Rethinking Automated Synthesis of MPSoC Architectures

Brett H. Meyer and Donald E. Thomas
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{bhm, thomas}@ece.cmu.edu

ABSTRACT

Emerging heterogeneous multiprocessors will have custom memory and bus architectures that must balance resource sharing and system partitioning to meet cost constraints. We propose an augmented simulated annealing synthesis tool that uses system performance and layout evaluation to drive simultaneous data mapping, memory allocation and bus synthesis. A detailed look at the resulting automated design process reveals an approach that, contrary to prior approaches, optimizes bus topology first rather than last, providing design insight for the development of future tools.

1. INTRODUCTION

The future of most embedded applications lies in single-chip low-power heterogeneous multiprocessors. These systems-on-chips will consist of tens of individually programmable processing elements (PEs). These systems will be heavily customized to obtain the best-tuned architecture for the set of applications specified. Given that both the underlying technology and the size of systems being designed are adding complexity to the design process, new synthesis techniques are needed to address design productivity. We present and analyze the foundation of a synthesis tool aimed at the architectural design of single-chip multiprocessor systems optimized for low latency and manufacturing cost.

This work was supported by the National Science Foundation through Grant CNS-0509193. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

Prior work has suggested a variety of approaches to exploring this design space. Many of the prior tools, however, impose limitations on the design space exploration by dividing synthesis into multiple exploration phases or iteration loops, exploring a different axis of the space in each. This approach makes the assumption that decisions made in earlier phases won't prevent exploration from finding globally optimal solutions.

We present and analyze a novel synthesis tool that searches for optimal embedded multiprocessor systems without the above limitation. Given a set of concurrent tasks and their processor assignments and a library of components, our augmented simulated annealing approach simultaneous performs data mapping, memory allocation, and bus architecture synthesis, all within a single iteration loop.

This unconstrained, single-phase approach allows our tool to consider a wider range of designs than the prior work. Our analysis reveals that as a result, our tool consistently approaches design in an unconventional fashion, providing insight into best practices for embedded multiprocessor design and for the development of future exploration tools.

2. PRIOR WORK

A large body of work has already been done to optimize both system memory organization and system interconnect. [1][2][3] all perform bus topology exploration, but require a description of the memories in the system and the data assigned to them (derived from memory allocation and data mapping, respectively) as inputs. [4] performs bus topology exploration after deriving a data mapping and memory allocation in a separate phase. [5] also explores bus topology after data mapping and memory allocation, but if constraints can't be satisfied with a mapping and allocation, the tool derives a new one before returning to bus topology exploration. Unlike our approach, these approaches all conduct bus topology exploration in isolation from data mapping and memory allocation. While [6][7] propose simultaneous mapping, allocation, and topology exploration, unlike our tool they do not floorplan to jointly optimize systems for performance and cost.

Many of the above approaches perform data mapping and memory allocation separately from bus topology exploration, or require mapping and allocation as input parameters. This makes the assumption that data mapping and memory allocation can be decided independently of bus topology without limiting design optimality. Our tool makes no such assumption, freeing it to best optimizing performance and cost for the target system.

3. SYNTHESIZING MPSOC ARCHITECTURES

This section defines the trade-offs in bus and memory system organization at this level and describes our synthesis formulation.

3.1 Design Space

There are two basic memory system organizations for handling the concurrency inherent in multiprocessor systems, the extremes of which are partitioning a system into a collection of subsystems to reduce contention, and globally sharing all resources to enable more cost effective implementation. Optimal bus and memory system design strikes a balance between resource sharing and system partitioning, driven by the conflicting goals of sharing to reduce cost (e.g., area) and partitioning to improve performance (e.g., bus utilization, execution time). Our synthesis approach meets this need by optimally distributing, sharing and segregating memory and interconnect resources to balance performance and cost in concurrent embedded systems.

3.2 Formulation

We have developed a synthesis tool that augments simulated annealing techniques with closed-form decision-making that prunes the design space and legalizes initially illegal system modifications. This formulation is intended to lay the groundwork for the development of more sophisticated and direct solutions by identifying the critical issues in the optimal design of embedded multiprocessor memory architectures.

Our simulated annealing approach iteratively permutes and evaluates the system in search of the global optimal design. Each exploration iteration (1) selects and performs a move, (2) prunes the system and redistributes data, and (3) evaluates the objective function and probabilistically accepts or rejects the permutation.

There are three basic move types, two of which are further broken into sub-types. *Data mapping* randomly moves a data array from one memory to another, enlarging the destination memory as necessary. *Bus* moves randomly change the bus topology by: *connecting* (*disconnecting*) a processor or memory to (from) a bus in the system or, *splitting* a highly utilized bus into two buses. *Memory* moves randomly change memory allocation by: *absorbing* a small, underutilized memory (and its data) into another larger memory, or *splitting* a large underutilized memory (and its data) into two smaller, better-utilized memories. Unlike the prior work, we allow *all* system modification steps in a single exploration phase.

Any move that modifies the system topology is followed by *pruning*. Pruning adjusts the size of memories that are too large for the data they contain and removes unutilized memories and buses, eliminating unnecessary cost. Any move that changes which bus(es) a memory (or processor) is connected to is followed by *redistribution*. Redistribution applies a simple heuristic to reassign data to memories, legalizing any illegal data mappings, since for example, when a processor is arbitrarily moved it may not be able to access all of its data.

3.3 Modeling System Cost

System cost, the objective function minimized by our annealing approach, is evaluated after each move is completed (possibly including pruning and redistribution) and is used to probabilistically determine if the move in question should be accepted or rejected. System cost is the weighted combination of total execution cost, or *latency,* and the system's *physical cost*:

$$\alpha \cdot latency + (1-\alpha) \cdot physical\ cost$$

where α determines the relative importance of latency and physical cost. The goal of our synthesis tool is to minimize *system cost* for a given α. Unlike the prior work, cost and performance are evaluated after each system modification.

Latency is evaluated by determining the total system execution time. The total system execution time is calculated by performing trace-based discrete event simulation (DES) on the dependency graph generated by combining memory access traces for each task in the system.

Physical cost is itself a weighted combination of system layout *area*, total bus *wire length*, and a *penalty* factor that increases the cost of systems with aspect ratios greater than one:

$$\beta \cdot area + (1-\beta) \cdot wire\ length + penalty$$

where β has been chosen so that system *area* and total bus *wire length* contribute equally to *physical cost*. *Area*, *wire length* and the *aspect ratio* are all determined with floorplanning. *Area* is the bounding box of the design and *wire length* is the sum of half-perimeter wire lengths of all buses. Floorplanning is performed by annealing a slicing tree representation of the system.

4. EXPERIMENTS AND RESULTS

We conducted a set of experiments to evaluate our technique and gain insight into effective embedded multiprocessor design practices. Using an example workload and fixed task-processor assignments, we explored the design space in search of pareto-optimal systems composed of elements from a component library. We performed design space exploration for a variety of α values and then examined the exploration process for a subset of the resulting design points.

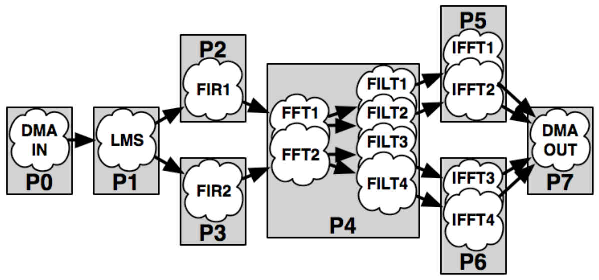

Figure 1: Software pipeline and task-processor mapping.

4.1 Workload

Our experimental workload is a DSP software pipeline executing on a concurrent hardware pipeline, illustrated in Figure 1. Data is introduced to the system by a hardware DMA engine (P0), and fed to a least-mean-squared (LMS) adaptive filter (on processor P1) for noise cancellation. Two different FIR filters are then separately applied (P2 and P3), followed by an FFT transform and more filtering (P4). IFFT is then applied to each filtered stream (P5 and P6) before the output is collected and sent off-chip by another hardware DMA engine (P7). Processors that execute more than one task execute each task to completion before starting the next. Memory access traces were generated for each task from optimized assembly kernels.

4.2 Library Components

The library of components used in our experiment consists of a small collection of processors, memories, and interconnect modules in a 90 nm process.

The basic processing element is an ARM7; all processors except P0 and P7, the DMA engines, are ARM7s. The DMA engines are modeled as small buffers for area consumption purposes. Processing elements are allowed multiple bus connections, but with a fixed area penalty applied for each connection after the first. All memories are SRAMs with a single read/write port, and are allowed only a single bus connection. The library contains SRAMs varying in size from 256B to 32kB by powers of two. There is only one bus in the component library, a single-transaction bus with no data buffering.

4.3 Design Space Exploration

To find a set of latency-cost pareto-optimal architectures, we conducted nine experiments over which we manipulated α, and therefore the relative importance of execution latency and physical cost. We selected α values over a range of 9e-4 to 1.1e-5, covering latency-cost ratios of 9:1 to 1:9. The results are summarized in Figure 2, which plots latency vs. cost for each resulting design.

Nearly all of the resulting design points are pareto-optimal, indicated with black diamonds. The non-pareto points are indicated with red squares. The designs have area ranging from 1.81-2.5 mm^2, and all have an aspect ratio under 1.25. The low cost designs experience heavy bus utilization, up to 94%, while the high performance designs experience much lower bus utilization, as low as 66%.

Despite the obvious differences between designs implementing anywhere from two to five buses, the systems have much in common. This stems from the common goal of balancing performance and cost in some proportion. Each system takes advantage of the structure of the given workload in some way, the most obvious of which is spatial partitioning into three macro pipeline stages: input (DMA, LMS), processing (FIR, FFT, FILT), and output (IFFT, DMA).

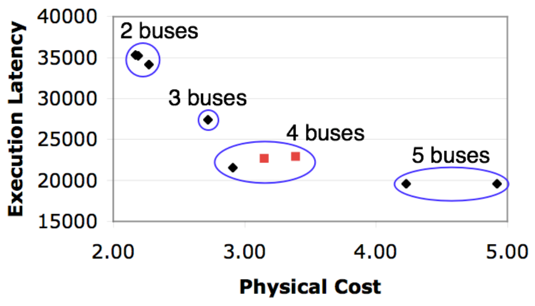

Figure 2: Latency vs. physical system cost for variable α.

When only two buses are available (low α), the input and output stages share a bus, while the processing stage makes use of the second. In systems with more resources (up to four buses), the partitioning is more obvious; e.g., input and output stages have their own bus, while the processing stage shares two buses. When five buses are available (high α), the partitioning is less strict in order to minimize bus utilization (therefore maximizing performance), but remains present.

4.4 Move Acceptance Trends

To gain insight into embedded multiprocessor design, we will now look more closely at three specific design points from the previous section: cost-centric design when $\alpha = 1.1e\text{-}5$, balanced cost-performance design when $\alpha = 1e\text{-}4$, and performance-centric design when $\alpha = 9\text{e-}4$. If design exploration proceeded in a consistent fashion for these three very different design points, we may gain insight into principles of effective design for embedded multiprocessors. Figure 3 illustrates the change in move acceptance for each of these three different design points.

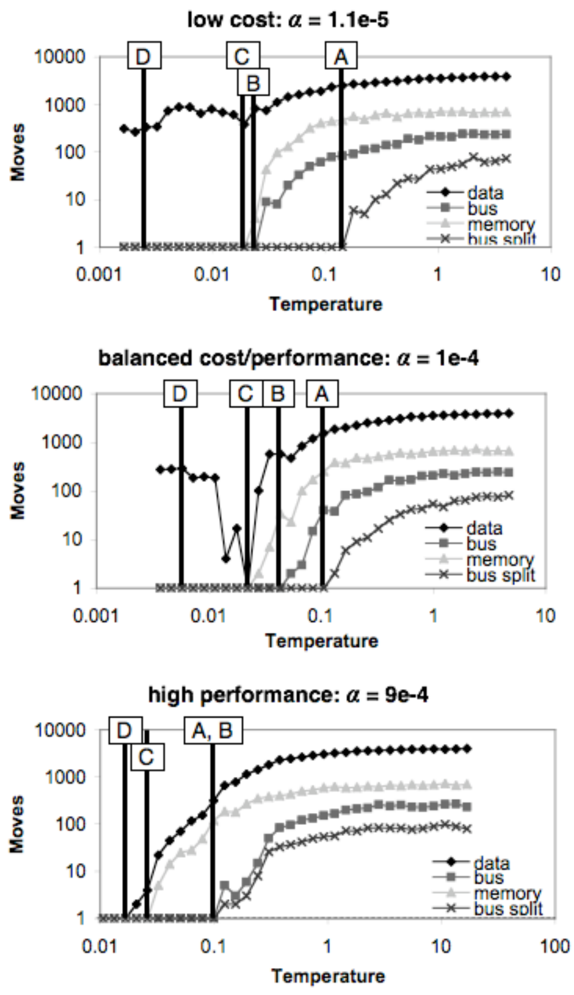

Figure 3: Accepted moves per iteration for three design points.

The lettered markers in Figure 3 indicate specific events in the course of design space exploration. For all three of the graphs, A marks the point when bus split moves are no longer accepted, B, when bus moves are no longer accepted, C, when memory moves are no longer accepted, and D, when the system is beginning to "freeze", i.e., accepted moves no longer appreciably change the system cost.

It is immediately clear from Figure 3 that exploration at the three design points exhibit the same basic progression: bus topology is fixed before memory allocation, which is fixed before data mapping. First, bus moves are no longer accepted, setting the number of buses in the system and the assignment of nodes to buses. At this point, the number of memories per bus can still change through local reallocation. Next, memory moves are no longer accepted, fixing the number of memories per bus, though data mapping moves and pruning may still change the size of the existing memories.

With only slight variations between them, this occurs for all three design points, in spite of the very different optimization targets (low cost, top; balanced cost and performance, middle; and high performance, bottom). For the low-cost and balanced designs, bus split moves stop being accepted before bus moves in general are no longer accepted. This means that the number of buses in the system is fixed before the location of nodes in the system, unlike what occurs in the high-performance design.

5. DISCUSSION

The results in section 4.3 illustrate our tool's ability to incrementally trade performance and cost, balancing system partitioning and resource sharing. Our investigation in section 4.4 makes it clear that this balancing and sharing proceeded in a consistent fashion for very different design points: first the bus topology was fixed, then the memory allocation, and finally the data mapping. This approach, however, is contrary to the literature. For most bus synthesis approaches, allocation and mapping are fixed before bus topology exploration and not revisited later.

The historical synthesis approach makes a critical assumption: data mapping and memory allocation can be selected independently of bus topology without any impact on design optimality. This is the same as assuming that optimal allocation and mapping can be performed without knowledge of the cost of system organization.

If this were the case for our workload and component library, then we would expect to see the acceptance of all basic move types converge to zero at approximately the same time, implying parallel optimization. Instead, for three very different design points, at least partially serial optimization occurs. Further, bus topology is finalized first and within the context of allocation and mapping, rather than last and in isolation. This implies a co-dependence of allocation and mapping with bus topology, rather than independence.

In the context of our application and design space, there are several reasons not to fix memory allocation before bus topology. Physical cost is a function both of wire length and area. Selecting a particular memory allocation early assumes that a different allocation couldn't improve the system significantly. For example, [4] breaks the memory space into pieces based on usage alone: memory shared by two processors is allocated to a single node, and each processor gets a single local memory. This assumes that breaking one shared memory into two or combining multiple scratch-pad memories into one is inherently a worse solution, where the realities of floorplanning may prove otherwise.

Our approach makes no assumptions about the order in which parts of the system should be optimized. As a result, in the discussed design points the average system cost was optimized up to an additional 24% after the bus topology was fixed. This provides the insight that optimization order matters, and optimizing the bus topology in isolation may not be the best approach. Rather, because of the implicit codependence of optimal allocation and mapping with topology, approaches that are sensitive to this are more likely to find truly globally optimal solutions.

6. CONCLUSIONS

We presented a synthesis tool targeting multiprocessor embedded system memory and bus architecture. Our tool uses an augmented simulated annealing approach to simultaneously explore the design space of data mapping, memory allocation, and bus-based interconnect, given a target application, task-processor mapping and component library, and is driven by the joint optimization of latency and cost.

Our tool effectively balance global resource sharing and system partitioning. Further, our results suggest a design approach contrary to the literature: resolve the bus topology first, followed by memory allocation and data mapping. This insight into the design process for such systems will help shape future design automation approaches.

7. REFERENCES

[1] J. Guo, *et al*. Energy/area/delay Trade-offs in The Physical Design of On-chip Segmented Bus Architecture. *SLIP 2006.*
[2] K. Lahiri, A. Raghunathan, S. Dey. Efficient exploration of SoC communication architecture design space. *ICCAD 2000.*
[3] H Jincao, *et al.* System-level point-to-point communication synthesis using floorplanning information. *ASP-DAC 2002.*
[4] S. Pasricha, *et al.* Floorplan-Aware Automated Synthesis of Bus-based Communication Architectures. *DAC 2005.*
[5] N. Thepayasuwan, A Doboli. Layout Conscious Approach and Bus Architecture Synthesis for Hardware/Software Codesign of Systems on Chip Optimized for Speed. *IEEE Trans. on VLSI Sys.*, 13(5), May 2005.
[6] S. Pasricha, N. Dutt. COSMECA: application specific co-synthesis of memory and communication architectures for MPSoC. *DATE 2006.*
[7] Sungchan Kim, *et al*. Efficient Exploration of On-Chip Bus Architectures and Memory Allocation. *CODES+ISSS 2004.*