On-line Scheduling of Tasks in Multi-DSP Systems *

Bernhard Rinner and Diethard Kaufmann
E-mail: rinner@iti.tu-graz.ac.at
Institute for Technical Informatics
Technical University of Graz, AUSTRIA

Abstract

We present a comparison of three different list scheduling algorithms for on-line scheduling of independent tasks. This comparison is based on runtime measurements of parallel applications taken on a multi-DSP system (TMS320C40) of up to 8 processors.

keywords:
list scheduling,
multi-DSP TMS320C40,
parallel processing

1 Introduction

Many applications in the area of digital signal processing are implemented on multi-processor systems to meet their high performance requirements. In such a multi-processor implementation the overall application is partitioned into a collection of separate cooperating and communicating modules, called tasks [4]. Tasks can execute in sequence or at the same time on two or more separate processing elements. The scheduling of the tasks, i.e., the ordering of their execution, plays a crucial role for the overall performance.

Scheduling is a *NP–complete* problem; it must satisfy the conditions of the tasks (e.g., time constraints and precedence relations) and of the computer architecture (e.g., number of processing elements and communication times) [7]. The objective is to find an optimal mapping with regard to a specific optimality criteria. A common optimality criteria is the sum of the completion time of all tasks.

There are many applications which do not have strong dependencies between the tasks. Especially in the field of digital signal processing, multi-processor implementations exploit the data parallelism of the DSP-algorithms. Examples of such implementations are audio or image processing applications where each task operates independently in its own frequency range or image area, respectively. However, even in those applications some characteristics, like the number of tasks and/or their execution times, are either often not known at compile time or may change during runtime. Therefore, on-line scheduling is necessary.

In this paper we compare three different list-scheduling algorithms for on-line scheduling. In Section 2, the scheduling algorithms are briefly described. Section 3

*This project is partially supported by the Austrian National Science Foundation *Fonds zur Förderung der wissenschaftlichen Forschung* under grant number P10411-MAT.

```
1  procedure LS-I()
2  begin
3    for all n processors i do
4      sum_i ← 0
5      tasklist_i ← ∅
6    endfor
7    for all tasks j do
8      find processor i with min. sum_i+p_ij
9      sum_i ← sum_i+p_ij
10     add {j} to tasklist_i
11   endfor
12 end
```

Figure 1: Pseudo code for the simple list scheduling algorithm (LS-I).

presents the experimental setup and the results. Section 4 concludes the paper.

2 List Scheduling Algorithms

In on-line scheduling, the execution time of the scheduling algorithm is included in the overall execution time. Heuristics are often applied for on-line scheduling to keep the execution time of the scheduling algorithm small. However, the heuristics differ in their qualities, i.e., the deviation in completion time between the generated and the optimal schedule. Hence, there is a tradeoff between the scheduler's execution time and it's quality, both influencing the overall execution time.

List scheduling [2] is an excellent scheduling heuristic for independent tasks on multi-processor implementations due to the following reasons.

- This scheduling heuristic exploits the execution times of the tasks or estimations of the execution times to improve the schedule.
- List scheduling guarantees that the deviation of the completion time is bounded by the factors n, $\frac{5}{2}\sqrt{n}$ or $(1+\sqrt{2})\sqrt{n}$ for the three scheduling algorithms where n denotes the number of processors.
- List scheduling has a short execution time; it is essentially linear in the number of tasks and number of processors.

The list scheduling algorithm generates the schedule using an individual tasklist for each processor. This scheduling algorithm requires the task execution times of all tasks on all processors. The execution time of task j on processor i is denoted as $\mathtt{p}_{ij}$. Hence, this scheduling heuristic is capable of scheduling tasks on heterogeneous multi-processor architectures. In the following sections, the list scheduling algorithm and two modifications are briefly described. A detailed description can be found in [6].

2.1 Simple List Scheduling

The simple list scheduling algorithm (LS-I) is based on [3] and is shown in Figure 1. In simple list scheduling, the assignment of tasks to processors is based on the actual processing time of each processor, $\mathtt{sum}_i$. Each task is assigned to the processor with the minimal sum of processing time and task execution time (line 8).

2.2 Modified List Scheduling

The modified list scheduling algorithm presented in Figure 2 was firstly introduced by [1]. It differs from the simple list scheduling algorithm mainly by two facts. First, for each processor a tasklist including all tasks is created. The tasks in these tasklists are sorted by their efficiency. The efficiency $\mathtt{e}_{ij}$ of a task j on processor i is defined as the ratio of the minimum execution time of task j over all processors to the execution time of task j on processor i

```
1  procedure LS-II()
2  begin
3    for all n processors i do
4      create tasklist_i sorted by e_ij
5      sum_i ← 0
6    endfor
7    while (not all tasks are assigned) do
8      find processor i with minimal sum_i
9      get next unassigned task j of tasklist_i
10     if ((∄ j) ∨ (e_ij < 1/√n)) then
11       deactivate processor i
12     else
13       assign task j to processor i
14       sum_i ← sum_i + p_ij
15     endif
16   endwhile
17 end
```

Figure 2: Pseudo code for the modified list scheduling algorithm (LS-II).

($\mathtt{e}_{ij} \leq 1$). Second, a task is only assigned to the processor with the shortest processing time, if the task has a high efficiency. If the efficiency drops below a limit, then the processor is deactivated and no more tasks are assigned to it (line 11).

In the third scheduling algorithm, LS-III, the ordering criteria for the tasklists is extended. Tasks with the same efficiency are ordered by their execution times.

3 Experimental Results

The comparison of the three scheduling algorithms is based on runtime measurements taken on a multi-DSP system from Transtech. This PC-based system is equipped with up to 8 TMS320C40 processors (TIM40 modules). Two parallel implementations are used for the comparison. In the first implementation, a set of randomly generated dummy tasks with predefined execution times is scheduled. In the second implementation, the list scheduling algorithms are applied in a high-performance simulator. Both parallel algorithms are implemented using the distributed operating system Virtuoso [8]. They are organized in a master/slave structure. The master processor is responsible for the task generation, the scheduling of all tasks onto the slave processors as well as the reception and processing of the tasks' results. All tasks are executed on the slave processors. The multi-DSP system is connected in a tree topology with the master processor as root. The root has at most 5 children. Hence, the tree topology of the 8-processor system has 5 processors at the first level and 2 processors at the second level.

task set	*mean [ms]*	*std. dev.*
T1	1	0.5
T2	10	5
T3	100	50

Table 1: Mean and standard deviation of the task execution times for the randomly generated tasks.

Two runtimes are used for the comparison: the runtime required for the scheduling of all tasks on the master processor, t_{sch}, and the runtime required for the execution of all tasks on the slave processors, t_{exe}. This runtime includes the communication time for the tasks' input and output data.

3.1 Randomly Generated Tasks

To evaluate the scheduling algorithms, a predefined number of independent dummy tasks is generated. At task generation, each task is assigned to a random task execution time which is utilized by the scheduling algorithms. On the slave processors, the tasks wait actively for the predefined task execution time and then they return an acknowledgment to the master processor. Three different sets of tasks are

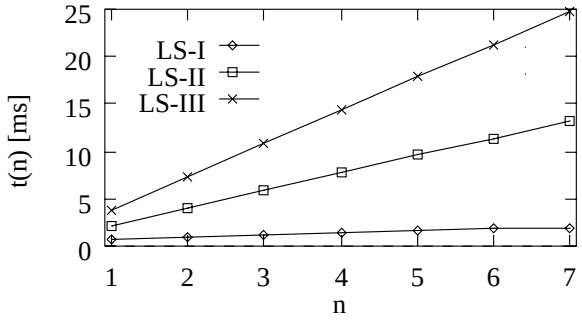

Figure 3: Runtimes of LS-I, LS-II and LS-III for scheduling 100 tasks on 1 to 7 slave processors.

used for the comparison. Each set consists of 100 tasks with normally distributed execution times. Table 1 shows the mean and standard deviation of the task execution times. The tasks are scheduled on 1 to 7 slave processors.

Figure 3 presents the runtimes of the scheduling algorithms dependent on the number of slave processors, n. LS-I has the shortest runtime; the runtime of LS-II and LS-III increases linear with the number of slave processors due to the sorting of the tasklists by the task efficiency e_{ij}.

Figure 4 shows the influence of the scheduling algorithm on the overall speedup. The overall speedup $S(n)$ is defined as the ratio of the single-processor runtime over the runtime of the parallel implementation using n slave processors $(t_{sch} + t_{exe})$. LS-I achieves a better speedup for short tasks (tasks set T1) than the other two scheduling algorithms due to the short runtime of the scheduling algorithm. In this case, t_{sch} and the communication overhead limit also the speedup. For longer task runtimes, the runtime of all tasks, t_{exe}, dominates the runtime of the parallel implementation. For these tasks, LS-II and LS-III achieve better speedups than LS-I.

3.2 Application Example

In this application example, the list scheduling algorithms are applied in a parallel implementation of a qualitative simulation algorithm [5]. In this application, the tasks are also scheduled onto n slave processors. However, there are two differences to the previous application. First, the slave processors are equipped with specialized coprocessors to reduce the runtime of some functions. Hence, the runtime for a task may be different on different slave processors. Second, the exact runtime of a task is not known at scheduling time. Therefore, estimations of the runtimes are used.

Table 2 shows the scheduling runtime, t_{sch}, and the runtime of all tasks, t_{exe}, for the three different simulation models M1, M2 and M3. Each model is partitioned into 30 tasks which are executed on 3 slave processors. In this special-purpose computer architecture, each slave processor is equipped with a different coprocessor. In most cases, LS-I achieves the shortest parallel runtime $(t_{sch} + t_{exe})$. The modified scheduling algorithms do not sufficiently reduce the task completion time to compensate for the increased scheduling time. Therefore, LS-I is finally applied in this implementation.

4 Conclusion

We presented a comparison of three list scheduling algorithms for on-line scheduling of independent tasks on multi-DSP systems. The modified list scheduling algorithms LS-II and LS-III achieve better results for tasks with longer execution times $(t_{sch} \ll t_{exe})$ and more slave processors. In these cases, the scheduling algorithms reduce the parallel execution time. For

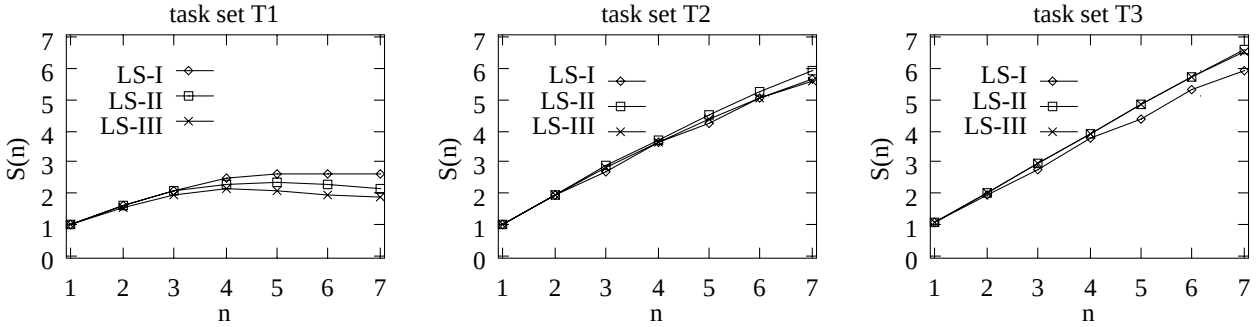

Figure 4: Achieved speedup $S(n)$ using 1 to 7 slave processors for the task sets T1, T2 and T3.

	LS-I		*LS-II*		*LS-III*	
model	$t_{sch}[ms]$	$t_{exe}[ms]$	$t_{sch}[ms]$	$t_{exe}[ms]$	$t_{sch}[ms]$	$t_{exe}[ms]$
M1	0.12	27.09	0.73	27.10	1.28	27.12
M2	0.12	2.75	0.71	1.47	1.28	1.67
M3	0.13	0.78	0.72	0.90	1.29	0.90

Table 2: Comparison of LS-I, LS-II and LS-III using 3 slave processors. Measurements are taken from a parallel high-performance simulator.

short task execution times ($t_{sch} \approx t_{exe}$), LS-I results in the shortest parallel execution time.

References

[1] E. Davis and J. M. Jaffe. Algorithms for Scheduling Tasks on Unrelated Processors. *Journal of the ACM*, 28(4):721–736, Oct. 1981.

[2] R. I. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, Mar. 1969.

[3] O. H. Ibarra and C. E. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM*, 24(2):280–289, Apr. 1977.

[4] T. Lewis and H. El-Rewini. Parallax: A Tool for Parallel Program Scheduling. *IEEE Parallel & Distributed Technology*, 1(5):155–166, May 1993.

[5] M. Platzner, B. Rinner, and R. Weiss. Parallel Qualitative Simulation. *Simulation Practice and Theory*, 1997. Elsevier Science Publishers B.V. In print.

[6] B. Rinner. *Design, Implementation and Experimental Evaluation of a Scalable Multiprocessor Architecture for Qualitative Simulation.* PhD thesis, Graz University of Technology, 1996.

[7] J. A. Stankovic, M. Spuri, M. D. Natale, and G. C. Buttazzo. Implications of Classical Scheduling Results for Real-Time Systems. *IEEE Computer*, 28(6):16–25, June 1995.

[8] E. Verhulst. Virtuoso: A virtual single processor programming system for distributed real-time applications. *Microprocessing and Microprogramming*, 40:103–115, 1994.