A Hardware-Software Cosynthesis Technique Based on Heterogeneous Multiprocessor Scheduling

Hyunok Oh
Soonhoi Ha
Dept. of Computer Engineering
Seoul National University, Seoul, Korea
{oho,sha}@comp.snu.ac.kr

ABSTRACT
In this paper, we propose a fast and simple heuristic for the cosynthesis problem targeting the system-on-chip (SOC) design. The proposed algorithm covers from implementation selection and resource sharing problem in SOC design to PE selection problems in distributed heterogeneous embedded (DHE) system design. The proposed solution also considers multiple design objectives. Through benchmark experimentation, it is proven that the proposed solution produces solutions of equivalent quality to the previously published results in the DHE design. Its execution speed is several orders of magnitude smaller for large examples. We envision that the proposed approach will be one of significant cosynthesis researches in the SOC design. In the DHE design, the proposed approach could be used as an initial solution to a probabilistic algorithm guaranteeing to obtain a better solution.

Keywords
Hardware-software cosynthesis, system-on-chip, distributed heterogeneous embedded system.

1. INTRODUCTION
As the complexity of embedded system grows consistently with technology improvement, hardware-software codesign has been actively investigated as a new design methodology to enhance the design power accordingly in terms of design quality and design time[1][2]. Codesign methodology allows the designer to explore the design space in the early stage of design cycle by evaluating architectures without implementation.

There are at least two groups of research activities targeting different style of architectures in mind. One group targets distributed heterogeneous embedded (DHE) systems which consists of several different types of processing engines such as processors, FPGAs, and ASICs[3][4]. The other group targets system-on-chip (SOC) design where processing engines are processor cores and hardware blocks[5][6][7]. In the abstraction layer, both groups deal with the same problem: given the input task graphs, determine the optimal architecture within design constraints. However, there is a significant gap in the way of dealing hardware components. An SOC research considers many implementation possibilities of the same task in the hardware, while the DHE research tends to regard an ASIC as a single processing engine for the task. To consider those implementation possibilities in the DHE design, it should consider each implementation as a separate processing engine to make the problem size huge. Thus, the task granularity assumed in the DHE research larger than the SOC research.

This paper stands between those two groups. While our target architecture is more oriented toward SOC design, our technique is more like architecture cosynthesis for the distributed heterogeneous embedded systems. We use task graphs of mixed grain as input graphs, where each node of a task graph is either predefined with a set of library blocks associated with different implementation possibilities or synthesized with hardware synthesis CAD tools. The task graphs are assumed acyclic and without conditional dependencies. An IP block can be regarded as a library block with a given implementation. We allow multiple task graphs with different iteration periods. Therefore, the size of input task graphs is usually large and the number of processing elements is also huge.

The main body of hardware-software codesign consists of four problems.

- Architecture selection: determine the communication architecture including memory structure and interconnection network structure.
- Component selection: determine the processing elements to be used. For hardware modules, an implementation of each task should be selected.
- Partitioning and scheduling: partition the input tasks into processing elements and perform static scheduling to determine the execution times of tasks.
- Performance evaluation: evaluate the quality of solution and check whether design constraints are met.

Note that these problems are not independent of each other. For example, the implementation of a task can be determined after the task is partitioned into hardware. Also, the optimal architecture can be determined after components are selected. In the proposed approach, we solve the last three problems at the same time as the cosynthesis problem. And, we solve the architecture selection problem and cosynthesis problem iteratively in our codesign framework as illustrated in figure 1.

Obviously, the cosynthesis problem is very difficult to solve. A great deal of recent work has addressed partitioning problem with a given target architecture in the context of the SOC design[5][6][7]. Partitioning problem alone is complicated to

This research has been funded by Korea Science and Engineering Foundation 97-2079.
adopt fast heuristics to be applied for real-life examples. In the SOC design, the cosynthesis problem is not widely discussed yet to our best knowledge except some partitioning works considering diverse hardware implementations[6][7]. On the other hand, hardware-software cosynthesis problem has been addressed in the DHE design where the problem size of interest is manageable with mathematical or probabilistic approach[3][4][8][9]. In particular, very recently Dick and Jha[10] presented a multiobjective genetic algorithm (called MOGAC) for hardware-software cosynthesis of distributed embedded systems. Since they formulate the problem carefully in a genetic algorithm framework, they achieved significant enlargement of applicable problem domain. They successfully considered multiple design objectives such as low cost and low power, and accelerated the algorithm speed to deal with huge task graphs. We believe, however, that their probabilistic algorithm is still limited in its applicability for the SOC design since the design space for hardware part is larger.

In this paper, we propose a fast and simple heuristic for the cosynthesis problem targeting the SOC design. The proposed solution can also consider multiple design objectives. Through benchmark experimentation, it is proven that the proposed solution produces solutions of equivalent quality to the previously published results in the DHE design including MOGAC. Its execution speed was order of magnitude smaller. We envision that the proposed approach will be one of significant cosynthesis researches in the SOC design. In the DHE design, the proposed approach could be used as an initial solution to probabilistic algorithm guaranteeing to obtain a better solution.

In the following section, we explain the proposed cosynthesis framework. The detailed discussion of the proposed heuristic is discussed in section 3. We will present the benchmark experiments to demonstrate the usefulness of the proposed technique and discuss the future works in the subsequent sections.

2. COSYNTHESIS FRAMEWORK

Since the target architecture can be regarded as a heterogeneous multiprocessor system in the abstract level, we devised a cosynthesis framework based on heterogeneous multiprocessor (HMP) scheduling algorithms. The proposed framework consists of two main parts as shown in figure 2: a heterogeneous multiprocessor scheduler and the task-PE allocation controller. The inputs to the cosynthesis framework are input task graphs and a task-PE profile table. The task-PE allocation controller modifies the task-PE time table at each iteration according to the design objectives. Suppose that we want to minimize both power and cost of a design within a time constraint, the task-PE allocation controller modifies the task-PE time table so that a task will not be scheduled on processing elements of high cost and high power. We combine multiple objectives into a single weighted sum of objectives, to define the overhead function. The detailed function of the task-PE allocation controller will be explained in the next section.

The performance evaluation block examines the design constraints and records the tradeoffs between multiple objectives if all design constraints are met. Tradeoffs are obtained by changing the weights in the overhead function. After obtaining a set of cosynthesis results with different combinations of multiple objectives, the performance evaluation block closes the cosynthesis iteration.

The inputs to the HMP scheduler are task graphs and a task-PE time table as shown in figure 3(c). The scheduler tries to schedule the task graphs to the processing elements in the shortest execution time based on the task-PE time table. On the other hand, the task-PE allocation controller modifies the task-PE time table at each iteration according to the design objectives. Suppose that we want to minimize both power and cost of a design within a time constraint, the task-PE allocation controller modifies the task-PE time table so that a task will not be scheduled on processing elements of high cost and high power. We combine multiple objectives into a single weighted sum of objectives, to define the overhead function. The detailed function of the task-PE allocation controller will be explained in the next section.

In this paper, we propose a fast and simple heuristic for the cosynthesis problem targeting the SOC design. The proposed solution can also consider multiple design objectives. Through benchmark experimentation, it is proven that the proposed solution produces solutions of equivalent quality to the previously published results in the DHE design including MOGAC. Its execution speed was order of magnitude smaller. We envision that the proposed approach will be one of significant cosynthesis researches in the SOC design. In the DHE design, the proposed approach could be used as an initial solution to probabilistic algorithm guaranteeing to obtain a better solution.

In the following section, we explain the proposed cosynthesis framework. The detailed discussion of the proposed heuristic is discussed in section 3. We will present the benchmark experiments to demonstrate the usefulness of the proposed technique and discuss the future works in the subsequent sections.

3. THE PROPOSED HEURISTIC

3.1 Task-PE Allocation Controller

Before explaining the proposed algorithm, it is necessary to introduce some terminologies. A task is called "locked" out of a PE if it cannot be scheduled on a processing element, and "unlocked" otherwise. We will use "node" and "task" interchangeably in this section.

If a node is locked out of a PE, the corresponding entry of the task-PE time table is set to infinity. Initially we unlock all PE’s of minimum overhead function for all tasks. Then, the HMP scheduler schedules all tasks into the PE’s of minimum overhead. If the time constraint is not met, we move on to the next iteration by unlocking other task-PE pairs of higher overhead value. In this section, we use a single objective, cost, for brevity as the overhead function to describe the detailed task-PE allocation procedure.

Let us consider an example of figure 3. Suppose that the time constraint is 32. Figure 3(b) indicates the task-PE profile table which contains execution time and cost. Pi’s indicate processing elements while Bi’s indicate different implementations in hardware. For instance, “P1(1)” means that if we allocate P1 then system cost increases by one. “4(3)” in A-P0-B0 means that the execution time of node A on P0-B0 is 4 and the cost is 3. Figure 3(c) shows the initial time table. All PE’s except the minimum cost PE are locked. After scheduling, all nodes are scheduled on P1. The...
schedule length (or makespan) becomes 37, which does not meet the time constraint.

If the scheduling result meets the given time constraint, cosynthesis is finished with an optimal solution. Otherwise we unlock a node from a certain PE, which allows the possibility of mapping the node to a non-minimum cost PE to reduce the schedule length. In order to determine a candidate PE for unlocking, we define the expected makespan decrement (EMD) and the expected cost increment (ECI) of each PE. Furthermore, we define the slack as the difference between the time constraint and the current schedule length in order to avoid unlocking an excessively high cost PE.

\[
EMD(i,j,k) = \text{execution time}(i,j^*,k^*) - \text{execution time}(i,j,k) \\
ECI(i,j,k) = \text{cost}(i,j,k) - \text{cost}(i,j^*,k^*)
\]

Slack = the schedule length – the time constraint

(i : node, j : processor, k : implementation)

\(j^*,k^*\): the minimum cost processing element or implementation

We unlock the selected PE by unlocking PE. Nodes will be mapped to it only when the total unlocking PE does not always imply that nodes are mapped to the unlocked PE. Nodes will be mapped to it only when the total schedule length is actually reduced considering communication overheads.

![Figure 3](image)

Figure 3. Example. (a) input graph (b) task-PE profile table (c) initial time table (d) EMD/ECI (e) modified time table after unlocking C-P0-B0 (f) result

We repeat this basic iteration until all unlocked PE’s are visited. For the example of figure 3, we compute EMDs and ECIs of all pairs to obtain the result as figure 3(d). The slack is 5(37-32=5).

Now, we choose the largest min(EMD,Slack)/ECI PE. We choose the C-P0-B0 PE and unlock it. The schedule length meets the time constraint. In this example, there is no PE to relock in cost reduction step. Figure 3(f) shows the scheduling result.

In order to extend the proposed heuristic to consider resource sharing, we revise the EMD of a task-PE pair. If there are n nodes to share a PE, we set the EMD of the first PE allocation as \(n^{\ast}\)EMD because n nodes can be scheduled on the same PE without increasing cost. When more than one PE’s are allocated, EMD of each PE instance except the first allocated PE instance is defined as the reduction of the schedule length ( = current makespan – makespan after unlocking the PE ).

If the time complexity of the heterogeneous multiprocessor cosynthesis is assumed to be \(O(S)\), n is the number of nodes, p is the number of processors, and i is the number of implementations per a node-processor pair, the time complexity of the proposed partition is \(O(n^{\text{piS}})\) since it takes \(O(n^{\text{piS}})\) to compute EMD of all PE’s.

### 3.2 Heterogeneous Multiprocessor Scheduling Algorithm

In this section, we briefly explain the BIL scheduling algorithm which we adopt as the heterogeneous multiprocessor scheduling algorithm in this paper. For detailed discussion, refer to [11].

The BIL scheduling algorithm is a list scheduling technique where a node is assigned a priority based on the best imaginary level (BIL). The BIL of a node is defined as follows.

\[
BIL(i,j) = E(i,j) + \max\{\min(BIL(d,j),\min(BIL(d,l,k)+C(i,d)))\}
\]

(where i is ith node, j is jth processor, d is child node of node i, E(i,j) is the execution time of node i on processor j and \(C(i,d)\) indicates the amount of IPC overhead between i and d.)

The BIL of node i on j represents the critical path length including the IPC overhead based on the assumption that the node is scheduled on processor j and all descendant nodes are also scheduled on the processor with minimum BIL value.

It is proven that the BIL scheduling algorithm produces an optimal result when the task graph is linear. For randomly generated examples of certain topologies, the performance of the BIL scheduling algorithm was shown close to optimal solutions [11].

At each scheduling step, the scheduler compares the priorities of the runnable nodes. The priority of a runnable node is adjusted considering the processor available time. The adjusted level of a node i on processor j, BIM (Best Imaginary Makespan), is defined as follows: \(BIM(i,j) = T(i,j) + BIL(i,j)\) since the node i cannot be scheduled on processor j before \(T(i,j)\). Note that a runnable node
has N different BIM values, one for each processor, if the total number of processors is N.

Let us consider figure 3. Assume that the input time table is figure 3(e). BIL(D,P0) and BIL(D,P2) are infinite and BIL(D,P1) is 15 since D has no child. BIL(C,P0) is 20 because 3(C’s execution time) + min(infinite(=BIL(D,P0)), 15(=BIL(D,P1)) + 2 (communication)) = 20. BIL(C,P1) is 20(=5 + BIL(B,P1)), BIL(C,P2) is infinite. BIL(B,P0) and BIL(B,P2) are infinite and BIL(B,P1) is 25. BIL(A,P0) and BIL(A,P2) are infinite and BIL(A,P1) is 32 since BIL(B,P1) is larger than BIL(C,P1). After computing BIL, BIL scheduler starts scheduling. The first runnable node is A. A is scheduled on P1. Now, two nodes B,C are runnable. BIM(B,P1) is 32 since T(B,P1) + BIL(B,P1) = 7(A’s finish time) + 25. BIM(C,P0) is 29 since T(C,P0) = 7(A’s finish time) + 2 (communication time between A and C). BIM(C,P1) is 25. Since the largest BIM is BIM(B,P1), B is scheduled on P1 since B cannot be scheduled on P0. And then C is scheduled on P0 because 29(=BIM(C,P0)) is smaller than 37(=BIM(C,P1) since T(C,P1) is 17(P1’s available time)). Finally D is scheduled on P1. Figure 3(f) shows the scheduling result.

4. EXPERIMENTAL RESULTS

In this section, we illustrate some examples to analyze the performance of the proposed algorithm compared with other algorithms originally developed for DHE system cosynthesis. The proposed algorithm is implemented in C++ and run on Ultraspac I (200MHz and 256 MB memory).

The examples used for performance benchmarks are collected from the papers which describe the referred algorithms. Some examples may have multiple task graphs, in which each task graph repeats its execution at input sample’s arrival whose period is specified independently, thus composing multirate examples. Some examples specify the deadlines of tasks, by which the associated task must complete its execution after input sample comes. We adopt the same optimization technique of hyperperiod contraction to reduce the effective number of task graph copies as MOGAC uses[10].

To support deadlines of tasks in the benchmark examples, we make a simple modification to the heterogeneous multiprocessor scheduler explained in the previous section. We compare the static level (BIL) of node i with (the largest deadline - node i’s deadline + execution time) and take the larger as the revised priority. This modification gives higher priority to the task of earlier deadline.

Table 1 compares the performance of the proposed algorithm with those of Hou’s algorithm[3], COSYN[9] and MOGAC[10] when they are applied to the unclustered(u) and clustered(c) version of Hou’s task graphs.

Table 2 compares the performance of the proposed algorithm with those of SOS[4], COSYN and MOGAC for Prakash and Parker’s task graphs[4]. “P&P1 <4> ” means that Prakash and Parker’s first task graph with the worst case finish time of four time units.

Table 3 compares the performance of the proposed algorithm with those of Yen’s system and MOGAC when each system is applied to Yen’s large random task graphs[8]. Yen’s random 1 has six independent tasks and each task consists of approximately eight subtasks. There are eight PE types available. Yen’s random 2 consisted of eight task graphs, each of which has approximately eight tasks. There are 12 PE types available. Both graphs have zero communication delay and communication link cost.

Table 4 compares the performance of the proposed algorithm with those of MOGAC when each is applied to MOGAC’s very large random graphs[10]. The first graph contains eight tasks , each of which has about 63 tasks. There are eight PE types and five link types. MOGAC solved the minimization of price and power. The table shows that the proposed algorithm obtains the same or better results with much shorter time than the MOGAC algorithm. From the observation that the MOGAC algorithm requires significantly longer CPU time compared with Yen’s example, we think that it is not scalable enough to handle the SOC design.

<table>
<thead>
<tr>
<th>Example</th>
<th>No. of Tasks</th>
<th>Hou</th>
<th>COSYN</th>
<th>MOGAC</th>
<th>Proposed algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hou1&amp;2(u)</td>
<td>20</td>
<td>170</td>
<td>10206</td>
<td>-</td>
<td>170</td>
</tr>
<tr>
<td>Hou1&amp;3(u)</td>
<td>20</td>
<td>240</td>
<td>11550</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hou3&amp;4(u)</td>
<td>20</td>
<td>210</td>
<td>7135</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hou1&amp;2(c)</td>
<td>8</td>
<td>170</td>
<td>14.9</td>
<td>170</td>
<td>5.1</td>
</tr>
</tbody>
</table>

Table 1. Hou’s Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>No. of Tasks</th>
<th>SOS</th>
<th>COSYN</th>
<th>MOGAC</th>
<th>Proposed algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>P&amp;P1 &lt;4&gt;</td>
<td>4</td>
<td>7</td>
<td>28</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>P&amp;P1 &lt;7&gt;</td>
<td>4</td>
<td>5</td>
<td>37</td>
<td>5</td>
<td>0.2</td>
</tr>
<tr>
<td>P&amp;P2 &lt;8&gt;</td>
<td>9</td>
<td>7</td>
<td>4511</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>P&amp;P2 &lt;15&gt;</td>
<td>9</td>
<td>5</td>
<td>385012</td>
<td>5</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 2. Prakash and Parker’s Examples
In addition to the previous examples, we compare the performance of the proposed algorithm with an integer linear programming (ILP) formulation. The proposed algorithm is applied to random graphs, each of which has 11, 17, 23 and 31 nodes and a small real example, JPEG, which has 8 nodes. We assume that there are one S/W PE and many H/W PE types available. For instance, the example of JPEG has one software module and about 40 hardware modules since each node has about 4-5 hardware implementations. On the average, experimental results indicate that the cost overhead of the proposed algorithm over ILP is just about 5%.

## 5. CONCLUSION AND FUTURE WORK

In this paper, we propose a fast and simple heuristic for the cosynthesis problem targeting system-on-chip (SOC). The proposed algorithm covers from implementation selection and resource sharing problem in SOC design to PE selection problems in distributed heterogeneous embedded system design.

Benchmark experimentation shows the promising result that the proposed algorithm gives good performance for cosynthesis problem with huge design space unlike the existent cosynthesis research for the distributed heterogeneous embedded (DHE) system design. In the DHE design, the proposed algorithm could be used as an initial solution to probabilistic algorithms guaranteeing to obtain a better solution.

Since the design objectives in the real-life examples are more than what our implementation currently considered, the future research will be aimed to include missing design objectives to the framework. Furthermore, we will extend the framework to handle cyclic task graphs with loops, since we meet many situations of cyclic task graphs with loops in video processing and graphics applications.

## 6. REFERENCES


