A List Scheduling Heuristic for Allocating the Task Graph to Multiprocessors

Janez Brest and Viljem Žumer
University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova 17, 2000 Maribor, Slovenia
E-mail: janez.brest@uni-mb.si, http://marcel.uni-mb.si/janez

Keywords: parallel processing, compiler, static scheduling

Received: July 2, 2002

In this paper we propose a new static scheduling algorithm for allocating the task graph without communication costs to fully connected multiprocessors. A global comparison is carried out for the proposed algorithm and three reported scheduling algorithms. The proposed algorithm outperforms the previous algorithms in terms of the generated schedule length using Standard Task Graph set.

1 Introduction

To efficiently execute a program on a multiprocessor system [11, 19, 20, 8, 18], it is essential to solve a minimum execution time multiprocessor scheduling problem [16, 13, 14, 2, 4, 5], which determines how to assign a set of tasks to processors and in what order those tasks should be executed to obtain the minimum execution time. The tasks can then be scheduled to the processors for execution by using a suitable scheduling algorithm, static in compile-time or dynamic in run-time [7, 9, 3]. The optimal static scheduling, except for a few highly simplified cases, is an NP-complete problem. Thus, heuristic approaches are generally sought to tackle the problem. Traditional static scheduling algorithms attempt to minimize the schedule length through iterative local minimization of the start times of individual tasks. On the other hand for example the Dynamic Level Scheduling (DLS) algorithm dynamically selects tasks during the scheduling process [15]. As optimal scheduling of tasks is a strong NP-hard problem, many heuristic algorithms have been introduced in the literature [6].

In this paper we proposed a low time complexity multiprocessor static scheduling algorithm called MCP/CLR without communication costs, which is based on critical path (CP) algorithm, such as, for example, the MCP [21] algorithm. It generates high quality scheduling solutions.

The remaining paper is organized as follows: In the next section, we present a brief overview of various approaches that have been proposed for the DAG scheduling problem. In Section 3, we present the proposed algorithm, and discuss its design principles. We present the experimental results in Section 4, and conclude the paper with some final remarks in Section 5.

2 The Multiprocessor Scheduling Problem

In static scheduling, a parallel program is presented by a directed acyclic graph (DAG) [19]. In a DAG, $G = (V, E)$, V is a set of v nodes, representing the tasks, and E is a set of e directed edges, representing the communication messages. Edges in a DAG are directed and, thus, capture the precedence constraints among the tasks. The cost of node n_i, denoted as $w(n_i)$, represents the computation cost of the task. The cost of the edge, emerges from the source node n_i and incidents on the destination node n_j, denoted by c_{ij}, represents the communication cost of the message. The source node of an edge is called a parent node, while the destination node is called a child node. A node with no parent is called an entry node and a node with no child is called an exit node. A node can only start execution after it has gathered all of the messages from its parent nodes. The *b-level* of a node is the length (sum of the computation costs only) of the longest path from this node to an exit node. The *t-level* of a node is the length of the longest path from an entry node to this node (excluding the cost of this node).

The objective of scheduling is to minimize the schedule length, which is defined as the maximum finish time of all the nodes, by properly assigning tasks to processors such that the precedence constraints are preserved.

The existing scheduling algorithms are classified into four categories by Ahmad and Kwok [2, 14]:

1. Bounded Number of Processors (BNP) Scheduling: A BNP algorithm schedules a DAG to a limited number of processors directly. The processors are assumed to be fully connected without any regard to link contention and scheduling of messages. The proposed algorithm belongs to this class.

2. Unbounded Number of Clusters (UNC) Scheduling: An UNC algorithm schedules a DAG to an unbounded

number of clusters. The clusters generated by these algorithms may be mapped onto the processors using a separate mapping algorithm. These algorithms assume the processors to be fully connected.

3. Arbitrary Processor Network (APN) Scheduling: An APN algorithm performs scheduling and mapping on an architecture in which the processors are connected via a network topology. An APN algorithm also explicitly schedules communication messages on the network channels, taking care of the link contention factor.

4. Task-Duplication-Based (TDB) Scheduling: A TDB algorithm duplicates tasks in order to reduce the communication overhead. Duplication, however, can be used in any of the other three classes of algorithms.

For our purpose, we will compare the proposed algorithm with three other BNP scheduling algorithms.

In a traditional scheduling algorithm, the scheduling list is statically constructed before node allocation begins, and, more importantly, the sequencing in the list is not modified.

The Earliest Task First (ETF) algorithm [10] uses static node priorities and assumes only a bounded number of processors [16, 17]. The High Level First with Estimated Time (HLFET) algorithm [1] assigns the nodes in a DAG to the processors, level by level.

Similar to the ETF and HLFET algorithms, the Modified Critical Path (MCP) algorithm [21] constructs a list of tasks before the scheduling process starts. The MCP algorithm uses the ALAP (As-Late-As-Possible) start time of a node as the scheduling priority. The MCP algorithm first computes the ALAP times of all the nodes, then constructs a list of nodes in ascending order of ALAP times. Ties are broken by considering the ALAP times of the children of a node. The MCP algorithm then schedules the nodes on the list one by one so that a node is scheduled to a processor that allows the earliest start time using the insertion approach. The MCP algorithm looks for an idle time slot for a given node. The algorithm is briefly described in Figure 1 [21, 16, 14]. The complexity of the MCP algorithm is $O(v^2 \log v)$.

(1) Compute the ALAP time of each node.
(2) For each node, create a list which consists of the ALAP times of the node itself and all its children in descending order.
(3) Sort these lists in ascending lexicographical order. Create a node list according to this order.
Repeat
(4) Schedule the first node in the node list to a processor that allows the earliest execution, using the insertion approach.
(5) Remove the node from the node list.
Until the node list is empty.

Figure 1: The MCP algorithm.

3 The Heuristic Algorithm

In this section we discuss some of the principles used in the design of proposed algorithm. To minimize the final schedule length, we select a node as it is selected in the MCP algorithm. At each step of the scheduling process, the first node is removed from the list of nodes (the list of nodes is sorted in increasing lexicographical order of the latest possible start times) and it is scheduled to a processor. While we are able to identify a selected node, we still need a method to select an appropriate processor for scheduling that node into the most suitable idle time slot. At each step, the algorithm needs to find the most suitable processor which contains the most suitable place in time for a selected node.

The MCP algorithm schedules the selected node to a processor that allows for the earliest start time. The proposed algorithm has another processor selection criteria and they are described as follows.

3.1 The MCP/CLR Algorithm

```
Build_ALAP();
Sort_ALAP();
// v is number of tasks
for (i = 0; i < v; i++)
{
   t_i = EST(ALAP(n_i));
   if a processor j exists where SL_j(i) ≤ t_i
   then
      schedule node n_i to a processor j where
           SL_j(i) − t_i is minimal
   else
      schedule node n_i to a processor that
      allows the earliest execution
}
```

Figure 2: The MCP/CLR algorithm.

The function $Build\_ALAP()$ computes the ALAP time of each node and creates a list, which consists of the ALAP times of the node itself and all its children in descending order. Function $Sort\_ALAP()$ sorts these lists in ascending lexicographical order as in the MCP algorithm.

Assumed that, in the scheduling process there are already scheduled $i-1$ nodes. Next selected node is n_i. $SL_j(i)$ is the schedule length of the step i of the scheduling process on the processor j. The MCP/CLR (*MPC/Close-Left-Right*) algorithm (see Fig. 2) tries to find a processor j for the selected node n_i. It is needed to distinguish two cases of the processor selection step. If a processor exists, say j, which satisfy that $SL_j(i)$ is less or equal to the earliest start time (EST) of the selected node n_i, our

algorithm assigns the selected node n_i to the processor j with the smallest value $SL_j(i) - t_i$. Otherwise it assigns the selected node n_i to a processor that allows the earliest execution (like the MCP algorithm), using non-insertion approach. The complexity of the MCP/CLR algorithm is $O(v^2 \log v)$, too.

3.2 Scheduling Example

In this section, we present an example to demonstrate the operation of the proposed algorithm using the task graph shown in Fig. 3. The task graph was drawn using the Graphlet Tool (*http://www.fmi.uni-passau.de/Graphlet*). The schedules of the algorithms are shown in Fig. 4. The entry and exit node are dummy. The MCP algorithm creates a list of edges and schedules the task graph onto the multiprocessor machine with 2 processors (processing elements) in the order: $n_1, n_2, n_5, n_3, n_8, n_7, n_4, n_{11}, n_{10}, n_9, n_6, n_{12}$. The HLFET and MCP/CLR schedule the nodes in the same order as the MCP algorithm. The ETF algorithm schedules the nodes in the order: $n_1, n_2, n_5, n_3, n_4, n_7, n_8, n_{10}, n_6$, n_9, n_{11}, n_{12}. The order of nodes n_4, n_7, n_8 and the processor selection during the scheduling process, have caused different schedules of the task graph, and therefore also different schedule lengths.

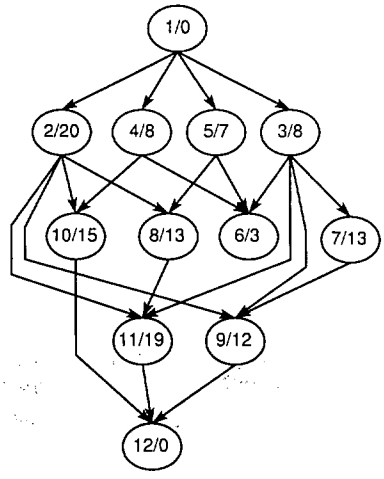

Figure 3: An example of a task graph with 12 nodes.

4 Results

In this section, we present the performance results of the proposed algorithm and compare them with the results of the HLFET, ETF and MCP algorithms.

We have implemented the scheduling algorithms on a SUN workstation using C/C++. They were evaluated by using a Standard Task Graph set: *http://www.kasahara.elec.waseda.ac.jp/schedule/*. The Standard Task Graph set has 900 task graphs with 50 to 2700 tasks.

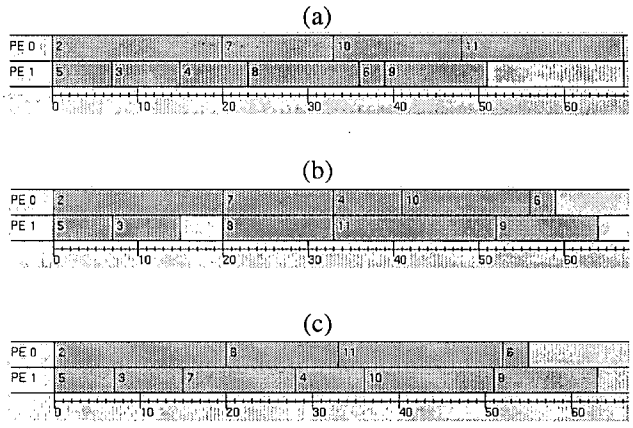

Figure 4: The schedules of the task graph on Fig. 3 generated by: (a) ETF algorithm (schedule length = 67 time units); (b) HLFET and MCP algorithms (schedule length = 64 time units); and (c) MCP/CLR algorithm (schedule length = 63 time units).

Table 3: Number of times the optimal schedule is found, and a global error

Algorithm	Optimal schedule	%	Global error
ETF	33	12.94	5577
HLFET	50	19.61	3189
MCP	57	22.53	1531
MCP/CLR	167	65.49	257

The results obtained in our experiments are shown in Table 1. The second and third columns indicate the name of the task graph instance and number of nodes, respectively. In next four columns results of the schedule length for the all of algorithms are shown, respectively. In the last column the optimal schedule length value is shown. If the optimal schedule is found, the schedule length value is boldface. For some problem instances, the optimal schedule length is not known.

In order to rank all the algorithms in terms of the schedule lengths, we made a global comparison [17]. We observed the number of times each algorithm performed better, worse or the same compared to each of the other algorithms. This comparison is presented in Fig. 5, where some boxes have the left and the right side. Each left side of the box compares two algorithms – the algorithm on the left side and the algorithm on the top. Each left side of the box contains three numbers preceded by ">", "<", and "=" signs which indicate the number of times the algorithm on the left performed better, worse, or the same, respectively, compared to the algorithm shown on the top. Each comparison is based on the total of 300 task graphs. Each right side of the box contains the number of times when one of algorithms, the algorithm on the left side or the algorithm on the top, find the optimal schedule length. Optimal sched-

Table 1: Schedule results of 50 task graph instances

	Graph	#Nodes	ETF	HLFET	MCP	MCP/CLR	Optimum
1	proto000.stg	452	**537**	**537**	**537**	**537**	537
2	proto001.stg	473	1191	1179	1179	**1178**	1178
3	proto002.stg	499	357	363	355	343	341
4	proto003.stg	164	**556**	**556**	**556**	**556**	556
5	proto004.stg	457	267	238	234	222	—
6	proto005.stg	404	758	749	**742**	**742**	742
7	proto006.stg	273	171	154	149	142	—
8	proto007.stg	499	492	**489**	**489**	**489**	489
9	proto008.stg	399	578	582	579	572	571
10	proto009.stg	438	**625**	**625**	**625**	**625**	625
11	proto010.stg	539	351	338	338	**334**	334
12	proto011.stg	759	513	496	494	484	—
13	proto012.stg	939	1804	1795	1795	**1793**	1793
14	proto013.stg	799	698	688	685	682	681
15	proto014.stg	636	523	520	516	509	—
16	proto015.stg	712	513	513	501	**491**	491
17	proto016.stg	641	1016	1026	1022	1006	—
18	proto017.stg	722	487	475	473	463	—
19	proto018.stg	730	704	706	704	701	700
20	proto019.stg	617	683	682	674	668	667
21	proto020.stg	1104	1523	1514	1511	1505	1504
22	proto021.stg	1145	644	632	616	**605**	605
23	proto022.stg	1189	1625	1620	1617	1610	1609
24	proto023.stg	1353	1619	1628	1624	1614	1612
25	proto024.stg	1218	1295	1291	1289	1283	1281
26	proto025.stg	1258	1193	1198	1194	1191	1188
27	proto026.stg	1239	1509	1502	1501	**1500**	1500
28	proto027.stg	1055	2003	2001	2001	2001	2000
29	proto028.stg	1424	1538	1506	1506	**1504**	1504
30	proto029.stg	1341	845	**830**	**830**	**830**	830
31	proto280.stg	1668	2821	2809	2806	**2800**	2800
32	proto281.stg	1622	977	970	921	897	896
33	proto282.stg	1793	3131	3131	3127	**3123**	3123
34	proto283.stg	1591	2479	2429	2428	2425	2422
35	proto284.stg	1703	4499	4458	4447	**4444**	4444
36	proto285.stg	1766	1333	1304	1285	**1268**	1268
37	proto286.stg	1703	945	940	930	**918**	918
38	proto287.stg	1615	1423	1422	1381	1360	1353
39	proto288.stg	1672	1938	1944	1935	1926	1925
40	proto289.stg	1642	1968	1922	1917	**1915**	1915
41	proto290.stg	2133	1453	1450	1438	**1428**	1428
42	proto291.stg	2122	2611	2605	2597	**2591**	2591
43	proto292.stg	2333	8022	8012	8011	**8009**	8009
44	proto293.stg	2089	1516	1477	1474	**1472**	1472
45	proto294.stg	2014	1336	1307	1273	1259	1257
46	proto295.stg	2168	1349	1329	1321	**1318**	1318
47	proto296.stg	2162	2589	2590	2575	**2563**	2563
48	proto297.stg	2136	7708	7710	7710	**7703**	7703
49	proto298.stg	2399	2492	2476	2474	**2471**	2471
50	proto299.stg	2205	1171	1153	1147	1142	1141

Table 2: Schedule length with respect to the optimal solution

	Quality of the solution (Error)	ETF	HLFET	MCP	MCP/CLR
1	0% (optimum)	33	50	57	167
2	< 5%	178	182	195	88
3	5% - 10%	30	21	3	0
4	10% - 15%	11	2	0	0
5	15% - 20%	3	0	0	0
6	Optimum not known	45	45	45	45
	Total	300	300	300	300

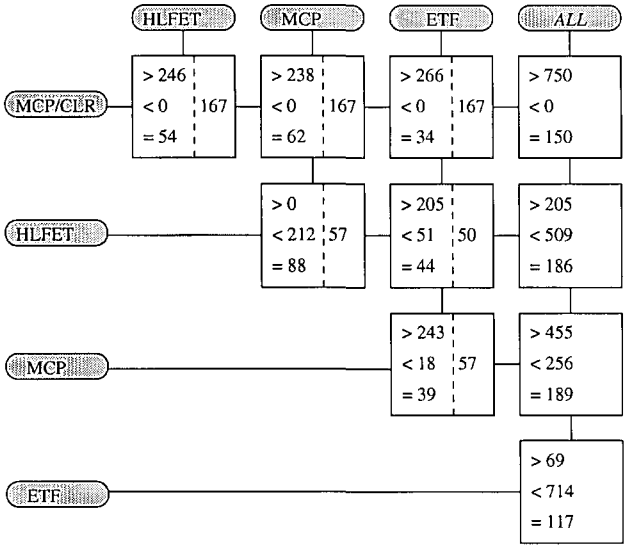

Figure 5: A global comparison of four algorithms in terms of better, worse, and equal performance.

ule lengths are known for 255 of all 300 task graphs. They were computed on a parallel machine using the ISH algorithm [13, 12]. For example, the MCP/CLR algorithm performed better than the MCP algorithm in 238 cases, never performed worse, and performed the same in 62 cases. The MCP/CLR algorithm or the MCP algorithm or both of them found optimal solution of the schedule length in 167 cases. An additional box for each algorithm compares that algorithm with all other algorithms combined.

The experimental results of the quality of the schedule length are summarized in Table 2. For example, the MCP/CLR algorithm found the optimal schedule length in 167 cases and, additionally, the solution within 5% in 88 cases.

Table 3 shows number of times the algorithm has found the optimal schedule, and global error which is defined as difference between the sum of all the optimal schedule values and the sum of all the schedule values generated by the algorithm.

It can be noticed that the proposed MCP/CLR algorithm outperformed three other well known algorithms. Based on these experiments, all the algorithms can be sorted in the following order: MCP/CLR, MCP, HLFET and ETF. The same order of the MCP and ETF algorithms can be found in [17], where communications are also assumed among the tasks.

5 Conclusion

This paper presents the static task scheduling algorithm which can schedule directed acyclic graphs (DAGs) with a complexity of $O(v^2 \log v)$, where v is the number of tasks in the DAG. The algorithm schedules the tasks and it is suitable for the graphs with arbitrary computation and without communication costs, and is applicable to the system with homogeneous fully connected processors. The performances of the proposed algorithm has been observed by comparing it with other existing bounded number of processor (BNP) scheduling algorithms in terms of the schedule length.

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(12):685–690, December 1974.

[2] I. Ahmad and Y.-K. Kwok. On parallelizing the multiprocessor scheduling problem. *IEEETPDS: IEEE Transactions on Parallel and Distributed Systems*, 10, 1999.

[3] J. Brest, V. Žumer, and M. Ojsteršek. Dynamic scheduling on a network heterogeneous computer system. *LNCS 1557*, pages 584–585, 1999.

[4] J. Brest and V. Žumer. A Performance Evaluation of List Scheduling Heuristics for Task Graphs without Communication Costs. Proceedings of the International Workshop on Parallel Processing (ICPP'00), pages 421–428, 2000.

[5] J. Brest, J. Jejčič, A. Vreže and V. Žumer. An Approximation Algorithm for the Static Task Schedul-

ing on Multiprocessors. VECPAR'2000 4th International Meeting on Vector and Parallel Processing, Vol. 1, pages 46–56, 2000.

[6] D. Darbha and D. P. Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEET-PDS: IEEE Transactions on Parallel and Distributed Systems*, 9, 1998.

[7] M. M. Eshagian, editor. *Heterogeneous Computing*. Artech House, Inc., Norwood, MA 02062, ISBN 0-89006-552-7, 1996.

[8] I. Foster. *Designing and Building Parallel Programs*. Addison-Wesley, ISBN 0-201-57594-9, 1995.

[9] E. Haddan. Load Balancing and Scheduling in Network Heterogeneous Computing. In M. M. Eshagian, editor, *Heterogeneous Computing*, pages 224–276, Norwood, MA 02062, ISBN 0-89006-552-7, 1996. Artech House, Inc.

[10] J. J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing*, 18(2):244–257, April 1989.

[11] K. Hwang and Z. Xu. *Advanced Computer Architecture: Technology, Architecture, Programming*. McGraw-Hill, New York, 1998.

[12] H. Kasahara, H. Honda, and S. Narita. Parallel processing of near fine grain tasks using static scheduling on OSCAR (optimally scheduled advanced multiprocessor). In IEEE, editor, *Proceedings, Supercomputing '90: November 12–16*, pages 856–864. IEEE Computer Society Press, 1990.

[13] H. Kasahara and S. Narita. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. on Computers*, 33(11):1023, November 1984.

[14] Y.-K. Kwok. *High-Performace Algorithms for Compile-Time Scheduling of Parallel Processors*. PhD thesis, The Hong Kong University of Science and Technology, 1997.

[15] Y.-K. Kwok and I. Ahmad. FASTEST: A practical low-complexity algorithm for compile-time assignment of parallel programs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 10(2):147–159, February 1999.

[16] Y.-K. Kwok and I. Ahmad. Parallel program scheduling technique. In Buyya Raykumar, editor, *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall - PTR, NJ, USA, 1999.

[17] Y.-K. Kwok and I. Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors:. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521, May 1996.

[18] M. Quinn. *Parallel Computing: Theory and Practice*. McGraw-Hill, 1994.

[19] B. Raykumar, editor. *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall - PTR, NJ, USA, 1999.

[20] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1998.

[21] M.-Y. Wu and D. D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343, July 1990.