

# Improving Parallel I/O Performance through Scheduling Algorithms based on Randomized, Distributed Edge Coloring Algorithms.\*

Dannie Durand,<sup>†</sup> Ravi Jain, David Tseytlin  
Bellcore  
445 South Street  
Morristown, NJ 07960

June 13, 1999

---

\*A version of this work has appeared in the Proceedings of the 1995 ACM Symposium of Parallel Algorithms and Architectures.

<sup>†</sup>Currently at Department of Molecular Biology, Princeton University Princeton, NJ 08544

**Running Head:** Improving Parallel I/O Performance through scheduling

**Contact Author:**

Dannie Durand  
Department of Molecular Biology  
Princeton University  
Princeton, NJ 08544  
tel: 609-258-6874  
fax: 609-258-0975  
durand@cs.princeton.edu

**Abstract**

A growing imbalance in CPU and I/O speeds has led to a communications bottleneck in distributed architectures, especially for data-intensive applications such as multimedia information systems, databases, and Grand Challenge problems. Our solution is to schedule parallel I/O operations explicitly. We present a class of decentralized scheduling algorithms that eliminate contention for I/O ports while maintaining an efficient use of bandwidth. These algorithms, based on edge-coloring and matching of bipartite graphs, rely upon simple heuristics to obtain shorter schedules. We use simulation to evaluate the ability of our algorithm to obtain near optimal solutions in a distributed context and compare our work with that of other researchers. Our results show that our algorithms produce schedules within 5% of the optimal schedule, a substantial improvement over existing algorithms.

# 1 Introduction

In the last decade, CPU performance has outstripped I/O performance and the size of data sets has increased rapidly, resulting in a growing communications bottleneck. This problem has instigated new research in parallel IO from low level solutions, such as disk striping, through operating system and compiler support for I/O to high level algorithmic solutions for “out-of-core” computations. In this paper, we propose and evaluate distributed algorithms to preschedule I/O requests to eliminate contention for I/O ports while maintaining an efficient use of bandwidth.

We formalize the distributed I/O scheduling problem in Section 2. In Section 3, we introduce our algorithm, which is based on edge coloring. Other applications of edge coloring to distributed communications problems are surveyed in Section 4. In Section 5 we present an experimental analysis of our algorithms based on simulation and compare them with previous work. Results show that our algorithm yields schedule lengths close to the optimum solution achieved in the centralized case; within 5% for some parameter choices. This is a substantial improvement over previous algorithms which give schedules 50% - 60% greater than the optimum. In Section 6, we discuss the communication complexity of the scheduling stage and enumerate the issues required for an analysis of running time.

## 2 The Distributed I/O Scheduling Problem

We consider communication in an architecture based on clients and servers connected by a crossbar network, such as processors and disks in a multiprocessor I/O subsystem, workstations and disk arrays in a network of workstations or disk caches and tape archives in a video-on-demand system. Every client can communicate with every server. Each client has a queue of data transfer requests (reads or writes) destined for various servers. Transfers take place in units of fixed-size blocks and preemption is permitted at block boundaries. Clients and servers can handle at most one data transfer at any given time, yet there is no hardware support for fast communication between clients that would allow the arbitration of conflicts in processing transfer

requests. The object is to process all requests as fast as possible without violating the “one data transfer at a time” constraint.

One approach is to use algorithms in which conflicts at the servers are managed dynamically using retry algorithms, buffering or by discarding packets. In contrast, our approach uses prescheduling to resolve conflicts. Data transfer requests are assigned to time slots during a scheduling stage. The data transfers are then executed according to the schedule in the data transfer stage. Requests that arrived during the data transfer stage are then scheduled in the next scheduling stage. The length of a request message is much shorter than the transfer length, so that the cost of sending prescheduling messages is (more than) offset by the reduction in the number of time slots required to complete the data transfers. This assumption is appropriate for data intensive, I/O bound applications.

An example is shown in Figure 1. Outstanding data transfer requests between two clients and three servers are represented as edges in a bipartite graph, with the clients on the left hand side and the servers on the right (Figure 1(a)). Clients  $c_1$  and  $c_2$  each have two pending requests. A conflict exists at server  $s_2$  which is the target of two requests. Figure 1(b) shows a schedule for this request graph with time slots on the x-axis and transfers scheduled for each slot on the y-axis. Transfers  $T_1$  and  $T_4$  are scheduled simultaneously in the first time slot, a legal schedule since  $T_1$  and  $T_4$  share neither a client nor a server. However, to avoid

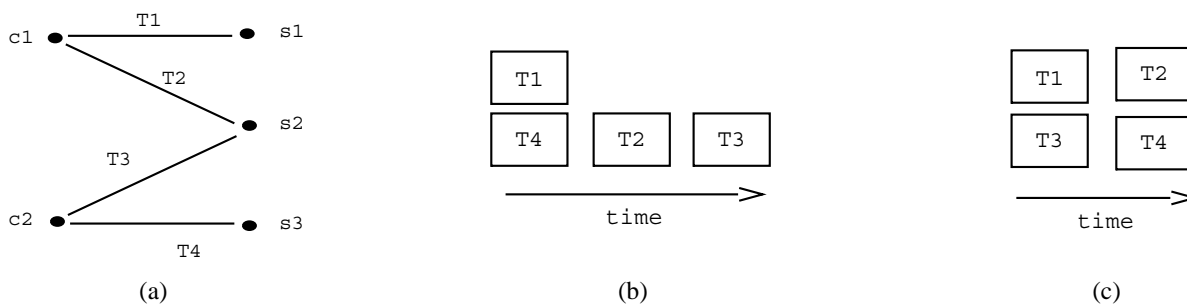


Figure 1: A transfer request graph with two possible schedules

the conflict at  $s_2$ ,  $T_2$  and  $T_3$  must be scheduled separately, resulting in a schedule with three slots. A shorter schedule can be obtained by scheduling  $T_3$  with  $T_1$  and  $T_2$  with  $T_4$  (Figure 1(c)).

Scheduling data transfers can be viewed as an edge coloring problem, where each time slot in a schedule is a matching in the bipartite transfer graph. An *edge coloring* is an assignment of colors to edges in  $E$  such that no vertex has two edges of the same color adjacent to it. A *matching* in a bipartite graph  $G = (A, B, E)$  is a subset  $E' \subset E$  such that no two edges in  $E'$  share a vertex. Each color in an edge coloring is a matching. To obtain an optimal edge coloring, every matching must be a *critical* matching; i.e., must include an edge adjacent to the highest degree node. In contrast, the primary goal of a matching algorithm is to obtain *maximum* matchings, which is not necessary for edge coloring. At least  $\Delta$  colors are required to edge color a bipartite graph with maximum degree  $\Delta$  [3].

Efficient centralized algorithms to obtain optimal edge colorings exist (see [12] for a survey). Previous simulation studies have shown that when the entire set of outstanding transfer requests is known, static scheduling algorithms can reduce the time required to complete a set of data transfers by up to 40% [13, 14]. However, in many distributed architectures, global information about I/O requests is not centrally available and clients (and similarly servers) have no shared memory or private shared network allowing fast communication between them. In the current paper, we present a distributed, randomized algorithm for prescheduling I/O requests based on on a small number of rounds of bidding between clients and servers.

### 3 A Distributed Edge Coloring Algorithm

Our algorithm is based on a distributed, two-step bidding system that computes a matching. In its simplest form, each client selects one of its incident edges uniformly at random and sends a message to the associated server, proposing to color that edge. In the second step, each server resolves conflicts by selecting, uniformly at random, one of the proposals received and sending back an accept message. Proposals from other clients are rejected. Communication is synchronous; that is, all clients (servers) communicate at fixed times. Clients assign the current color to the winning edges and remove those edges from the graph. A fresh, new color is obtained and the process is repeated until all edges are colored.

This simple bidding procedure is the inner loop of the parameterized algorithm shown in Figure 2 (when

$Ncolors = 1$ ). The matching obtained execution of the inner loop can be quite sparse. To improve the

```

1. While (G = (A, B, E) is not empty)
2.   {
3.     Get Ncolors new colors.
4.     For i = 1 to Npasses
5.       {
6.         For all clients: Assign colors to edge(s).
7.         For all servers: Resolve conflicts between bids.
8.       }
9.     Delete colored edges and vertices of zero degree from G.
10.  }
```

Figure 2: A parameterized scheduling algorithm

schedule length, we use two approaches to obtain denser matchings. The first is to use *multiple passes* (*MPASSES*), where a *pass* is a single round trip in the bidding process. With *MPASSES*, those clients whose proposals were rejected in the first pass have an opportunity to assign the current color to a different adjacent edge in subsequent passes. We call the second approach the *Highest Degree First* or *HDF* heuristic. When the clients send their proposals to the servers, they also send their current degree. Each server grants the request of the highest degree client, with ties broken arbitrarily. By favoring high-degree nodes, *HDF* increases the probability of obtaining a critical matching. Note that *MPASSES* explicitly increases the size of the matchings and only indirectly increases the probability of a critical matching.

The time to generate the schedule depends on the number of rounds of bidding required to edge color the graph. If additional bandwidth is available, the scheduling stage can be shortened by computing several matchings simultaneously (when  $Ncolors > 1$ ). At the beginning of each iteration of the outer loop in Figure 2, called a *phase*,  $Ncolors$  fresh new colors are selected. Each client selects up to  $Ncolors$  edges uniformly at random and then assigns a random permutation of the current colors to these edges. It then sends  $Ncolors$  messages to the servers. In this case, the servers must resolve conflicts between bids of the same color, selecting up to  $Ncolors$  winners.

## 4 Related Work

A number of studies have modeled distributed routing problems as edge coloring on a communication graph and presented solutions based on randomized bidding schemes similar to ours. These approaches apply a similar idea to different architectural models and hence focus on optimizing different parameters.

In the optical computing (OCP) model [1], an  $h$ -relation [19] is a routing problem where every processor has at most  $h$  messages to send and is the destination of at most  $h$  messages. This is analogous to edge coloring a general graph of maximum degree,  $h$ . The focus is on minimizing the number of steps required to route all  $h$  messages (or color all edges) under the constraints of the OCP model. Much of this work deals with the difficulty of edge coloring sparse graphs [9, 10, 16]. Gereb-Graus and Tsantilas [8] posit that in order to achieve an optimal coloring,  $h$  must be bounded below by  $\log n \log \log n$ , where  $n$  is the number of nodes, and present an algorithm that is optimal to a constant factor for that case. Goldberg *et al.* [9] show that sparse graphs may be routed faster if messages are first sent to intermediate nodes to reduce contention by balancing the communication load. This approach is not possible in our model since there is no client-client or server-server communication.

As part of a divide-and-conquer approach to edge-coloring general graphs, Panconesi and Srinivasan [17, 18] present a distributed edge coloring algorithm for bipartite graphs. Their algorithm (*PS*) is based on the same two-step bidding routine that we used but does not use any additional machinery to improve the size of the matchings. *PS* also computes several matchings at every phase, but unlike our algorithm, uses an adaptive approach to selecting the number of fresh new colors used in each phase. Initially  $N_{colors} = \Delta$ , where  $\Delta$  is the maximum degree of the graph. In subsequent phases,  $N_{colors} = \Delta(i)$ , a probabilistic estimate, based on an extension to Chernoff-Hoeffding bounds, of the degree of the graph in phase  $i$ . These bounds are not valid for sparse graphs, so the algorithm switches to a randomized, deterministic vertex-coloring algorithm proposed by Luby [15] once  $\Delta(i)$  drops below  $\delta$ , the threshold of validity. Panconesi and Srinivasan show analytically that their algorithm requires at most  $1.6\Delta + O(\log^{1+\delta} n)$  colors to edge-color the entire graph in at most  $O(\log \Delta)$  phases. Both Gereb-Graus and Tsantilas and Panconesi and Srinivasan require the initial

degree of the graph to estimate the current degree of the graph at each stage and to estimate the number of steps required to complete a coloring. Unfortunately, determining the true degree of a distributed graph requires  $O(\log \Delta)$  communication, making these algorithms difficult to use in practise.

In extensions of this work, Dubhashi, Panconesi and Grable describe faster randomized, distributed edge coloring algorithms [5, 4, 11] that require at most  $O((1 + O(1))\Delta)$  colorings with high probability. These latter algorithms, which were designed for general graphs, are not suitable for our application because they assume substantial computational power on the part of the servers, have an arbitrarily small but nonzero failure probability and require a large initial palette of colors, which could lead to the problem of holes in the schedule described on page 11.

In a packet switching context, Anderson *et al.* [2] present a randomized, distributed algorithm for computing bipartite matchings to route data cells from inputs to outputs in DEC's AN2 ATM communications switch. Because of their application, this work is geared towards meeting real time constraints and minimizing delay associated with individual requests rather than minimizing the time required to complete a set of outstanding requests.

The distributed bidding algorithms surveyed here focus on different applications and, hence, address different issues: schedule length, matching size, fairness, the time to compute a schedule and obtaining good performance when the graph is sparse. In this paper we focus on heuristics to obtain short schedules. In the discussion section, we discuss the implications of our work for some of these other goals and suggest problems for future work.

## 5 Experimental Evaluation of Schedule Length

Previous work [13, 14] has shown that prescheduling data transfers can speedup parallel I/O subsystems when the entire I/O load can be analyzed by a single processor. Can such an improvement also be achieved in a distributed I/O environment where centralized scheduling is not possible? Lacking complete information concerning the graph, distributed edge coloring algorithms generally cannot obtain the optimal schedule

length,  $\Delta$ . We used simulation to compare the performance of our distributed algorithms with the centralized optimum as a function of algorithmic parameters ( $Npasses$ ,  $HDF$ ,  $Ncolors$ ) and properties of the input graph (size, density, uniformity). We also looked at the impact of computing several matches simultaneously on the schedule length.

The simulator takes  $Ncolors$  and  $Npasses$  as parameters and selects winners either uniformly at random or using the  $HDF$  heuristic. Given a bipartite graph representation of a set of transfer requests, it computes a schedule using the distributed edge coloring algorithm described above. The output is normalized schedule length, the schedule length divided by  $\Delta$ , allowing the comparison of graphs of different sizes and densities. The assignment of transfers to time slots can also be viewed graphically.

Experiments were run on  $N \times N$  random bipartite graphs, where  $N \in \{16, 32, 64\}$ , with an edge density of one half (i.e.,  $G = (A, B, E)$ , where  $|A| = |B| = N$  and  $E = N^2/2$ ). Multiple edges were allowed, since multiple requests between the same client and server are possible in a distributed system. Two types of graphs, *uniform* and *hotspot*, were generated as test data. In uniform graphs, the expected degree of every vertex is  $N/2$ . In hotspot graphs, a distinguished vertex has an expected degree of  $Hd$ , where  $d$  is the expected degree of the other vertices and  $H \in \{2, 4\}$ . These graphs, designed to study performance on a biased I/O load, contained either a single client hotspot or a single server hotspot. The results are presented graphically below. For uniform graphs, every data point is an average of 100 experiments; that is, ten colorings each of ten graphs. Because hot graphs have greater variance, we ran 225 executions of each hotspot experiment (15 colorings of 15 graphs). The error bars are 99% confidence intervals.

## 5.1 Impact of Heuristics for Improving Schedule Length

Schedule lengths obtained by our algorithm, when schedules are computed one matching at a time ( $Ncolors = 1$ ), are shown in Figure 3(a). The basic bidding strategy results in schedules that are 20% to 40% longer than optimal. The data shows that both  $HDF$  and  $MPASSES$  improve schedule lengths, with eight passes bringing schedule lengths within 5% of optimal for all graphs studied.

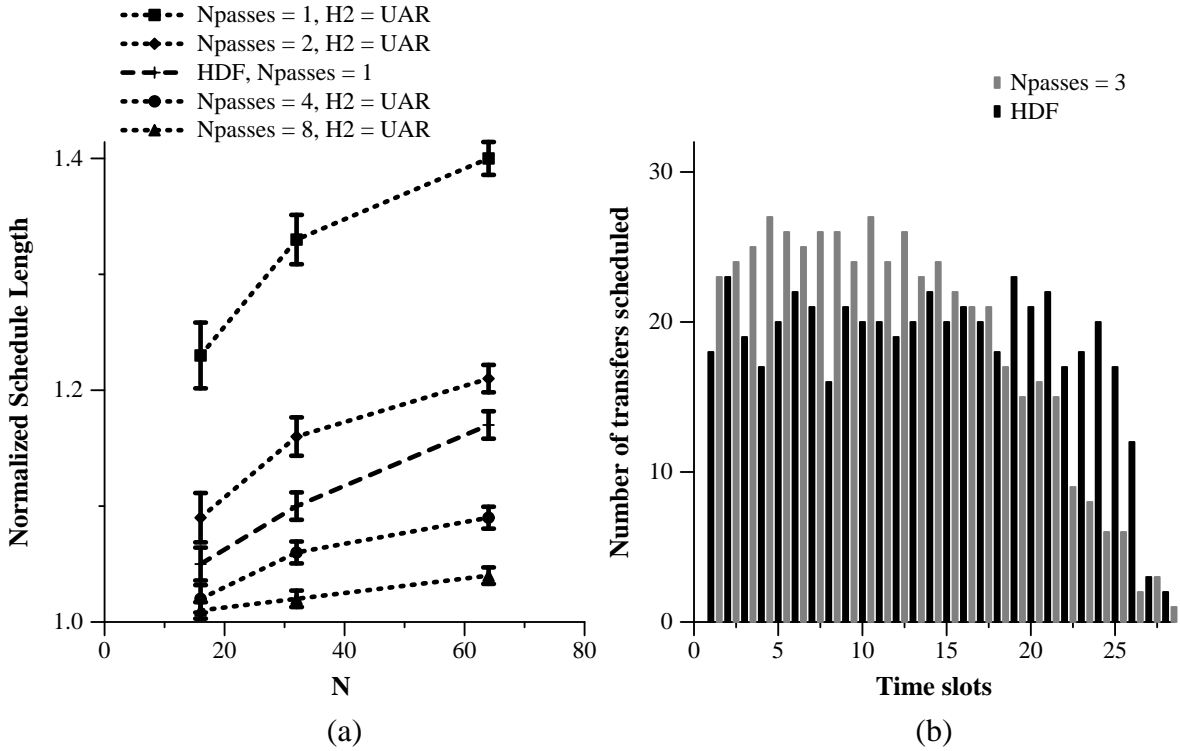


Figure 3: Impact of *HDF* and *MPASSES* on schedule length when  $N_{colors} = 1$ : (a) normalized schedule length as a function of graph size, (b) comparison of two different schedules, obtained using *HDF* and three passes, for the same  $32 \times 32$  graph.

Comparison of *HDF* with *MPASSES* for these experiments, suggests that *HDF* gives about the same schedule length as three passes. Juxtaposing one schedule obtained with each method for the same problem instance shows that the schedules they generate are qualitatively different. Figure 3(b) shows matchings obtained in two executions that gave exactly the same schedule length. We see that *HDF* gives better load balancing; except for the last two matchings, all matchings are quite large and of the same size. The execution based on three passes shows larger matchings at the beginning of the execution but the size of the matchings begins to decrease about half way through the run.

We also ran the simulator on hotspot graphs to see how the distributed algorithm behaves on biased I/O loads (Figure 4). The experiments taken as a whole show that the greater the bias, the closer the performance of the distributed algorithms to the centralized optimum. An optimal edge coloring requires that every matching be a critical matching. The distributed algorithm is suboptimal because it is not always

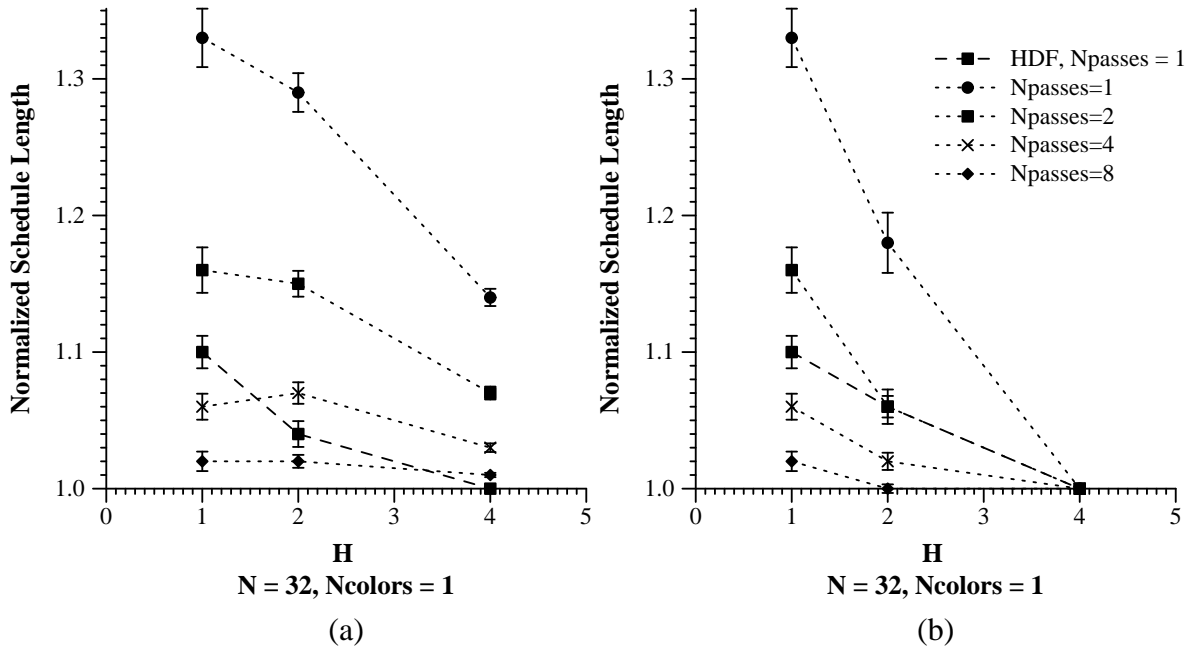


Figure 4: Impact of  $HDF$  and  $MPASSES$  on hotspot graphs when  $Ncolors = 1$ : (a) client hotspots, (b) server hotspots.

possible to find a highest degree node in the absence of global information. However, the greater the value of  $H$ , the greater the probability that the nondeterministic bidding procedure will succeed in coloring an edge required for a critical matching.

Comparing the plots in Figure 4 shows that server hotspots are easier to color than client hotspots. When the highest degree node is a server, a critical matching will be obtained if that server gets at least one bid for every color. When there is a hotspot on the server side, the probability that the highest degree server will get a bid is very high. Client hotspots do not have this advantage since they must still compete with all other bidding clients. However,  $HDF$  does very well with client hotspots, since  $HDF$  favors the client that is the coloring bottleneck.

## 5.2 Impact of Heuristics for Reducing the Number of Phases

In Section 3, we observed that the number of phases required to edge color a graph can be reduced by computing several matches simultaneously. Our algorithm allows the user to set  $Ncolors$ , the number of

simultaneous matches, as a fixed parameter. In *PS* several matches are also computed simultaneously, but the number of simultaneous matches is reduced adaptively. What is the impact of bidding several matches at once on schedule length? We used our simulator to study this question experimentally. For comparison purposes, we also implemented a version of *PS* adapted for bipartite graph models of client-server architectures, called *mPS* that continues the bidding process until all edges are colored instead of switching to Luby’s algorithm.

Figure 5(a) shows that *mPS* requires between  $1.5\Delta$  and  $1.6\Delta$  colors to edge color the graphs we studied, which is comparable to Panconesi and Srinivasan’s analytical prediction. Our algorithm achieves its

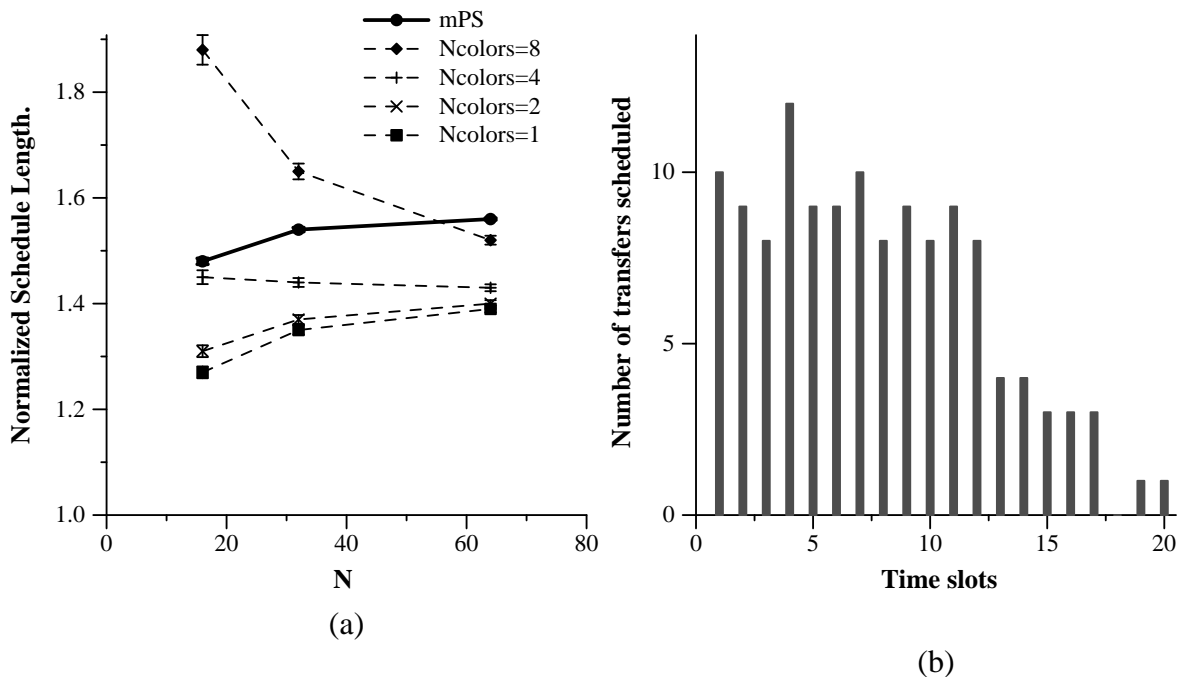


Figure 5: Impact of *Ncolors* on schedule length: (a) normalized schedule length as a function of graph size when  $N_{passes} = 1$  and winners are selected uniformly at random, (b) a schedule from a single execution on a  $16 \times 16$  uniform graph.

performance when  $N_{colors} = 1$ . As  $N_{colors}$  increases, schedule length increases as well, particularly for the smaller graphs. This can be understood by examining the coloring shown in Figure 5(b). The matchings are quite full for the first half of the run but then begin to decrease in size. In particular, color 18 is not used at all. In a given phase, a palette of  $N_{colors}$  colors will only be used optimally if  $N_{colors}$  critical matchings are

obtained. This requires that in each phase, a highest degree node colors  $Ncolors$  adjacent edges. Because of the randomness in the algorithm, this becomes decreasingly likely as the ratio of  $Ncolors$  to  $\Delta(i)$  increases. If  $Ncolors$  is larger than  $\Delta(i)$ ,  $\Delta(i) - Ncolors$  time slots will not be used at all, creating “holes” in the schedule. An analysis of the maximum number of holes that can occur is presented in Durand *et al.* [6]. Even when holes do not occur, unless  $Ncolors$  is small compared with  $\Delta$ , colors will be used inefficiently, leading to sparser matchings. Although  $mPS$  reduces the number of colors used in each phase to  $Ncolors = \Delta(i)$ , it yields relatively poor schedules, suggesting that although  $Ncolors = \Delta(i)$  is the minimum number of colors needed to color the entire remaining graph at each phase, it is too many colors to generate good matchings.

Figure 6 shows how  $MPASSES$  and  $HDF$  behave on 32x32 uniform graphs when  $Ncolors > 1$ . Clearly,  $MPASSES$  is less sensitive than  $HDF$  to increased values of  $Ncolors$ . This is not surprising, since  $MPASSES$

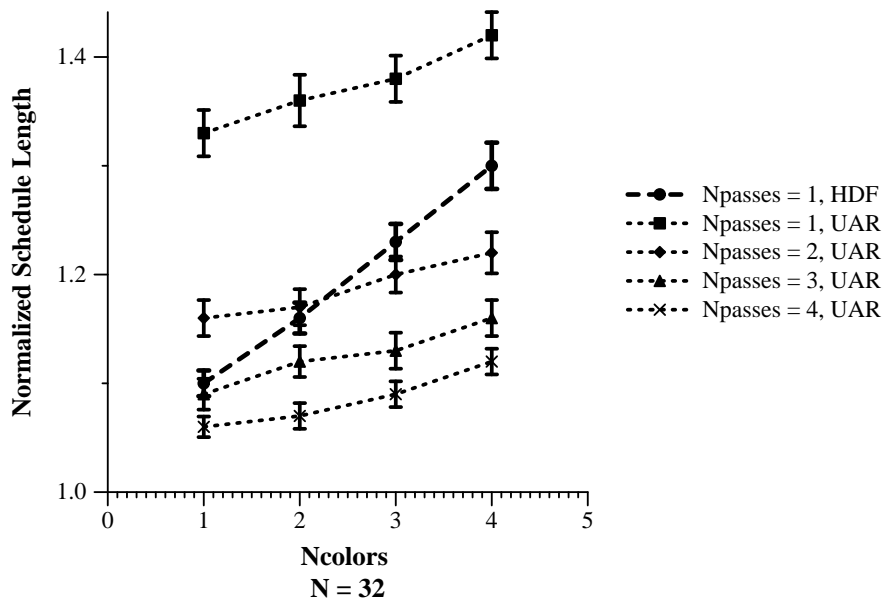


Figure 6: Interaction of multiple colors per phase with  $MPASSES$  and  $HDF$  on a 32x32 uniform graph.

compensates for the sparse matchings that result when  $Ncolors$  is larger than or comparable to  $\Delta$ . For hotspot graphs, the tendency of large values of  $Ncolors$  to inflate schedule length decreases as  $H$  increases (Figure 7). This is not surprising since schedule length increases when  $Ncolors \sim > \Delta$ , which occurs less frequently when the initial degree is high.  $mPS$  seems only mildly sensitive to hotspots in the client case and

not sensitive at all in the server case. This is because in *mPS* the value of *Ncolors* scales with the degree of the graph.

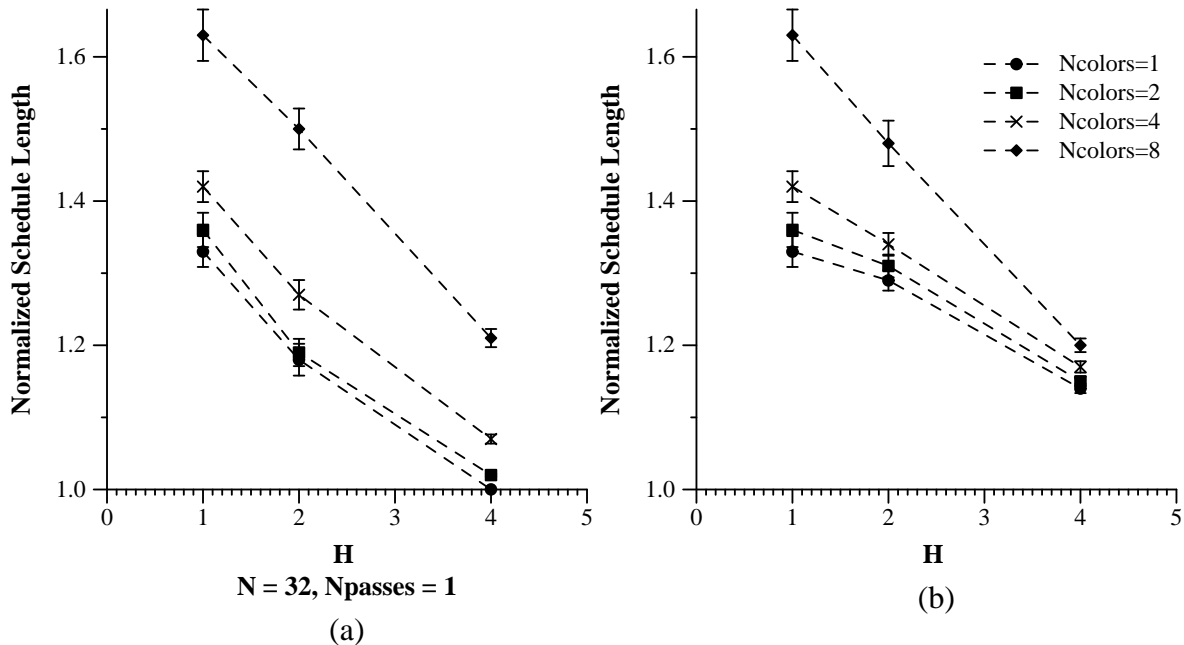


Figure 7: Impact of *Ncolors* on hotspots in a 32x32 graph when *Npasses* = 1 and winners are selected uniformly at random: (a) server hotspot, (b) client hotspot.

## 6 Discussion

Our experiments in the previous section have shown that a randomized, distributed scheduling algorithm, based on edge coloring, can achieve schedules very close to those obtained in the deterministic, centralized case, suggesting the appropriateness of this approach for parallel I/O. The results in Figure 4 show that both *HDF* and multiple passes can be used to obtain shorter schedules. *HDF* can only be used if there is sufficient computational power on the servers. The optimal number of passes will depend on the trade off between scheduling time and schedule length, which in turn depends on the communications cost associated with each pass.

Figure 4(b) shows that *HDF* makes more consistent use of the bandwidth available, whereas *MPASSES* is more sensitive to the sparseness of the graph. In architectures where the bandwidth is limited, *HDF* will

provide better load balancing. In dynamic applications, such as the packet switch described by Anderson *et al.* [2], *MPASSES* would make poor use of bandwidth if the traffic is bursty but maximize the use of bandwidth for consistently heavy traffic loads. In contrast, *HDF* would utilize bandwidth well under varied loads but could raise issues of fairness and starvation since nodes with few requests would be rarely serviced.

This study has focussed on schedule length and has not addressed the issues of bandwidth and the time required to compute the schedule. In order to address these questions, an architectural model is needed. First, although scheduling eliminates conflicts during the data transfer stage, conflicts can still occur in the scheduling stage. The cost of resolving conflicts between messages depends on the communications protocol. The communications costs for our algorithm were discussed in Durand *et al.* [7] for two such protocols, a randomized crossbar network and communication via preassigned time slots. Second, the cost of the scheduling stage is proportional to the number of phases times the cost per phase. Estimating the cost per phase requires also requires an architectural model with specific computational and communication costs and bandwidth constraints. Once the cost of the scheduling stage has been determined, the speedup associated with prescheduling can be estimated and the benefits of prescheduling versus other protocols, such as retry, packet loss and exponential backoff, can be analyzed.

Finally, an architectural model is required to compare our approach with other algorithms. Adaptively reducing the number of colors per phase, allows PS to generate a schedule in  $O(\log \Delta)$  time. In comparison, we require  $\approx (1 + \epsilon)\Delta$  phases, where experimental results suggest that  $\epsilon$  is generally less than .5. However, Panconesi and Srinivasan permit each processor to receive and send one message to each neighbor (up to  $O(N\Delta)$  communications) during each phase, as well as unlimited computation. Our algorithm requires only  $O(N)$  messages per phase. Furthermore, as our experimental results with  $Ncolors > 1$  show, although increased values of  $Ncolors$  can reduce the number of phases, a price is paid for this performance improvement in increased schedule length. A more detailed model of interprocessor communication is necessary in order to fully understand the complexity of our algorithm and that of PS.

## 7 Conclusion

In this article we have presented a randomized, distributed edge coloring algorithm to preschedule data transfers in a fully connected distributed architecture. Prescheduling data transfers eliminates contention for I/O ports while maintaining efficient use of bandwidth. Using simulation, we measured the length of the schedule required to complete a set of data transfers with these algorithms. We evaluated two approaches to reducing the schedule length: the *Highest Degree First* or HDF heuristic, which favors nodes with a heavy I/O load, and *Multiple Passes (MPASSES)* which uses additional communication to increase bandwidth utilization. Experimental results showed both approaches yielded good performance. When compared with the optimal schedule achieved in the centralized case, our best algorithm yielded schedules within 5% of the optimal solution. Previous work by other authors required roughly 50% to 60% above the optimal schedule.

Our simulations also gave insight into how the two methods differed in achieving these results. From looking at traces of individual runs, we concluded that *MPASSES* is more suitable for dynamic applications such as communications switches and I/O in multitasked systems. *HDF* is more appropriate for batch oriented problems like “out-of-core” algorithms. *MPASSES* incurs a higher communication cost whereas *HDF* requires greater computational power on the disks. Our parameterized approach offers an algorithm that can be tailored to a wide variety of architectural characteristics. We also studied computing several matchings simultaneously to reduce the time required to generate schedules. This approach was shown to incur a penalty in increased schedule length.

## Acknowledgements

We thank Bill Aiello, Sandeep Bhatt, Alessandro Panconesi, Aravind Srinivasan and Mark Sullivan for useful discussions.

## References

- [1] ANDERSON, R. J., AND MILLER, G. L. Optical Communication for Pointer-based Algorithms. Tech. Rep. CRI-88-14, Computer Science Department, University of Southern California, 1988.
- [2] ANDERSON, T. E., OWICKI, S., SAXE, J. B., AND THACKER, C. P. High-Speed Switch Scheduling for Local-Area Networks. *ACM Transactions on Computer Systems* 11, 4 (Nov. 1993), 319–352.
- [3] BERGE, C. *Graphs*. North Holland, 1985.
- [4] DUBHASHI, D., GRABLE, D. A., AND PANCONESI, A. Near-Optimal Distributed Edge Coloring via the Nibble Method. Invited for a special issue of Theoretical Computer Science dedicated to ESA 95, under review, 1997.
- [5] DUBHASHI, D., AND PANCONESI, A. Near-Optimal Distributed Edge Coloring. In *Algorithms: Third Annual European Symposium (Lecture notes in computer science)* (September 1995), P. Spirakis, Ed., Springer Verlag, pp. 448–459.
- [6] DURAND, D., JAIN, R., AND TSEYTLIN, D. Distributed scheduling algorithms to improve the performance of parallel data transfers. In *Input/Output in Parallel and Distributed Computer Systems*, R. Jain, J. Werth, and J. C. Brown, Eds. Kluwer, 1996, pp. 245–268.
- [7] DURAND, M., JAIN, R., AND TSEYTLIN, D. Applying Randomized Edge Coloring Algorithms to Distributed Communication: An Experimental Study. In *1995 Symposium on Parallel Algorithms and Architectures* (July 1995), pp. 264–274.
- [8] GEREB-GRAUS, AND TSANTILAS. Efficient Optical Communication in Parallel Computers. In *1992 Symposium on Parallel Algorithms and Architectures* (1992), pp. 41–48.
- [9] GOLDBERG, L. A., JERRUM, M., LEIGHTON, T., AND RAO, S. A Doubly Logarithmic Communications Algorithm for the Completely Connected Optical Communication Parallel Computer. In *1993 Symposium on Parallel Algorithms and Architectures* (1993), pp. 300–310.

- [10] GOLDBERG, L. A., JERRUM, M., AND MACKENZIE, P. D. An  $\omega(\sqrt{\log \log n})$  Bound for Routing in Optical Networks. In *1994 Symposium on Parallel Algorithms and Architectures* (1994), pp. 147–156.
- [11] GRABLE, D., AND PANCONESI, A. Nearly Optimal Distributed Edge Colouring in  $O(\log \log n)$  Rounds. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (1997), pp. 278–285.
- [12] JAIN, R. Scheduling data transfers in parallel computers and communications systems. Tech. Rep. TR-93-03, Univ. Texas at Austin, Dept. of Comp. Sci., Feb. 1993.
- [13] JAIN, R., SOMALWAR, K., WERTH, J., AND BROWNE, J. Scheduling Parallel I/O Operations in Multiple Bus Systems. *Journal of Parallel and Distributed Computing* 16 (Dec. 1992), 352–362.
- [14] JAIN, R., SOMALWAR, K., WERTH, J., AND BROWNE, J. Heuristics for Scheduling Parallel I/O Operations. *IEEE Transactions on Parallel and Distributed Systems* (1997). To appear.
- [15] LUBY, M. Removing Randomness in Parallel Computation without a Processor Penalty. In *Proceedings of the IEEE Symposium on Foundations of Computer Science* (1988), pp. 162–173.
- [16] MACKENZIE, P. D., PLAXTON, C. G., AND RAJARAMAN, R. On Contention Resolution Protocols and Associated Probabilistic Phenomena. In *Proceedings of the ACM Symposium on the Theory of Computation* (1994), vol. 26.
- [17] PANCONESI, A., AND SRINIVASAN, A. Fast Randomized Algorithms for Distributed Edge Coloring. In *Proceedings of the 1992 ACM Symposium on Parallel and Distributed Computing* (Aug. 1992), pp. 251–262.
- [18] PANCONESI, A., AND SRINIVASAN, A. An Extension of the Chernoff-Hoeffding Bounds with an Application to Distributed Edge Coloring. To appear in *SIAM Journal of Computing*, 1997.
- [19] VALLIANT, L. G. General Purpose Parallel Architectures. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier, 1990, p. 967. Chapter 18.