Simultaneous Synthesis of Buses, Data Mapping and Memory Allocation for MPSoC

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

ABSTRACT

Heterogeneous multiprocessors are emerging as the dominant implementation approach to embedded multiprocessor systems. In addition to having processing elements suited to the target applications, these systems will also have custom memory and bus architectures. Because of performance and cost constraints, these systems must be carefully designed to balance system partitioning and resource sharing. The sheer size of the design space requires that tools be able to do this balancing. We have developed an augmented simulated annealing synthesis tool that uses system performance and layout evaluation to drive simultaneous data mapping, memory allocation and bus synthesis. Performing these optimizations at the same time, our tool is able to explore a larger design space and take advantage of cost-saving resource sharing unavailable to previous approaches that allocate memories before synthesizing buses. This results in 20% cost reduction for high-performance designs as well as 27% for low-cost designs in comparison with an approach that performs memory allocation and data mapping separately from bus synthesis.

Categories and Subject Descriptors

J.6.1 [**Computer Applications**]: Computer-aided Engineering—*Computer-aided design (CAD)*

General Terms

Design, Performance

Keywords

Embedded multiprocessor systems-on-chip, bus architecture synthesis, memory allocation, data mapping, partitioning, sharing

1. INTRODUCTION

The future of most embedded applications lies in single-chip heterogeneous multiprocessors. These systems-on-chips will consist of tens of individually programmable processing elements (PEs) and will be customized to obtain the best trade-off of the designers' requirements. Given that design complexity is increasing because of both the underlying technology and the size of systems being designed, new synthesis techniques are needed to address design productivity. We present and demonstrate a synthesis tool aimed at the architectural design of single-chip multiprocessor systems optimized for low latency and manufacturing cost.

Prior work has suggested a variety of approaches for exploring this design space. A representative tool takes as input a particular hardware/software task decomposition, determines an appropriate memory allocation and data mapping, and then performs bus synthesis. In these approaches, synthesis is often divided into multiple exploration phases, such that memory allocation is determined independently of and prior to bus synthesis, and bus topology is determined assuming a fixed memory allocation. In the case of memory allocation and bus synthesis, however, dividing the process into phases limits the explorable design space and can lead to dramatically increased system cost.

Multiphase approaches increase costs by over-partitioning the system, limiting opportunities to share resources. System partitioning physically divides system resources like buses and memories, dedicating resource access to specific processing elements to improve performance. Resource sharing, on the other hand, time-multiplexes the accesses of multiple processing elements to resources, reducing cost. There is an inverse relationship between sharing and partitioning, as illustrated in Figure 1. Partitioning carried to its logical conclusion completely divides a system into subsystems, with each processor (P) accessing only its own memories (M). Sharing carried to its logical conclusion, on the other hand, makes all resources available to all processors. System optimization is the process of finding the best application-specific balance of sharing and partitioning. When multiphase approaches partition systems in early design phases because the performance impact of sharing can't be quantified, resources are locked up that could otherwise be shared or eliminated. This restricts the design space and solution quality.

Our contention is that the best way to balance partition-

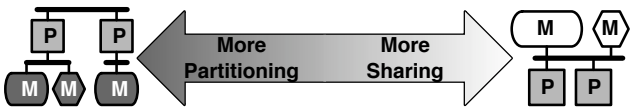

Figure 1: Partitioning reduces opportunity to share resources by restricting access to them.

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria

Table 1: Memory Allocation and Bus Synthesis Approaches in the Literature

Ref	Phases	Private	Co-Synthesis	Notes
[1]	2	Yes	No	Local memories on dedicated buses. Memory allocation is never evaluated.
[2]	2	Yes	No	Same as in [1].
[3]	3	Yes	No	Local memories on dedicated buses. After bus synthesis, shared memories are sometimes merged if performance evaluation allows.
[4]	1	No	Yes	Only co-synthesis approach. Local memories restricted to nearby buses. Memory allocation evaluated from performance perspective only.

ing and sharing to truly optimize performance and cost is to do it in the context of system cost and performance evaluation. In the case of memory allocation and bus synthesis, however, system cost and performance can't be evaluated until the bus topology has been determined because of the role area and wire length play in system cost and performance. Memory allocation and bus topology must therefore be explored simultaneously so that the implications of design changes can be accurately measured.

We present a novel synthesis tool that simultaneously explores memory allocation, data mapping and bus synthesis for embedded multiprocessor systems-on-chip. Our approach combines this exploration in a single phase, evaluating both performance and system cost with each design change. By treating memory allocation and data mapping as first class design space axes along with bus synthesis, our approach is able to capture how the design axes interact, exposing a larger design space and allowing us to better balance system partitioning and resource sharing. This results in 20% cost reduction for high-performance designs as well as 27% for low-cost designs when our single-phase approach is compared with a multiphase approach that separately allocates memories and synthesizes buses.

2. RELATED WORK

Computer-aided design approaches, especially in response to increasing design size and complexity, frequently divide system design into phases. The high-level design space of embedded multiprocessor design is large, consisting of countless combinations and organizations of processing elements and other components, and task and data mappings to those components. Design is greatly simplified if only one aspect of the system is considered at a time. For multiprocessor embedded systems, this is often manifested as the division of design into (1) task decomposition and mapping, (2) memory allocation and data mapping, and (3) bus synthesis. The underlying assumption is that a system can be globally optimized through separate local optimization steps.

A large body of other work has been done to address (3) in isolation [5][6][7][8][9]. Unlike our approach, these all assume a fixed memory allocation as input, severely limiting the opportunities to trade off system partitioning and resource sharing.

The literature also contains a number of proposals for how best to approach problems (2) and (3) together. For a selection of these techniques, Table 1 lists the number of phases the approach breaks memory allocation and bus synthesis into, whether local memories are private, and whether local or shared memories are co-synthesized with buses.

In approaches [1] and [2], phase one identifies data local to each processor, and assigns it to a memory allocated on a bus dedicated to that processor, heavily partitioning the system. Shared memories are also allocated in phase one, divided so two processors only access the same memory if they access the same address space.

[3] allocates dedicated local memories in phase one, then performs bus synthesis in phase two. In phase three, shared memories are merged provided they are accessed by the same processors and the merging won't introduce delay due to contention. Though this approach adjusts memory allocation after bus synthesis, since it doesn't perform floorplanning to measure system cost, it assumes that combining is always beneficial. This approach also only considers merging memories once, making it heavily dependent on the initial memory allocation.

[4] co-synthesizes memories and buses, exploring memory allocation and bus synthesis at the same time. This approach evaluates system performance to determine if memories should be merged or split. Shared data and local data may be mapped to the same memory provided that the local memory is allocated to a bus the processor can access directly (rather than through a bridge). This approach performs co-synthesis only for performance, however, and doesn't consider system cost at all.

All of the above approaches place restrictions on where or how memories are allocated in an effort to expose performance to later design stages, but inevitably lead to increased costs. Even [4], the only bus-memory co-synthesis approach, enforces keeping local data close, restricting resource sharing. Though the system partitioning employed improves performance by ensuring that local data is available quickly, doing so typically requires the costly integration of additional resources. Private buses may be underutilized, and statically allocated memories may result in suboptimal floorplanning compared with approaches that can combine or divide memories during bus synthesis. Partitioning systems without concern for the cost implications has the unintended consequence of limiting resource sharing opportunities, which in turn restricts designers from finding cost-optimal, and therefore generally optimal designs.

3. SIMULTANEOUS SYNTHESIS AND EVALUATION

We have developed a novel synthesis tool that avoids the resource sharing limitations of the prior work by allowing data mapping, memory allocation, and bus synthesis to be explored simultaneously in a process driven by performance and cost evaluation. Our approach, illustrated in Figure 2, employs simulated annealing techniques [10] augmented with closed-form decision-making in the form of deterministic design refinement. After selecting an initial temperature using the approach in [11], the system is iteratively permuted and cooled according to a constant cooling schedule until it remains frozen (unchanged) for three iterations (parameters detailed in Section 4.4).

In each iteration, the annealer selects and performs a move that modifies the system's data mapping, memory allocation, or the bus-based interconnect. Then, the system is pruned of unused resources and the data mapping is legal-

```
while (! frozen) {
    /* modify system */
    randomly select and apply move
    if (new topology) {
        dofloorplan = TRUE
        prune system

        if (node was moved)
            legalize data mapping
    }

    /* evaluate system */
    if (dofloorplan)
        floorplan      // to evaluate cost
    DES      // to evaluate latency
    accept/reject move
}
```

Figure 2: System modification and evaluation sequence. Repeats once per move until the system is frozen.

ized. Next, floorplanning is done as needed, and physical cost and execution latency, the two components of the objective function, are evaluated.

Pruning and data mapping legalization reduce the exploration burden of the annealer without limiting access to optimal designs. In our experiments, we found that even when we consider all of these design axes simultaneously, our approach is able to complete design space exploration in 6-8 hours, comparable to other approaches.

3.1 Move Set

The construction of the move set is motivated by our observation that redistributing resources is a more effective way to trade off system partitioning and resource sharing than creating and removing resources. For example, moves do not create or destroy memories or buses, but rather split, merge, or move resources about in the system, in effect partitioning or sharing resources among the different processing elements in the system. At present, systems are restricted to processing elements, SRAM memories and flat (non-hierarchical) bus-based interconnect.

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 memory (and its data) into another larger memory or *splitting* a large memory (and its data) into two smaller memories.

3.2 Pruning and Data Mapping Legalization

Any move that modifies the system topology triggers *pruning*. Pruning searches the system for memories that could be replaced with smaller memories without displacing data, and removes unused memories and buses, eliminating unnecessary cost in the system. Pruning memories does not restrict the design space, as data moves occur independently of the storage available in the destination memory (the move is performed, then legalized by growing the memory if necessary). Pruning buses also has negligible impact on the design space. As moves that disconnect nodes from buses must reduce cost on average to be accepted, that a bus is unused indicates that the system may not need it. There is no move to remove buses; pruning allows the system to automatically gravitate toward the number and arrangement of buses that best fits the design constraints.

Any move that changes which bus(es) a memory (or processor) is connected to triggers *data mapping legalization*. When a node is moved in the system, there's a chance that some processors may not be able to access all of their data. Rather than require that all moves be legal initially (a very strict requirement which makes design space exploration very difficult), data mapping legalization finds a new legal assignment for the inaccessible data given the requested topology change.

Together, *pruning* and *data mapping legalization* reduce the size of the design space that must be explored by reducing multiple designs to a single design point or alleviating the annealer of the need to traverse large numbers of inferior designs to reach a better system organization.

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 data mapping legalization) and is used to determine if the design change in question should be accepted or rejected. Moves that decrease *system cost* are automatically accepted. Moves that degrade *system cost* are accepted with probability $\exp(-\Delta E/T)$, where T is the temperature of the current iteration and ΔE is the change in *system cost* that results when the move is applied.

System cost is the weighted combination of total execution cost, or latency, and the system's physical cost:

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

where α is selected by the designer based on the relative importance of performance and cost to the given design. The goal of our approach is to minimize system cost for a given α.

Physical cost is a function of system area, wire length, and aspect ratio. Since square designs are more easily manufactured, the *baseline* physical cost is scaled to penalize floorplans with high aspect ratios.

$$\begin{aligned} baseline &= \beta \cdot area + (1 - \beta) \cdot wire\ length \\ physical\ cost &= baseline \cdot (0.75 + 0.25 \cdot aspect\ ratio) \end{aligned} \quad (2)$$

β is chosen so that system area and total bus wire length contribute equally to the baseline physical cost.

System *area*, *wire length* and the *aspect ratio* are all determined with floorplanning. *Area* is the bounding box of the design. *Wire length* is the sum of half-perimeter wire lengths of all buses, evaluated using the smallest rectangle enclosing the center of each module connected to a bus. *Aspect ratio* is the longer dimension of the design divided by the shorter (so it is always ≥ 1). Floorplanning is performed by annealing a slicing tree representation of the system [12].

Once floorplanning has completed, we use bus topology information and the system floorplan to determine the bus speed for the system. The delay for each bus is first determined using the approach proposed in [13]. Then, system bus cycle length is determined by the speed of the slowest bus in the system. This bus cycle time is subsequently used in latency evaluation.

Latency is evaluated by determining the total system exe-

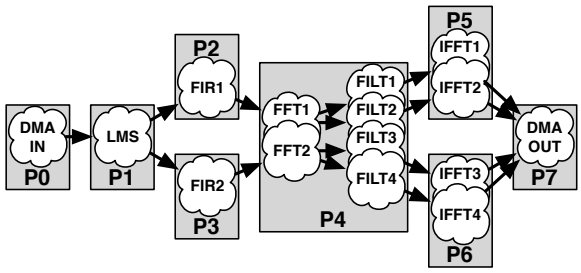

Figure 3: Software pipeline and task-processor mapping.

cution time. This is accomplished by first gathering memory access traces to express the partially ordered sequence of all memory accesses that occur in the system, including any dependencies (e.g., one trace may not be able to start reading memory until another has finished writing). The total system execution time is then calculated by performing discrete event simulation (DES) on the dependency graph generated by combining all memory access traces in the system. Memory accesses are modeled as events which propagate through the system from processors to memories (request) and back again (data), according to the protocol of the given interconnect modules. We acknowledge that DES does not scale and will limit the scalability of our approach. We employ DES for proof-of-concept purposes; replacing it with a method that scales better is the subject of future work.

4. EXPERIMENTS AND RESULTS

We conducted experiments to compare our simultaneous synthesis and exploration (SSE) technique with a technique that performs system synthesis in two phases. Using an example workload and fixed task-processor mappings, we used our technique to find latency-cost pareto-optimal memory architectures and interconnect configurations for a particular workload and a small component library. We compare our results with an approach that performs static memory allocation and data mapping prior to bus synthesis.

4.1 Workload

Our experimental workload is the DSP software pipeline illustrated in Figure 3. Data is introduced to the system by a hardware DMA engine (P_0), and fed to a least-mean-squared (LMS) adaptive filter (on processor P_1) for noise cancellation. Two different FIR filters are then separately applied to the noise-free data (P_2 and P_3), after which the filtered data is transformed into the frequency domain for the further application of four filters (P_4). Each filtered stream is then transformed back into the time domain (P_5 and P_6) before being collected and sent off-chip by another hardware DMA engine (P_7). Processors that execute more than one task execute each task to completion before starting the next. Memory access traces were generated by hand 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 130 nm process.

All processors operate at 133 MHz, and with the exception of P_0 and P_7, 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 a fixed area penalty of 10% with respect to an ARM7 is 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. The delay of the slowest bus in the system, a function of bus length and the number of attached nodes, is used to determine the system bus speed. System bus speed is selected from 33, 66, 100, and 133 MHz.

ARM7 area was derived from publicly available area data [14]; the ARM7 with single bus port is assumed to be square. We derived DMA and memory area using CACTI 4.2 [15]. When the processing elements are connected to more than one bus, the area penalty is assessed in such a way that the devices grow wider (they do not maintain their original aspect ratio).

4.3 Two-Phase Comparison Approach

We compared our SSE technique to a two-phase exploration approach. The two-phase approach statically allocates memories in phase one according to the method described in [1] and then performs bus topology optimization using our formulation in phase two. All data accessed by a single processor is mapped to a single local memory attached to a bus dedicated to that processor. All shared data is divided into shared memories based on which processors access it. Shared arrays are only mapped to the same memory if they are accessed by the same set of processors.

After local and shared memories have been allocated, the comparison approach uses the same annealing process described in Section 3 to find the best bus topology, but with a few modifications. No data mapping or memory allocation moves are attempted under the two-phase approach; only the bus topology moves are available during annealing. Further, only shared memories and processors may be connected or disconnected from the buses in the system. No nodes beyond the dedicated local memory and its accessing processor are connected to or disconnected from the dedicated buses.

4.4 Parameters

The majority of parameters for the system annealer and floorplanning annealer are fixed for all of the experiments we conducted. The two exceptions to this are α, used to adjust the relative importance of latency and system cost in the objective function (detailed in Section 4.5), and the system annealer move distribution, which was different depending on which approach, either simultaneous or two-phase exploration, was being used.

For SSE, data, memory, and bus moves, are selected 75%, 20%, and 5% of the time respectively. Because of the varying number of possible system configurations that result from the different system move types, each move type is selected with a different probability. Once the move type has been randomly selected, all move subtypes are equally likely to be selected.

For the two-phase comparison approach, the move distribution is restricted to bus moves. Each bus move subtype is equally likely to occur.

For the system annealer (both approaches), we performed 5000 moves per iteration, used a constant cooling rate of 0.8, and a freezing threshold of 0.01. We performed sensitivity analysis to validate this selection of annealing parameters with respect to the move distributions, and empirically determined that this set represented the best trade-off between

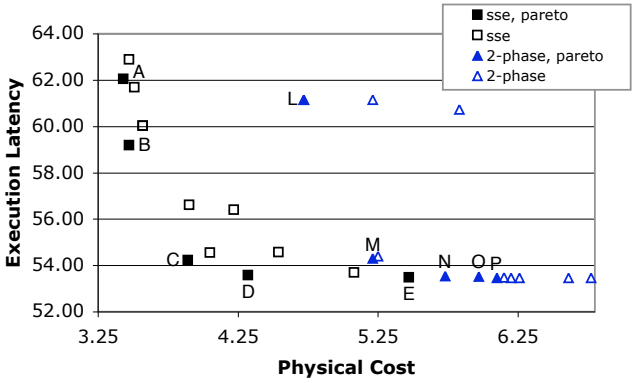

Figure 4: SSE shares resources to deliver similar performance for more than 20% less cost than the two-phase approach.

design quality and annealer execution time for both our approach and the comparison approach. For our approach, because it explores data mapping and memory allocation at the same time as bus synthesis, there are many equivalent designs. The comparison approach by contrast has to apply relatively more bus moves to reach the same number of designs because only the bus topology is malleable.

For the floorplanning annealer, we performed 200 moves per iteration, used a constant cooling rate of 0.8, and a freezing threshold of 0.05. As for the system annealer, we performed sensitivity analysis to validate this selection of parameters. We also empirically determined that for the sorts of designs we considered, $\beta = 0.75$ equally balanced the importance of design wire length and area.

4.5 Results

We executed both our SSE technique and the two-phase comparison approach with a variety of different α values, varying the ratio of importance of performance and cost from 9:1 to 1:9. Since the two-phase approach results in, on average, more costly designs, we used a separate set of α values, for those experiments. For SSE, α varied from 0.1 to 0.9. For the two-phase approach, α varied from 0.2 to 0.95. We also conducted a second set of experiments with a different random seed, consisting of five more data points for SSE and the two-phase approach, selected from the same α ranges as before. A third set of experiments explores the boundaries of the partitioning-sharing continuum for both SSE and the two-phase approach.

The cost-performance points for the first two sets of experiments are plotted in Figure 4. The designs vary 18% with respect to latency and 93% with respect to cost, and implement from two to six buses. The squares represent design points for our approach, triangles design points for the comparison approach. The filled squares and triangles are pareto points for the respective approaches, and are summarized in Table 2 and Table 3. These tables report the execution latency in μs, area in mm^2, half-perimeter wire length in mm (WL), aspect ratio (AR), physical cost (Cost), number of shared buses (SB), bus speed in MHz (BS), and average shared bus utilization (ABU). α values marked with an asterix (*) indicate designs generated with the alternative random seed. On a Pentium 4 workstation with 2 GB of RAM, annealing a single design point took 6-8 hours.

As can be seen in Figure 4, our approach results in lower cost solutions, both for low latency designs and low cost designs. Comparing low cost design L with A and B, our approach achieves the same latency (2% worse, 3% better respectively) and is 27% less costly. Comparing low latency designs M with C and N with D, our approach achieves the same latencies (within 0.2%) and is 20-25% less costly.

To validate our results, we conducted additional experiments with a different random seed, and explored the boundaries of the partitioning and sharing continuum for each approach. This resulted in design points similar to those in the first set of experiments, indicating our technique is independent of the precise sequence of attempted moves.

Next, we conducted additional experiments using $\alpha = \{0.01, 0.99\}$ to determine the extent to which each approach can reduce cost through resource sharing and improve performance through system partitioning. The results of this experimentation are reported in Table 4, where SSE produced designs U and V the two-phase approach produced designs Y and Z.

When $\alpha = 0.99$, our approach (V) exhibits execution latency within 0.1% of the corresponding design under the comparison approach (Z). The reason for this difference, however marginal, is that strict system partitioning is an excellent way to approach reducing latency and establishes a degree of structure that is difficult to replicate with a simulated annealing approach. The performance advantage, however, comes at a significant cost for designs other than those at the extreme partitioning end of the continuum, with our approach reducing cost by 20% or more for both low latency and low cost designs.

When $\alpha = 0.01$, our approach (U) achieves 36% lower cost compared to the corresponding two-phase design point (Y). Though a performance penalty is paid (89% increased latency) for such a low-cost design, such a design is impossible for a two-phase approach that partitions the system; with only one shared bus for each design, no further sharing is possible.

4.6 Discussion

The reason for the cost differential observed above can be exposed by comparing the area and wire length for designs from the two approaches. For the designs compared above, our approach generates designs with 17% less area on average. This is a direct result of being able to make memory allocation adjustments, merging or splitting memories, to address the floorplans of individual design points. While

Table 2: SSE Pareto Design Points

	Alpha	Latency	Area	WL	AR	Cost	SB	BS	ABU
A	0.20	62.05	3.14	3.82	1.15	3.43	2	133	0.78
B	*0.50	59.19	3.10	4.04	1.16	3.47	2	133	0.76
C	0.70	54.23	3.19	5.59	1.10	3.89	3	133	0.56
D	0.90	53.59	3.36	7.19	1.00	4.32	3	133	0.56
E	*0.90	53.49	3.65	10.87	1.01	5.47	5	133	0.34

Table 3: Two-phase Pareto Design Points

	Alpha	Latency	Area	WL	AR	Cost	SB	BS	ABU
L	0.20	61.15	3.71	7.74	1.00	4.72	2	100	0.52
M	*0.35	54.31	4.00	8.36	1.10	5.21	2	133	0.44
N	0.55	53.54	3.98	10.21	1.14	5.73	4	133	0.22
O	*0.65	53.53	4.07	11.40	1.05	5.97	4	133	0.22
P	0.90	53.47	3.96	11.50	1.17	6.10	5	133	0.18

Table 4: Performance and Cost Corner Cases

	Alpha	Latency	Area	WL	AR	Cost	SB	BS	ABU
U	0.01	191.43	3.01	2.64	1.15	3.02	1	66	0.95
V	0.99	53.47	4.26	14.37	1.04	6.85	5	133	0.34
Y	0.01	100.86	4.13	6.19	1.09	4.75	1	66	0.95
Z	0.99	53.43	4.55	14.01	1.03	6.96	5	133	0.18

the two-phase approach assumes that a single memory allocation will serve for all designs, our approach allows memory allocation to change based on the designer's desired balance of performance and cost.

Designs generated by our approach also have, on average, 40% less bus wire length. This is a direct result of needing fewer buses to deliver the same performance. Though the two approaches use approximately the same number of shared buses, the two-phase approach also implements six largely unutilized dedicated buses, one for each ARM7. Providing dedicated buses independent of performance and cost impact results in over-designed systems.

These results corroborate our findings in [16]. We observed that if buses and memories are co-synthesized, the bus topology is the first aspect of the design to freeze, with memory allocation and data mapping optimization reducing system cost an additional 24% after that point. Approaches that restrict memory allocation or perform it in isolation are unable to expose its apparent synergy with bus synthesis.

In effect, because our approach considers the performance and cost impact of memory allocation, data mapping, and bus synthesis all at the same time, we are able to better balance resource sharing and partitioning. The absence of barriers to resource sharing and system partitioning allows our approach not only to improve performance by isolating subsystems from one another, but also to reduce system cost by sharing resources when the performance impact is tolerable. This results in systems that offer dramatic cost savings and minimal performance losses compared with multi-phase design approaches.

5. CONCLUSIONS

We have demonstrated that in order to optimize multiprocessor memory allocation and bus synthesis, the two must be conducted simultaneously, and in the presence of accurate performance and cost evaluation. Computer-aided design approaches to memory allocation and bus synthesis frequently perform the two in isolation or impose other restrictions that, though intended to improve system performance, increase cost by limiting the opportunity to share resources. Bus-memory co-synthesis, coupled with evaluation techniques that can identify when partitioning and sharing are the most profitable, is the best way to expose the entirety of the partitioning-sharing continuum, and therefore find the optimal balance.

We presented a synthesis tool that 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 and a task-processor mapping. Annealer moves permute the system by adjusting how resources are shared and partitioned among processing elements. Results from the synthesis of a DSP software pipeline demonstrate the annealer's ability to judiciously balance global resource sharing and system partitioning, exposing a larger design space and wider opportunity to improve performance through system partitioning and reduce cost through resource sharing. Our experimentation revealed that our approach can deliver low-latency and low-cost designs with cost reductions of 20% and 27% respectively when compared with an approach that separately allocates memories and synthesizes buses.

6. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation through Grant CNS-0509193 and by the Semiconductor Research Consortium through contract 2005-HJ-1312. 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. We would like to thank Alex Bobrek, Ryan Johnson, and our reviewers for their thoughtful reading and helpful comments.

7. REFERENCES

[1] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane, "Floorplan-aware automated synthesis of bus-based communication architectures," in *DAC '05*, 2005.

[2] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Constraint-driven bus matrix synthesis for MPSoC," in *ASP-DAC '06*, 2006.

[3] S. Pasricha and N. Dutt, "Cosmeca: application specific co-synthesis of memory and communication architectures for MPSoC," in *DATE '06*, 2006.

[4] S. Kim, C. Im, and S. Ha, "Efficient exploration of on-chip bus architectures and memory allocation," in *CODES+ISSS '04*, 2004.

[5] J. Guo, A. Papanikolaou, P. Marchal, and F. Catthoor, "Energy/area/delay trade-offs in the physical design of on-chip segmented bus architecture," in *SLIP '06*, 2006.

[6] K. Lahiri, A. Raghunathan, and S. Dey, "Efficient exploration of the soc communication architecture design space," in *ICCAD '00*, 2000.

[7] J. Hu, Y. Deng, and R. Marculescu, "System-level point-to-point communication synthesis using floorplanning information," in *ASP-DAC '02*, 2002.

[8] S. Maguerdichian, M. Drinic, and D. Kirovski, "Latency-driven design of multi-purpose systems-on-chip," in *DAC '01*, 2001.

[9] N. Thepayasuwan and A. Doboli, "Layout conscious approach and bus architecture synthesis for hardware/software codesign of systems on chip optimized for speed," *IEEE Trans. on VLSI*, vol. 13, no. 5, 2005.

[10] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, *Modern Heuristic Search Methods*. New York, NY, USA: Wiley, 2001.

[11] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *ICCAD '86*, 1986.

[12] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *DAC '86*, 1986.

[13] J. Cong and Z. Pan, "Interconnect performance estimation models for design planning," *IEEE Trans. on CAD of ICs and Syst.*, vol. 20, June 2001.

[14] "ARM7TDMI." http://www.arm.com/products/CPUs/ARM7TDMI.html.

[15] "CACTI 4.2." http://quid.hpl.hp.com:9081/cacti/.

[16] B. H. Meyer and D. E. Thomas, "Rethinking automated synthesis of MPSoC architectures," in *NSF Next Generation Software Program*, March 2007.