Computing and Informatics, Vol. 24, 2005, 341–350

A MACHINE ASSIGNMENT MECHANISM FOR COMPILE-TIME LIST-SCHEDULING HEURISTICS

Tarek Hagras, Jan Janeček

Department of Computer Science and Engineering
Czech Technical University in Prague
Prague, Czech Republic
e-mail: `tarek@felk.cvut.cz, janecek@cs.felk.cvut.cz`

Manuscript received 18 November 2003; revised 3 May 2005
Communicated by Ladislav Hluchý

Abstract. Finding an optimal solution for a scheduling problem is NP-complete. Therefore, it is necessary to use heuristics to find a good schedule rather than evaluating all possible schedules. List scheduling is generally accepted as an attractive approach, since it combines low complexity with good results. List scheduling consists of two phases: a task prioritization phase where a certain priority is computed and assigned to each task, and a machine assignment phase where each task (in order of its priority) is assigned a machine that minimizes a suitable cost function. This paper presents a machine assignment mechanism that can be used with any list-scheduling algorithm. The mechanism is called Reverse Duplicator Mechanism and outperforms the current mechanisms.

Keywords: Compile-time scheduling, machine assignment mechanisms, list-scheduling, homogenous computing systems.

1 INTRODUCTION

Efficient schedule of parallel programs is one of the most essential and difficult issues to achieve high performance in both homogeneous and heterogeneous computing environments [1]. The main objective of *scheduling mechanism* is to map tasks to machines and order their executions so that precedence requirements are satisfied and minimum overall completion time (makespan) is achieved. When the structure of the parallel program in terms of its task execution times, task dependencies and size of communicated data is known a priori, the application is represented by the static model and scheduling can be accomplished statically at compile-time. In the general form of a static task scheduling, the application is represented by the *directed acyclic graph* (DAG) [2, 3], in which nodes represent application tasks and edges represent inter-task data dependencies. Each node is labeled by the computation cost (expected computation time) of the task and each edge is labeled by the communication cost (expected communication time).

Finding an optimal solution for the scheduling problem is NP-complete [4, 5]. Therefore, it is necessary to have heuristics to find a good schedule rather than evaluate all possible schedules. Most scheduling heuristics algorithms are based on list scheduling [2, 3, 5–8]. List scheduling consists of two phases: *a task prioritizing phase* where a task list (L) that contains all tasks is constructed and a priority is computed and assigned to each task in L, and *a machine assignment phase* where each task (in order of its priority) is assigned a machine that minimizes a suitable cost function. The scheduling heuristic is called *static* if the machine assignment phase starts after finishing the task prioritizing phase [2, 5, 9] and it is called *dynamic* if the two phases are interleaved [10, 11].

This paper presents a machine assignment mechanism called Reverse Duplicator (RD) that outperforms the current mechanisms in low complexity. The mechanism can be used for the machine assignment phase with any list-scheduling algorithm. It can be also used for both homogeneous and heterogenous environments. The remainder of this paper is organized as follows. The next section describes the current machine assignment mechanisms. Section 3 presents the suggested mechanism. In Section 4, the performance comparison of the examined mechanisms is presented. Section 5 provides the conclusion.

2 CURRENT MACHINE ASSIGNMENT MECHANISMS

This section presents the current machine assignment mechanisms used in static list-scheduling algorithms. These mechanisms are: *non-insertion* [3, 4, 8] and *insertion based* [6, 9] mechanisms.

2.1 Non-Insertion Based Mechanism

The non-insertion (NI) mechanism tries to assign each task $v_i \in L$ a machine $p_m \in P$, that allows the task to be executed as early as possible. The *Task Start Time* on a machine TST is defined as

$$\mathrm{TST}(v_i, p_q) = \max_{v_n \in prnt(v_i)} \{\mathrm{RT}(p_q), \mathrm{FT}(v_n) + k \cdot c_{n,i}\} ,$$

where:

$prnt(v_i)$ is the set of immediate predecessors of v_i,

$\mathrm{RT}(p_q)$ is the time when p_q is available,

$\mathrm{FT}(v_n)$	is the completion time of the parent node v_n,
$c_{n,i}$	is the communication cost between v_n and v_i, and
$k = 1$	if the machine assigned to parent task v_n is not p_q, and $k = 0$ otherwise.

2.2 Insertion Based Mechanism

The insertion based (IB) mechanism considers a possible insertion of each task $v_i \in L$ in the earliest idle time slot between two already scheduled tasks on a given machine. For each task v_i, the absolute start time on a machine p_q is computed as follows

$$\mathrm{AST}(v_i, p_q) = \max_{v_n \in prnt(v_i)} \{\mathrm{FT}(v_n) + k \cdot c_{n,i}\},$$

where:

$prnt(v_i), \mathrm{FT}(v_n), c_{n,i}$ and k are the same as $\mathrm{TST}(v_i, p_q)$ in non-insertion based mechanism.

A task v_i can be inserted into the machine p_q, which contains the node sequence $\{v_{q_1}, v_{q_2}, ..., v_{q_x}\}$, after task v_{q_y} if

$$\mathrm{TST}(v_{q_{y+1}}, p_q) - \max\{\mathrm{AST}(v_i, p_q), \mathrm{TFT}(v_{q_y}, p_q)\} \geq w_i$$

where:

$\mathrm{TST}(v_{q_y}, p_q)$	is the task v_y starting time on p_q,
$\mathrm{TFT}(v_{q_y}, p_q)$	is the task v_y finishing time on p_q,
w_i	is the task v_i computation cost, and
$\mathrm{TST}(v_{q_{x+1}}, p_q)$	is equal to ∞.

The machine p_m that minimizes v_i start time is selected.

3 PROPOSED MECHANISM

Basically, task-duplication algorithms try to duplicate the parent-tree or some selected parents of a current selected task to an unbounded number of machines. The goal of this duplication is to minimize or optimize the start time of the duplicated parents to be able to select the machine that minimizes the start time of the selected task. This big number of duplications increases the algorithm complexity, while optimality is still far from being achieved. Considering an unbounded number of machines as a target computation environment is still unpractical.

The main idea of the proposed machine assignment mechanism (Figure 1) is to:

1. select the machine that minimizes the start time of the current selected task,
2. examine the idle time left by the selected task on the selected machine for duplicating one selected parent,

3. confirm this duplication if it will reduce the start time of the current selected task.

In contrast to the basic idea of general duplication algorithms, the proposed mechanism selects the machine and then checks for duplication. Instead of examining one task at each step, the mechanism examines one task and one parent, which does not increase the complexity of the classical non-insertion based machine assignment mechanism.

The following five definitions should be given to clarify the proposed mechanism:

Definition 1. The *Task Start Time on a machine* TST is defined as follows:

$$\mathrm{TST}(v_i, p_q) = \max_{v_n \in prnt(v_i)} \{\mathrm{RT}(p_q), \mathrm{FT}(v_n) + k \cdot c_{n,i}\},$$

where:

$prnt(v_i)$	is the set of immediate parents of v_i,
$\mathrm{RT}(p_q)$	is the time when p_q is available,
$\mathrm{FT}(v_n)$	is the completion time of parent v_n,
$c_{n,i}$	is the communication cost between v_n and v_i, and
k	is equal to 1, if the machine assigned to parent task v_n is not p_q and is equal to 0 otherwise.

Definition 2. The Duplication Time Slot:

$$\mathrm{DTS}(v_i, p_m) = \mathrm{TST}(v_i, p_m) - \mathrm{RT}(p_m).$$

Definition 3. *The Critical Parent* is the parent v_{CP} (scheduled on p_q) of v_i (tentatively scheduled on p_m) whose data arrival time to v_i is the latest.

Definition 4. $DAT(v_{CP2}, p_m)$ is the data arrival time of the *second* critical parent v_{CP2} on p_m.

Definition 5. The Duplication Condition is

$$\mathrm{DTS}(v_i, p_m) > w_{CP}$$

and

$$\mathrm{TST}(v_{CP}, p_m) + w_{CP} < \mathrm{TST}(v_i, p_m).$$

If the *duplication condition* is satisfied the mechanism works as follows:

1. duplicate the v_{CP} on p_m at the later of $\mathrm{RT}(p_m)$ and $\mathrm{TST}(v_{CP}, p_m)$,
2. update $\mathrm{RT}(p_m)$,
3. assign v_i to p_m at the later of $\mathrm{RT}(p_m)$ and $DAT(v_{CP2}, p_m)$.

```
while not the end of $L$ do
    dequeue $v_i$ from $L$
    for each machine $p_q$ in the machine set $P$ do
        compute TST$(v_i, p_q)$
    select the machine $p_m$ that minimizes TST of $v_i$
    select $v_{CP}$ and $v_{CP2}$ of $v_i$
    if the duplication condition is satisfied
        if TST$(v_{CP}, p_m) \leq$ RT$(p_m)$
            duplicate $v_{CP}$ on $p_m$ at RT$(p_m)$
            RT$(p_m)$ = RT$(p_m) + w_{CP}$
        else
            duplicate $v_{CP}$ on $p_m$ at TST$(v_{CP}, p_m)$
            RT$(p_m)$ = $TST(v_{CP}, p_m) + w_{CP}$
        if $DAT(v_{CP2}, p_m) >$ RT$(p_m)$
            assign $v_i$ to $p_m$ at $DAT(v_{CP2}, p_m)$
            RT$(p_m)$ = $DAT(v_{CP2}, p_m) + w_i$
        else
            assign $v_i$ to $p_m$ at RT$(p_m)$
            RT$(p_m)$ = RT$(p_m) + w_i$
    else
        assign task $v_i$ to $p_m$ at TST$(v_i, p_m)$
        RT$(p_m)$ = TST$(v_i, p_m) + w_i$
```

Fig. 1. Proposed machine assignment mechanism

4 MECHANISMS COMPLEXITY

Complexity is usually expressed in terms of the number of nodes v, the number of edges e, and the number of machines p. The mechanisms complexity is shown in Table 1.

Algorithm	Complexity
RD	$O(pv^2)$
NI	$O(pv^2)$
IB	$O(pv^3)$

Table 1. Mechanisms complexity

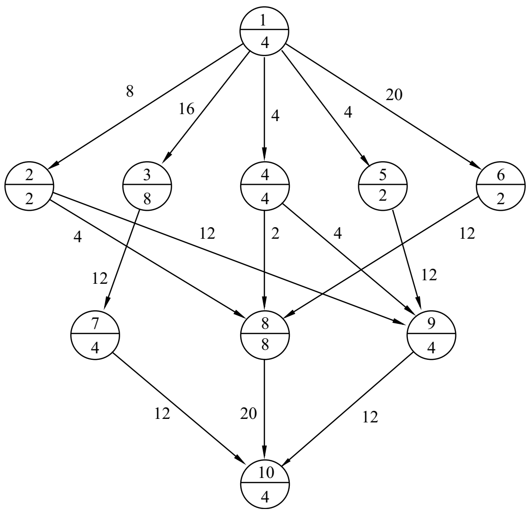


Fig. 2. Application graph

5 EXPERIMENTAL RESULTS AND DISCUSSION

This section presents the performance comparison of the examined mechanisms (non-insertion (NI) and insertion based (IB) mechanisms) in addition to the proposed mechanism (reverse duplicator (RD)). For this purpose, we used a list L generated by the well known list scheduling algorithm, Modified Critical Path (*MCP*) [8]. For the application graph in Figure 2, the list L generated using MCP heuristic is $\{v_1, v_6, v_3, v_2, v_4, v_5, v_8, v_7, v_9, v_{10}\}$ and the scheduling of the examined mechanisms is shown in Figure 3. In Figure 3 b), the gray tasks are the inserted tasks and in Figure 3 c), the gray tasks are the duplicated tasks. A large number of randomly generated task graphs with variant characteristics and the following comparison metrics are used for the comparison.

5.1 Comparison Metrics

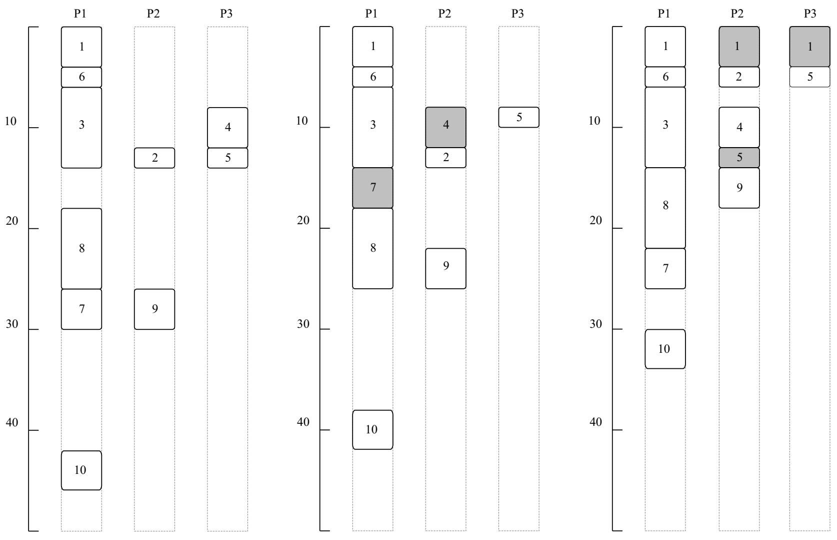
The comparisons of the mechanisms are based on the following metrics.

Makespan. The makespan is defined as the overall completion time and can be specified as follows:

$$\text{makespan} = \text{FT}(v_{exit}),$$

where: $\text{FT}(v_{exit})$ is the finish time of the scheduled exit task.

Scheduling Length Ratio (SLR). The main performance measure is the scheduling length (makespan). Since a large set of task graphs with different properties



a) Non-insertion (Makespan = 46)

b) Insertion based (Makespan = 42)

c) Reverse Duplicator (Makespan = 34)

Fig. 3. Schedules produced by the examined mechanisms

is used, it is necessary to normalize the schedule length to the lower bound, which is called the Schedule Length Ratio (SLR). The SLR is defined as

$$SLR = \frac{\text{makespan}}{\sum_{i \in CT} w_i}.$$

The denominator is the sum of the computation costs of the tasks on a critical path (CP). The average SLR is used in our experiments.

Quality of Schedules. The percentage number of times that a mechanism produced better, worse, and equal quality of schedule compared to every other mechanism is counted in the experiments.

5.2 Random Graph Generator

The random graph generator was implemented to generate application DAGs with various characteristics that depend on several input parameters. The generator requires the following input parameters to build weighted DAGs:

- number of tasks in the graph v,
- graph levels l,

- communication to computation ratio CCR, which is defined as the ratio of the average communication cost to the average computation cost.

In all experiments, graphs with single entry and single exit node were considered. In each experiment, the values of parameters were selected from the following sets:
$v \in \{20, 40, 60, 80, 100, 120\}$,
$0.2\,v \leq l \leq 0.8\,v$,
$CCR \in \{0.5, 1.0, 2.0\}$.

5.3 Performance Results

The performances of the mechanisms were compared with respect to different graph size. The experiments were repeated for each v from the v set given above. For each v, 1 000 graph were generated using random selection for CCR and levels (l) (given above) for each graph. The average SLR for each v is given in Figure 4. For all experiments 16 full connected machines were used. In general the performances of the IB are better than the NI and the RD outperformed them both.

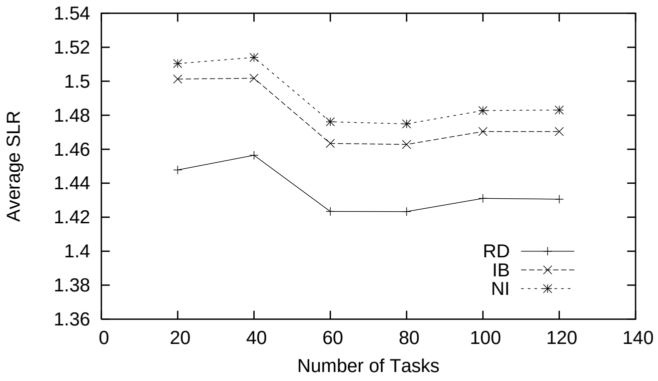


Fig. 4. Average SLR

Finally, the percentage of situations that each mechanism in the experiments produced better (B), equal (E) or worse (W) scheduling length compared to every other mechanism were counted for all generated graphs. Each cell in Table 2 indicates the comparison results of the mechanism at the left with the mechanism at the top.

6 CONCLUSION

In this paper we presented a simple machine assignment mechanism called Reverse Duplicator. The mechanism can be used with list-scheduling heuristics for both

		RD	IB	NI
RD	B		89.62 %	93.62 %
	E		4.72 %	4.57 %
	W		5.67 %	1.82 %
IB	B	5.67 %		31.28 %
	E	4.72 %		64.08 %
	W	89.62 %		4.63 %
NI	B	1.82 %	4.63 %	
	E	4.57 %	64.08 %	
	W	93.62 %	31.28 %	

Table 2. Pairwise comparison of the examined mechanisms

limited and unlimited number of machines. The performance of the mechanism was examined using variant random generated graphs. Three comparison matrices were used to measure its performance. The reverse duplicator mechanism outperformed both the non-insertion and insertion based mechanisms having the same complexity as the non-insertion based mechanism.

REFERENCES

[1] Feitelson, D.—Rudolph, L.—Schwiegelshohm, U.—Sevcik, K.—Wong, P.: Theory and Practice in Parallel Job Scheduling. JSSPP, 1997, pp. 1–34.

[2] Kwok, Y.—Ahmed I.: Benchmarking the Task Graph Scheduling Algorithms. Proc. IPPS/SPDP, 1998.

[3] Liou, J.—Palis, M.: A Comparison of General Approaches to Multiprocessor Scheduling. Proc. Int'l Parallel Processing Symp., 1997, pp. 152–156.

[4] Khan, A.—McCreary, C.—Jones, M.: A Comparison of Multiprocessor Scheduling Heuristics, ICPP, Vol. 2, 1994, pp. 243–250.

[5] Hagras, T.—Janeček, J.: A High Performance, Low Complexity Algorithm for Compile-Time Job Scheduling in Homogeneous Computing Environments. IEEE Proc. Int'l Conf. Parallel Processing Workshops (ICPP03 workshops). October 2003, pp. 149–155.

[6] Kwok, Y.—Ahmed, I.: Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graph to Multiprocessors. IEEE Trans. Parallel and Distributed Systems, Vol. 7, 1996, pp. 506–521.

[7] Zhou, H.: Scheduling DAGs on a Bounded Number of Processors. Int'l Conf., Parallel and Distributed Processing Techniques and Applications, 1996.

[8] Min-You, W.—Gajski, D.: Hypertool: A Programming Aid for Message-Passing Systems. IEEE Trans. Parallel and Distributed Systems, Vol. 1, 1990, No. 3.

[9] Topcuoglu, H.—Hariri, S.—Min-You, W.: Task Scheduling Algorithm for Heterogeneous Processors. Heterogeneous Computing Workshop, 1999, pp. 3–14.

[10] Hwang, J.—Chow, Y.—Anger, E.—Lee, C.: Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. SIAM Journal on Computing, Vol. 18, 1989, No. 2, pp. 244–257.

[11] Sih, G.—Lee, E.: A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. IEEE Trans. In Parallel and Distributed Systems, Vol. 4, 1993, No. 2, pp. 75–87.

Tarek Hagras received his M. Sc. degree in computer engineering from Asyut University, Asyut, Egypt, and his Ph. D. degree in computer engineering and informatics from Czech Technical University in Prague, Czech Republic, in 1998 and 2005, respectively. Currently, he is a lecturer at Higher Institute of Energy, Aswan, Egypt. He is a member of International Society of Computers and their Applications (ISCA), IEEE and its Computer and Communication Societies, and Egyptian Syndicate of Professional Engineers. His research interests include parallel and distributed systems and task scheduling in homogeneous and heterogeneous computing systems.

Jan Janeček is an associate professor in the Department of Computer Science and Engineering at the Czech Technical University in Prague. He received his M. Sc. degree and his Ph. D. degree in technical cybernetics from the Czech Technical University in Prague in 1973 and 1981, respectively. Currently he lectures on local area networks, advanced Technologies of computer networks, distributed systems and applications of embedded systems. His research focuses on distributed computation, middleware technologies, networking and embedded applications. He has led and participated in research teams working on projects dealing with networking technologies (X.25 PAD, ATM switch, VoIP software), software implementation tools (embedded C and Pollux compilers, efficient SOAP parser), distributed quorum algorithms, support for asynchrony in distributed applications, and effectiveness of middleware technologies. He is a member of IEEE and its Computer and Communication Societies, and serves as a vice chairman to the Czech Chapter of the IEEE Computer Society.