Analysis, Evaluation, and Comparison of Algorithms for Scheduling Task Graphs on Parallel Processors

Ishfaq Ahmad, Yu-Kwong Kwok
Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong.
Min-You Wu
Department of Computer Science, State University of New York at Buffalo, New York.

Abstract[1]

In this paper, we survey algorithms that allocate a parallel program represented by an edge-weighted directed acyclic graph (DAG), also called a task graph or macro-dataflow graph, to a set of homogeneous processors, with the objective of minimizing the completion time. We analyze 21 such algorithms and classify them into four groups. The first group includes algorithms that schedule the DAG to a bounded number of processors directly. These algorithms are called the bounded number of processors (BNP) scheduling algorithms. The algorithms in the second group schedule the DAG to an unbounded number of clusters and are called the unbounded number of clusters (UNC) scheduling algorithms. The algorithms in the third group schedule the DAG using task duplication and are called the task duplication based (TDB) scheduling algorithms. The algorithms in the fourth group perform allocation and mapping on arbitrary processor network topologies. These algorithms are called the arbitrary processor network (APN) scheduling algorithms. The design philosophies and principles behind these algorithms are discussed, and the performance of all of the algorithms is evaluated and compared against each other on a unified basis by using various scheduling parameters.

Keywords: Algorithms, Multiprocessors, Parallel Processing, Software, Task Graphs.

1 Introduction

Given an edge-weighted directed acyclic graph (DAG), also called a task graph or macro-dataflow graph, the problem of scheduling it to a set of homogeneous processors to minimize the completion time has intrigued researchers since the advent of parallel computers. Since, the problem has been identified as NP-complete in its general forms [10], and polynomial time solutions are known only in a few restricted cases [7], research effort in this area has resulted in a myriad of heuristic algorithms [5], [21]. While each heuristic individually seems to be efficient, a plethora of research has ensued a number of questions: how effective are these algorithms? how sensitive are they to various scheduling parameters? how do they compare against each other on a unified basis? what are the most effective performance measures? how to classify various algorithms? and what possible improvements can be made for a better performance? In this paper we try to answer some of these questions by examining a number of recently proposed algorithms. We start by classifying these algorithms into the following four groups:

- ***Bounded Number of Processors (BNP) scheduling:*** These algorithms schedule the DAG to a bounded number of processors directly. The processors are assumed to be fully-connected.
- ***Unbounded Number of Clusters (UNC) scheduling:*** These algorithms schedule the DAG to an unbounded number of clusters. The processors are assumed to be fully-connected. The technique employed by these algorithms is also called *clustering*.
- ***Task Duplication Based (TDB) scheduling:*** These algorithms also schedule the DAG to an unbounded number of clusters but employ task duplication technique to further reduce the completion time.
- ***Arbitrary Processor Network (APN) scheduling:*** These algorithms perform scheduling and mapping on the target architectures in which the processors are connected via a network of arbitrary topology.

We discuss six BNP, five UNC, six TDB, and four APN scheduling algorithms. We analyze their design philosophies and characteristics, and assess their merits and deficiencies. The rest of this paper is organized as follows. In the next section, we describe the generic DAG model and discuss its variations and suitability to different situations. We describe the BNP scheduling algorithms in Section 3, and the UNC algorithms in Section 4. Section 5 describes the TDB algorithms. The APN algorithms are discussed in Section 6. The performance results and comparisons are presented in Section 7, and Section 8 concludes the paper.

2 The DAG Model

The DAG is a generic model of a parallel program consisting of a set of processes among which there are dependencies. Each process is an indivisible unit of execution, expressed by an atomic node. An atomic node has one or more inputs. When all inputs are available, the node is triggered to execute. After its execution, it generates its outputs. In this model, a set of v nodes $\{n_1, n_2, ..., n_v\}$ are connected by a set of e directed edges, each of which is denoted by (n_i, n_j), where n_i is called the parent and n_j is called the child. A node without parent is called an entry node and a node without child is called an exit node. The weight of a node, denoted by $w(n_i)$, is equal to the process execution time. Since each edge corresponds to a message transfer from one process to another, the weight of an edge, denoted by $c(n_i, n_j)$, is equal to the message transmission time. Thus, $c(n_i, n_j)$

1. This research was supported by the Hong Kong Research Grants Council under contract number HKUST179/93E.

becomes zero when n_i and n_j are scheduled to the same processor because intraprocessor communication time is negligible compared with the interprocessor communication time.

3 BNP Scheduling Algorithms

Most BNP scheduling algorithms are based on the *list scheduling* technique [1], [12], [19], [21]. List scheduling is a class of scheduling heuristics in which the nodes are assigned priorities and placed in a list arranged in a descending order of priority. The node with a higher priority will be examined for scheduling before a node with a lower priority. If more than one node has the same priority, ties are broken using some method.

The two main attributes for assigning priority are the *t-level* (top level) and *b-level* (bottom level). The *t-level* of a node n_i is the length of the longest path from an entry node to n_i (excluding n_i). Here, the length of a path is the sum of all the node and edge weights along the path. The *t-level* of n_i highly correlates with n_i's earliest start time, denoted by $T_S(n_i)$, which is determined after n_i is scheduled to a processor. The *b-level* of a node n_i is the length of the longest path from node n_i to an exit node. The *b-level* of a node is bounded by the length of the *critical path*. A critical path (CP) of a DAG, is a path from an entry node to an exit node, whose length is the maximum. It should be noted that some BNP scheduling algorithms do not take into account the edge weights in computing the *b-level*. To distinguish such definition of *b-level* from the one we described above, we call it the *static b-level*.

Different algorithms use the *t-level* and *b-level* in different ways. Some algorithms assign a higher priority to a node with a smaller *t-level* while some algorithms assign a higher priority to a node with a larger *b-level*. Still some algorithms assign a higher priority to a node with a larger (*b-level* – *t-level*). In general, scheduling in a descending order of *b-level* tends to schedule critical path nodes first while scheduling in an ascending order of *t-level* tends to schedule nodes in a topological order. The composite attribute (*b-level* – *t-level*) is a compromise between the previous two cases. In the following, we discuss very briefly six BNP scheduling algorithms. The detailed steps of the algorithms are omitted due to space limitations.

HLFET Algorithm. The HLFET (Highest Level First with Estimated Times) algorithm [1] is one of the simplest list scheduling algorithms using *static b-level* as node priority.

ISH Algorithm. The ISH (Insertion Scheduling Heuristic) algorithm [14] uses a simple but effective idea of inserting nodes into holes created by the partial schedules.

MCP Algorithm. The MCP (Modified Critical Path) algorithm [23] uses an attributed called ALAP time of a node as node priority. The ALAP times of the nodes on the CP are just their *t-levels*.

ETF Algorithm. The ETF (Earliest Time First) algorithm [3] computes, at each step, the earliest start times for all ready nodes and then selects the one with the smallest start time, which is computed by examining the start time of the node on all processors exhaustively.

DLS Algorithm. The DLS (Dynamic Level Scheduling) algorithm [22] uses as node priority an attribute called *dynamic level* (DL) which is the difference between the *static b-level* of a node and its earliest start time on a processor.

LAST Algorithm. The LAST algorithm [4] is not a list scheduling algorithm, and its main goal is to minimize the overall communication.

4 UNC Scheduling Algorithms

The basic technique employed by the UNC scheduling algorithms is called *clustering* [5], [11], [12]. At the beginning of the scheduling process, each node is considered as a cluster. In the subsequent steps, two clusters are merged if the merging reduces the completion time. This merging procedure continues until no cluster can be merged. Usually, no backtracking is allowed in order to avoid formidable time complexity.

The clustering strategy is particularly designed for DAGs with non-zero edge weights. If all edge weights are zero, the CP length of the original DAG gives the shortest completion time. The clustering process is so designed that when two clusters are merged and the weights of the edges across the two clusters are zeroed, the new CP length of the resulting DAG becomes shorter than the one before the merging. An optimal clustering results in a number of clusters such that the CP length of the clustered DAG cannot be further reduced. At this point, the completion time is minimized. In order to facilitate the subsequent cluster mapping step, the secondary goal of the UNC scheduling algorithms is to minimize the number of clusters. In the following, we discuss five UNC scheduling algorithms. In the discussion, we will use the term *cluster* and *processor* interchangeably since in the UNC scheduling algorithms, merging a single node cluster to another cluster is analogous to scheduling a node to a processor.

EZ Algorithm. The EZ (Edge-zeroing) algorithm [20] selects clusters for merging based on edge weights. At each step, the algorithm zeros the edge with the largest weight.

LC Algorithm. The LC (Linear Clustering) algorithm [13] iteratively merges nodes to form a single cluster based on the CP. The merged nodes are removed and the merging process repeats.

DSC Algorithm. The DSC (Dominant Sequence Clustering) algorithm [24] is designed based on the *Dominant Sequence* (DS) of a graph. The DS is the CP of the partially scheduled DAG. A distinctive feature of the algorithm is that in order to lower the time complexity, the *t-level* of a node is computed incrementally and the *b-level* does not change until the node is scheduled.

MD Algorithm. The MD (Mobility Directed) algorithm [23] selects a node n_i for scheduling based on an attribute called the *relative mobility*, which is defined as:

If a node is on the current CP of the partially scheduled

$$\frac{\text{Cur\_CP\_Length} - (\text{b-level}(n_i) + \text{t-level}(n_i))}{w(n_i)}$$

DAG, the sum of its *b-level* and *t-level* is equal to the current CP length. The MD algorithm scans from the earliest idle time slot on each cluster and schedules the node into the *first* idle time slot that is large enough for the node.

DCP Algorithm. The DCP (Dynamic Critical Path) algorithm [15] is designed based on the value of *mobility*, defined as: (Cur_CP_Length – (b-level (n_i) + t-level (n_i))) . The DCP algorithm uses a *lookahead* strategy to find a better cluster for a given node. In addition to computing the value of $T_S(n_i)$ on a cluster, the DCP algorithm also computes the value of $T_S(n_c)$ on the same cluster. Here, n_c is the child of n_i that has the largest communication and is called the *critical child* of n_i. The DCP algorithm schedules n_i to the cluster that gives the minimum value of the sum of these two attributes.

5 TDB Scheduling Algorithms

The TDB (Task Duplication Based) scheduling algorithms described below assume the availability of an unbounded number of processors. The principal rationale behind the TDB scheduling algorithms is to reduce the communication overhead by redundantly allocating some tasks to multiple processors. In duplication-based scheduling, different strategies can be employed to select ancestor nodes for duplication. Some of the algorithms duplicate only the direct predecessors whereas some other algorithms try to duplicate all possible ancestors. There is a trade-off between performance and time complexity of the algorithm. In the following, we describe six TDB scheduling algorithms.

PY Algorithm. The PY algorithm (named after Papadimitriou and Yannakakis) [19] uses an attribute to approximate the absolute achievable lower bound of the start time of a node. It is shown [19] that the schedule length generated is within a factor of 2 from the optimal.

LWB Algorithm. We call the algorithm [8] the LWB (Lower Bound) algorithm based on its main procedure: it first determines the lower bound start time for each node, and then identifies a set of critical edges in the DAG. The paths containing the critical edges are scheduled to the same processor. It is shown in [8], the LWB algorithm can generate optimal schedules for DAGs in which node weights are strictly larger than any edge weight.

DSH Algorithm. The DSH (Duplication Scheduling Heuristic) algorithm [14] considers each node in a descending order of their priorities. The DSH algorithm first determines the start time of the node on the processor *without* duplication of any ancestor. Then, it considers the duplication in the idle time period from the finish time of the last scheduled node on the processor and the start time of the node currently under consideration.

BTDH Algorithm. The BTDH (Bottom-Up Top-Down Duplication Heuristic) algorithm [6] is essentially an extension of the DSH algorithm described above. The major improvement of the BTDH algorithm over the DSH algorithm is that the algorithm keeps on duplicating ancestors of a node even if the duplication time slot is totally used up in the hope that the start time will eventually be minimized.

LCTD Algorithm. The LCTD algorithm [5] first constructs linear clusters and then identifies the edges among clusters that determines the completion time. It tries to duplicate the parents corresponding to these edges to the reduce the start times of some nodes in the clusters.

CPFD Algorithm. The CPFD (Critical Path Fast Duplication) algorithm [2] is based on partitioning the DAG into three categories: critical path nodes (CPN), in-branch nodes (IBN) and out-branch nodes (OBN). An IBN is a node from which there is a path reaching a CPN. An OBN is a node which is neither a CPN nor an IBN. The main strength of the CPFD algorithm is that it tries to start each CPN as early as possible on a processor by recursively duplicating the IBNs (and also other CPNs) reaching it.

6 APN Scheduling Algorithms

The algorithms in this class take into account specific architectural features such as the number of processors as well as their interconnection topology. These algorithms can schedule tasks on the processors and messages on the network communication links. Scheduling of messages may be dependent on the routing strategy used by the underlying network. The mapping, including the temporal dependencies, is therefore implicit — without going through a separate clustering phase. There are not many reported algorithms that belong to this class. In the following, we discuss four such algorithms.

MH Algorithm. The MH (Mapping Heuristic) algorithm [9] first assigns priorities by computing the *static b-levels* of all nodes. A ready node list is then initialized to contain all entry nodes ordered in decreasing priorities. Each node is scheduled to a processor that gives the smallest start time.

DLS Algorithm. The DLS (Dynamic Level Scheduling) algorithm [22] described earlier can also be used as an APN scheduling algorithm. To use it as a APN scheduling algorithm, it requires the message routing method to be supplied by the user.

BU Algorithm. The BU (Bottom-Up) algorithm [18] first finds out the CP of the DAG and then assigns all the nodes on the CP to the same processor at once. Afterwards, the algorithm assigns the remaining nodes in a reversed topological order to the processors. The node assignment is guided by a load-balancing processor selection heuristic which attempts to balance the load across all given processors.

BSA Algorithm. The BSA (Bubble Scheduling and Allocation) algorithm [16] constructs a schedule incrementally by first injecting all the nodes to the *pivot processor*, defined as the processor with the highest degree. Then, the algorithm tries to improve the start time

of each node (hence "bubbling" up nodes) by migrating it to one of the adjacent processor of the *pivot processor* if the migration can improve the start time of the node. Essentially, after a node is migrated from *pivot processor* to another processor, not only the node itself is "bubbled up" but its successors as well. After all nodes on the *pivot processor* are considered, select the next processor in the processor list to be the new *pivot processor*. The process is repeated by changing the *pivot processor* in a breadth first order.

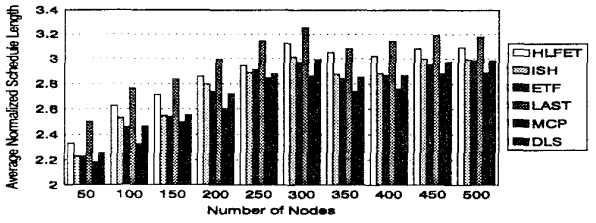
7 Performance Results and Comparison

In this section, we present the performance results and comparisons of the scheduling algorithms of all four classes described above. The algorithms were implemented on a SUN SPARC IPX workstation. The experimental results used a set of 250 random task graphs. Our main rationale for selecting random graphs as a test suite is that they contain as their subset a variety of graph structures. This avoids any bias that an algorithm may have towards a particular graph structure. Furthermore, random graphs have indeed been used extensively in previous studies on scheduling. For generating the complete set of 250 graphs, we varied three parameters: size, *communication-to-computation ratio* (CCR) and *parallelism*. The size of the graph was varied from 50 to 500 nodes with increments of 50. The weight of each node was randomly selected from a uniform distribution with mean equal to the specified average computation cost. The weight of each edge was also randomly selected from a uniform distribution with mean equal to the product of the average computation cost and the CCR. Five different values of CCR were selected: 0.1, 0.5, 1.0, 2.0 and 10.0. The parallelism parameter determined the average number of children nodes for each node. Five different values of parallelism were chosen: 1, 2, 3, 4 and 5. The algorithms were compared within their own class, although some comparison of UNC and BNP algorithms was also carried out. The comparisons were made using the following six measures.

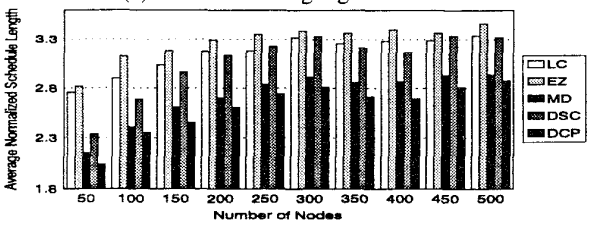
- **Normalized Schedule Length (NSL):** Schedule length is the prime performance measure of a scheduling algorithm. The NSL of an algorithm is obtained by dividing the schedule length produced by the algorithm to the lower bound (defined as the sum of weights of the nodes on the original critical-path). It should be noted that the lower bound may not always be possible to achieve, and the optimal schedule length may be larger than this bound.
- **Pair-Wise and Global Comparisons:** In the pair-wise comparison, we measured the number of times an algorithm produced better, worse or equal schedule length compared to each other algorithm within the same class. In the global comparison, an algorithm was collectively compared with all other algorithms in the same class.
- **Best Solutions:** For each of the 250 graphs, we simply counted the number of times an algorithm produced the shortest schedule length compared to other algorithms.

7.1 Comparing the NSLs

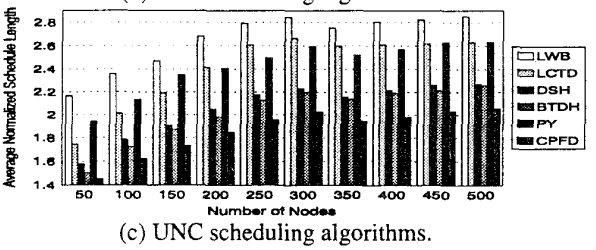
The normalized schedule lengths (NSL) for all of the algorithms are given in Figure 1. Each bar in this figure is



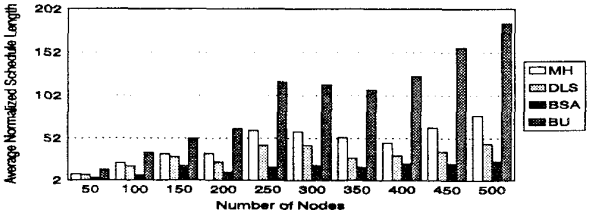
(a) BNP scheduling algorithms.



(b) TDB scheduling algorithms.



(c) UNC scheduling algorithms.



(d) APN scheduling algorithms.

Figure 1: The average normalized schedule lengths produced by various scheduling algorithms.

the average of 25 tests cases with various values of CCR and parallelism; the results showing the impact of CCR — and hence granularity — and parallelism are not included here due to space limitations. From Figure 1, we notice that behavior of these algorithms was consistent in terms of their relative performance for various number of nodes in the graph. Out of the BNP scheduling algorithms, the performance of the MCP algorithm was the best among all the algorithms. The LAST algorithm was outperformed by all other algorithms.

For the UNC scheduling algorithms, we can observe that the DCP and MD algorithms performed significantly

better as compared to the rest of the algorithms. The values of NSL for the DSC and LC algorithms were closed. The EZ algorithm was outperformed by all other algorithms. The relatively inferior performance of the EZ algorithm indicates that clustering for the minimization of the communication alone is not enough for reducing the schedule length. This comparison also indicates that the critical-path algorithms are superior as compared to other algorithms.

For the TDB scheduling algorithms, we can observe large variations in the performance of these algorithms. For example, CPFD was significantly better than the other algorithms. The performance of DSH and BTDH was close but much better than PY, LCTD and LWB. The NSLs produced by LWB were almost 50% larger than those of CPFD. These results also indicate that although the PY algorithm *guarantees* a schedule length within a factor of 2 from the optimal, much shorter schedule lengths are possible.

Although the BNP algorithms are designed for limited number of processors (they take this number as a parameter), we ran each algorithm with a very large number of processors such that the number of processors became virtually unlimited. From this experiment, we noted the average number of processors used by these algorithms for each graph size (the numbers of processors used are omitted due to space limitations). In the next experiment, we reduced the number processors to 50% of that average. These results are shown in Figure 2. Here, no significant differences in the NSLs as well as the relative performance of these algorithms were observed. One possible reason for this behavior is that the schedule length is dominated by the scheduling of CP nodes. In the case of a very large number of processors, the non-CP nodes are spread across many processors, while in the case of a fewer number of processors, these nodes are packed together without making much impact on the overall schedule length.

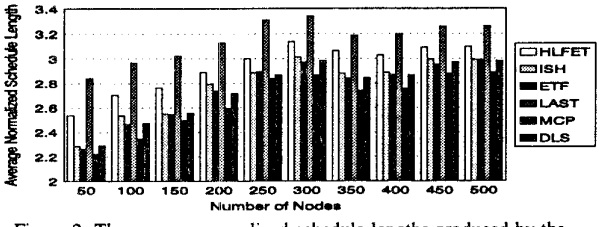


Figure 2: The average normalized schedule lengths produced by the BNP scheduling algorithms for task graphs of various sizes given only 50% of the average number of processors.

For the APN scheduling algorithms, the target architectures included an 8-node ring, an 8-node hypercube, a 4×2 mesh, and an 8-node clique. The average values of these NSLs across all topologies are also depicted in Figure 1. One reason for the much larger NSLs in these cases is that the numbers of processors used were much smaller. We deliberately used very small number of processors to make the experiments more realistic. For example, a 500-node task graph is scheduled to 8 processors[1].

These results suggest that there can be substantial difference in the performance of these algorithms. For example, significant differences can be noticed between the NSLs of BSA and BU. The performance of DLS was relatively stable with respect to the graph size while MH yielded fairly long schedule lengths for large graphs. As can be noticed, the BSA algorithm performed admirably well for large graphs. One of the main reasons for the better performance of BSA is an efficient scheduling of communication messages that can have a drastic impact on the overall schedule length. In terms of the impact of the topology, one can notice that all algorithms performed better on the networks with more communication links.

7.2 Pair-Wise Comparison

Next, we present a pair-wise and a global comparison among the algorithms by observing the number of times each algorithm performed better, worse or the same compared to every other algorithm in 250 test cases. This comparison for the BNP scheduling algorithms is given in a graphical form shown in Figure 3. Here, each box compares two algorithms — the algorithm on the left side and the algorithm on the top. Each box contains three numbers preceded by '>', '<' and '=' signs which indicate the number of times the algorithm on the left performed better, worse, and the same, respectively, compared to the algorithm shown on the top. For example, the DLS algorithm performed better than the MCP algorithm in 66 cases, worse in 162 cases and the same in 22 cases. For the global comparison, an additional box ("ALL") for each algorithm compares that algorithm with all other algorithms combined. Based on these results, we rank these BNP algorithms in the following order: MCP, ISH, DLS, HLFET, ETF, and LAST. This ranking essentially indicates the quality of scheduling based on how often an algorithm performs better than the others. Note, however, that a ranking of these algorithms based on NSLs shown in Figure 1 is different: MCP, DLS, ETF, ISH, HLFET, and LAST. This ranking indicates the quality of scheduling based on the average performance of the algorithm. An algorithm which outperforms other algorithms more frequently but has a lower rank based on the average NSL indicates that it produces long schedule lengths in some cases.

The pair-wise and global comparison of UNC scheduling algorithms is depicted in Figure 4. These results clearly indicate that the DCP algorithm is better than all other algorithms. Both DCP and MD outperformed EZ and LC by a large margin while DSC was marginally better than LC. Based on these results, we rank these UNC algorithms in the following order: DCP, MD, DSC, LC, and EZ. Interestingly, this ranking does not change using the NSLs shown in Figure 1.

1. The number of processors used by a typical UNC algorithm is very large — the LC algorithm, for instance, uses more than 100 processors for a 500-node task graph

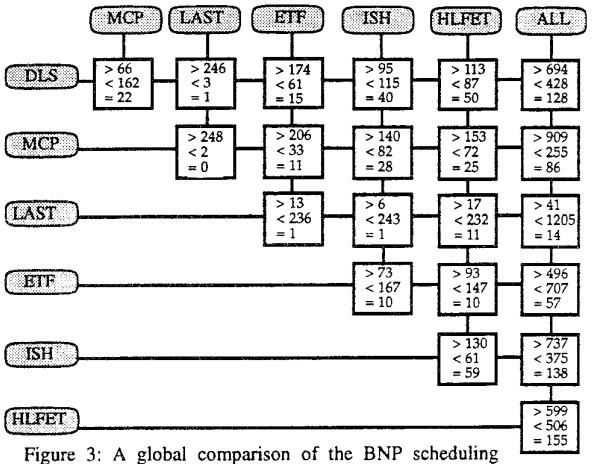


Figure 3: A global comparison of the BNP scheduling algorithms in terms of better, worse and equal performance.

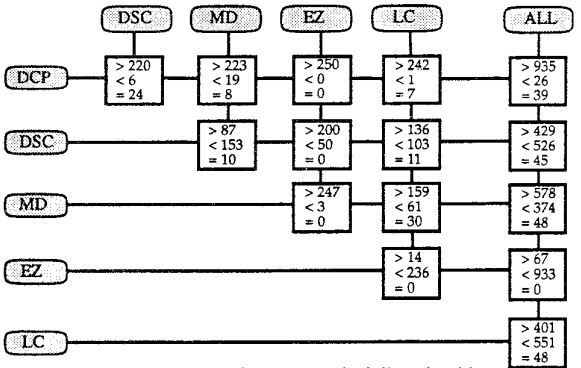


Figure 4: A comparison of the UNC scheduling algorithms in terms of better, worse and equal performance.

In the pair-wise comparison for the TDB scheduling algorithms, shown in Figure 5, we notice that CPFD was better than all other algorithms by a large margin — it was outperformed in only 5 cases. The unexpected result was the comparison of the LWB algorithm compared to other algorithms. Collectively, LWB was second only to the CPFD algorithm but, as shown above in Figure 1, its average performance was the worst. This is because it generates optimal solutions in many cases when CCR is small but becomes inefficient when CCR is larger than 1. Based on the results of Figure 1, these algorithms can be ranked in the order: CPFD, LWB, BTDH, DSH, LCTD and PY. Based on the results of Figure 5, we make the following ranking: CPFD, BTDH, DSH, PY, LCTD and LWB.

In the pair-wise comparison of the APN scheduling algorithms shown in Figure 6, BSA outperformed the other three algorithms in a large number of cases while DLS performed better than MH. The BU algorithm was outperformed by all other algorithms. In terms of performance, these algorithms can be ranked in the order: BSA, DLS, MH, and BU.

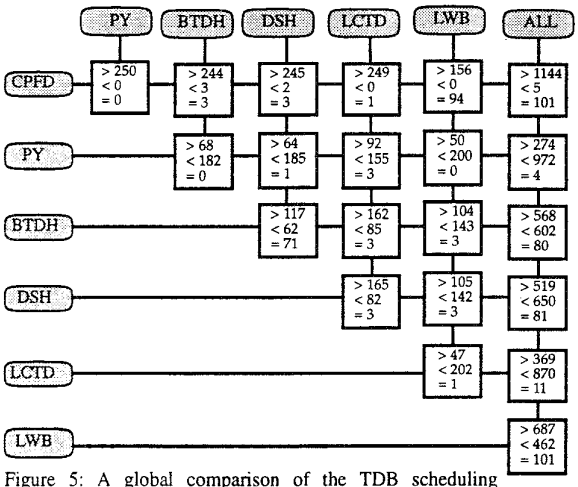


Figure 5: A global comparison of the TDB scheduling algorithms in terms of better, worse and equal performance.

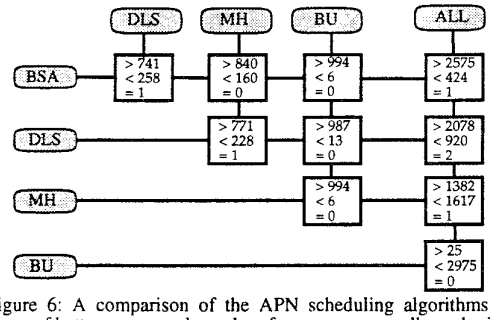


Figure 6: A comparison of the APN scheduling algorithms in terms of better, worse and equal performance across all topologies.

7.3 Best Performance

Table 1 shows the number of times each algorithm yielded the best solution out of 250 test cases (for the ANP scheduling algorithms, there were 1000 test cases). For the BNP scheduling algorithms, the MCP algorithm generated the best solution for 129 times — which is more than 50% of the number of test cases. In the category of the UNC scheduling algorithms, the DCP algorithm generated the best solution in about 90% of the cases. Similarly, in the category of TDB algorithms, the CPFD algorithm generated the best solution in almost all cases while, in the category of the ANP algorithms, the BSA algorithm generated the best solution in about 60% of the cases. The LAST, EZ, PY, and BU algorithms did not yield the best solution in any single case.

BNP Alg.		UNC Alg.		TDB Alg.		APN Alg.	
MCP	129	DCP	225	CPFD	246	BSA	599
ISH	101	DSC	30	LWB	94	DLS	244
HLFET	79	MD	15	DSH	5	MH	59
DLS	64	LC	7	BTDH	5	BU	0
ETF	18	EZ	0	LCTD	1		
LAST	0			PY	0		

Table 1: The number of times an algorithm produced the shortest schedule length.

8 Conclusions and Future Work

Our study has revealed several important findings. For both the BNP and UNC classes, algorithms emphasizing the accurate scheduling of nodes on the critical-path are in general better than the other algorithms. Dynamic critical-path is better than static critical-path, as demonstrated by both the DCP and DSC algorithms. Insertion is better than non-insertion—for example, a simple algorithm such as ISH employing insertion can yield dramatic performance. Dynamic priority is in general better than static priority, although it can cause substantial complexity gain — for example the DLS and ETF algorithms have higher complexities. However, this is not always true — one exception, for example, is that the MCP algorithm using static priorities performs the best in the BNP class. A BNP algorithm can be used as an UNC algorithm assuming infinite number of processors. However, BNP algorithms are designed for a bounded number of processors. BNP algorithms usually use *b-level*, *t-level*, or combination of both, as the criterion for selecting nodes to schedule. UNC algorithms, on the other hand, usually use mobility as the major criteria. An UNC algorithm can be used for a bounded number of processors if the number of processors is not smaller than the number of clusters generated. Exploitation of other topological properties of the graph such as the concept of critical child used by the DCP algorithm can result in a dramatic improvement in schedule lengths. Low complexity algorithms such as DSC and LC can outperform some of the higher complexity algorithms. The APN algorithms can be fairly complicated because they take into account more parameters. Further research is required in this area. The effects of topology and routing strategy need to be determined.

A number of research prototypes have been designed and implemented, showing good performance on a group of carefully selected examples [9], [17]. The current researches concentrate on further elaboration of various techniques, such as reducing the scheduling complexities, improving computation estimations, and incorporating network topology and communication traffic.

References

[1] T.L. Adam, K. Chandy and J. Dickson, "A Comparison of List Scheduling for Parallel Processing Systems," *Communications of the ACM*, vol. 17, no. 12, pp. 685-690, Dec. 1974.

[2] I. Ahmad and Y.K. Kwok, "A New Approach to Scheduling Parallel Programs Using Task Duplication," Proc. of *Int'l Conf. on Parallel Processing*, vol. II, pp. 47-51, Aug. 1994.

[3] J.J. Hwang, Y.C. Chow, F.D. Anger, and C.Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," *SIAM J. Comp.*, pp. 244-257, Apr. 1989.

[4] J. Baxter and J.H. Patel, "The LAST Algorithm: A Heuristic-Based Static Task Allocation Algorithm," Proc. of *Int'l Conference on Parallel Processing*, vol. II, pp. 217-222, Aug. 1989.

[5] H. Chen, B. Shirazi and J. Marquis, "Performance Evaluation of A Novel Scheduling Method: Linear Clustering with Task Duplication," Proc. of *Int'l Conf. on Parallel and Distributed Systems*, pp. 270-275, Dec. 1993.

[6] Y.C. Chung and S. Ranka, "Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed-Memory Multiprocessors," Proc. of *Supercomputing'92*, pp. 512-521, Nov. 1992.

[7] E.G. Coffman and R.L. Graham, "Optimal Scheduling for Two-Processor Systems," *Acta Informatica*, vol. 1, pp. 200-213, 1972.

[8] J.Y. Colin and P. Chretienne, "C.P.M. Scheduling with Small Computation Delays and Task Duplication," *Operations Research*, pp. 680-684, 1991.

[9] H. El-Rewini and T.G. Lewis, "Scheduling Parallel Programs onto Arbitrary Target Machines," *Journal of Parallel and Distributed Computing*, vol. 9, no. 2, pp. 138-153, Jun. 1990.

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

[11] A. Gerasoulis and T. Yang, "A Comparison of Clustering Heuristics for Scheduling DAG's on multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 16, no. 4, pp. 276-291, Dec. 1992.

[12] A.A. Khan, C.L. McCreary and M.S. Jones, "A Comparison of Multiprocessor Scheduling Heuristics," Proc. of *Int'l Conf. on Parallel Processing*, vol. II, pp. 243-250, Aug. 1994.

[13] S.J. Kim and J.C. Browne, "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures," *Proc. of Int'l Conference on Parallel Processing*, vol. II, pp. 1-8, Aug. 1988.

[14] B. Kruatrachue and T.G. Lewis, "Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems," Technical Report, Oregon State University, Corvallis, OR 97331, 1987.

[15] Y.K. Kwok and I. Ahmad, "A Static Scheduling Algorithm Using Dynamic Critical Path for Assigning Parallel Algorithms Onto Multiprocessors," Proc. of *Int'l Conf. on Parallel Processing*, vol. II, pp. 155-159, Aug. 1994.

[16] Y.K. Kwok and I. Ahmad, "Bubble Scheduling: A Quasi Dynamic Algorithm for Static Allocation of Tasks to Parallel Architectures," to appear in Proc. of *7th IEEE Symposium on Parallel and Distributed Processing*, Oct. 1995.

[17] T.G. Lewis and H. El-Rewini, "Parallax: A Tool for Parallel Program Scheduling," *IEEE Parallel and Distributed Technology*, May 1993, vol. 1, no. 2, pp. 64-76.

[18] N. Mehdiratta and K. Ghose, "A Bottom-Up Approach to Task Scheduling on Distributed Memory Multiprocessor," Proc. of *Int'l Conf. on Parallel Processing*, vol. II, pp. 151-154, Aug. 1994.

[19] C.H. Papadimitriou and M. Yannakakis, "Towards an Architecture-Independent Analysis of Parallel Algorithms," *SIAM J. of Comp.*, vol. 19, no. 2, pp. 322-328, Apr. 1990.

[20] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors*, MIT Press, Cambridge, MA, 1989.

[21] B. Shirazi, M. Wang and G. Pathak, "Analysis and Evaluation of Heuristic Methods for Static Scheduling," *Journal of Parallel and Distributed Computing*, no. 10, pp. 222-232, 1990.

[22] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 75-87, Feb. 1993.

[23] M.Y. Wu and D.D. Gajski, "Hypertool: A Programming Aid for Message-Passing Systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 330-343, Jul. 1990.

[24] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, no. 9, pp. 951-967, Sep. 1994.