AN EFFECTIVE SCHEDULING ALGORITHM TARGETING HIGH UTILIZATION OF RECONFIGURABLE DEVICES IN RECONFIGURABLE CO-SYNTHESIS SYSTEM ARCHITECTURE

Ji-Han Park[(1)], Sang-Hoon Kwak[(1)], Fahad Ali Mujahid[(1)], Jeong-A Lee[(2)] and Dong-Soo Har[(1)]

[(1)] Department of Information and Communications, Gwangju Institute of Science and Technology, Republic of Korea.
[(2)] Department of Computer Engineering, Chosun University, Republic of Korea.

hardon@gist.ac.kr

ABSTRACT

A heuristic algorithm that maps data-processing tasks onto either CPU or reconfigurable devices, while targeting high utilization of reconfigurable devices, is presented. The algorithm tries to minimize the number of reconfigurable devices with the help of run-time-reconfiguration and various reconfiguration overhead time into account, while the system configuration satisfies the resource constraints. Experimental results show this algorithm's relative effectiveness to conventional co-synthesis algorithm in embedded computing systems.

1. INTRODUCTION

Hardware-software co-synthesis creates an embedded computing system architecture to meet performance, power and cost goals [1]. This paper describes a new scheduling algorithm for reconfigurable co-synthesis system architecture in distributed, embedded computing systems. The algorithm synthesizes a system consisting of CPU, memory, bus, and many reconfigurable devices.

Conventional architecture uses a heterogeneous shared memory multiprocessor and application-specific hardware (e.g., ASICs, custom SoCs) as the target architecture, as shown in Figure 1. A CPU executes simple tasks, while computationally intensive tasks are performed on the application-specific hardware.

In contrast, we use reconfigurable devices (e.g., FPGA) instead of application-specific hardwares to execute many tasks, since they can be reconfigured during execution, even if they have reconfiguration time overhead.

This paper is organized as follows. Section 2 describes related works. Section 3 describes scheduling algorithm for reconfigurable co-synthesis system architecture using reconfigurable devices. Section 4 discusses experimental results and finally we conclude in section 5.

2. RELATED WORKS

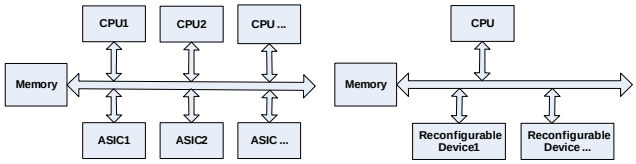

Figure 1 Conventional Architecture (Co-synthesis by ASICosyn) and Reconfigurable Architecture (Co-synthesis by Proposed Algorithm).

Design teams today must choose to implement logic either in application-specific hardware or reconfigurable device. Each of these offerings has distinct advantages: performance and density for application-specific hardware, vs. Turn-Around-Time and flexibility for reconfigurable devices [2]. If we use reconfigurable device instead of application-specific hardware to satisfy system constraints and to achieve a minimized cost of the system, we can get these advantages additionally.

Systems implemented with reconfigurable devices can make use of their reprogrammability in two ways: Compile-Time Reconfiguration (CTR) or Run-Time Reconfiguration (RTR) [5]. We will use a RTR method to reconfigure reconfigurable devices because we can execute many tasks on a few reconfigurable devices instead of multiple application-specific hardwares.

ASICosyn is a co-synthesis tool for embedded computing systems. The algorithm synthesizes a distributed multiprocessor architecture as shown in Figure 1 and allocates processes to the CPUs and ASICs in such a manner that the allocation and scheduling meet the system constraints, while the cost of the system is minimized. [3]. The tool heuristically finds an optimal solution considering conventional architecture only as shown in the Figure 1.

We use a task graph model [1] to describe each application. Application is partitioned into task graph, which is a directed acyclic graph [3]. In a task graph, nodes represent tasks that may have been moderated to

Pseudo Codes

```
// task[i] – store task information
// reconfigurable device (R.D.)
Read(system specification);   // Read a system specification from a input file
Initial_solution();                    // Make a initial solution
while(i < number of tasks )   // Move tasks from reconfigurable devices to CPU
{
        delete(R.D., task[i]);
        add(CPU, task[i]);
        if(meet = deadline) { // Check the deadline
           add(R.D., task[i]);
        }
        i++;
}
while(i < number of R.D.)     // Reduce the number of R.D. as much as possible
{
        if(possible(R.D.[i]) // Try to remove R.D. using blank space and RTR
           delete(R.D.[i]);
        i++;
}
```

Figure 2. Outline of Scheduling Algorithm.

large granularity; the directed edges represent data dependencies between tasks [4]. Data dependencies means that task placed in a child node can not run before a task placed in a parent node does. Each task in a task graph has information of execution time on CPU or on reconfigurable device, execution time deadline, and data dependencies between tasks.

3. SCHEDULING ALGORITHM

We propose a new scheduling algorithm for reconfigurable co-synthesis system architecture as shown in Figure 2.

Our algorithm's primary objective is to meet the rate constraint and the secondary objective is to minimize total implementation cost using reprogrammablility of reconfigurable device. The total implementation cost of the system is obtained as

$$\sum_{i \in CPUs} Cost(CPU_i) + \sum_{j \in HardwareDevices} Cost(HardwareDevices_j)$$

The outline of the algorithm consists of the following steps.

1. Read initial specification of a system and find an initial solution assuming that we have infinite number of reconfigurable devices.
2. Iteratively reduce reconfigurable device numbers by moving tasks to CPU.
3. Further reduce the number of reconfigurable devices by moving tasks assigned to reconfigurable devices from low utilized reconfigurable device to other devices considering run-time-reconfiguration and data dependencies

In step 1, read initial specification of a system from data file, which is based on Monet [6] architectural exploration

Table 1. Experiment Example.

ASIC/ Task Name	Fast Speed	Fast Area (Cost)	Slow Speed	Slow Area (Cost)
A1	3	30	5	20
B1	3	30	4	20
CPU/ Task Name	Speed	**Reconfigurable Device/ Task Name**	Speed	Cost
A1	9	A1	7	30
B1	15	B1	9	30

system and initial solution is constructed by assigning each task in the task graphs to many reconfigurable devices.

In step 2, we try to find candidate tasks which have small difference between execution time on CPU and reconfigurable device including reconfigurable time. We sort tasks according to ascending order of difference and then assign a task to CPU from reconfigurable device. It implies that implementing the task on reconfigurable device can not achieve much speed-up, as compared to implementing the task on CPU. This procedure iteratively continues when all the possible tasks move to the CPU while satisfying execution time deadline.

In step 3, we reschedule tasks which are assigned to reconfigurable devices. We use a RTR; If one task is being executed on a reconfigurable device, we can execute other tasks on other reconfigurable devices. With the task completed, reconfigurable device is ready to reconfigure other tasks. Before the end of execution of the task placed in parent node, the tasks placed on a child node can reconfigure themselves on reconfigurable devices and start to run after the task on parent node.

A iteration of the reconfigurable device number reduction procedure tries to reduce the number of the reconfigurable devices by eliminating lightly loaded reconfigurable device after moving the tasks on those reconfigurable devices to other reconfigurable devices.

By the proposed algorithm, the tasks on reconfigurable device of low utilization are moved to other possible reconfigurable devices, according to RTR and data dependencies. Thereby, the system cost will be decreased.

4. EXPERIMENT RESULTS

Before carrying out any experiment, we considered impact factors such as cost, execution time and reconfiguration time of tasks that have been assigned to reconfigurable devices.

We calculate our reconfigurable device's performance based on the ASIC performance. In real design, we might

either choose a high speed implementation with higher cost (larger area), or choose a low speed implementation with lower cost (smaller area) [1].

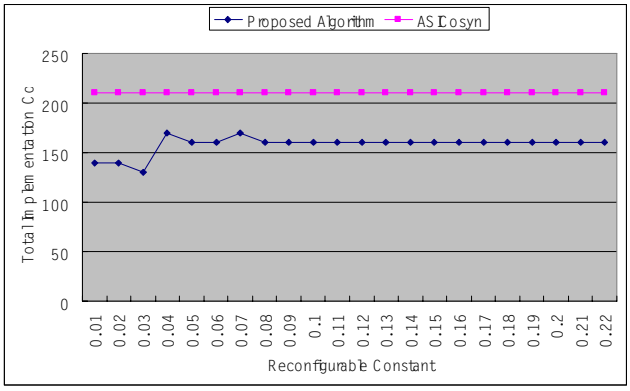

Figure 2. 'EX1' Example.

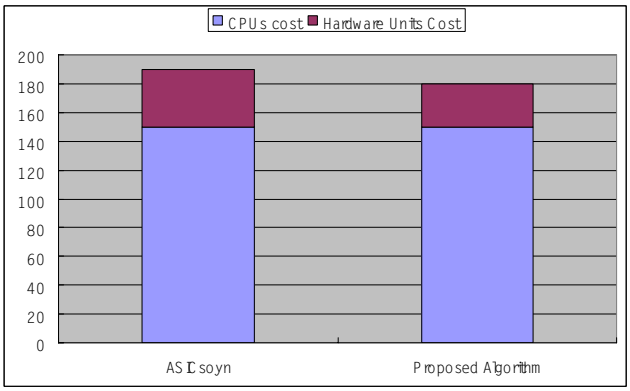

Figure 3. 'EX2' Example.

We assume that the cost of reconfigurable device is the same as that of high speed ASIC and the execution time of our reconfigurable device takes an average of CPU execution time and the low speed ASIC execution time as shown in Table 1. In general, reconfigurable device runs faster than CPU but slower than ASIC [2].

The reconfiguration time can be calculated by an equation given by

Reconfiguration Time = *area* x Δ

where

Δ = reconfigurable constant
(time unit/area unit)

During the experiments, we changed the reconfigurable constant while satisfying execution time deadline of the system. Figure 2, 3, 4 show the statistics obtained from three different experiments. The data indicated by 'EX1,' 'EX2,' and 'EX3' were obtained by Monet tool of Mentor Graphics. The maximum total implementation cost of the system is smaller than that of the ASICosyn except 'EX3' case, even if it includes the reconfiguration time of the reconfigurable device.

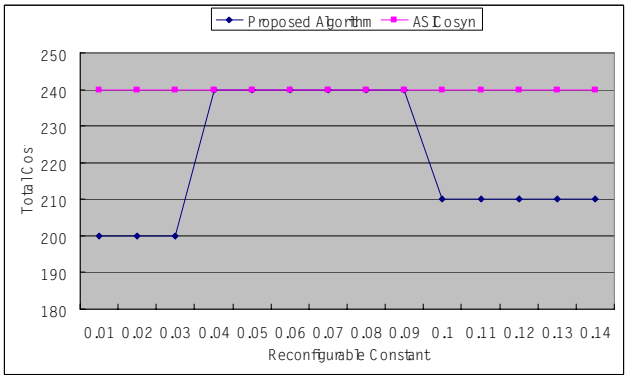

Figure 4. 'EX3' Example.

In case of 'EX1' case, ASICosyn synthesized two CPUs and single ASIC whereas our algorithm synthesized single CPU and two reconfigurable devices. Our algorithm used more hardware units but they executed many tasks taking advantage of run-time reconfiguration. Thus, our algorithm's total implementation cost is smaller than ASICosyn's one as shown in Figure 2.

In case of 'EX2' case, the total implementation cost was fixed. ASICosyn synthesized single CPU and two ASICs whereas our algorithm synthesized single CPU and single reconfigurable device. These results are similar but our algorithm found other critical path using single reconfigurable device instead of two ASICs. Thus, our algorithm's total implementation cost is smaller than ASICosyn's one as seen in Figure 3.

In case of 'EX3' case, it didn't have enough empty time slots between task's executions until reconfigurable constant was smaller than '0.09'. Because of this, our algorithm's total implementation cost was same as ASICosyn's one. However the reconfiguration time of reconfigurable devices became more longer, the algorithm used the reconfiguration time to reconfigure reconfigurable devices. Thus, after '0.09' mark, the total implementation cost of the system synthesized by the proposed method is smaller than that of ASICosyn-based one as shown in Figure 4.

5. CONCLUSIONS

In this paper we described a new scheduling algorithm for reconfigurable co-synthesis architecture. The algorithm synthesizes a system consisting of CPU, memory, bus and many reconfigurable devices. We use a run-time reconfiguration method to reconfigure reconfigurable devices, since we can execute many tasks on a fewer number of reconfigurable devices instead of multiple application-specific hardwares. The scheduling algorithm

increases the utilization increase of reconfigurable devices as much as possible.

Our experimental results demonstrate that the total cost of the system can be made smaller than that of the conventional architecture even if it includes the reconfiguration time of the reconfigurable device.

Acknowledgments

This work has been supported in part by Chosun University research funds, 2003, in part by the Center for Distributed Sensor Network (CDSN) at GIST, in part by IC Design Education Center (IDEC), and in part by the GIST Technology Initiative (GTI).

REFERENCES

[1] Wayne Wolf and Jorgen, Staunstrup "Hardware/Software co-design: Principles and Practice". Kluwer Academic Publishers. 1997.

[2] Paul S. Zuchowski, Christopher B. Reynolds, Richard J. Grupp, Shelly G. Davis, Brendan Cremen, Bill Troxel "A Hybird ASIC and FPGA Architecture". Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on 10-14 Nov. 2002 Page(s):187 – 194, 2002.

[3] Yuan Xie and Wayne Wolf, "Co-synthesis with custom ASICs". Proc. Of ASP_DAC 2000, pp. 129-135, 2000.

[4] Yuan Xie and Wayne Wolf, "ASICosyn: co-synthesis of conditional task graphs with custom ASICs" Proc. Of ASIC 2001, pp. 130-135, 2001.

[5] B.L. Hutchings and M. J. Wirthlin, "Implementation approaches for reconfigurable logic applications" in Field-Programmable Logic and Applications (FPL'1995) (W. Moore and W. Luk, eds.), (Oxford, England), pp. 419-428, Springer-Verlag, Berlin, Aug. 1995.

[6] Mentor Graphics Company, "Monet reference manual".