ON OBJECTIVE FUNCTION SELECTION IN LIST SCHEDULING ALGORITHMS FOR DIGITAL SIGNAL PROCESSING APPLICATIONS

Jan Jonsson and Jonas Vasell

Department of Computer Engineering,
Chalmers University of Technology, S-412 96 Göteborg, Sweden
Email: {janjo, vasell}@ce.chalmers.se

ABSTRACT

In this paper we discuss the choice of objective function in list scheduling algorithms for scheduling data flow graphs onto multiprocessor architectures. A majority of the list scheduling algorithms used in practice utilize a global strategy wherein actor static levels are used for making scheduling decisions. When fine-grain DSP applications such as FIR or elliptical filters need to be scheduled on architectures that consist of commodity part processors and a general interconnection network whose interprocessor communication cost cannot be ignored, a traditional list scheduling algorithm is in many cases not the best choice. In an experimental study we compare these global strategies to local strategies that utilize load balancing. The study reveals that global strategies suffer from flaws that could cause local strategies to yield more than 10% shorter schedule lengths on the average. In particular we find that a novel Earliest Finish Time (EFT) strategy exhibits very good performance.

1. INTRODUCTION

For many digital signal processing (DSP) applications, very short data sample periods are required. Many scheduling methods (*e.g.* [1][2]) only allow different instances of the application graph to be executed in a non-overlapping fashion, thus forcing the sample period to be dictated by the *makespan*, or schedule length, for the graph. If the sample period is required to be shorter than the makespan, overlapping scheduling techniques (*e.g.* [3][4]) can be used for exploiting pipelining parallelism in the application.

In many modern DSP design systems, *e.g.* Ptolemy [5], *list scheduling* is the main technique for scheduling DSP applications on multiprocessor architectures. List scheduling is a non-overlapping scheduling technique wherein each schedulable actor in the application is assigned a priority and placed in a list ordered by the actor priorities. Actor priorities are calculated using an *objective function* that takes as input an actor/processor pair and leaves as output an objective value that will be used as actor priority. At each scheduling step the schedulable actor with the highest priority is assigned to an idle processor. Traditional list scheduling algorithms (*e.g.* [6]) use as an actor's priority a *static level* that represents the maximum "length" in execution time from the actor to the output actors. Modern list scheduling algorithms (*e.g.* [1][7]) use dynamic priorities that change for each new scheduling step to take into account architecture parameters such as communication cost. Hu [6] showed that the strategy with static levels yields a minimal makespan for systems with tree-structured application graphs, unity actor execution time and negligible communication cost. For more realistic assumptions about the application and the architecture, minimizing the makespan is an NP-complete problem [8] and it is thus very likely that optimal solutions can only be guaranteed using computationally intractable exhaustive search techniques. For this reason, many greedy heuristic strategies have been proposed in the list scheduling class [1][7][9].

A list scheduling algorithm can be classified as *processor-driven* or *graph-driven* [10]. In the processor-driven approach the set of schedulable actors is updated only when no more schedulable actors are available or an actor cannot be scheduled because all processors are busy. In this situation, one or more processors are forced to complete their execution and become idle, thereby generating new schedulable actors or idle processors. The processor-driven approach provides a means for load balancing between processors using a global time to track the increasing sequence of processor completion times [11]. Unfortunately, the processor-driven approach is not capable of overlapping actor execution time with interprocessor communication delays in a multiprocessor architecture with non-negligible interprocessor communication cost. In contrast, the graph-driven approach updates the set of schedulable actors following the scheduling of each actor and the completion times of all processors are immediately recalculated. The graph-driven approach is able to overlap actor execution with communication delays and can thereby produce shorter makespans.

In this paper we will investigate how different objective functions affect the scheduled makespan. Objective functions with global strategies (static/dynamic levels) are compared to objective functions with local strategies (earliest start/finish times).

2. EXECUTION MODEL

We assume a homogeneous Synchronous Data Flow (SDF) [12] execution model. In data flow, a program is represented as a directed graph in which the nodes represent data flow actors and the arcs represent data dependencies between actors. Data are represented as tokens flowing along the arcs. An actor is executed when all necessary data are available on its input arcs. The graphs used in SDF is a special class of data flow graphs wherein the actors lack data dependency in their execution patterns. We will restrict our execution model even further by assuming that the application is described as a homogeneous SDF graph, *i.e.* a graph in which each actor always produces and consumes exactly

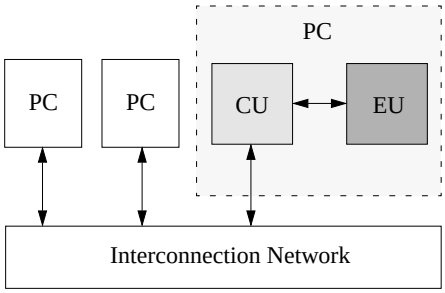

Figure 1. Basic architecture model

one token per invocation. This does not impose restrictions on the applicability of the presented results as a general SDF graph always can be converted into a homogenous SDF graph [13].

We assume a multiprocessor architecture based on the model depicted in Figure 1. The architecture consists of a set of homogeneous processor cells (PCs) attached to an interconnection network. Processor cells communicate with each other using messages that carry tokens to and from data flow actors. The communication cost in the interconnection network is assumed to be non-negligible, and may depend on the distance between communicating processors as well as on other communication taking place in the network. Each processor cell consists of an *execution unit* (EU) and a *communication unit* (CU). The execution unit executes data flow actors in a fully-static, non-preemptive fashion, *i.e.* each actor is invoked at a predetermined time and executes to completion without interruption from other actors. The communication unit handles token message flow in the network and operates in parallel with the execution unit.

3. EXPERIMENTAL SETUP

We have used an experimental platform based on the execution model described in the previous section. All modeling and simulation necessary for the experiments was performed within the FEAST evaluation framework [14].

The processor cells in the architecture were connected via a shared bus topology. The architecture size was selected in the range of 2 to 16 processor. The communication cost between two processor cells was assumed to be one time unit per transmitted data item. Network contention was not taken into account. The communication cost between two actors residing on the same processor cell was assumed to be zero.

A graph-driven list scheduling algorithm was used for the experiment. For each scheduling step, the list scheduler constructed actor/processor pairs from all schedulable actors (those whose predecessors have been scheduled) and all idle processors in the architecture. The quality of each actor/processor pair was then evaluated using an objective function. During the actor/processor pair evaluation, communication channels were tentatively established between an actor and its predecessors to take into account the overhead introduced by the interconnection network. After evaluating all actor/processor pairs, the pair that yielded the best objective value was selected. In case of a draw between two or more actor/processor pairs, the pair was selected that yielded the earliest actor start time. Finally, static scheduling information (*i.e.* actor start times) for the next scheduling step was generated. No attempts was made by the list scheduler to exploit "schedule-holes", *i.e.* unused time intervals earlier in the schedule [15].

Four objective functions were used in the experiment: Highest Static Level (HSL), Highest Dynamic Level (HDL), Earliest Start Time (EST), and Earliest Finish Time (EFT).

The HSL heuristic is a graph-driven extension of Hu's work [6] that allows arbitrary graph structure and non-uniform actor execution time. The objective of HSL is to select the actor/processor pair that yields the highest static level. This strategy exhibits graph balancing characteristics in that it gives preference to actors on an estimated critical path in the data flow graph.

The HDL heuristic is a further refinement of HSL that takes communication cost into account [1]. The objective of HDL is to select the actor/processor pair that yields the highest dynamic level, *i.e.* the biggest difference between an actor's static level and the earliest time at which the actor can be scheduled on the processor. Similar to the HSL strategy, HDL exhibits graph balancing characteristics. Because HDL takes communication cost into account it is able to compensate for inadequate estimations of the static levels. However, this compensation cannot take place during the initial scheduling steps when the earliest actor start times will be too low to have any major impact on the dynamic level. Therefore, HDL will initially behave like HSL and give preference to actors on an estimated critical path. For subsequent scheduling steps, the earliest actor start times will increase and exceed the actor static levels. At this point, HDL will give preference to actor/processor pairs that yields the earliest actor start time.

The EFT heuristic is a novel heuristic whose objective is to select the actor/processor pair that yields the earliest actor finish time. This strategy exhibits processor load balancing characteristics in that it attempts to keep the earliest available processor times as low as possible.

The EST heuristic is a graph-driven version of the work presented in [7]. The objective of EST is to select the actor/processor pair that yields the earliest actor start time. Like the EFT heuristic, EST exhibits processor load balancing characteristics. However, because EST does not take actor execution time into account, the load balancing effect will not be as good as for EFT when actor execution times are non-uniform.

For each choice of objective function, a set of 500 data flow graphs was generated using a random graph generator. Each graph contained between 75 and 100 nodes and the execution time of each node was chosen at random assuming a uniform probability distribution in the range of 3 to 5 time units. The number of predecessors to each node was chosen at random to be 1 or 2, and the depth of the graph was chosen at random in the range of 8 to 10 levels; this to mimic data flow graphs for typical fine-grain DSP applications where (a) unary and binary actors are prevalent, (b) the graph is fairly deep, and (c) there is a low distribution of data flow actor execution time. The number of data items in each token passed between a pair of actors was chosen in a way such that the communication cost ratio (CCR) of the average token communication cost over the average actor execution time corresponded to 0.5, 1.0, 2.0 or 4.0.

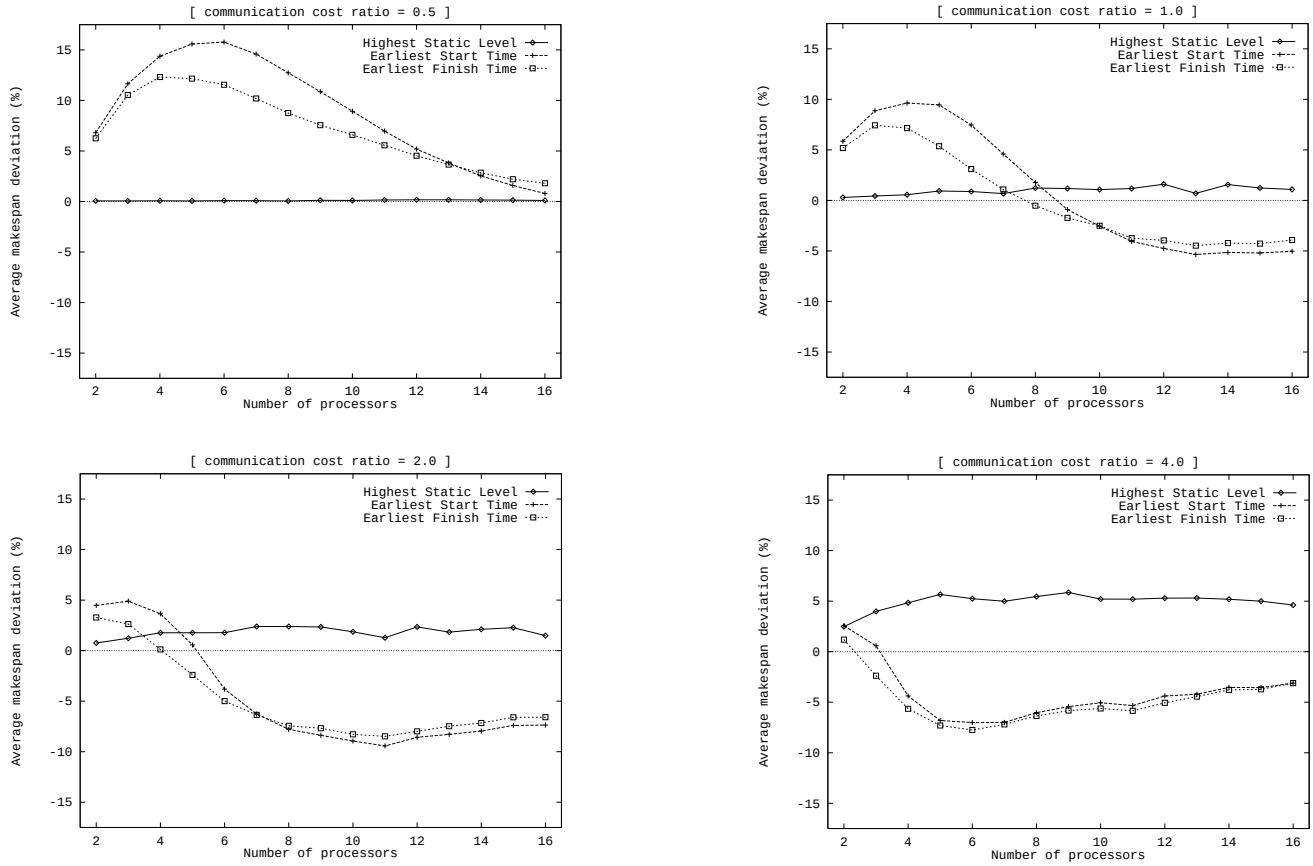

Figure 2. Average makespan deviation for different objective function heuristics relative to the HDL heuristic.

4. EXPERIMENTAL EVALUATION

The diagrams in Figure 2 summarize the results attained when the generated data flow graphs were scheduled using different combinations of objective function, communication cost ratio, and architecture size. The diagrams show the average makespan deviation in percent for each objective function relative to HDL for each value of CCR and architecture size. In the diagrams, negative deviation means shorter makespan.

The EST/EFT makespan deviation curves have two extreme points: one maximum that occurs for a low number of processors and one minimum that occurs for a medium number of processors. This curve shape can be explained as follows. HDL concentrates on an estimated critical path during the initial scheduling steps, whereas EST/EFT use a load balancing strategy. For a low number of processors, static levels will be a good estimate on the critical path and thus a much better heuristic to use than the load balancing strategy which lacks global knowledge about the application graph. For low CCR, the EST/EFT curve maxima exceed +10%, but as CCR increases, any communication overhead will invalidate the estimated static levels used by HDL and the EST/EFT curve maxima will decrease towards zero. For a medium number of processors, HDL will give preference to actors on the estimated critical path and discriminate against other actors when the number of processors is too limited to allow for full extraction of graph parallelism. Then the discriminated actors may have to be executed in sequence and potentially give rise to a new critical path. EST/EFT exhibit no such behaviour and their relative performances will be improved. Also, more communication channels will be established for this architecture size and the estimated static levels used by HDL will be less accurate as CCR increases. At best, the EST/EFT curve minima are approximately -10%. For a larger number of processors, however, the HDL discriminating effect disappears and the EST/EFT makespan deviations will again increase towards zero (this was verified through measurements with the number of processors set as high as 64).

For low CCR, the EST strategy is approximately 5% worse than EFT. This is because the load balancing strategy used by EFT is more effective as it considers actor execution time. Furthermore, when CCR is low, HSL is comparable to HDL, but as CCR increases, the HSL makespan deviation increases because HSL does not compensate for communication cost.

To investigate the effect of execution time distribution on the makespan, we have performed two complementary studies. In the first study, a uniform execution time (4 time units) was used for all actors in the graph. In the second study, the execution times were chosen in the range of 1 to 7 time units. Because of space limitation, we will only describe the results briefly. When uniform execution time is assumed, EST and EFT exhibit identical

behaviour. For low CCR, the worst EST/EFT makespan deviations exceed +15%. When CCR increases, the best EST/EFT makespan deviations are approximately -12%. When the execution time is distributed between 1 and 7, a behaviour similar to the one in Figure 2 is observed. However, the best EST/EFT makespan deviations are only around -5%.

5. CONCLUSIONS

In this paper we investigate the impact of objective function selection in list scheduler algorithms for digital signal processing applications. Experimental studies reveal that commonly used global strategies that give preference to estimated critical paths in many situations are outperformed by local strategies that employ load balancing. For example, when the average communication cost exceeds the average actor execution time, the increase in performance for a local strategy can be more than 10% on the average.

The general observations from the experiments can be summarized as follows:

(i) When the number of processors is below a critical point, a global strategy such as HSL or HDL should be used; otherwise a local strategy such as EST or EFT should be used.

(ii) The critical point decreases towards fewer processors with increasing communication cost ratio.

(iii) Among the local strategies EST and EFT, the preferred choice is EFT; especially when the number of processors is low.

(iv) The relative performance between any pair of objective functions tends to decrease with increasing distribution of actor execution time.

ACKNOWLEDGEMENTS

The work presented here was supported by a grant from the Volvo Research Foundation and the Volvo Educational Foundation.

REFERENCES

[1] G. C. Sih, 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, February 1993, pp. 175-187.

[2] G. C. Sih, E. A. Lee. “Declustering: A New Multiprocessor Scheduling Technique.” *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 6, June 1993, pp. 625-637.

[3] P. D. Hoang, J. M. Rabaey. “Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput.” *IEEE Trans. on Signal Processing*, Vol. 41, No. 6, June 1993, pp. 2225-2235.

[4] J. Jonsson, J. Vasell. “Real-Time Scheduling for Pipelined Execution of Data Flow Graphs on a Realistic Multiprocessor Architecture.” In *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Proc.*, Atlanta, Georgia, May 7-10, 1996, pp. 3314-3317.

[5] J. L. Pino, S. Ha, E. A. Lee, J. T. Buck. “Software Synthesis for DSP Using Ptolemy.” *Journal of VLSI Signal Processing*, Vol. 9, No. 1-2, January 1995, pp. 7-21.

[6] T. C. Hu. “Parallel Sequencing and Assembly Line Problems.” *Operations Research*, Vol. 6, No. 6, November 1961, pp. 841-848.

[7] J.-J. Hwang, Y.-C. Chow, F. D. Anger, C.-Y. Lee. “Scheduling Precedence Graphs in Systems with Interprocessor Communication Times.” *SIAM Journal on Computing*, Vol. 18, No. 2, April 1989, pp. 244-257.

[8] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[9] H. El-Rewini, T. G. Lewis. “Scheduling Parallel Program Tasks onto Arbitrary Target Machines.” *Journal of Parallel and Distributed Computing*, Vol. 9, No. 2, June 1990, pp. 138-153.

[10] M. Al-Mouhamed, A. Al-Maasarani. “Performance Evaluation of Scheduling Precedence-Constrained Computations on Message-Passing Systems.” *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 12, December 1994, pp. 1317-1322.

[11] B. Kruatrachue, T. Lewis. “Grain Size Determination for Parallel Processing.” *IEEE Software*, Vol. 5, No. 1, 1988, pp. 23-32.

[12] E. A. Lee, D. G. Messerschmitt. “Synchronous Data Flow.” *Proc. of the IEEE*, Vol. 75, No. 9, September 1987, pp. 1235-1245.

[13] E.A. Lee. “A Coupled Hardware and Software Architecture for Programmable DSPs.” Ph.D. Thesis, Department of EECS, University of California Berkeley, May 1986.

[14] J. Jonsson, J. Vasell. “Evaluation and Comparison of Task Allocation and Scheduling Methods for Distributed Real-Time Systems.” In *Proc. of the IEEE Workshop on Real-Time Applications*, Montreal, Canada, October 21-25, 1996, pp. 226-229.

[15] S. Selvakumar, C. Siva Ram Murthy. “Scheduling Precedence Constrained Task Graphs with Non-Negligible Intertask Communication onto Multiprocessors.” *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 3, March 1994, pp. 328-336.