Static Task Scheduling for Configurable Multiprocessors

Daniel Shapiro
Computer Architecture Research Group
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada
dshap092@site.uottawa.ca

Michael Montcalm
Computer Architecture Research Group
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada
mmont044@site.uottawa.ca

Miodrag Bolic
Computer Architecture Research Group
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada
mbolic@site.uottawa.ca

Voicu Groza
Computer Architecture Research Group
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada
groza@site.uottawa.ca

ABSTRACT

Our task scheduling pass implemented in the COINS compiler uses a Binary Linear Programming model for scheduling a program into a multiprocessor system-on-chip where each processor can be accelerated with instruction set extensions. We compare our work to state of the art approaches and estimate an average speedup of 4.01 in application execution time compared to a sequential approach. We estimate on average a 1.43 times speedup over the use of multiprocessor scheduling without instruction set extensions.

Categories and Subject Descriptors

1.2 [**Scheduling, HW/SW Partitioning, HW/SW Interface Synthesis**]; 1.4 [**System on Chip (SoC) and Multiprocessor**]

Keywords

Multiprocessor static scheduling; Application specific instruction-set processor; Instruction set extension

1. INTRODUCTION

Modern embedded systems, from high-performance simulators to integrated multimedia cell phones, are designed using a wide range of processor architectures and bus standards. In recent years the complexity of these embedded systems has increased while the time to market window has decreased. Under such competitive stresses Electronic System-Level (ESL) design tools have emerged which promise to enable fast implementation of complex embedded systems using automated hardware/software co-design and co-generation. These tools are able to automatically add custom instructions called Instruction Set Extensions (ISEs) to the instruction set of a configurable processor. Due to the continuing trend 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. The solution mechanism would provide the most benefit if it can guarantee that the result is a global optimal solution, and execute relatively quickly.

An embedded system having several processors, each of which are customized to the tasks which it will perform, is called a configurable heterogeneous multiprocessor system-on-chip, or heterogeneous MP-SoC. As embedded system designs trend towards configurable multiprocessor architectures, the scheduling of software onto the various processors becomes a significant issue. Even more troubling is the question of how to map hardware accelerators onto a configurable embedded system because the task to processor assignment changes the utilization of the instruction set extensions, and therefore changes the runtime of the application. [8] sets out these problems as open questions. This motivated us to seek out a solution in the literature. The majority of current methods for static task scheduling for MP-SoCs with ISEs use stochastic methods or iterative improvement which can get stuck at a local maximum in the solution space. Our first contribution is to solve these problems, often with a guarantee of optimality from the model solver, using operations research mathematical modeling.

Instruction Set Extension (ISE) identification requires the discovery and selection of candidate instruction groups for implementation as custom instructions. The first step in identifying ISEs for implementation is instruction enumeration, where an algorithm searches the set of instructions for valid potential ISEs. The enumeration algorithm can either be exhaustive or a heuristic. For a potential ISE to be valid, the datapath of the instruction must be a convex subgraph of a basic block dataflow graph; the outputs of the instructions making up the ISE must not lead back to inputs of the ISE. Once the set of potential ISEs has been found, a subset

of them must be selected for implementation. The ISEs are chosen based on the the amount of time they save, the number of times they appear in the program, the frequency with which they are called, and the amount of hardware available. The implementation must not exceed the design constraints such as hardware size.

The addition of ISEs to the MP-SoC scheduling model adds another layer of complexity that must be resolved in order to create a task list that is capable of being scheduled. To accomplish this, the model must be aware of changes in the execution time of each task, as the execution time of a task is decreased when ISE hardware acceleration is added to a processor. This was achieved using a set of data in the model which shows the runtime of the task in question. The model takes this data in to account when scheduling the tasks, placing the extended tasks in processors capable of executing them. Scheduling ISEs and tasks into processors can be accomplished when we make some simplifying assumptions. Specifically, we assume that the application code is statically scheduled and the number of processors is fixed. We further assume that a given area of the chip is available for hardware accelerators to be implemented, and we do not account for the area required for routing busses between hardware modules. We also do not model the cache.

The design of the Binary Linear Program (BLP) representing our model focused in part on creating a faster and more exact model for scheduling in MP-SoCs. While heuristic algorithms such as genetic algorithms and interior points methods may execute faster than the BLP approach, they fail to provide provably optimal and repeatable results. The solution provided by our BLP model is the global maximum unless the model runs for too long and we interrupt it to select the best feasible solution. Furthermore, the nature of linear programming models delivers deterministic results. The BLP model used control dataflow graph (CDFG) information from the COINS compiler, and was implemented using the LINGO math programming language [17]. The model was integrated into COINS under the assumption that each task is a basic block. In order to schedule loops, each loop must be completely unrolled before scheduling begins. This unrolling must be done for all loops in the system to create static code in the form of a Directed Acyclic Graph (DAG) which can be scheduled by the BLP model. We used profiling information to determine the sequence and frequency of basic block execution.

2. BACKGROUND

The survey in [7] covers several possible solutions for the ISE identification problem. [11] surveys several multiprocessor task scheduling algorithms, as well as the current set of assumptions used in their development. We follow similar assumptions as those in [11] for the multiprocessor model, working on the Task Precedence Graph style of scheduling. Our technique in scheduling follows the b-level approach, minimizing the critical path of the graph. While we have observed that several solutions for multiprocessor task scheduling have been published, as well as solutions for uniprocessor ISE identification, there are only a few papers that address the challenge of applying ISE identification to a multiprocessor system.

Approaches that automate the creation of an MP-SoC with ASIPs are [10, 20, 13, 19]. The first, [10] has yet to integrate the ASIP functionality into their tool. The other two are both based on tools that do not have both MP-SoC and ASIP automation. [20] managed to add MP-SoC automation to the Tensilica tools in 2007, while [13] modified the Altera tools in 2006. Without the modifications, Tensilica does not support MP-SoC automation, and Altera supports neither (though both allow for user-defined MP-SoC automation). The approach taken in [19] simultaneously extends the processors as it schedules tasks for them. The benefit here is that no task with ISEs is ever scheduled outside the processor that was extended for it, and the scheduling is much easier. However, as the tasks are extended as they are scheduled, a global approach to the creation of ISEs is impossible, which limits the number of isomorphic instructions that can be uncovered.

Other approaches opt for creating a design environment that facilitates the creation of multiprocessor systems without custom instructions. [9] shows that the approach used by the Cmpware CMP-DK is to focus design efforts on software by abstracting away the hardware. The downside of this approach is that the processors themselves are treated as black-boxes, which prevents the customization that we wish to provide using ISEs. [6] focuses on analysis of applications based on FPGA arrays to produce speedup based on power consumption. They achieve an average of a 40x speedup per watt in comparison to the Xeon processor, with a maximum of a 300x speedup for Gaussian Number Generation benchmarks. While this is extremely important for low power applications, the fact that the overall execution time of the program in seconds may be many times longer than the Xeon processor, the applications for designs requiring speed alone are limited.

[16] found that an existing Mixed Integer Linear Program (MILP)-based solution was a bad fit for the multiprocessor task scheduling problem. In the [16] approach, the algorithm for task scheduling is broken down into a master and sub-problem. The master problem is a simple constraint optimization, producing several potential solutions, which in turn are analyzed by the sub-problem. The sub-problem learns new constraints to prune suboptimal potential solutions from each potential solution it processes. Their algorithm can find solutions for multiple topologies, whereas we have assumed non-blocking communication for a fully connected network. Their results from both MPEG decoding and IPv4 packet forwarding benchmarks were equal or better than the DLS algorithm by [18] for 2, 4, and 6 processors. In addition, [16] was able to schedule a number of tasks that could not be completed by the MILP. The remaining programs and modifications in the field of MPSoC and ASIP automation support either one automation method [14, 12] or none [4, 1, 3, 2].

Our results will show that given appropriate constraints, a BLP model can perform well in terms of solution quality and execution time. Unlike the simulated annealing and SAT solver approach shown in [16], our results are deterministic and therefore predictable.

3. MULTIPROCESSOR STATIC SCHEDULING BLP MODEL

Our initial model, called MPS, addressed only the scheduling of tasks into processors, and it is described in this section. We will begin by introducing the variables and constants used in the model, and then we will describe the objective function and the constraints. The example we use in this section is the same one used in [16] and was taken from [18]. Many of the variable names in [16] are used in our model for the same purpose in order to ease the comparison between approaches.

We now introduce the model variables and constants. Each task V_i has an execution time w, and a start time S. Each task is associated with a communication delay *comm* which represents the time that the task must wait to receive data from a predecessor task. For our example the tasks will be called $V1$ through $V8$. The dependencies among tasks is shown in Fig. 1. Each dependency arc is associated

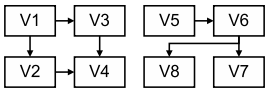

Figure 1: Dependency graph for the example described in [18].

with a communication delay c required to transmit data between tasks that are not assigned to the same processor, and the binary variable xc which indicates when two tasks are assigned to the same processor. All pairs of tasks (A, B) are associated with three numbers: $T(A, B)$ which is a constant, $xd(A, B)$ which is a binary decision variable, and $xdPossible(A, B)$ which is a continuous variable which ends up being restricted to the two values 0 and 1 by linear constraints. $xdPossible(A, B)$ allows us to track the possibility that two tasks are independent, and $xd(A, B)$ orders independent tasks running in the same processor. The transitive closure T for every pair of tasks is calculated before the model is initiated, and represents the reachability of some task A by some task B by traversing task dependencies. We have assumed that the transitive closure of a task and itself is 1. The complete transitive closure array is presented below.

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

We assume a fixed number of processors and for this example we set the number of processors to 2. Each processor is associated with an execution time $Ptime$ required to complete all of the tasks assigned to it. For every task and every processor there is a binary decision variable $X(TASK, PROCESSOR)$ which represents the assignment of the task to the processor. X is 1 if the task is assigned to the processor, and 0 otherwise.

The execution time of each task $V1$ to $V8$ is represented by the constants $[3, 3, 3, 7, 2, 3, 5, 6]$. The variable w indexes these constants such that $w(V1) = 3$, $w(V2) = 3$, and so on. The communication delays are modeled similarly using c and the constant array $[9, 5, 7, 4, 3, 10, 4]$. Effectively whenever c or w are used in the model, they are really representing a constant.

The objective of the model is to minimize the execution time of the application on the MP-SoC (called the *makespan*), and the start times of the tasks in the application.

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

We define a helpful constant $WCET$ which represents the worst case execution time of the model. $WCET$ is the sum of all communication and computation times and is used later on in the model when we must force certain constraints to be non-binding.

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

We now begin to introduce the model constraints, starting with the processing time for each processor P. The communication cost *comm* is defined in Eq. 10.

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

Eq. 4 is used to ensure that each task is assigned to exactly 1 processor. If task duplication is allowed (to reduce communication costs) then this constraint can be removed. We assumed that tasks cannot be duplicated.

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

For each processor, the total time for the assigned tasks must be less than the maximum cycle time (the makespan). Put another way, the makespan is equal to the time of the longest executing processor. Technically it is greater than or equal to the longest execution time of a processor, but the objective function minimizes the makespan and therefore it will be exactly on the constraint and equal to the longest execution time. The constraint for this relationship is presented in Eq. 5.

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

We must set the ordering of all tasks in the system, and for that reason the binary variable $xc(A, B)$ is set to 1 when two tasks A and B are in different processors, and otherwise is 0. Also, xc takes on a large number when A and B are the same task in order to prevent an ordering relationship between a task and itself. In the following constraints ID is an identity matrix with the same size as T.

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

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

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

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

Eq. 6 through 9 compute which of two different, independent tasks on the same processor will be scheduled first, and result in the truth table shown in Tab. 1 for the variable xc.

Table 1: Truth table for xc given the constraints 6 through 9

$X(V_i, P)$	$X(V_j, P)$	$xc(V_i, V_j)$
0	0	0
0	1	1
1	0	1
1	1	0

The constraint of Eq. 10 calculates the communication cost between two tasks.

$$(comm(V_j) = (\sum c(V_i, V_j) * xc(V_i, V_j))\forall V_i), \forall V_j \quad (10)$$

Looking at the data dependency in the CDFG of a program, some tasks are known to be dependent on the data from other tasks. All successor tasks must begin after their predecessor task(s), and therefore we define a constraint for each successor with respect to its predecessors as shown in Eq. 11. Because the tasks with no successor will have no upper bound, we minimize the start times of the tasks in the objective function defined in Eq. 1. This brings the execution times of tasks without successors to their minimum as required.

$$S(V_i) \geq S(V_j) + w(V_j) + comm(V_j)\forall V_j \rightarrow V_i \quad (11)$$

We must also define the precedence among non-dependent tasks so that the model does not allow them to execute concurrently on the same processor, and this is achieved using the constraints of Eq. 12 through 16. Eq. 16 ensures that given two tasks, one task is first and the other is second if they are assigned to the same processor. It creates a situation where $xd(V_i, V_j) = 1$ or $xd(V_i, V_j) = 1$, but never both at the same time. Implicit in 12 is 13 but it is stated for clarity.

$$xdPossible(V_i, V_j) \leq 1 - T(V_i, V_j), \forall V_i, V_j \quad (12)$$

$$xdPossible(V_i, V_j) \leq 1 - T(V_j, V_i), \forall V_i, V_j \quad (13)$$

$$xdPossible(V_i, V_j) \leq 2 - T(V_i, V_j) - T(V_j, V_i), \forall V_i, V_j \quad (14)$$

$$xdPossible(V_i, V_j) \geq 1 - T(V_i, V_j) - T(V_j, V_i), \forall V_i, V_j \quad (15)$$

$$xdPossible(V_i, V_j) = xd(V_i, V_j) + xd(V_j, V_i), \forall V_i, V_j \quad (16)$$

We clarify the implications of the constraints of Eq. 12 through 15 in the truth table shown in Tab. 2. We see that the constraints perform a logical-nor operation on xdPossible. And so in Eq. 16 when $xdPossible(A, B)$ is 0, both $xd(A, B)$ and $xd(B, A)$ must be 0. On the other hand, when $xdPossible(A, B)$ is 1, then either $xd(A, B)$ or $xd(B, A)$ must be 1, and the other will be 0.

We wish to set constrains which order all independent tasks in the model. 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. 17. Any

Table 2: Truth table for $xdPossible(A, B)$ given the constraints 12 through 15

$T(A, B)$	$T(B, A)$	$xdPossible(A, B)$	$xdPossible(B, A)$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

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 $V_i \neq V_j$.

$$R(V_i, V_j) = 1 - (1 - T(V_i, V_j)) * (1 - T(V_j, V_i))\forall V_i, V_j \quad (17)$$

For each pair of independent tasks we assure that they do not execute at the same time if they are in the same processor by using the constraints of Eq. 18. Put another way, for two independent tasks the variable xd loosens the constraint of the predecessor task which would have forced it to execute before its successor, but it does not loosen the constraint on the successor task with regards to the predecessor. Furthermore, xc loosens the precedence constraints when the independent tasks are in different processors because their ordering no longer matters. The R variable is used to unbind this constraint for tasks that are not independent.

$$\begin{aligned} S(V_i) + WCET * (xd(V_i, V_j) + R(V_i, V_j) \\ +xc(V_i, V_j)) \geq S(V_j) + w(V_j) + comm(V_j)\forall V_i, V_j \end{aligned} \quad (18)$$

The results from the model assuming 2 processors is the set of start times [0,3,6,9,0,2,5,10], task-to-processor assignments [1,1,1,1,2,2,2,2], and makespan 16. The starting point for the MPS model was [16], and our contribution was to formulate a BLP to solve the same type of problem.

4. MULTIPROCESSOR SCHEDULING WITH ISE SELECTION

In order to get the model from the previous section to schedule tasks into heterogeneous processors, and to configure those processors to best reduce the makespan of the scheduled code, several constraints were added from the Instruction-Set Extension (ISE) domain. Specifically, the knapsack problem constraints and data from the instruction selection problem were added to MPS. This expanded model is called IMP, and we begin by presenting the data added to the model and then proceed to explain the new variables along with the added and changed constraints.

When the scheduling model is called we assume that the candidate instruction set extensions have been enumerated using an instruction enumeration algorithm such as [15] with an I/O constraint of (4,4). $INC(ISE, TASK)$ is a binary decision variable which specifies when an ISE should be included in a specific task. ISEs are represented in the model as having associations to any instructions which other ISEs may also want to use. Therefore we are able to perform conflict resolution between mutually exclusive ISEs at the same time as the ISE selection. Each conflict-prone instruction in the model is represented with a name (k_i), and the list of names is included as data in the model.

Each conflicting use of a node by an ISE is marked (e.g. $ISE\_USES = [ISE7 \rightarrow k3, ISE8 \rightarrow k3]$). As shown in Eq. 19 and 20 TU is the number of times a node is used by (i.e. implemented in) any ISE.

$$TU(Node) \geq \sum INC(ISE_i, V) \quad \forall V, ISE_i \rightarrow V, Node \rightarrow ISE_i \tag{19}$$

$$TU(Node) \leq 1 \forall Node \tag{20}$$

ISEs themselves are represented with the labels $ISE1$ to ISE_N, and the hardware cost (HC) of implementing each ISE is represented in the model by constants. Equivalent ISEs are marked as being part of an equivalence class: $EQUIV = [ISE3 \rightarrow L1, ISE4 \rightarrow L1]$. $EQIV\_HC$ defines the cost of implementing any ISE which is part of an equivalence class. The saved time resulting from each use of an ISE is represented by clock cycles in the array SC. The available chip area for implementing ISEs in the design is represented by a knapsack capacity constant (e.g. $CAP = 30$).

$CU(PROCESSOR, CLASS)$ is used to calculate when a class of equivalent ISEs has been used already in a given processor, and therefore all subsequent uses of that class do not require additional hardware. $ST(TASK)$, calculated with the constraint of Eq. 21, is a measure of the time actually saved by a task due to the addition of ISEs to the processor that the task was assigned to. Eq. 22, 23 and 24 are modified versions of Eq. 3, 11, and 18 respectively.

$$ST(V) = \sum (SC(ISE_i) * INC(ISE_i, V)) \forall ISE_i \rightarrow V \forall V \tag{21}$$

$$Ptime(P) + X(V, P) * WCET \geq S(V) + w(V) + comm(V) - ST(V) \tag{22}$$

$$S(V_i) \geq S(V_j) + w(V_j) + comm(V_j) - ST(V_j) \forall V_j \rightarrow V_i \tag{23}$$

$$S(V_i) + WCET * (xd(V_i, V_j) + R(V_i, V_j) + xc(V_i, V_j)) \geq S(V_j) + w(V_j) + comm(V_j) - ST(V_j) \quad \forall V_i, V_j \tag{24}$$

For each ISE a constant UL is 1 when the ISE is not equivalent to other ISEs, or 0 when it is part of any equivalence class. As shown in Eq. 28, the cost of implementing equivalent ISEs and non-equivalent ISEs is computed separately using the cost of unlisted ISEs ULC (Eq. 25) and the cost for listed ISEs in each processor $LIC(PROCESSOR)$ (Eq. 26). In the case when two equivalent ISEs are implemented in different processors, they both contribute to the hardware size. However, when two equivalent ISEs are implemented in the same processor, then the hardware cost of implementation is only the cost of one instance of the ISE.

$$ULC = \sum INC(ISE_i, V) * HC(ISE_i) * UL(ISE_i) \quad \forall ISE_i \rightarrow V \tag{25}$$

$$LIC(P) = \sum CU(P, Class) * EQIV\_HC(Class) \forall Class \tag{26}$$

$$CU(P, Class) \geq \sum ((INC(ISE_i, V) * (1 - UL(ISE_j))) - (1 - X(V, P))) \forall ISE_i \in Class, \forall ISE_j \rightarrow V, \forall P \tag{27}$$

$$ULC + \sum (LIC(P)) \leq CAP, \forall P \tag{28}$$

5. EXPERIMENTAL RESULTS

The results from this sections were derived from a set of randomly generated DAGs ranging in size from 10 to 32 tasks and varying communication to computation ratios (CCR). We compared to results presented in [5]. The algorithms used to generate these results are load balancing with minimized communications (LMBC), DeClustering (DC), a variant of largest processing time (LPTDC), and preferred path selection (PPS) as well as our own algorithms (MPS, IMP). Tab. 3 shows the results of several algorithms presented in [5] as well as MPS and IMP. Execution time of our models for CCR=0 was approximately 3 minutes before the solver was interrupted to obtain a feasible solution. Often the feasible solution was the optimal solution; the speedup was 2 in some cases. In Tab. 3 we only indicate an optimal solution was found when the solver guaranteed that the objective was at a global optimum. For the CCR=1 tests the model execution ranged from a few seconds to a minute. A loose hardware constraint was used for IMP in order to obtain the maximum speedup, and binding the hardware constraint slightly increases the model execution time (by a few seconds for CCR=1, and a minute on average for CCR=0). Some models required as much as 5 minutes to find the optimal solution. The results are percentage difference from the optimal schedule length calculated as $r = \frac{FoundLength - OptimalLength}{OptimalLength}$. As mentioned in [16] a MILP approach, which is similar to a BLP will fail for problems where there are few task dependencies. The extreme of such a case is seen in Tab. 3 in the rows with a CCR of 0. For such programs each basic block is independent from all the others. Real programs do not often have this characteristic, and it is easy to identify. And so we see potential for MPS and IMP as algorithms that are called when communication is found in the CFG.

Tab. 4 presents the average speedup for the tests described in Tab. 3. We can see that when there are no dependencies between tasks, the model can find a high speedup but cannot guarantee the speedup optimality because there are so many possible solutions. On the other hand, when many precedence constraints are present as is the case with a CCR of 1, we find that the speedup is lower and the model objective moves quickly to a global maximum. This makes perfect sense because the large number of communicated variables when CCR is 1 causes the design space to shrink dramatically. We can see in Tab. 4 that for programs with inter-basic block communication, there is a 43% improvement for IMP over the theoretical maximum of a 2 times speedup for 2 processor MPS model.

6. CONCLUSIONS AND FUTURE WORK

As future work we expect to re-implement the models from this paper using lp_solve so that they can be included in free and open source products. We hope to integrate the output from our models back into the COINS compiler, and then to emit machine code and hardware for the designed system. At this point our model decides upon a hardware/software partition and multiprocessor schedule, but is not integrated with the code generation of COINS or our hardware generation tools. We also intend to investigate the tradeoff between adding processors to the system versus adding ISEs.

Table 3: Percentage longer than the optimal schedule length for algorithms presented in [5], Section 3 without ISEs, and Section 4 with ISEs. These results are for a 2 processor model. A star indicates a provably optimal solution.

# Tasks	CCR	% Longer than Optimal Schedule					
		LBMC	DC	LPTDC	PPS	MPS	IMP
10	0	44	30	1	5	1	-58
10	1	26	35	37	40	0*	-54*
12	0	41	29	10	35	1	-50
12	1	33	30	45	43	0*	-53*
14	0	34	21	16	42	1	-49
14	1	57	37	36	38	0*	-53*
16	0	25	46	12	29	0	-50
16	1	47	10	31	50	0*	-52*
18	0	48	22	15	35	0	-52
18	1	52	11	71	23	0*	-39*
20	0	50	19	15	21	1	-49
20	1	52	24	38	58	0*	-52*
22	0	43	17	21	34	1	-43
22	1	46	0	24	104	0*	-51*
24	0	46	32	16	30	6	-50
24	1	52	29	32	56	14	-48
26	0	44	21	6	27	0	-51
26	1	44	12	12	87	0*	-46*
28	0	46	10	8	41	5	-51
28	1	44	30	21	91	0*	-47
30	0	48	13	5	42	0	-47
30	1	56	6	41	89	0*	-52*
32	0	44	19	10	58	1	-48
32	1	51	0	53	92	29	-30

Table 4: Average speedup calculated for MPS and IMP from the tests in Tab. 3. This calculation is with respect to a uniprocessor approach.

CCR	MPS	IMP
0	1.97	4.01
1	1.45	2.86

Using the models presented in this paper, we calculated a 4.01 times speedup over a uniprocessor approach. We efficiently modeled both static scheduling and instruction set extension identification for a configurable multiprocessor. In some cases this approach can guarantee that the solution is a global maximum.

7. REFERENCES

[1] Coware platform architect. http://www.coware.com/products/platformarchitect.php.

[2] Ramp: Research accelerator for multiple processors. http://ramp.eecs.berkeley.edu/.

[3] Xilinx edk. http://www.xilinx.com/tools/platform.htm.

[4] A. Bakshi, V. K. Prasanna, and A. Ledeczi. Milan: A model based integrated simulation framework for design of embedded systems. In *Proc. LCTES '01*, pages 82–93, New York, NY, USA, 2001. ACM.

[5] 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.*, 33(8):2155–2177, 2006.

[6] M. Flynn, R. Dimond, O. Mencer, and O. Pell. Finding speedup in parallel processors. In *Proc. ISPDC '08.*, pages 3–7, July 2008.

[7] C. Galuzzi and K. Bertels. The instruction-set extension problem: A survey. In *Proc. 4th Int. workshop on Reconfigurable Computing, ARC '08:*, pages 209–220, 2008.

[8] D. Goodwin, C. Rowen, and G. Martin. Configurable multi-processor platforms for next generation embedded systems. In *Proc. ASP-DAC '07.*, pages 744–746, Jan. 2007.

[9] S. Guccione. Programming configurable multiprocessors. In *Proc. 19th IEEE International, Parallel and Distributed Processing Symposium, 2005.*, pages 4 pp.–, April 2005.

[10] A. Kumar, M. Balakrishnan, P. Panda, K. Paul, et al. Srijan - a methodology for synthesis of asip based multiprocessor socs. Technical report, Embedded Systems Group, Department of Computer Science and Engineering, Indian Institute of Technology Delhi, 2005.

[11] Y. K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, 1999.

[12] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *Proc. LCTES/SCOPES '02*, pages 18–27, New York, NY, USA, 2002. ACM.

[13] H. Nikolov, T. Stefanov, and E. Deprettere. Systematic and automated multiprocessor system design, programming, and implementation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(3):542–555, March 2008.

[14] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere. Daedalus: Toward composable multimedia mp-soc design. In *Proc. 45th ACM/IEEE, DAC 2008.*, pages 574–579, June 2008.

[15] C. Ozturan, G. Dundar, and K. Atasu. An integer linear programming approach for identifying instruction-set extensions. In *Proc. 3rd IEEE/ACM/IFIP, International Conference on Hardware/Software Codesign and System Synthesis*, pages 172–177, 2005.

[16] 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*, pages 1–6, April 2007.

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

[18] G. Sih and E. Lee. A compile-time scheduling heuristic for interconnection-constrained

heterogeneous processor architectures. *Parallel and Distributed Systems, IEEE Transactions on*, 4(2):175–187, Feb 1993.

[19] F. Sun, N. Jha, S. Ravi, and A. Raghunathan. Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors. In *Proc. 18th Int. Conf. on VLSI Design, 2005.*, pages 551–556, Jan. 2005.

[20] S. Xu and H. Pollitt-Smith. Optimization of hw/sw co-design: Relevance to configurable processor and fpga technology. In *Proc. CCECE 2007*, pages 1691–1696, April 2007.