A Performance Evaluation of CP List Scheduling Heuristics for Communication Intensive Task Graphs

Benjamin S. Macey and Albert Y. Zomaya*
Parallel Computing Research Laboratory
The University of Western Australia
Nedlands, WA, 6907, Australia
{maceyb,zomaya}@ee.uwa.edu.au

Abstract

List-based priority schedulers have long been one of the dominant classes of static scheduling algorithms. Such heuristics have been predominantly based around the "critical path, most immediate successors first" (CP/MISF) priority. The ability of this type of scheduler to handle increased levels of communication overhead is examined in this paper. Three of the more popular list scheduling heuristics, HLFET [1] and ISH and DSH [10], plus the Mapping Heuristic [4,6] are subjected to a performance based comparison, with results demonstrating their inadequacies in communication-intensive cases. Performance degradation in these instances is partly due to the level alteration problem, but more significantly to conservative estimation of communication costs due to the assumption of zero link contention. The significance of this component of communication is also examined in this paper.

1. Introduction

Task scheduling is one of the most challenging problems facing parallel programmers today. It is known to be NP-complete in its general form as well as in many restricted cases [3, 16]. By restricting the scheduling problem, either through assumptions made about the target machine or limitations imposed upon the task graph, many researchers have been able to obtain optimal scheduling algorithms [3,8,15]. To solve the problem in the general case sub-optimal heuristics, which have the advantage of polynomial time-order, must be used.

List-based heuristics, one of the earliest proposed solutions, involve assigning priorities to each task and then scheduling tasks ready-to-run in order of this priority. Using the classification of Al-Mouhamed and Al-Maasarani [2] they are graph-driven. The Mapping Heuristic, on the other hand, incorporates modifications so as to make it processor-driven.

List scheduling heuristics were originally devised with the assumption of zero inter-task communication costs. Even with this simplification, the scheduling problem remains NP-complete. However, research [7] has shown that in the absence of communication, list scheduling heuristics have a guaranteed performance within 50% of the optimum. Common sense suggests that different priority schemes will be better suited to different cases, giving better performance. Hence, the method of assignment of task priority is important. Adam et al [1] demonstrated experimentally that the "critical path" (CP) or "highest level first" (HLF) scheme is within 5% of the optimal in 90% of cases.

More recently, interest in message passing MIMD architectures has highlighted the significance of communication costs in scheduling. For these architectures the assumption that communication overhead is negligible no longer holds true [14], making the scheduling problem much more complex. Under these conditions the CP list heuristics suffer from the problem of level alteration, which has so far eluded satisfactory solution.

2. Models

An acyclic parallel program can be easily described using an "enhanced directed acyclic graph" or EDAG [9], where nodes represent tasks, and arcs are communication dependencies and/or precedence relations. A task graph can be characterized by the ratio of average communication to processing cost, its C/P ratio. Another measure of the level of communication involved in a task graph is its average degree, the number of arcs in the task graph divided by the number of nodes. These characteristics are combined in the CCR ratio, which is the ratio of the total amount of communication to computation.

* This work was supported by Australian Research Council grant 04/15/412/131,

These measures can be used to classify the level of communication in the task graph as high, medium or low.

This work assumes the target architecture to be a fully connected message passing system comprising an arbitrary number of processing elements (each with a separate I/O processor performing store-and-forward routing) connected via a network of links. With many scheduling algorithms communication is further assumed to be contention free, meaning that multiple messages can be using the same link at the same time without affecting each other. In this work, this is only true in the initial scheduling stages. When rescheduling is performed, the connection network is assumed to be made up of bi-directional links, which means that messages traveling in either direction along a link do not interfere with each other. Static message routing is used.

The scheduling heuristics considered in this paper are based on the macro-dataflow model of execution. A task receives all inputs before commencing execution and then runs without pre-emption until completion, at which point it sends any outputs to its successors. The algorithms are deterministic or static, meaning that all information must be available *a priori*.

3. Scheduling Heuristics

The basic list scheduling heuristic (LSH) used in this comparison is a modification of that examined by Adam et al [1]. The CP/MISF priority (or HLFET as it was then called) is based on the task's level (the length of the longest path from that node to an exit node). Traditionally, only task execution times are considered when calculating levels; communication costs then play no part in scheduling decisions, except as they affect task starting times.

Kruatrachue's Insertion Scheduling Heuristic (ISH) represents an improvement over LSH in that it tries to assign ready tasks to the idle time slots resulting from communication delays. Essentially the algorithm attempts to avoid the effects of latencies by overlapping communication with computation. Several variants are possible [5]:

ISH0: The hole task in this case is any task from the ready queue that fits into the idle slot.

ISH2. To be a hole task, a task must be able to execute within the idle slot but must not be able to start earlier on any other processor, a condition enforced by a routine to find each candidate hole task's "best" processor.

The list scheduling heuristics can be further improved by the duplication of certain key tasks on multiple processors so as to reduce interprocessor communication. The Duplication Scheduling Heuristic (DSH) attempts to reduce latency and improve a task's starting time by the re-calculation of some of its inputs in preference to their transmission through the connection network.

The recursive nature of the algorithm means that at each task allocation step, tasks for duplication are considered in a depth-then-breadth manner - tracing back through the task's "latest immediate predecessor" (the predecessor responsible for the task's greatest wait) or LIP chains to find tasks for duplication.

As with ISH there are several versions of the DSH algorithm:

DSH1: At most a single task is duplicated in the idle slot created by a task assignment.

DSH2: The task duplication process exhaustively checks the task's LIP chains, continuing until either the idle slot is filled, an entry node has been reached, or the LIP task is already assigned to the same processor.

El-Rewini and Lewis' Mapping Heuristic (MH) is a modified list-scheduling algorithm. Ready tasks are still stored in a prioritized queue, but scheduling is time-based, with allocations made as processors become available. MH also uses routing tables with dynamic routing to track network traffic, allowing estimates of contention delays to be included in communication costs. However, these tables are only updated retrospectively (i.e. after both ends of each communication arc have been assigned) so they are not very up-to-date or accurate.

4. Simulating Contention Delay

There are two components to any communication delay: the time required to transmit the message over an empty route, and the time spent waiting for links on the route to become empty. The first component is a linear function of both the message size and the number of individual links comprising the route. If static routing is used, this delay can be easily calculated given the source and destination processors. The second component, the contention delay, is much harder to calculate during scheduling because it is a function of the message traffic in the system and hence varies with time. To allow inclusion of contention delays in the final schedule a second pass must be used to create a communication schedule and then simulate the combined schedules.

As implemented in WINSCHED [13], this second pass uses routing tables and a time-ordered event list in a manner similar to MH. Processor allocations are known at this stage so static scheduling can be used to route messages through the network, updating routing tables with each link traversal. This allows the routing information to be timely and accurate.

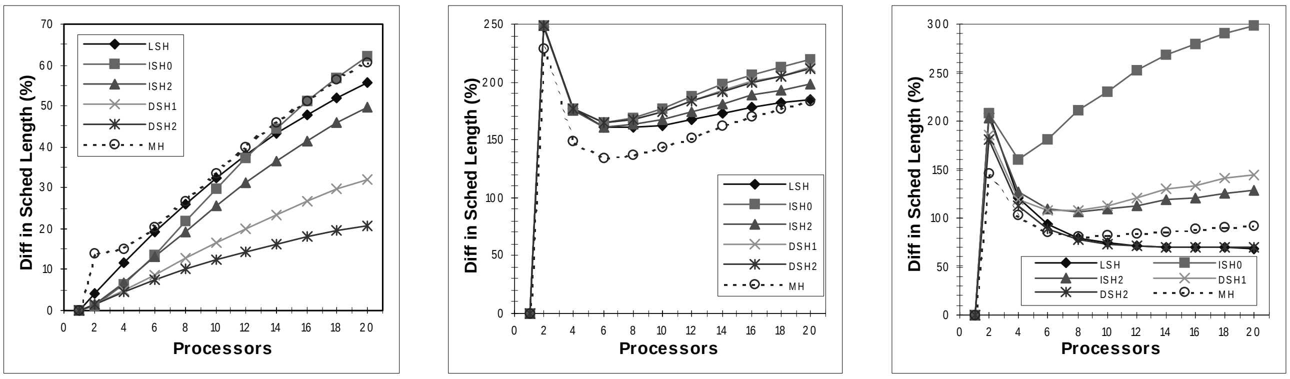

Figure 1. Increase in schedule lengths (%) showing significance of contention delays. a) Linear component significance. b) CD significance (comm ignored in first pass). c) CD significance (first pass includes comm)

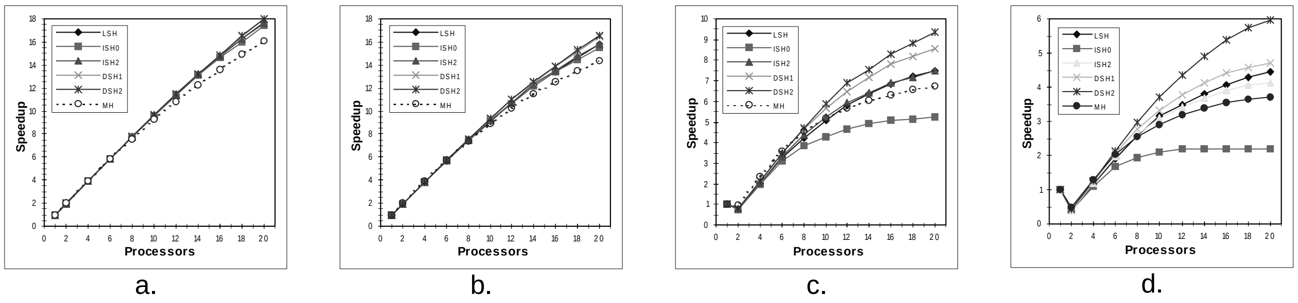

Figure 2. Speedup curves for random task graphs with increasing levels of communication. a) CCR 0.1 b) CCR 1.0 c) CCR 5.0 d) CCR 10.0

5. Performance Comparison

WINSCHED, a scheduling tool similar to Task Grapher [11] and Parallax [12], was used to generate and schedule 600 random task graphs - the results of which were averaged and are presented in this section. The task graphs varied in size from 50 to 500 nodes, with C/P and degree varied from 0.1 to 10.

5.1. Significance of Contention Delays

In this section, the significance of the link contention component of communication delays is examined. This is done by comparing schedule lengths predicted by the first pass of scheduling with those produced by rescheduling. This is put into context by comparison with the difference in first pass schedule lengths when the linear component of communication is included and ignored (Figure 1a).

Figures 1b and 1c show the difference in the schedule lengths before and after the second pass of scheduling. As can be seen from Figure 1b, when communication delays are ignored in the first stage of scheduling, the resultant schedules are completely unrealistic. In Figure 1c, the linear component of communication is considered when scheduling, yet the inclusion of contention delays by the second pass still causes a significant degradation in schedule length.

The typical curve shape of Figure 1c peaks at a small number of processors and then improves somewhat as more processors are added to the system. This is due to the number of links in a fully connected architecture increasing exponentially with the number of processors, thereby significantly reducing contention for larger machines.

MH seems to perform quite well with respect to the correlation between schedule lengths before and after the second pass, at least for smaller architectures. The conservative estimation of communication costs made in the first pass is largely responsible for this result. However, the individual estimates of contention delay are not that accurate, so overall the algorithm does not perform that well after the second pass. As the number of processors, and hence links, in the machine is increased, the accuracy of these estimates is even further diminished.

DSH2 also performed well in this experiment. This is because task duplication in preference to predecessor communication reduces the number of messages required to be sent, and hence has a compounded effect after contention delays are included in the communication model.

A comparison of Figures 1a and 1c shows that the effect of contention delays on schedule lengths is greater than that of the linear component of communication. This means that results quoted for schedulers which assume zero contention (the majority of research to date) are completely unrealistic. Just as the relaxation of the assumption of zero communication changed the face of scheduling, so must the inclusion of link contentions in the communication model.

5.2. Varying levels of communication

To examine the effects of increasing levels of communication on the performance of the heuristics, experimental results were sub-divided into four sets on the basis of the level of communication in the task graph (CCR). The schedule lengths were then averaged and are presented in Figure 2 in the form of speedup curves. In the previous section, the significance of contention delays was demonstrated. Consequently, the schedule lengths used have been produced by WINSCHED’s second pass.

Figure 2 clearly indicates the superiority of the DSH heuristics, especially in the communication intensive cases. In such instances the large idle slots resulting from lengthy communication delays allow the benefits of task insertion and/or duplication to be realized. High degree, a characteristic of communication-intensive task graphs, provides a large number of predecessors for choice of duplication, resulting in DSH2 outperforming the other heuristics.

ISH0 is by far the worst of the heuristics. Its poor performance arises from the “greedy” manner in which it chooses tasks for insertion. Since ISH2 uses a more conservative selection policy, it is guaranteed to produce better schedules than LSH after the first pass. As can be seen from the figures, the concentration of tasks resulting from insertion leads to a high level of link contention which far outweighs its benefits.

MH also performs rather poorly after the second pass. Even though it handles inclusion of realistic contention delays relatively well (as seen in the previous section), this cannot make up for its poor performance during the first pass as a result of unrealistic communication estimates.

It can be seen that list based heuristics are unable to satisfactorily handle communication beyond a certain level. This is evidenced by the large increases in schedule lengths when more realistic communication models were used. At high levels of communication, estimates of communication costs made in the first scheduling pass become meaningless due to the increased significance of contention delays.

References

1. T.L. Adam, K.M. Chandy and J.R. Dickson. A comparison of list schedules for parallel processing systems. *Communications of the ACM*, 17:685-9, 1974.
2. M. Al-Mouhamed and A. Al-Maasarani. Performance evaluation of scheduling precedence-constrained computations on message-passing systems. *IEEE Trans. Parallel and Distributed Systems* 5(12):1317-1322, 1994.
3. E.G. Coffman. *Computer and Job-shop Scheduling Theory.* Wiley, 1976.
4. H. El-Rewini. *Task Partitioning and Scheduling on Arbitrary Parallel Processing Systems*. PhD Thesis, Oregon State University, 1989.
5. H. El-Rewini, H. Ali and T. Lewis. *Task Scheduling in Parallel and Distributed Systems*, PTR Prentice Hall, 1994.
6. H. El-Rewini and T. Lewis. Scheduling parallel programs onto arbitrary target machines. *Journal of Parallel and Distributed Computing,* 9:138-53, 1990.
7. R.L. Graham. Bounds for certain multiprocessing anomolies. *Bell System Tech Journa,l* 45:1563-81, 1966.
8. T.C. Hu. Parallel sequencing and assembly line problems. *Operations Research,* 9:841-8, 1961.
9. J. Hwang, Y.C. Chow, F.D. Anger and C.Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal of Computing,* 18:244-57, 1989.
10. B. Kruatrachue. *Static Task Scheduling and Grain Packing in Parallel Processing Systems*. PhD Thesis, Oregon State University, 1987.
11. T. Lewis, H. El-Rewini, P. Fortner, J. Chu and W. Su. Task grapher: A tool for scheduling parallel program tasks. *Proc. 5th Distributed Memory Computing Conference*, 1171-1178, 1990.
12. T. Lewis and H. El-Rewini. Parallax: a tool for parallel program scheduling. *IEEE Parallel and Distributed Technology,* 1:62-72, 1993.
13. B.S. Macey and A.Y. Zomaya. A comparison of scheduling heuristics for communication intensive task graphs. *Cybernetics and Systems,* 28:535-546, 1997.
14. D.M. Pase. *A comparative analysis of static parallel schedulers where communication costs are significant.* PhD Thesis, Oregon Graduate Inst. Science and Technology, 1989.
15. C. Papadimitriou and M. Yannakakis. Scheduling interval ordered tasks. *SIAM Journal of Computing,* 5:73-82, 1976.
16. J. Ullman. NP-complete scheduling problems. *Journal of Computer System Science,* 10:384-93, 1975.