ITS: AN ILP-BASED COMBINED INSTRUCTION/TASK STATIC SCHEDULING ALGORITHM

Michael Montcalm, Daniel Shapiro, Voicu Groza, Miodrag Bolic

School of Information Technology and Engineering
Computer Architecture Research Group, University of Ottawa
{ mmont044, dshap092, groza, mbolic } @site.uottawa.ca

ABSTRACT

Our combined task and instruction static scheduling algorithm implemented in the COINS compiler uses an Integer Linear Programming model to find a schedule for a program on a symmetric multiprocessor system-on-chip. We compare our work to state of the art approaches and on average we find a speedup as high as 1.49 compared to a static task scheduling approach without instruction scheduling. Depending on the computation to communication ratio of the application we estimate an average speedup of 2.55 to 2.63 in application execution time compared to sequential code.

Index Terms— MPSoC, Static scheduling, Integer linear program, Multiprocessor

1. INTRODUCTION

In recent years the complexity of embedded systems has increased with the addition of multiple types of multiprocessor architectures and bus standards. While the complexities are increasing, the time to market window is decreasing. As the trend continues towards more configurable and flexible systems, the ability to exploit parallelism, schedule tasks, and balance the number of processors for maximum performance is one that requires a solution. We consider such a multiprocessor scheduling problem and model the design space. The ideal algorithm would be able to provide an optimal design space exploration solution, while finding the solution in a reasonable amount of time.

An embedded system having multiple processors, each of which may have several issue slots presents an opportunity to schedule code at both the task and instruction levels. Large tasks are more easily scheduled into the processors which have communication delays between them, while instructions of these tasks are more easily scheduled inside each processor over multiple functional units (ALUs) which have no communication cost between them. To schedule the tasks to processors, we make several assumptions in order to make the problem tractable. We assume that the application code is statically scheduled and that the number of processors is fixed. We additionally assume that all processors are on a fully connected network and the communication costs from one processor to another are identical. During task scheduling we assume that no tasks can run in parallel in the same instruction slot on the same processor.

The design of the Integer Linear Program (ILP) representing our two models ISM (Instruction Scheduling) and MPS (Task Scheduling) focused on solving the scheduling of both tasks and instructions in embedded multiprocessors. ISM is used to perform fine grained optimization and then MPS is used to perform coarse grained optimization. While heuristic algorithms such as genetic algorithms may execute faster than the ILP approach, they fail to provide provably optimal and repeatable results. Due to the nature of linear programming our models deliver deterministic results. The solutions from each ISM and MPS model is the global maximum provided that the model executes to completion. In cases when the solver fails to exit ISM or MPS, it is interrupted and the current best feasible solution is used.

Employing the COINS compiler, MPS uses control dataflow graph (CDFG) information and ISM uses instruction information. The models were implemented using the LINGO math programming language [1] and integrated into COINS under the assumption that each task is a basic block. Each CDFG is converted into the form of a Directed Acyclic Graph (DAG) so that the graph nodes can be scheduled by the ILP model. We assumed that having tasks duplicated to reduce communication between processors is not allowed.

2. PRIOR ART

Several approaches to solving scheduling at both the task and instruction level have been discussed in the literature. The survey of several algorithms for task scheduling in [2] shows that heuristics and declutstering list scheduling algorithms give suboptimal solutions compared to other methods. The results in [2] show only the overall speed of the program in comparison to the optimal solution and not the amount of time required to solve each problem. The usability of these optimal

Thanks to NSERC for funding this research.

solutions may be small as the time required to find the solution may be unacceptable. The issue of time required to find an optimal solution was presented in [3] where Mixed Integer Linear Program (MILP)-based solutions were found to be a bad fit for the multiprocessor task scheduling problem. However, as shown in [4], a Binary Linear Program (BLP) can be used with constraints on the task scheduling problem such that optimal results can be obtained in a reasonable amount of time for small CDFGs.

Both task and instruction level scheduling are NP-complete, but scheduling at the instruction level is much harder as the problem is much larger. Each task can have in it several hundred instructions, which causes a model to go from taking several minutes to find a global optimum to requiring an unreasonable amount of time. Integer Linear Programming (ILP) based approaches have met with optimal but slow running solutions [5]. ILP-based solutions to the instruction scheduling problem have been discussed extensively in the literature. For example, [6] uses an ILP to schedule clusters of instructions into a IA-64 VLIW processor, and [7] uses an ILP model for multiprocessor RISC architectures. The approach of [6] could be easily applied to another VLIW architecture such as Tensilica's Xtensa [8]. Our goal was to extend this line of research to include multiple VLIW RISC processors typically used for Digital Signal Processing (DSP). Unlike [6] we do not attempt to control the granularity of tasks, or consider resource pressure. As well, cyclic dependencies in the CDFG are not allowed in our approach, whereas [7] can vary the loop unrolling degree. We model the start time of each instruction in each pipeline of each processor, using a time-based model as described in [9]. This problem has been investigated in [10], where instruction scheduling is performed on traces, as opposed to whole programs. They show a speedup as high as 200% over an unmodified program. We expect to see an even higher speedup as we do both local and global design space exploration.

3. INSTRUCTION AND TASK SCHEDULING (ITS) ALGORITHM OVERVIEW

The primary problem model schedules instructions in a basic block to a set number of functional units (ALUs) in a processor. Our assumptions are that the primary model does not see the other basic blocks, nor their communication. It is also assumed that all communication between basic blocks occurs at the beginning and end of each block. After finding a valid assignment of instructions to functional units given the constraints of computation time and dependencies, the primary problem hands this list of shortened basic blocks to the secondary problem model to have the basic blocks assigned to processors. The secondary model receives the set of tasks (where each task is a basic block) from the primary problem. The secondary model then schedules the tasks to each processor in the system, given the constraints of computation time, communication time, and dependencies. The high level algorithm for our approach is presented in Alg. 1. We first discover the execution time of each basic block (w) on a VLIW processor. We then find the best case schedule of basic blocks into processors in a Symmetric Multi-Processor (SMP) system.

Algorithm 1 High Level Algorithm

1: Generate list of basic blocks $blocks$ from CDFG
2: **for** ($basicBlock \in blocks$) **do**
3: $w(basicBlock) = INSN\_SCHEDULE\_PASS(basicBlock)$
4: **end for**
5: $makespan = TASK\_SCHEDULE\_PASS(blocks, w)$

3.1. Primary Problem: ISM

For each basic block of application source code, Algorithm 2 is used to discover an assignment of instructions to functional units of a VLIW processor. The objective is to minimize the schedule length.

Algorithm 2 INSN_SCHEDULE_PASS

1: Create Transitive Closure T and Identity Matrix ID and write them into model ISM
2: **for** ($instruction \in basicBlock$) **do**
3: Find execution time of $instruction$
4: Find dependencies of $instruction$
5: Write execution time and dependencies into model ISM
6: **end for**
7: $w(basicBlock) = model(ISM)$
8: $RETURNw(basicBlock)$

3.2. Secondary Problem: MPS

We find an assignment of tasks to processors in Algorithm 3 using the instruction-scheduled tasks received from the models of the primary problem. The objective is again to minimize the schedule length, this time using tasks instead of instructions.

Algorithm 3 TASK_SCHEDULE_PASS

1: Create Transitive Closure T and Identity Matrix ID and write them into model MPS
2: **for** ($basicBlock \in blocks$) **do**
3: Find dependencies of $basicBlock$
4: Write $w(basicBlock)$ and dependencies into model MPS
5: **end for**
6: $RETURNmodel(MPS)$

4. MODEL DEFINITION

For the ISM and MPS models of ITS we use variations of the same constants and variables. The objective of each model is the same, to minimize the schedule length (also called the

makespan) as shown in Eq. 1. For ISM this objective translates as the minimum schedule length of a sequence of instructions inside a VLIW RISC processor. Whereas for MPS this translates as the minimum schedule length when basic blocks are assigned into a multiprocessor system.

$$Objective = MIN(makespan) \tag{1}$$

The constant $WCET$ represents the worst case execution time as shown in Eq. 2. It is the sum of all communication and computation times. T is the transitive closure of a graph. For ISM the graph is the dataflow graph (DFG), whereas for MPS the graph is the CDFG. ID is an identity matrix with the same size as T.

$$WCET = \sum c(V_i, V_j) + \sum w(V), \forall V, V_i, V_j \tag{2}$$

Having presented the objective of the model, we now turn our attention to the constraints. As we can see in Eq. 3 the processing time for each processor P is equal to the latest start time S of a task assigned to P plus the execution time of the task, plus the delay in the task due to communication. Whereas c is the communication delay between two tasks (V_i, V_j) when they are assigned to different processors, $comm$ captures the choice to communicate that results from a particular task schedule. The communication penalty is therefore $comm$ as defined in Eq. 10, and for ISM $comm$ is always 0.

$$\begin{aligned} Ptime(P) + WCET * X(V, P) \geq \\ w(V) + S(V) + comm(V), \forall V, P \end{aligned} \tag{3}$$

Eq. 4 ensures that each task is assigned to only one processor, and that each instruction is assigned to only one ALU in a processor. If task duplication for multiprocessors is allowed (to reduce communication costs) then this constraint can be removed in MPS.

$$(\sum X(V, P) = 1 \forall V) \forall P \tag{4}$$

The makespan for MPS is the longest execution time of the processors in the system, and for ISM it is the critical path length. The objective function minimizes the makespan in Eq. 5 and therefore it will be exactly on the constraint and equal to the longest execution time.

$$(makespan \geq Ptime(P)) \forall P \tag{5}$$

In MPS the communication between all processors in the system must be defined in order to find a valid schedule for the multiprocessor. Therefore the binary variable $xc(A, B)$ is set to 1 when two tasks A and B are in different processors, and otherwise is 0. The four constraints of Eq. 6 to 9 are only included in MPS, and these constraints perform an exclusive-or operation on two tasks and only allows xc to be 1 when the tasks are in different processors.

$$\begin{aligned} (X(V_i, P) - X(V_j, P) \leq \\ xc(V_i, V_j) + WCET * ID(V_i, V_j), \forall V_i, V_j) \forall P \end{aligned} \tag{6}$$

$$\begin{aligned} (X(V_j, P) - X(V_i, P) \leq \\ xc(V_i, V_j) + WCET * ID(V_i, V_j), \forall V_i, V_j) \forall P \end{aligned} \tag{7}$$

$$\begin{aligned} (X(V_i, P) + X(V_j, P) + WCET * ID(V_i, V_j) \geq \\ xc(V_i, V_j), \forall V_i, V_j) \forall P \end{aligned} \tag{8}$$

$$\begin{aligned} (2 - X(V_i, P) - X(V_j, P) + WCET * ID(V_i, V_j) \geq \\ xc(V_i, V_j), \forall V_i, V_j) \forall P \end{aligned} \tag{9}$$

The constraint in Eq. 10 calculates the communication cost between two tasks. For ISM we set $comm$ to 0 for all instructions.

$$(comm(V_j) = (\sum c(V_i, V_j) * xc(V_i, V_j)) \forall V_i), \forall V_j \tag{10}$$

Task/instruction dependencies are defined in a constraint for each successor with respect to each of its predecessors as shown in Eq. 11. $S(i)$ is the start time of task/instruction i.

$$S(V_i) \geq S(V_j) + w(V_j) + comm(V_j) \forall V_j \rightarrow V_i \tag{11}$$

Precedence among non-dependent tasks is achieved using the constraints of Eq. 12 through 15. Eq. 15 ensures that given two tasks/instructions they do no execute concurrently in the same processor/pipeline. It creates a situation where $xd(V_i, V_j) = 1$ or $xd(V_j, V_i) = 1$, but never both at the same time. These constraints perform a logical-nor operation on xdPossible, such that neither task/instruction is dependent for xdPossible to be 1.

We find each pair of tasks that are independent using the data from the transitive closure array T, and the result is stored in the variable R as shown in Eq. 16. Any pair of tasks V_i, V_j where $T(V_i, V_j) = 0$ and $T(V_j, V_i) = 0$ are independent as long as $V_i \neq V_j$.

$$xdPossible(V_i, V_j) \leq 1 - T(V_i, V_j), \forall V_i, V_j \tag{12}$$

$$xdPossible(V_i, V_j) \leq 2 - T(V_i, V_j) - T(V_j, V_i), \forall V_i, V_j \tag{13}$$

$$xdPossible(V_i, V_j) \geq 1 - T(V_i, V_j) - T(V_j, V_i), \forall V_i, V_j \tag{14}$$

$$xdPossible(V_i, V_j) = xd(V_i, V_j) + xd(V_j, V_i), \forall V_i, V_j \tag{15}$$

$$R(V_i, V_j) = 1 - (1 - T(V_i, V_j)) * (1 - T(V_j, V_i)) \forall V_i, V_j \tag{16}$$

For two independent tasks the variable xd forces the task precedence indicated by $xdPossible$, as shown in Eq. 17. The R variable is used to unbind this constraint for tasks that are not independent.

$$\begin{aligned} S(V_i) + WCET * (xd(V_i, V_j) + R(V_i, V_j) \\ + xc(V_i, V_j)) \geq S(V_j) + w(V_j) + comm(V_j) \forall V_i, V_j \end{aligned} \tag{17}$$

5. EXPERIMENTAL RESULTS

The results from this section were derived from a set of randomly generated DAGs ranging in size from 10-32 tasks and varying communication to computation ratios (CCR). We compared the results of MPS and ITS to load balancing with minimized communications (LMBC), declustering (DC), a variant of largest processing time declustering (LPTDC), and the preferred path selection (PPS) algorithms presented in [2]. For the ITS model each processor can issue instructions to two functional units each cycle, and for all other models only one instruction per cycle per processor is allowed. The algorithms of Table 1 are from [2], the MPS model from [4], and Section 1. A star indicates a provably optimal solution. The negative numbers under the ITS column indicate that the addition of instruction scheduling increases performance past the optimal of task scheduling alone. Any solution for which a provably optimal schedule length could not be found is assumed to have an optimal schedule length of half the sequential time, which is the theoretical limit on the speedup for these tests.

Table 1. Percentage longer than the optimal schedule length on 2 single issue processors.

# Tasks	CCR	% Longer than Optimal Schedule					
		LBMC	DC	LPTDC	PPS	MPS	ITS
10	0	44	30	1	5	1	-46*
10	1	26	35	37	40	0*	-49
12	0	41	29	10	35	1	-11*
12	1	33	30	45	43	0*	-47
14	0	34	21	16	42	1	0
14	1	57	37	36	38	0*	-47
16	0	25	46	12	29	0	-21
16	1	47	10	31	50	0*	-47
18	0	48	22	15	35	0	-23
18	1	52	11	71	23	0*	-37
20	0	50	19	15	21	1	-26
20	1	52	24	38	58	0*	-46
22	0	43	17	21	34	1	-16
22	1	46	0	24	104	0*	-43
24	0	46	32	16	30	6	-30
24	1	52	29	32	56	14	-42
26	0	44	21	6	27	0	-16
26	1	44	12	12	87	0*	INC
28	0	46	10	8	41	5	-15
28	1	44	30	21	91	0*	-40
30	0	48	13	5	42	0	-27
30	1	56	6	41	89	0*	INC
32	0	44	19	10	58	1	-3
32	1	51	0	53	92	29	-29

Table 2. Average speedup calculated for MPS and ITS from the tests in Tab. 1. This calculation is with respect to a single processor. Note that ITS has two processors with two issue slots per processor.

CCR	MPS	ITS
0	1.97	2.55
1	1.45	2.63

6. CONCLUSIONS & FUTURE WORK

The Instruction/Task Scheduling (ITS) algorithm has been developed as a hierarchical method for performing scheduling for static applications. We found a speedup as high as 2.63 over a sequential approach, and an improvement of 118% over [4] for programs with computation and communication. The ITS algorithm also found solutions 63% better than the 200% speedup of [10]. Our future research will include refining this idea, adding constraints to reduce the runtime of the model, and investigating the addition of instruction set extensions to the model.

7. REFERENCES

[1] L. Schrage, *Optimization Modeling with LINGO*, Lindo Systems Inc., Chicago, IL., 6th edition, 2006.

[2] T. Davidović and T. G. Crainic, "Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems," *Comput. Oper. Res.*, vol. 33, no. 8, pp. 2155–2177, 2006.

[3] N. Satish, K. Ravindran, and K. Keutzer, "A decomposition-based constraint optimization approach for statically scheduling task graphs with communication delays to multiprocessors," in *Proc. DATE '07*, April 2007, pp. 1–6.

[4] Michael Montcalm Daniel Shapiro and Miodrag Bolic, "Static task scheduling for configurable multiprocessors," RFC 1, University of Ottawa, Jan 2010, http://carg.site.uottawa.ca/doc/TR–2010–1.pdf; accessed April 20, 2010.

[5] Chien-Ming Chen, Chien ming Chen, and Chung ta King, "Using integer linear programming for instruction scheduling and register allocation in multi-issue processors," in *in Multi-Issue Processors. Computers and Mathematics with Applications*, 1997.

[6] D. Kästner and S. Winkel, "Ilp-based instruction scheduling for ia-64," *SIGPLAN Not.*, vol. 36, no. 8, pp. 145–154, 2001.

[7] K. Srinivasan and K. S. Chatha, "Integer linear programming and heuristic techniques for system-level low power scheduling on multiprocessor architectures under throughput constraints," *Integration, the VLSI Journal*, vol. 40, no. 3, pp. 326 – 354, 2007.

[8] "Tensilica," http://www.tensilica.com/.

[9] D. Kästner and R. Wilhelm, "Operations research methods in compiler backends," *Mathematical Communications*, vol. 3, no. 2, pp. 159–183, 1998.

[10] N. Vachharajani, M. Iyer, C. Ashok, M. Vachharajani, D. I. August, and D. Connors, "Chip multi-processor scalability for single-threaded applications," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 44–53, 2005.