0.5000	0.4490	0.5499	0.5836	0.6438	0.5828
$k = 2.0$	0.5000	0.4490	0.4233	0.4482	0.5052	0.4394
$k = 1.5$	0.5000	0.3529	0.3139	0.3253	0.2559	0.3085
$k = 1.0$	0.5000	0.3529	0.2314	0.2282	0.1719	0.1335

Table 4b: $\tilde{\psi}(P, P')$ for partitioning algorithms.

P'	2	4	8	16	32	64
$P = 2$	1.0000	0.0909	0.0426	0.0209	0.0104	0.0104
$P = 4$	–	1.0000	0.4681	0.2304	0.1147	0.1147
$P = 8$	–	–	1.0000	0.4921	0.2451	0.2450
$P = 16$	–	–	–	1.0000	0.4980	0.4977
$P = 32$	–	–	–	–	1.0000	0.9993
$P = 64$	–	–	–	–	–	1.0000

Table 5a: $\tilde{S}(W, P)$ for diamond dags.

P	2	4	8	16	32	64
$k = 1.50$	0.6957	0.7310	0.7550	0.7725	0.7863	0.7977
$k = 1.25$	0.6957	0.6972	0.7020	0.7069	0.7117	0.7163
$k = 1.00$	0.6957	0.6615	0.6434	0.6318	0.6238	0.6182
$k = 0.75$	0.6957	0.6243	0.5808	0.5503	0.5277	0.5100
$k = 0.50$	0.6957	0.5861	0.5163	0.4671	0.4303	0.4016

Table 5b: $\tilde{\psi}(P, P')$ for diamond dags.

P'	2	4	8	16	32	64
$P = 2$	1.0000	0.1250	0.0278	0.0078	0.0025	0.0009
$P = 4$	–	1.0000	0.2222	0.0625	0.0200	0.0069
$P = 8$	–	–	1.0000	0.2812	0.0900	0.0312
$P = 16$	–	–	–	1.0000	0.3200	0.1111
$P = 32$	–	–	–	–	1.0000	0.3472
$P = 64$	–	–	–	–	–	1.0000

expected parallel execution time is

$$I\!E(T(W,P)) \approx \left(\frac{N}{P} + 2dH_P\right)\frac{1}{\lambda}, \quad \text{when } d \gg P.$$

The average-case average-speed is $\tilde{S}(W,P) \approx (1 + \phi(N,P,d))^{-1}$, where

$$\phi(N,P,d) = \Theta\left(\frac{P\log P}{d}\right) = \Theta\left(\frac{P\log P}{\sqrt{N}}\right).$$

To keep a constant average-speed, we need to keep $N = \Theta((P\log P)^2)$. This is illustrated in Table 5a, where we display the average-case average-speed $\tilde{S}(W,P)$ for diamond dags with $d = 2P(\log_2 P)^k$ and $N = 4P^2(\log_2 P)^{2k}$. Table 5b provides the average-case isospeed scalability $\tilde{\psi}(P,P')$ for diamond dags with $d = P\log_2 P$ and $N = P^2(\log_2 P)^2$, that is,

$$\tilde{\psi}(P,P') = \Theta\left(\frac{P(\log P)^2}{P'(\log P')^2}\right).$$

It is clear that compared with the previous task precedence graphs, diamond dags have much lower scalability. For example, the value $\tilde{\psi}(16, 1024) = 0.0025$ means that compared with a system of size 16, the amount of work performed by a processor is 400 times more when the system size is scaled to 1024, if the same average-speed is to be achieved.

Table 6: Summary of results.

Graph	Growth Rate of Work	Average-case Isospeed Scalability
IT	$N = \Theta(P\log P)$	$\tilde{\psi}(P,P') = \Theta\left(\frac{\log P}{\log P'}\right)$
IC	$N = \Theta(P\log P)$	$\tilde{\psi}(P,P') = \Theta\left(\frac{\log P}{\log P'}\right)$
ST	$N = \Theta(P(\log P)^2)$	$\tilde{\psi}(P,P') = \Theta\left(\left(\frac{\log P}{\log P'}\right)^2\right)$
PA	$N = \Theta(P(\log P)^2)$	$\tilde{\psi}(P,P') = \Theta\left(\left(\frac{\log P}{\log P'}\right)^2\right)$
DD	$N = \Theta((P\log P)^2)$	$\tilde{\psi}(P,P') = \Theta\left(\frac{P(\log P)^2}{P'(\log P')^2}\right)$

10 Summary

We have analyzed the average-case performance of parallel computations on multiprocessor systems using a probabilistic model. Our performance metrics are average-speed and isospeed scalability. It is found that the scalability of a parallel computation is determined by its task precedence graph, i.e., the structure of a parallel algorithm. Table 6 summarizes our analytical results for the task precedence graphs discussed in this paper.

References

[1] A.Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: measuring the scalability of parallel algorithms and architectures," *IEEE Parallel and Distributed Technology*, vol.1, no.3, pp.12-21, 1993.

[2] J.L. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, vol.31, pp.532-533, 1988.

[3] K. Hwang and Z. Xu, *Scalable Parallel Computing*, McGraw-Hill WCB, 1998.

[4] A.H. Karp and H. P. Flatt, "Measuring parallel processor performance," *Communications of the ACM*, vol.33, pp.539-543, 1990.

[5] V. Kumar and A. Gupta, "Analysis of scalability of parallel algorithms and architectures: a survey," *Proc. International Conference on Supercomputing*, pp.396-405, 1991.

[6] V. Kumar, *et al.*, *Introduction to Parallel Computing*, Benjaming/Cummings, 1994.

[7] K. Li and Y. Pan, "On the impact of communication overhead on the average-case scalability of random parallel programs on multicomputers," *Informatica – An International Journal of Computing and Informatics*, vol.21, pp.279-291, 1997.

[8] K. Li, Y. Pan, H. Shen, and S.-Q. Zheng, "A study of average-case speedup and scalability of parallel computations on static networks," to appear in *Mathematical and Computer Modelling*.

[9] D. Nussbaum and A. Agarwal, "Scalability of parallel machines," *Communications of the ACM*, vol.34, pp.57-61, 1991.

[10] S. Sahni and V. Thanvantri, "Performance metrics: keeping the focus on runtime," *IEEE Parallel and Distributed Technology*, vol.4, no.1, pp.43-56, 1996.

[11] J.P. Singh, J. L. Hennessy, and A. Gupta, "Scaling parallel programs for multiprocessors: methodology and examples," *Computer*, vol.26, pp.42-50, 1993.

[12] X.-H. Sun and L. Ni, "Scalable problems and memory-bounded speedup," *Journal of Parallel and Distributed Computing*, vol.19, pp.27-37, 1993.

[13] X.-H. Sun and D. T. Rover, "Scalability of parallel algorithm-machine combinations," *IEEE Transactions on Parallel and Distributed Systems*, vol.5, no.6, pp.599-613, 1994.

[14] M. Willebeek-LeMair, A.P. Reeves, and C.H. Ning, "Characterization of multicomputer systems: a transfer ration approach," *Proceedings of International Conference on Parallel Processing*, vol.II, pp.171-178, 1990.