LogGP Analysis of Parallel Sorting Algorithms

Greg Tracy, University of Wisconsin, Madison

Abstract

The need for solutions to large, complex computations has grown considerably. As a result, so has the need for models that can accurately predict the performance of parallel systems without constructing the algorithms themselves. Parallel software developers have a need for analyzing design changes in a quick and efficient manner. The LogGP model was developed to help answer this problem. In this paper, I apply the LogGP model to the analysis of parallel sorting algorithms. This paper outlines equations used to model the performance of parallel sorting and compares the results to models built using the LogP model as well as comparisons with actual measurements made on the target hardware.

1 Introduction

This paper investigates the use of the parallel computation model, LogGP [AISS95] in the analysis of parallel sorting algorithms in a network of workstations (NOW). Although parallel sorting algorithms are fairly well developed, an accurate model provides some insight into performance characteristics of the sort as well a quick and efficient means for testing improvements to the algorithm. The work done in this research is the foundation for models of more advanced parallel applications. The problems researched in parallel sorting apply to lots of other parallel applications. The duality of computation and communication is a common attribute for lots of interesting distributed applications.

In previous work [DCMS96], the LogP [CKPS93] model was applied to a set of sorting algorithms running on the Thinking Machines CM-5. Those models were accurate to within 12% of the measured results on the CM-5 and provided compeling evidence that the LogP model provided the necessary accuracy for parallel sorts. This research investigates and produces results which indicate that the use of the LogGP model will result in more accurate models for parallel sorting algorithms due to the size of messages which are transmitted between nodes. The LogP model assumes “small”, fixed sized messages being passed between nodes, and therefore does not account for the true behavior of the NOW sorting algorithm.

This paper will outline the LogGP model created to measure the expected performance of sorting in a NOW environment. I will present results for a wide range of node configurations and compare the results of the LogGP model to a LogP model of the same problem. I will also present data for a series of model variations in an attempt to more accurately model the current NOW-Sort implementation. This is followed up with an

analysis of possible improvements to the algorithm and a comparison to the lower bound of the I/O subsystem. The goals of this model will be to accurately estimate the running time of a set of sorting algorithms outlined in the NOW literature. It will allow performance estimation with variable node counts, link bandwidth, and link latency.

2 NOW-Sort

The NOW-Sort Project [AACHP97] at UC-Berkeley used a set of parallel sorting algorithms to break a performance benchmarking record. They held the premise that networks of commodity workstations could provide the same performance as that of the large, parallel processors with shared memory architectures. The sorting problem was used as a vehicle to prove that these networks could provide a cost effective solution to large parallel problems.

Unlike a single, parallel machine, however, the nodes of a cluster must communicate via the network. The LogGP model captures this communication aspect through the measurement of network characteristics and applying them to the distribution phase of the sorting algorithm.

The baseline algorithm for parallel sorting is as follows:

1. Read – Each node will read all keys and corresponding records from local disk into memory.
2. Distribute – Each node will then send all keys and data to the respective node responsible for sorting and storing the data.
3. Sort – Each node will sort all of the keys and data it has received.
4. Write – Each node will write its keys and data to local disk.

This research focuses on applying the LogGP model to the second step of this algorithm - distribution. Together with measurements made on the department's NOW cluster, a complete analysis is made of the sorting performance on a range of node configurations.

To better understand the importance of the distribution phase, a sample measurement was taken from a sort run on the department's NOW cluster. The configuration consists of eight nodes each with four disks and a total of 1000K records are sorted. Figure 1 shows the distribution of time spent by a single node. The performance of the sort is bounded by the performance of the distribution phase (57%) as well as the I/O subsystem. The distribution algorithm of the NOW-Sort coalesces data before sending to another

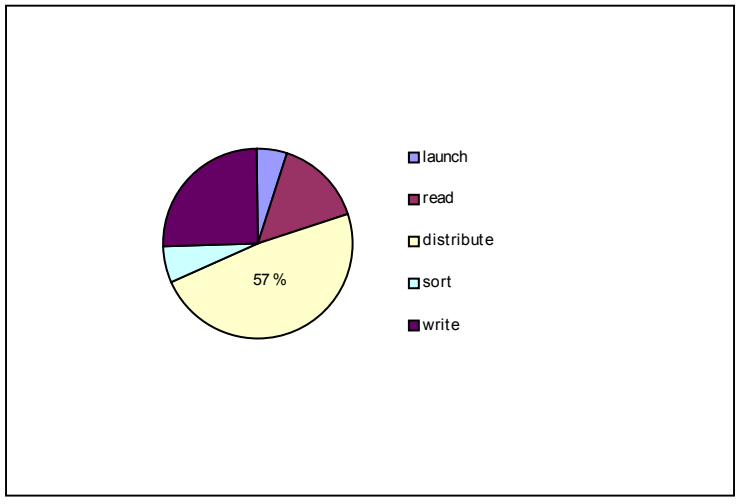

Figure 1: Cost Breakdown of a Parallel Sort (8 nodes, 4 disks, 1000 kilokeys)

node in the cluster. Thus, an accurate model must take into consideration that communication packets are of large, variable sizes. The LogGP model captures this behavior by using the bandwidth obtained for long messages.

3 LogGP Parameters

In order to create an accurate model, it is important to derive accurate parameters L, o, g, and G. Micro-benchmarks similar to those in [AISS95] and [SV99] were created to measure these parameters. It must be noted, however, that these parameters were not measured on the same hardware that the other NOW-Sort data elements were collected. At the time of this writing, I did not have access to the cluster. As a result, these measurements were made on a comparable configuration - Pentium III, 800MHz workstations running the Linux 2.2.17 operating system connected via 100Mbps Ethernet.

Using micro-benchmarks to measure response times from the network as well as response times from network system calls, the following parameter values were calculated: L=100□sec, o=16□sec, G=0.03□sec. Without the access to the NOW cluster it was not possible to accurately measure the g parameter. As a result, it was estimated to be equal to the o parameter.

4 Model Equations

To understand the importance of the G parameter, we first start with the LogP model. This focuses on the distribution phase of the sorting algorithm. In the LogP model, the amount of time it takes to distribute *k* bytes from one node to another is:

$$T_{dist}(k) = o_{send} + ((k-1)/w) * \max(g,o) + L + o_{recv} \qquad (1)$$

The LogP parameters are used in their traditional context. The *w* parameter indicates the message size so more then one word can be transferred in a single message. The sending rate of each message will be the maximum of the available transmission rate and the transmission overhead. From start to finish, the time consists of the overhead, o_{send}, to get the data on the wire, the time to transmit each remaining message, the latency, *L*, of transmitting the data, plus the overhead, o_{recv}, for the receiver to read the data off the wire.

In contrast, the LogGP model allows us to transmit *k* bytes in just one large message:

$$T_{dist}(k) = o_{send} + (k-1)G + L + o_{recv} \qquad (2)$$

In order to reflect the parallel nature of the sort, equation (2) must be extended further. Since all nodes are distributing data at the same time, each node must alternate between sending data and receiving data. These operations cannot overlap, however, which is reflected in the factor of two in equation (3).

$$T_{dist}(k) = 2(k-1)G + 2L + 2o \quad (3)$$

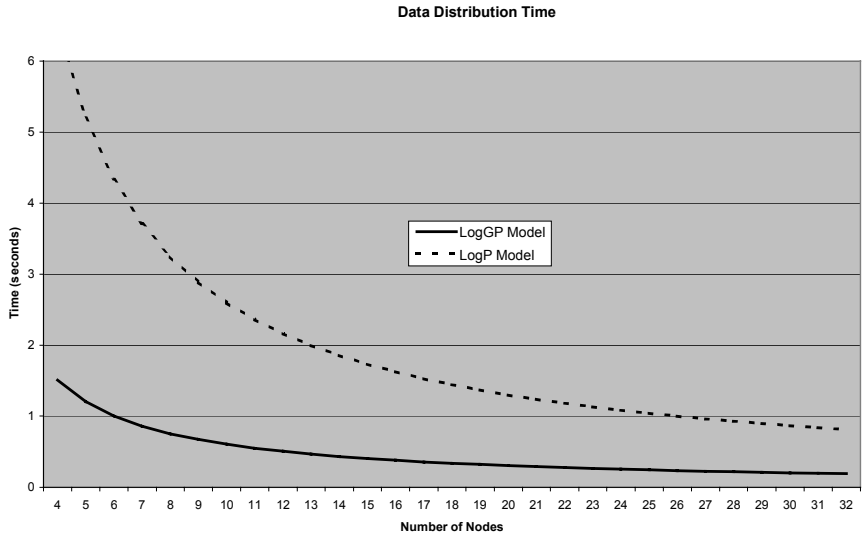

Figure 2: Comparison of the distribution times using LogP and LogGP models when sorting 1000K records.

Figure 2 shows the advantage of grouping small messages into one large one. This extension of the LogP model allows the nodes to take advantage of the higher bandwidth attained when sending large packets on the network. It will be shown that this new LogGP model approaches the actual performance measured on the NOW-Sort cluster. Using the LogP model with "small" messages produces results which are off by a factor of four.

5 Variations of Distribution Model

In practice, the NOW-Sort distribution algorithm performs the distribution phase by reading all local keys off of disk and distributing data in successive transmissions to its peers. This behavior was captured in the model with a few modifications.

5.1 Read All, Send All

If multiple, large messages are to be sent one after another, this distribution time can be modeled with the following equation:

$$T_{dist}(k) = 2p(k-1)G + (p-1)g + pL + 2o \quad (4)$$

This expands on (3). The transmission times for *p* messages are summed and we add in the gap period, *g*, between each message. Up until now we have separated the two overhead values between sender and receiver, but they are treated as being equal and are combined into one term in equation (4). The node can overlap the transmission overhead of the successive messages with the latency, *L*, of the previous message. That is why

there are still only 2 overhead cycles in the equation. Depending on the amount of data being sorted by each node, the distribution phase could include any number of transmissions of size *k*.

5.2 Delayed Transmission

In an attempt to refine this model further, two other factors were taken into consideration. First, the messages are *not* sent successively. In fact, in an attempt to avoid flooding the network and receiver queues at the nodes, the distribution is executed in a more orderly fashion. The following pseudo-code fragment outlines the algorithm:

```
for( i=p+1; i < P-1; i = (i+1)% P )
{
    send_data(i);
    barrier();
}
```

If all cluster nodes were ordered from 0 to *P*, where *p* identifies the current node; the for loop begins with the node immediately following our own in the list and loops through *P-1* of the nodes in the cluster. This forces each receiver to have only one unique sender. For each node, we transmit the portion of data it is responsible for sorting and writing but originated on *p's* disk. Implicit in the *send_data*()[11] call is a network read if data has arrived. Although it isn't explicitly shown in the above algorithm, reads are occurring. After each call to *send_data*(), each node blocks on a barrier which is a global synchronization point. When a node reaches the barrier it sends *P-1* messages to the rest of the nodes and then waits for *P-1* acknowledgements before proceeding. This synchronization step prevents slower nodes from getting overwhelmed with message processing. Time constraints prevented me from adding the barrier delays into the model.

The second modification to (4) is the consideration that not all messages are of the same length. Specifically, the last message will be of variable length depending on how many records will be sorted and how many nodes are in the configuration. (4) can be expressed more accurately by:

$$T_{dist}(k) = [\sum(k_i\text{-}1)G] + (p\text{-}1)g + pL + 2o \qquad (5)$$

And removing the g factor due to the barrier call (limiting the rate at which packets are sent) produces the following model:

$$T_{dist}(k) = 2p(k\text{-}1)G + 2(k_{rem})G + 2L + 2o \qquad (6)$$

[1] The NOW-Sort code was written in Split-C [CDGKLEY93] which is built on top of Active Messages [ECGS92]. Active Messages is a lightweight version of remote procedure calls which passes a handler with each send call which instructs the receiving node to execute some piece of code.

5.3 Overlap Distribution with I/O

Exploiting under-utilized resources in the NOW cluster is perhaps the best way to improve overall performance. One of the prime advantages of using the LogGP modeling technique is the ability to experiment with algorithm changes quickly and easily. In this research I experimented with overlapping the disk reads with the distribution phase. Through the simultaneous reading and sending, we can hide the majority of the cost associated with the distribution phase. The following equations outline this new technique:

$$T_{read+dist} = T_{read} + T_{dist\text{-}i} \qquad (7)$$

$$T_{dist\text{-}i}(n) = (P\text{-}1)(k\text{-}1)G + 2L + 2o \qquad (8)$$

Equation (7) states that the sum of the read and distribution phases is equal to the time it takes to read all of the data plus the cost of distributing the last piece of data to its peers. The last distribution phase (8) assumes the average case which is that the last read operation contains data which must be distributed to only half of its peers[2]. On average, the size of the message *k* is considerably smaller then the read all, transmit all approach found in (4). There is certainly some room for error in (8). Time permitting, it would be beneficial to create a probabilistic model to better represent the number of packets sent and the size of each packet for the last distribution phase. However, the majority of the distribution phase is overlapped with the read phase.

6 Results

The results of modeling (5) and (6) are presented in Figure 3 together with actual measurements made on the NOW-Sort cluster in the department. The good news is that the LogGP models have produced results which are very close to the measured results for sorts of 1000K records. Unfortunately, there is very little difference between the models resulting from (5) and (6). This is primarily due to the fact that as the *pk* product gets bigger, this product dominates the equations. The model becomes roughly equal to *2pkG*.

It is presumed that the difference in the tails of these curves can be attributed to two factors. First, as the number of nodes in the sorting cluster increases, the amount of time spent in the barrier call increases. In order to produce a more accurate model, this behavior must be understood better and integrated into the model. Second, while the micro-benchmarks showed that we could approach the network's peek bandwidth when transferring 64K data packets, it is reasonable to believe that as the number of nodes increases so does the network congestion. As a result, the available bandwidth may decrease resulting in slower distribution times.

[2] The number of outgoing and incoming packets will be 2*[(P-1)/2]. This simplifies to (P-1).

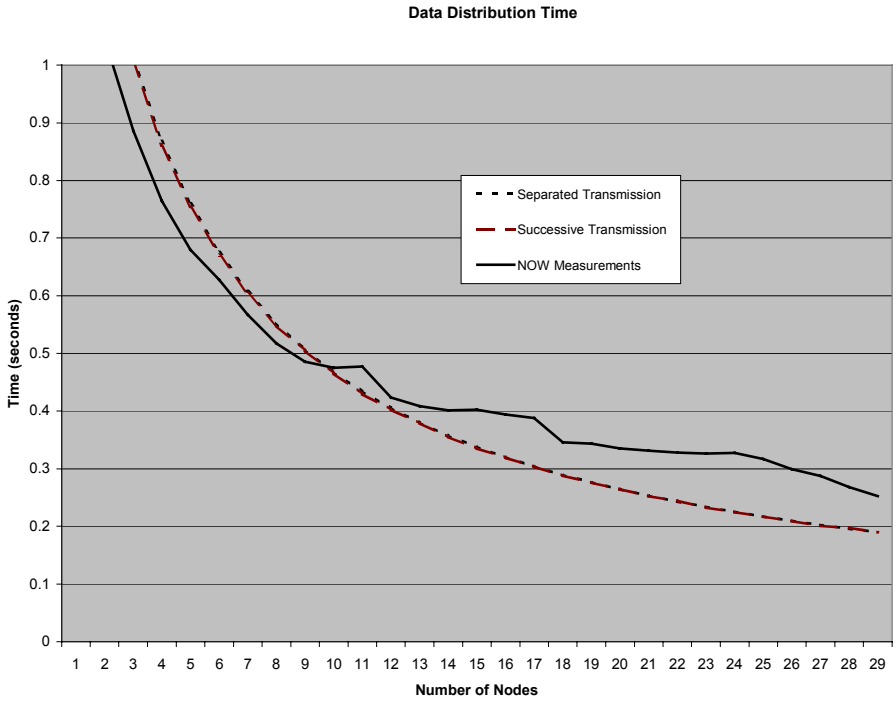

Figure 3: Comparison of the distribution times using a sample of LogGP models along with the actual distribution time measured on the NOW-Sort implementation.

Figure 4 presents the same results with the overlap model added. It is clear that there is a lot to be gained by overlapping the read and distribution phases of the sort. The NOW-Sort team has not yet implemented this version of the algorithm so I am not able to validate the results at this time. Intuitively, however, this is a promising direction to take the algorithm.

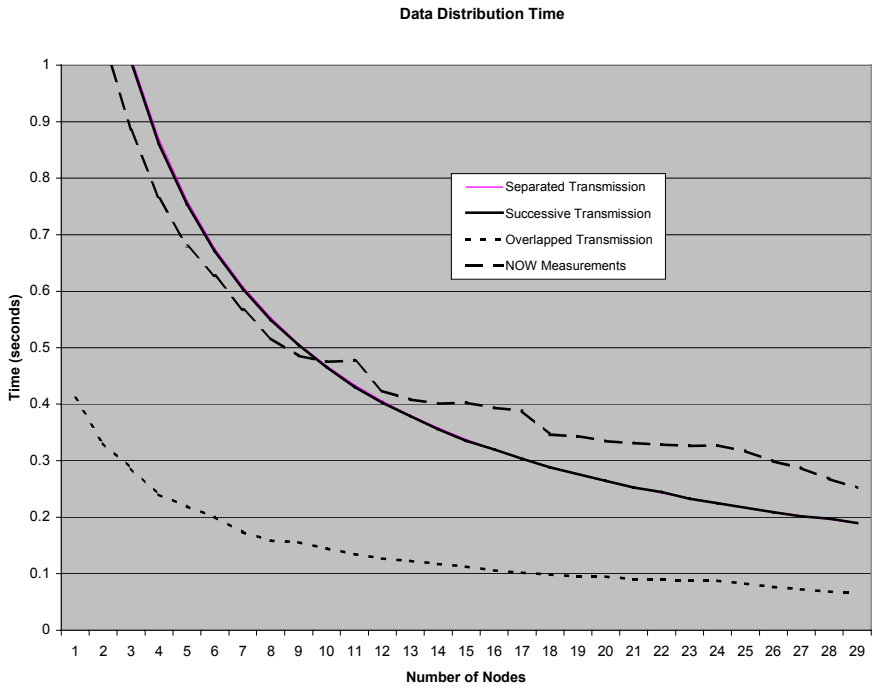

Figure 4: Comparison of the three LogGP models including the *overlap* model and the measured NOW-Sort data

6.1 Total Sort Times

Equations (3) and (4) are the foundation for my sorting models. When T_{dist} is inserted into our overall timing model, we come up with the following equation for each node p:

$$T_{sort} = T_{read}(N/P) + T_{dist}(N/P) + T_{sort}(N/P) + T_{write}(N/P) \qquad (9)$$

All other timing measurements: $T_{read,}$ $T_{sort,}$ and T_{write} were obtained from the NOW-Sort cluster and stay constant no matter which model is used for the T_{dist} phase. The overall performance of the sort is presented in Figure 5. In addition to the variations on the distribution model (including the LogP distribution model), the lower bound of the I/O subsystem is shown to provide a gauge as to how much improvement can be expected. The lower bound is based on the measurements produced by the current NOW-Sort implementation. This is not to say that the I/O performance cannot be improved in and of itself.

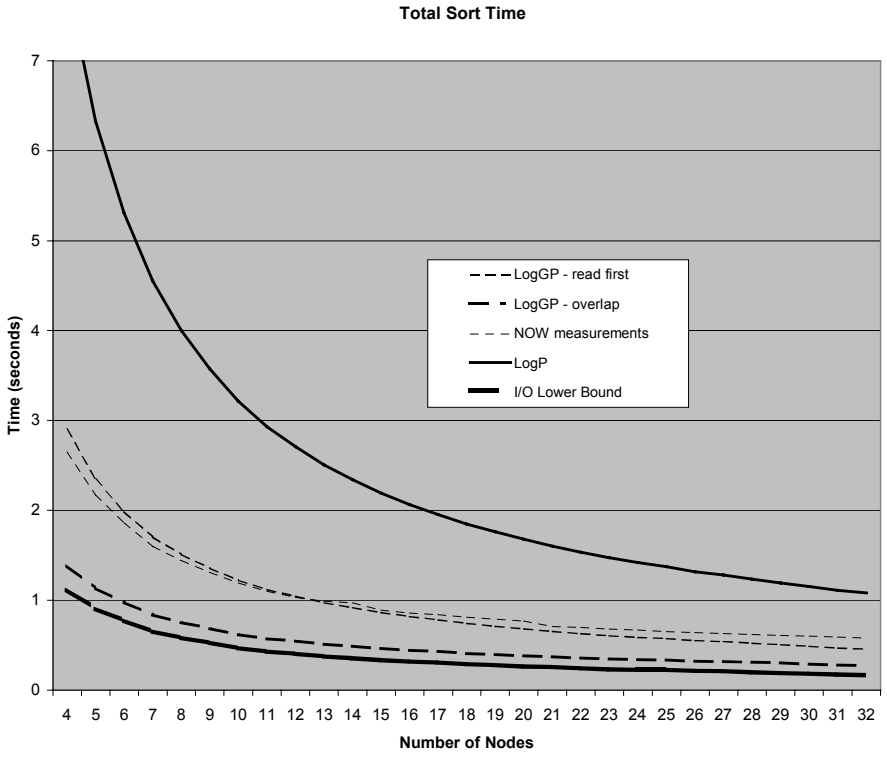

Figure 5: Comparison of total sort time for all sorting models presented thus far. The lower bound on the I/O subsystem is also shown to gauge how much room the models have to improve.

7 Conclusion

In this paper I presented a sequence of parallel application models in an attempt to better estimate the performance of the NOW-sort application. The primary focus was to understand the importance of the G factor in parallel applications which send large messages. The comparison between the LogP and LogGP model proved that there is considerable improvement in the accuracy. The LogGP equations for the distribution phase of the sort application provide an excellent means for predicting overall performance. In addition, it was shown with the overlap equation (7) that LogGP is an excellent tool for making quick and efficient evaluations of algorithm changes.

There is still a lot of work to be done in the scope of this research, however. It would be advantageous to run the benchmarking code on the actual NOW cluster to obtain more accurate model parameters. In addition, the distribution of data across cluster nodes should be more thoroughly analyzed to better understand the impact of the overlap model. Once that is completed, implementing this overlap in the NOW-Sort software would allow me to validate the results.

In addition, if the model were to be truly effective as a tool for analyzing software changes, the I/O subsystem would need to be modeled. Having said this, the area of

parallel sorting has been heavily researched and a lot of the problems associated with network and I/O bottlenecks in a parallel sorting environment have been analyzed and perhaps even solved. This application proved to be a good chance to put the LogGP model to use, but a bigger contribution can be made by applying this model to more advanced parallel applications. Applications which require more complex and highly developed communication patterns.

References

[AISS95] Alexandrov, A., M. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP Model", *Proc. 7th Ann. ACM Symp. On Parallel Algorithms and Architectures*, Santa Barbara, CA, July 1995.

[AACHP97] Arpaci-Dusseau, A. C., R. H. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, D. A. Patterson, "High-Performance Sorting on Networks of Workstations", *SIGMOD*, Tucson, Arizona. May, 1997.

[CDGKLEY93] Culler, D., A. Dusseau, S. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, K. Yelick. "Parallel Programmng in Split-C". *Supercomputing '93*, 1993.

[CKPS93] Culler, D., R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. Von Eiken, "LogP: Towards a Realistic Model of Parallel Computation", *Proc. 4th ACM SIGPLAN Symp. On Principles and Practice of Parallel Programming (PpoPP '93)*, San Diego, CA, May 1993.

[DCSM96] Dusseau, A., D. E. Culler, K. E. Schauser, and R. P. Martin, "Fast Parallel Sorting Under LogP: Experience with the CM-5", *IEEE Transactions on Parallel and Distributed Systems*, Volume 7, No. 8, 1996.

[ECGS92] Eicken, T. von, D. E. Culler, S. C. Goldstein, K. E. Schauser, "Active messages: A Mechanism for Integrated Communication and Computation", *Proceedings of the 19th Annual Symposium on Computer Architecture*, May 1992.

[FAV97] Frank, M. I., A. Agrawal, and M. K. Vernon, "LoPC: Modeling Contention in Parallel Algorithms", *Proc. 6th ACM SIGPLAN Symp. On Principles and Practice of Parallel Programming (PpoPP '97)*, Las Vegas, NV, June 1997.

[SV99] Sundaram-Stukel, D., M. K. Vernon, "Predictive Analysis of a Wavefront Application Using LogGP", *Proc. 7th ACM SIGPLAN Symp. On Principles and Practice of Parallel Programming (PpoPP '99)*, Atlanta, GA, May 1999.