Hardware/Software Partitioning of Synchronous Dataflow Graphs in the ACS domain of Ptolemy

Literature Survey
March 23, 1999

Gayathri Manikutty
Heather Hanson

Table of Contents

Abstract

We present a survey of research publications in the areas of hardware/software codesign and adaptive computing, and discuss plans for implementing a partitioning tool into a design environment. The partitioning tool is based upon a binary partitioning algorithm known as Global Criticality/Local Phase. This algorithm considers multiple resource constraints (for example, physical layout area and time to execute) and iteratively maps sections of the system into hardware and software implementations until it reaches a solution that satisfies the design constraints.

Introduction and Context of Work

Hardware/Software Codesign

Embedded processors for real-time systems must meet stringent requirements in terms of performance and power dissipation while keeping the product cost low and development cycle short. One successful method of designing embedded systems is exploiting the synergism of hardware and software through their concurrent design. Franke and Purvis [1] define co-design as “the system design process that combines the hardware and software perspectives from the earliest stages to exploit flexibility and efficient allocation of function.” An important phase in co-design is hardware/software partitioning, which maps each node of the application into hardware or software and selects an appropriate implementation. The difficulty is to choose from the multiplicity of mapping and implementation alternatives available such that the overall design is optimized, and to do so quickly.

Adaptive Computing

One research area well suited for hardware/software codesign is adaptive computing. Adaptive computing, alternately known as configurable or reconfigurable computing, combines an FPGA with a microprocessor, either a general-purpose processor or DSP. The system’s hardware adapts to different tasks by re-programming logic circuits as needed. It is a hardware equivalent to executing one program, then switching to another [2].

Adaptive computing applications are designed for a specific configuration type: run-time or compile-time configuration. A design with run-time configuration executes in stages, re-using the logic to execute parts of the application as needed. While this minimizes the area required, it is difficult to schedule and the system speed is slower due to reprogramming time. A compile-time configuration design will program the FPGA for a specific task without any reprogramming during execution [3]. For example, a universal telephone would detect the protocol of an incoming call and program the FPGA to process the data properly, then switch to another protocol for a later call. In our project, we will focus exclusively on compile-time configurations.

Ptolemy and the ACS domain

Ptolemy is an object-oriented framework developed by the University of California at Berkeley for simulating and prototyping heterogeneous systems [4]. It is organized into domains of computation models and systems. A recent development is the new Adaptive Computing Systems (ACS) domain that supports multiple output “targets” (implementation types) for each task or subtask in a system [5]. Applications modeled in this domain include signal processing and communication systems, such as modems, and are typically implemented in a mixture of reconfigurable hardware, such as FPGA's, and software.

Design objective

Our contribution is incorporating a mapping tool into the ACS domain; the tool will assist designers by performing near-optimal partitioning into hardware and software modules. We base the mapping in the Global Criticality/Local Phase (GCLP) algorithm [6]. The GCLP algorithm was prototyped within the Ptolemy software environment to provide design assistance in partitioning but was not released.

Summary of previous work

Partitioning a design into hardware and software sections has been the focus of ongoing research [7,8,9]. Here, we present a brief overview of this partitioning problem and a heuristic to solve it.

The Binary Partitioning problem

The SDF domain in Ptolemy translates the task level description of an application into a Directed Acyclic Graph (DAG) where each node represents a computation and the arcs represent the data and control precedence between the nodes. The problem is to decide the mapping for each node of the DAG--either into hardware or into software--and arrive at a schedule (the start times for each node) subject to latency constraints (the upper bound on the total application execution time) and resource constraints (which includes availability of hardware and software resources, namely program and data memory) such that the total area of nodes mapped to hardware is minimum. The area and latency estimates for hardware and software mappings of all nodes, as well as the communication and interfaces, must be known for an effective partition.

Solving this partitioning problem is computationally intensive. Integer Linear Programming could be used to solve constrained optimization problems such as the binary partitioning problem but exact solutions are intractable. A heuristic known as the Global Criticality/Local Phase (GCLP) algorithm--with computational complexity of $O(N^2)$--has been proposed to solve these problems [6].

GCLP algorithm

The GCLP algorithm first calculates the criticality of nodes like a list scheduling algorithm [10]. However, unlike list scheduling, which either optimizes for the finish time or for the area of the node by serially traversing all the nodes, the GCLP algorithm adaptively selects the objective function to be minimized at each step based on both time and area, in the form of global criticality and local phase values.

At each time step, the schedule for all the mapped nodes is known. Using the latency constraint, the remaining time to complete execution of the as-yet unmapped nodes is computed. If it is not possible to map all the unmapped nodes into software and complete execution by the latency bound, some of the unmapped nodes are moved to the hardware and the finish time is recomputed. The fraction of unmapped nodes moved from software to hardware at a given step gives the Global Criticality (GC) at that step. A high GC implies that time is the critical resource.

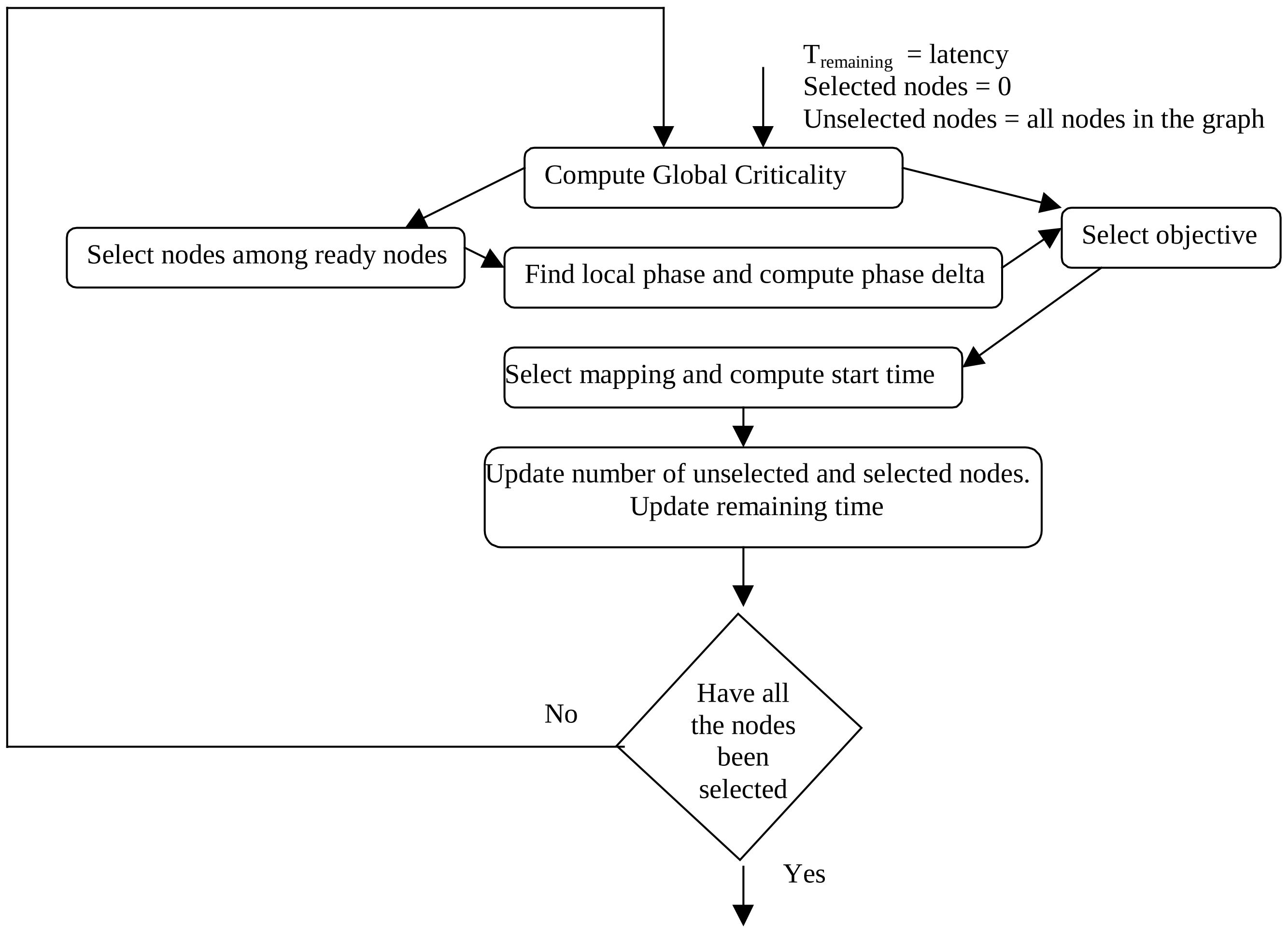
$T_{remaining}$ = latency
Selected nodes = 0
Unselected nodes = all nodes in the graph
Compute Global Criticality
Select nodes among ready nodes
Find local phase and compute phase delta
Select objective
Select mapping and compute start time
Update number of unselected and selected nodes.
Update remaining time
Have all the nodes been selected
No
Yes

Figure 1: GCLP flow chart [2]

The local phase of a node is used to in conjunction with the GC to determine appropriate mapping choices. Local properties of the nodes are classified into three types: extremity, repeller and normal. An extremity node is one that would use a large amount of a given resource—area or time—in a given hardware or software implementation and thus should be mapped to the other implementation. A repeller node is classified based on its structure: bit manipulations are more suited for hardware and hence a node performing many bit manipulations would be a software repeller. A node that falls into neither of these groups is a normal node.

Figure 1 shows a flow chart for the iterative GCLP algorithm. As Dr. Kalavade writes in [8]: a node is selected for mapping from a set of ready nodes (unmapped nodes whose predecessors have been mapped and scheduled). Using the global criticality and local phase delta (which quantifies the local mapping preference of the node under consideration), the mapping objective is selected and using this objective, the mapping is determined. The algorithm repeats until all nodes are mapped.

Design Environments

Several research groups have built design environments for specifically for adaptive computing. We will present three such environments to illustrate the current state of this research area.

PAM-Blox

PAM-Blox is a design environment for high-performance FPGA designs. The basic blocks within the PAM (programmable adaptive memories) are low-level circuit elements, which the designer combines to form functional units. Building a design in the PAM Blox environment is analogous to programming in assembly language [11]. The PAM environment is one of the earlier adaptive computing environments, and has been used in several applications.

DEFACTO

DEFACTO is another design environment, developed at the University of Southern California. Its high-level tool combines a CAD environment with compiler technology suited for adaptive computing. The design input is an abstract specification in either C or MATLAB; the user adds application-specific annotations like timing information. The target architecture consists of general-purpose processor (GPP) and multiple configurable computing units (CCUs) such as FPGAs. The partitioning algorithm iteratively divides the design into the microprocessor (software) and CCU (hardware) sections. DEFACTO produces an HDL representation for the CCUs and C code for the GPP. The output is architecture independent and re-targetable[12].

PeaCE

PeaCE (Ptolemy extension as Codesign Environment) is a recent development led by Soonhoi Ha of Seoul National University, a former member of the Ptolemy development team. Although not specifically dedicated to adaptive computing, a test application of a DSP and FPGA suggests that it is suited for this area. The PeaCE design environment differs from standard Ptolemy in that it uses a subset of Ptolemy's computational models and has a different internal data abstraction: an extension of the SDF model is used as for functional units, with finite state machines as control units. The codesign methodology in PeaCE begins with design space exploration and moves on to extended partitioning, then to cosimulation and completes with hardware/software synthesis. The output files are in C and VHDL[13].

Analysis of current research

Though created for slightly different goals, each of these design environments was developed to manage information and design resources, with software extensibility and design re-use in mind. Some environments, like PAM-Blox, are intended for users who have chosen system implementations prior to using the design environment and will focus on maximizing performance for a given partitioning. Others, such as PeaCE, use codesign as the primary design method, optimizing the design on a system level before proceeding to optimize individual tasks.

Codesign is becoming an increasingly important design methodology as performance goals increase while time-to-market decreases, and it is an effective methodology for designing adaptive computing systems’ distinct processor and FPGA components. Ptolemy’s ACS domain could benefit from a hardware/software partitioning tool to assist designers explore the design space of hardware and software implementations. The GCLP algorithm is an appropriate choice for a partitioning tool since it handles the binary decision of hardware or software targets available in the ACS domain.

Plans for design implementation

We will integrate the GCLP algorithm developed by Dr. Asawaree Kalavade into a partitioning tool for Ptolemy’s ACS domain. We have installed a local version of Ptolemy, and will proceed by integrating a standalone GCLP program into the ACS domain and recompiling Ptolemy. We are investigating the scheduler and target mechanisms within Ptolemy [14]. We are focusing on the Synchronous Data Flow (SDF) formal model since SDF graphs can be statically scheduled and are synthesizable in Ptolemy.

We anticipate the next release of the ACS domain will include FPGA targets; we will begin working on the current version and migrate to the next version when it is available. With the new features installed, we will use the GCLP algorithm to partition a simple application (as yet unspecified) and compare the results with an optimal partitioning algorithm.

References

1. D. W. Franke, M. K. Purvis, “Hardware/Software Codesign: A Perspective,” *Proc. Of 13th Intl. Conference on Software Engineering*, pp. 334 –352, Austin, Texas, USA, May 1991.

2. J. Villasenor, W. H. Mangione-Smith, “Configurable Computing,” *Scientific American*, pp. 66-71, June 1997.

3. L. Hutchings, M. J. Wirthlin, “Implementation Approaches for Reconfigurable Logic Applications,” In W. Moore and W. Luk, eds. *Field-Programmable Logic and Applications,* pp. 419-428. Springer, 1995.

4. J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *Int. J. Computer Simulation*, vol. 4, pp. 155-182, April 1994.

5. E. A. Lee, et al., University of California at Berkeley, *The Almagest*, Volumes 1-3, Regents of the University of California, 1995.

6. A. Kalavade, E. A. Lee, "A Global Criticality/Local Phase driven Algorithm for the Constrained Hardware/Software Partitioning Problem,” *Proc. of Codes/CASHE'94, Third Intl. Workshop on Hardware/Software Codesign*, pp. 42-48, Sept. 22-24, 1994.

7. A. Kalavade, E. A. Lee, "A Hardware/Software Codesign Methodology for DSP applications,” *IEEE Design and Test of Computers*, pp. 16-28, Sept. 1993.

8. A. Kalavade, E. A. Lee, "The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling, and Implementation-bin Selection," *Journal of Design Automation for Embedded Systems*, pp. 126-163, vol. 2, no. 2, March 1997.

9. A. Kalavade, “System Level Codesign of Mixed Hardware-Software Systems,” Technical Report UCB/ERL 95/88, Ph.D. Dissertation, Dept. of EECS, University of California, Berkeley, September, 1995.

10. T. C. Hu, “Parallel Sequencing and Assembly Line Problems,” *Operations Research* 9(6) , pp. 841-848, Nov. 1961.

11. O. Mencer, M. Morf, M. J. Flynn, “PAM-Blox: High Performance FPGA Design for Adaptive Computing,” *IEEE Symposium on FPGAs for Custom Computing Machines* (FCCM), Napa Valley, 1998.

12. K. Bondalapati, P. Diniz, P. Duncan, J. Granacki, M. Hall, R. Jain, H. Ziegler, "DEFACTO: A Design Environment for Adaptive Computing Technology," To appear in *Proceedings of the 6th Reconfigurable Architectures Workshop* (RAW'99), Springer-Verlag, 1999.

13. “PeaCE (Ptolemy as Codesign Environment),” http://mirage.snu.ac.kr/research/peace/PeaCE.html, June 11, 1998.

14. J. L. Pino, S. S. Bhattacharyya and E. A. Lee, “A Hierarchical Multiprocessor Scheduling System for DSP Applications," *Proc. IEEE Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, October 29 - November 1, 1995.