Memory System Connectivity Exploration.*

Peter Grun
pgrun@cecs.uci.edu

Nikil Dutt
dutt@cecs.uci.edu

Alex Nicolau
nicolau@cecs.uci.edu

Center for Embedded Computer Systems
University of California, Irvine, CA 92697-3425, USA

Abstract

In programmable embedded systems, the memory subsystem represents a major cost, performance and power bottleneck. To optimize the system for such different goals, the designer would like to perform Design Space Exploration, evaluating different memory modules from a memory IP library, and selecting the most promising designs. However, while the memory modules are important, the rate at which the memory system can produce the data for the CPU is significantly impacted by the connectivity architecture between the memory subsystem and the CPU. Thus, it is critical to consider the connectivity architecture early in the design flow, in conjunction with the memory architecture. We present a connectivity architecture exploration approach, evaluating a wide range of cost, performance, and energy connectivity architectures. When coupled with our memory modules exploration approach, we can significantly improve the system behavior. We present experiments on a set of large real-life benchmarks, showing significant performance improvements for varied cost and power characteristics, allowing the designer to tailor the performance, cost and power of the programmable embedded system.

1 Introduction

In contemporary programmable embedded systems, memory represents a major cost, performance and power bottleneck [21]. In order to optimize the system for such varying goals, the system designer would like to evaluate different combinations of memory modules from an IP library, such as caches, SRAMs, DMAs, etc., performing Design Space Exploration (DSE) of the memory architecture. However, the cost, bandwidth and power footprint of the memory system is influenced by both the memory modules employed, as well as the connectivity components which transfer the data between the memory modules, and the CPU. While the memory modules configuration and characteristics are important, often the connectivity structure has a comparably large impact on the system performance, cost and power; thus it is critical to consider connectivity early in the design flow. We present here such an approach, where we perform connectivity exploration, evaluating a wide range of connectivity configurations using components from a connectivity IP library, such as standard on-chip busses (e.g., AMBA busses [1]), MUX-based connections, and off-chip busses. Our approach significantly improves the performance of the system for varying cost, and power consumption, allowing the designer to best tradeoff the different goals of the system.

In [12] we presented the exploration of the memory modules, based on the access patterns exhibited by the application, assuming a simple connectivity model. In this paper we extend this work by performing connectivity exploration in conjunction with the memory modules exploration, to improve the behavior of the memory-connectivity system. There are two possible approaches to improving the memory system behavior: (a) a synthesis-oriented, optimization approach, where the result is a unique "best" solution, and (b) an exploration-based approach, where different memory system architectures are evaluated, and the most promising designs following a pareto-like shape are provided as the result, allowing the designer to further refine the choice, according to the goals of the system. In this paper we follow the second approach. We guide the design space search towards the pareto points in different design spaces (such as the cost/performance, and performance/power spaces), pruning the non-interesting designs early in the exploration process, and avoiding full simulation of the design space.

In Section 2 we present related work in the area of connectivity and memory architecture exploration, in Section 3 we present the flow of our approach. In Section 4 we use an example application to illustrate our exploration strategy, and in Section 5 we show the details of our Connectivity Exploration (ConEx) algorithm. We conclude with a set of experiments showing the cost, performance and power tradeoffs obtained by our coupled memory and connectivity exploration, and a short summary in Section 7.

2 Related Work

There has been related work in four main areas: (I) High-Level Synthesis, (II) System-on-Chip core-based systems, (III) Interface synthesis. and (IV) Layout and routing of connectivity wiring,

(I) In High-Level Synthesis, Narayan et al. [20] synthesize the bus structure and communication protocols to implement a set of virtual communication channels, trading off the width of the bus and the performance of the processes communicating over it. Daveau et al. [7] present a library based exploration approach, where they use a library of connectivity components, with different costs and performance. We complement these approaches by exploring the connectivity

*This work was partially supported by grants from NSF (MIP-9708067), DARPA (F33615-00-C-132) and a Motorola Fellowship.

design space in terms of all the three design goals: cost, performance and power simultaneously.

Wuytack et. al. [22] present an approach to increase memory port utilization, by optimizing the memory mapping and code reordering. Our technique complements this work by exploring the connectivity architecture, employing connectivity components from an IP library. Catthoor et al. [2] address memory allocation, packing the data structures according to their size and bitwidth into memory modules from a library, to minimize the memory cost, and optimize port sharing. Forniciari et al. [9] present a simulation based power estimation for the HW/SW communication on system-level busses, aimed at architectural exploration. We use the connectivity and memory power/area estimation models from [2] to drive our connectivity exploration.

(II) In the area of System-on-Chip architectures, Givargis et al. [10] present a connectivity exploration technique which employs different encoding techniques to improve the power behavior of the system. However, due to their platform-based approach, where they assume a pre-designed architecture platform which they tune for power, they do not consider the cost of the architecture as a metric. Drinic et al. [18] present an on-chip bus network design methodology, optimizing the allocation of the cores to busses to reduce the latency of the transfers across the busses. Lahiri et al. [17] present a methodology for the design of custom System-on-Chip communication architectures, which propose the use of dynamic reconfiguration of the communication characteristics, taking into account the needs of the application.

(III) Recent work on interface synthesis [4], [5] present techniques to formally derive node clusters from interface timing diagrams. These techniques can be used to provide an abstraction of the connectivity and memory module timings in the form of Reservation Tables [15]. Our algorithm uses the Reservation Tables [11, 14] for performance estimation, taking into account the latency, pipelining, and resource conflicts in the connectivity and memory architecture.

(IV) At the physical level, the connectivity layout and wiring optimization and estimation has been addressed. Chen et al. [3] present a method to combine interconnect planning and floorplanning for deep sub-micron VLSI systems, where communication is increasingly important. Deng et al. [8] propose the use of a 2.5-D layout model, through a stack of single-layer monolithic ICs, to significantly reduce wire length. We use the area models presented in [3] and [8] to drive our high-level connectivity exploration approach.

To our knowledge, none of the previous approaches have addressed connectivity exploration in conjunction with memory modules architecture, considering simultaneously the cost, performance, and power of the system, using a library of connectivity components including standard busses (such as AMBA [1], mux-based connections, and off-chip busses). By pruning the non-interesting designs early in the design flow, and simulating only the most promising architectures, we allow the designer to explore the connectivity architectures space, to best trade off the different goals of the system.

3 Our approach

Figure 1 shows the flow of our approach. The Connectivity Exploration (ConEx) approach is part of the MemorEx Memory System Exploration environment. Starting from the input application in C, our Access Pattern-based Memory Exploration (APEX) [12] algorithm first extracts the most active access patterns exhibited by the application data structures, and explores the memory module configurations to match the needs of these access patterns; however, it assumes a simple connectivity model. Our ConEx Connectivity Exploration approach starts from this set of selected memory modules configurations generated by APEX, and explores the most promising connectivity architectures, which best match the performance, cost and power goals of the system. Since the complete design space is very large, and evaluating all possible combinations in general is intractable, at each stage we prune out the non-interesting design configurations, and consider for further exploration only the points which follow a pareto-like curve shape in the design space.

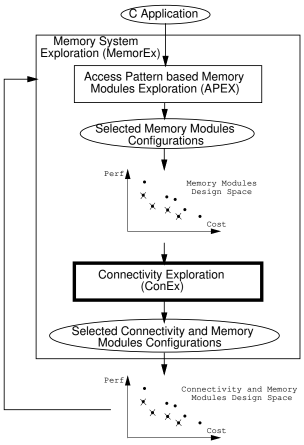

Figure 1. The flow of our Exploration Approach.

Starting from a memory architecture containing a set of memory modules, we map the communication channels between these modules, the off-chip memory and the CPU to connectivity modules from a connectivity IP library. Figure 2 (a) shows the connectivity architecture template for an example memory architecture, containing a cache, a stream buffer, an on-chip SRAM, and an off-chip DRAM. The communication channels between the on-chip memory modules, the off-chip memory modules and the CPU can be implemented in many ways. One naive implementation is where each communication channel is mapped to one connectivity module from the library. However, while this solution may generate good performance, in general the cost is prohibitive. Instead, we cluster the communication channels into groups based on their bandwidth requirement, and map each such cluster to connectivity modules. Figure 2 (b) shows an example connectivity architecture implementing the communication channels, containing two on-chip busses, a dedicated connection, and an off-chip bus.

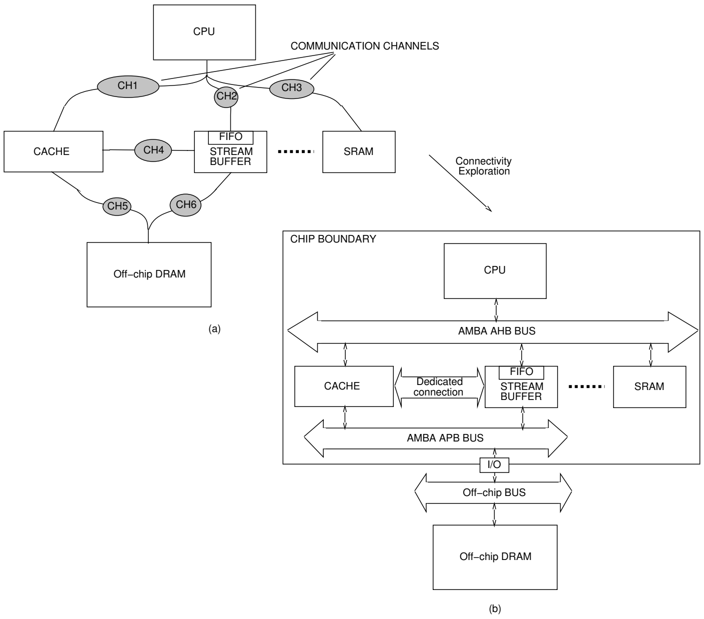

Figure 2. (a) The Connectivity Architecture Template and (b) An Example Connectivity Architecture.

4 Illustrative example

We use the compress benchmark (SPEC95) to illustrate the cost, performance, and power trade-offs generated by our connectivity exploration approach. The benchmark contains a varied set of data structures and access patterns, presenting interesting opportunities for customizing the memory architecture and connectivity.

The connectivity exploration is part of our larger memory exploration approach. First we perform Access Pattern-based Memory Exploration (APEX) [12], to determine a set of promising memory modules architectures. For each such memory modules architecture, a set of different connectivity architectures are possible, each resulting in different cost, performance and power characteristics. Our Connectivity Exploration approach (ConEx) starts from the memory modules architectures generated by APEX, and explores the connectivity configurations, using components from a connectivity library (such as the AMBA busses [1], MUX-based connections, etc.), trading off the cost, performance and power for the full memory system, taking into account both the memory modules and the connectivity structure.

For our compress illustrative example benchmark, APEX selects the most promising memory modules configurations. The resulting memory architectures employ different combinations of modules such as caches, SRAMs, and DMA-like custom memory modules storing well-behaved data such as linked lists, arrays of pointers, streams, etc. [12]. Figure 3 shows the memory modules architectures explored by APEX for the compress example. The X axis represents the cost of the memory modules in basic gates, and the Y axis represents the overall miss ratio (we assume that accesses to on-chip memory such as the cache or SRAM are hits, and accesses to off-chip memory are misses). APEX prunes the non-

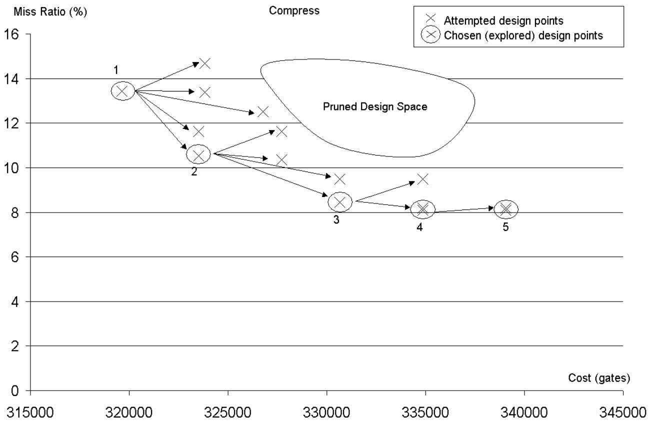

Figure 3. The most promising memory modules architectures for the compress benchmark.

interesting designs, on the inside of the pareto curve, choosing only the most promising cost/performance architectures for further exploration. The points labeled 1 through 5 represent the selected memory modules designs, which will be used as the starting point for the connectivity exploration.

Each such selected memory architecture may contain multiple memory modules with different characteristics, and communication requirements. For each such architecture, different connectivity structures, with varied combinations of connectivity modules from the library may be used. For instance, the memory modules architecture labeled with 3 in Figure 3 contains a cache, a memory module for stream accesses, a memory module for self-indirect [1] array references [12], and an off-chip DRAM. When using dedicated or MUX-based connections from the CPU to the memory modules, the latency of the accesses is small, at the expense of longer connection wires. Alternatively, when using a bus-based connection, such as the AMBA System bus (ASB) [1], the wire length decreases, at the expense of increased latency due to the more complex arbitration needed. Similarly, when using wider busses, with pipelined or split transaction accesses, such as the AMBA High-performance bus (AHB) [1], the wiring and bus controller area increases further. Moreover, all these considerations impact the energy footprint of the system. For instance, longer connection wires generate larger capacitances, which may lead to increased power consumption.

Figure 4 shows the ConEx connectivity exploration for the compress benchmark. The X axis represents the cost of the memory and connectivity system. The Y axis represents the average memory latency, including the latency due to the memory modules, as well as the latency due to the connectivity. The average memory latency is reduced from 10.6 cycles to 6.7 cycles, representing a 36% improvement [2], while trading

[1] We call self-indirect the array references which use the current array element value to compute the index for the next array element access [12].

[2] Please note that in order to keep the figures clear, we did not include the uninteresting designs exhibiting very bad performance (many times worse than the best designs). While those designs would increase even further the

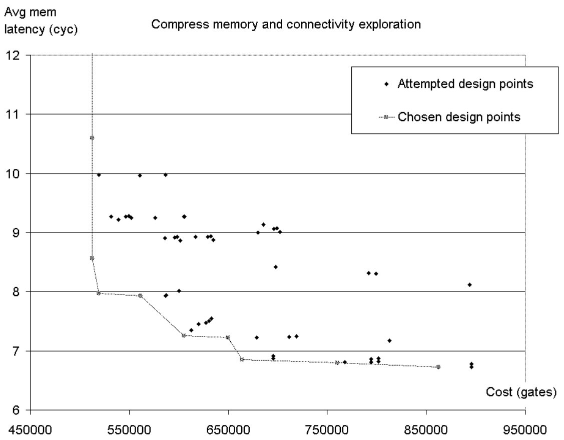

Figure 4. The connectivity architecture exploration for the compress benchmark.

off the cost of the connectivity and memory modules.

Alternatively, for energy-aware designs, similar tradeoffs are obtained in the cost/power or the performance/power design spaces (the energy consumption tradeoffs are presented in the Section 6). In this manner we can customize the connectivity architecture, thus substantially improving the memory and connectivity system behavior, and allowing the designer to trade off the different goals of the system.

5 Connectivity Exploration Algorithm

Our Connectivity Exploration (ConEx) algorithm is a heuristic method to evaluate a wide range of connectivity architectures, using components from a connectivity IP library, and selecting the most promising architectures, which best trade-off the connectivity cost, performance and power.

Figure 5 shows our Connectivity Exploration algorithm. The input to our ConEx algorithm is the application in C, a set of selected memory modules architectures (generated by the APEX exploration step [12]), and the connectivity library. Our algorithm generates as output the set of most promising connectivity/memory modules architectures, in terms of cost, performance and power.

For each memory modules architecture selected in the APEX memory modules exploration stage [12], multiple connectivity implementations are possible. Starting from these memory modules architectures, we explore the connectivity configurations by taking into account the behavior of the complete memory and connectivity system, allowing the designer to tradeoff the cost, performance and power of the design. The ConEx algorithm proceeds in two phases: (I) Evaluate connectivity configurations (II) Select most promising designs.

(I) Evaluate connectivity configurations. For each memory architecture selected from the previous APEX Memory Modules Exploration phase [12], we evaluate different connectivity architecture templates and connectivity allocations using components from the connectivity IP library. We estimate the cost, performance and power of each such connectivity archi-

```
Procedure ConnectivityExploration(Memory Modules Architecture mem_arch)
Input: The C Application and the Memory Modules Architecture mem_arch,
  the Connectivity Library
Output: The most promising Connectivity Design Points
begin
  Profile the Memory Modules Architecture mem_arch
  Construct the Bandwidth Requirement Graph (BRG)
  Allocate each arc in the BRG to a logical connection cluster
  connect_design_points = φ
  do{
    if number_of_logical_connections ≤ max_cost_constraint
      Allocate the logical connections to physical connections
        from the Connectivity Library
      Estimate the Cost, Performance and Power of connecitivity architecture
      Add this connectivity architecture to connect_design_points
    Merge the two logical connection clusters with lowest bandwidth requirement
      hierarchycally into a larger cluster
  }while(more clusters can be merged)
  return connect_design_points;
end

Algorithm ConEx
Input: C Application, Selected Memory Modules Architectures
Output: The Combined Memory Modules and Connectivity Design Points
  w/ best cost/performance/power trade-offs
begin
  Phase I:
    combined_design_points = φ
    For each selected memory module architecture mem_arch
      connect_design_points = ConnectivityExploration(mem_arch)
      Select the local most promising connectivity desing points from
        connect_design_points
      Add selected design points to combined_design_points
  Phase II:
    Simulate the design points from combined_design_points
    Select the global most promising combined memory modules and connectivity
      design points from combined_design_points
end
```

Figure 5. Connectivity Exploration algorithm.

tecture, and perform an initial selection of the most promising design points for further evaluation.

We start by profiling the bandwidth requirement between the memory modules and CPU for each memory modules architecture selected from APEX, and constructing a Bandwidth Requirement Graph (BRG). The Bandwidth Requirement Graph (BRG) represents the bandwidth requirements of the application for the given memory modules architecture. The nodes in the BRG represent the memory and processing cores in the system (such as the caches, on-chip SRAMs, DMAs, off-chip DRAMs, the CPU, etc.), and the arcs represent the channels of communication between these modules. The BRG arcs are labeled with the average bandwidth requirement between the two modules.

Each arc in the BRG has to be implemented by a connectivity component from the connectivity library. One possible connectivity architecture is where each arc in the BRG is assigned to a different component from the connectivity library. However, this naive implementation may result in excessively high cost, since it does not try to share the connectivity components. In order to allow different communication channels to share the same connectivity module, we hierarchically cluster the BRG arcs into logical connections, based on the bandwidth requirement of each channel. We first group the channels with the lowest bandwidth requirements into logical

performance variation, in general they are not useful.

connections. We label each such cluster with the cumulative bandwidth of the individual channels, and continue the hierarchical clustering. For each such clustering level, we then explore all feasible assignments of the clusters to connectivity components from the library, and estimate the cost, performance, and power of the memory and connectivity system.

(II) Select most promising designs. In the second phase of our algorithm, for each memory and connectivity architecture selected from Phase I we perform full simulation to determine accurate performance and power metrics. We then select the best combined memory and connectivity candidates from the simulated architectures.

While in the Phase I we selected separately for each memory module architecture the best connectivity configurations, in the Phase II we combine the selected designs and choose the best overall architectures, in terms of both the memory module and connectivity configuration.

The different design points present different cost, performance and power characteristics. In general, these three optimization goals are incompatible. For instance, when optimizing for performance, the designer has to give up either cost, or power. Typically, the pareto points in the cost/performance space have a poor power behavior, while the pareto points in the performance/power space will incur a large cost. We select the most promising architectures using three scenarios: (a) In a power-constrained scenario, where the energy consumption has to be less then a threshold value, we determine the cost/performance pareto points, to optimize for cost and performance, while keeping the power less then the constraint, (b) In a cost-constrained scenario, we compute the performance/power pareto points, and (c) In a performance-constrained scenario, we compute the pareto points in the cost-power space, optimizing for cost and power, while keeping the performance within the requirements.

(a) In the power-constrained scenario, we first determine the pareto points in the cost-performance space. A design is on the pareto curve if there is no other design which is better in both cost and performance. We then collect the energy consumption information for the selected designs. The points on the cost-performance pareto curve may not be optimal from the the energy consumption perspective. From the selected cost-performance pareto points we choose only the ones which satisfy the energy consumption constraint. The designer can then tradeoff the cost and performance of the system to best match the design goals.

(b) In the cost-constrained scenario, we start by determining the pareto points in the performance-power space, and use the system cost as a constraint. Conversely, the pareto points in the performance-power space are in general not optimal from the cost perspective.

(c) When using the performance as a constraint, we determine the cost-power pareto points.

For performance and power estimation purposes we use a time-sampling technique [16], which significantly speeds the simulation process. While this may not be highly accurate compared to full simulation, the fidelity is sufficient to make good incremental decisions guiding the search through the design space. To verify that our heuristic guides the search towards the pareto curve of the design space, we compare the exploration results with a full simulation of all the memory and connectivity mapping alternatives for two large examples. Indeed, as shown in Section 6, our algorithm successfully finds the best points in the design space, without requiring full simulation of the design space.

6 Experiments

We performed a set of experiments on a number of large multimedia and scientific applications to show the performance, cost and power tradeoffs generated by our approach.

The memory architectures selected by our memory modules exploration presented in [12] have been used as starting point for our connectivity exploration. We present here the experimental results taking into account the cost, performance and power for the full memory system, including both the memory and the connectivity architecture.

Our exploration algorithm guides the search towards the points on the pareto curve [3] of the design space, pruning out the non-interesting designs.

In order to verify that our Design Space Exploration (DSE) approach successfully finds the points on the pareto curve, we compare the exploration algorithm results with the actual pareto curve obtained by fully simulating the design space.

6.1 Experimental Setup

We simulated the design alternatives using our simulator based on the SIMPRESS [19] cycle accurate memory model, and SHADE [6]. We assumed a processor based on the SUN SPARC [4], and we compiled the applications using gcc. The library of connectivity modules contains information such as the resource usage, latency, pipelining, parallelism, split transaction model, and bitwidth, and the exploration algorithm selects automatically the different connectivity architectures, estimates, and prunes the design space, guiding the search towards the most promising designs.

We used a set of large real-life multimedia and scientific benchmarks. Compress and Li are from SPEC95, and Vocoder is a GSM voice encoding application.

We use a time-sampling [16] estimation to guide the walk through the design space, pruning out the designs which are not interesting. The time sampling alternates "on-sampling" and "off-sampling" periods, assuming a ratio of 1/9 between the on and off time intervals. We then use full simulation for the most promising designs, to further refine the tradeoff choices. The time-sampling estimation does not have a very good absolute accuracy compared to full simulation. However, we use it only for relative incremental decisions to guide the design space search, and the estimation fidelity is sufficient to make good pruning decisions.

[3] Assuming a two dimensional cost-performance design space, a design is on the pareto curve, if there is no other design which is better in terms of both cost and performance

[4] The choice of SPARC was based on the availability of SHADE and a profiling engine; however our approach is clearly applicable to any other (embedded) processor as well

6.2 Results

Figure 6 shows the analysis of the most promising design points for the compress benchmark. The X axis represents the cost of the memory and connectivity architecture, and the Y axis represents the average memory latency including both the memory and connectivity latencies (e.g., due to the cache misses, bus multiplexing, or bus conflicts). The design points *a* through *k* represent the most promising selected memory-connectivity architectures. Architectures *a* and *b* represent two instances of a traditional cache-only memory configuration, using the AMBA AHB split transaction bus, and a dedicated connection. The architectures *c* through *k* represent different instances of novel memory and connectivity architectures, employing SRAMs to store data which is accessed often, DMA-like memory modules to bring in predictable, well-known data structures (such as lists) closer to the CPU, and stream buffers for stream-based accesses. Architecture *c* contains a linked-list DMA-like memory module, implementing an self-indirect data structure, using a MUX-based connection. This architecture generates a roughly 10% performance improvement for a small cost increase, over the best traditional cache architecture (*b*). The architecture *d* represents the same memory configuration as *c*, but with a connectivity containing both a MUX-based structure and an AMBA APB bus. Similarly, architectures *e* through *k* make use of additional linked-list DMAs, stream buffers, and SRAMs, with MUX-based, AMBA AHB, ASB and APB connections. Architecture *g* generates a roughly 26% performance improvement over the best traditional cache architecture (*b*), for a roughly 30% memory cost increase. Architecture *k* shows the best performance improvement, of roughly 30% over the best traditional cache architecture, for a larger cost increase. Clearly, our memory-connectivity exploration approach generates a significant performance improvement for varied cost configurations, allowing the designer to select the most promising designs, according to the available chip space and performance requirements.

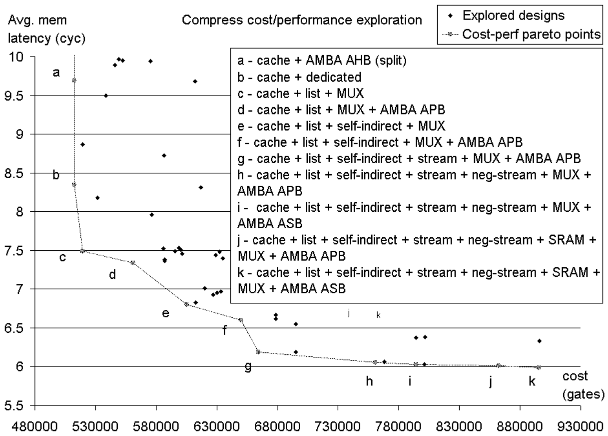

Figure 6. Analysis of the cost/perf pareto architectures for the compress benchmark.

In the following we present the exploration results for the compress, li, and vocoder benchmarks. Due to space limitations, we show only the selected most promising cost/performance designs, in terms of their cost (in basic gates), average memory latency, and average energy consumption per access. For more experimental results, please refer to [13]. In Table 1 the first column shows the benchmarks, the second, third and fourth columns show the cost, average memory latency and energy consumption for the selected design simulations. The simulation results show significant performance improvement for varied cost and power characteristics of the designs, for all the benchmarks. For instance, when using different memory and connectivity configurations, the performance of the compress and li benchmarks varies by an order of magnitude. The energy consumption of these benchmarks does not vary significantly, due to the fact that the connectivity consumes a small amount of power compared to the memory modules.

Benchark	Cost [gates]	Avg mem latency [cycles]	Avg energy [nJ]
Compress	480775	69.66	13.24
	512232	62.76	13.52
	512332	9.69	13.80
	512532	8.35	14.36
	519388	7.49	14.44
	561112	7.34	14.39
	604941	6.80	14.47
	649849	6.60	14.39
	664029	6.19	14.46
	760543	6.05	14.47
	793971	6.03	14.54
	862176	6.01	14.31
	895604	5.99	14.38
li	480775	57.59	10.42
	494992	57.48	10.43
	512232	50.29	10.70
	512332	9.18	10.98
	512532	7.76	11.54
	605767	6.97	11.57
	664029	6.87	11.58
	760543	6.84	11.59
vocoder	156806	16.37	5.05
	169370	13.28	5.33
	169481	5.09	5.61
	169703	3.60	6.17
	175865	3.40	6.43

Table 1. Selected cost/performance designs for the connectivity exploration.

Table 2 presents the coverage of the pareto points obtained by our memory modules and connectivity exploration approach. Column 1 shows the benchmark, and column 2 shows the category: Time represents the total computation time required for the exploration, Coverage shows the percentage of the points on the pareto curve actually found by the exploration. Average distance shows the average percentile deviation in terms of cost, performance and energy consumption, between the pareto points which have not been covered, and the closest exploration point which approximates them. Column 3 represents the results for the Pruned exploration approach, where only the most promising design points from the memory modules exploration are considered for connectivity space exploration. Column 4 shows the Neighborhood exploration results, where the design points in the neighborhood of the selected points are also included in the exploration, and the last Column shows the results for the brute-force full space exploration, where all the design points in the exploration space are fully simulated, and the pareto curve is fully determined.

The average cost, performance and power distance shows the average distance between the points on the pareto curve and the corresponding closest points found by the exploration, as the percentile deviation on the corresponding axes. If this average distance is small, means that even though a design point on the pareto curve has not been found, another design with very close characteristics (cost, performance, power) is provided (there are no significant gaps in the coverage of the pareto curve).

Benchmark	Category	Pruned	Neighborhood	Full
compress	Time	2 days	2 weeks	1 month
	Coverage [%]	50%	65%	100%
	Avg. cost dist [%]	0.84%	0.59%	0%
	Avg. perf. dist [%]	0.77%	0.60%	0%
	Avg. energ. dist [%]	0.42%	0.28%	0%
vocoder	Time	24 min	29 min	50 min
	Coverage [%]	83%	100%	100%
	Avg. cost dist [%]	0.29%	0%	0%
	Avg. perf. dist [%]	2.96%	0%	0%
	Avg. energ. dist [%]	0.92%	0%	0%

Table 2. Pareto coverage results for our Memory Architecture Exploration Approach.

In the Pruned approach during each Design Space Exploration phase we select for further exploration only the most promising architectures, in the hope that we will find the pareto curve designs without fully simulating the design space. Neighborhood exploration expands the design space explored, by including also the points in the neighborhood of the points selected by the Pruned approach. We omitted the li example from Table 2 due to the fact that the Full simulation was infeasible.

The Pruned approach significantly reduces the computation time required for the exploration. Moreover, full simulation of the design space is often infeasible. While in general, due to its heuristic nature, the pruned approach may not find all the points on the pareto curve, in practice it finds a large percentage of them, or approximates them well with close alternative designs. For instance, the coverage for the vocoder example shows that 83% of the designs on the pareto curve are successfully found by the Pruned exploration. While the Pruned approach does not find all the points on the pareto curve, the average difference between the points on the pareto and the corresponding closest points found by the exploration is 0.29% for cost, 2.96% for performance, and 0.92% for energy. In the compress example the computation time is reduced from 1 month for the Full simulation to 2 days, at the expense of less pareto coverage. However, while only 50% of the compress designs are exactly matched by the Pruned approach, for every pareto point missed, very close replacements points are generated, resulting in an average distance of 1.95%, 1,83%, and 1.76% in terms of cost, performance and power to the closest overall pareto point. Thus, our exploration strategy successfully finds most of the design points on the pareto curve without fully simulating the design space. Moreover, even if it misses some of the pareto points, it provides replacement architectures, which approximate well the pareto designs.

The Neighborhood exploration explores a wider design space than the Pruned approach, providing a better coverage of the pareto curve, at the expense of more computation time. For instance, for the Vocoder example, it finds 100% of the pareto points.

By performing combined exploration of the memory and connectivity architecture, we obtain a wide range of cost, performance and power tradeoffs. Clearly, this type of results are difficult to determine by analysis alone, and require a systematic exploration approach to allow the designer to best trade off the different goals of the system.

7 Summary

In order to optimize the memory system for varying goals, such as power, cost and performance, the system designer would like to evaluate different memory architectures, mixing and matching different memory modules from a memory IP library, and performing Design Space Exploration (DSE). However, while the memory modules are important, often the connectivity between these modules have an equally significant impact on the system behavior. We presented here our Connectivity Exploration approach (ConEx), which trades off the connectivity performance, power and cost, using connectivity modules from a library, and allowing the designer to choose the most promising connectivity architectures for the specific design goals. We generate significant performance improvements for incremental costs, and explore a design space beyond the one traditionally considered, allowing the designer to efficiently target the system goals. By intelligently exploring the design space, we guide the search towards the architectures with the best cost/performance/power characteristics, and avoid the expensive full simulation of the design space.

We presented a set of experiments on large multimedia and scientific examples, where we explored a wide range of cost, performance and power tradeoffs, by customizing the memory and connectivity architecture to fit the needs of the applications.

8 Acknowledgments

We would like to acknowledge and thank Ashok Halambi, Prabhat Mishra, Srikanth Srinivasan, Partha Biswas, Aviral Shrivastava, Radu Cornea and Nick Savoiu, for their contributions to the EXPRESS/ EXPRESSION project.

References

[1] ARM AMBA Bus Specification. *http://www.arm.com/armwww.ns4/html/AMBA?OpenDocument*.

[2] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. *Custom Memory Management Methodology*. Kluwer, 1998.

[3] H-M. Chen, H. Zhou, F. Young, D. Wong, H. Yang, and N. Sherwani. Integrated floorplanning and interconnect planning. In *ICCAD*, 1999.

[4] P. Chou, R. Ortega, and G. Borriello. Interface co-synthesis techniques for embedded systems. In *ICCAD*, 1995.

[5] K.-S. Chung, R. Gupta, and C. L. Liu. Interface co-synthesis techniques for embedded systems. In *ICCAD*, 1996.

[6] R. Cmelik and D. Keppel. Shade: A fast instruction set simulator for execution profiling. Technical report, SUN MICROSYSTEMS, 1993.

[7] J-M. Daveau, T. Ben Ismail, and A. Jerraya. Synthesis of system-level communication by an allocation-based approach. In *ISSS*, 1995.

[8] Y. Deng and W. Maly. Interconnect characteristics of 2.5-d system integration scheme. In *ISPD*, 2001.

[9] W. Forniciari, D. Sciuto, and C. Silvano. Power estimation for architectural exploration of hw/sw communication on system-level busses. In *CODES*, 1999.

[10] Tony Givargis and Frank Vahid. Interface exploration for reduced power in core-based systems. In *ISSS*, 1998.

[11] P. Grun, N. Dutt, and A. Nicolau. Memory aware compilation through accurate timing extraction. In *DAC*, 2000.

[12] P. Grun, N. Dutt, and A. Nicolau. Apex: Access pattern based memory architecture exploration. In *To appear in ISSS*, 2001.

[13] P. Grun, N. Dutt, and A. Nicolau. Connectivity exploration for embedded systems. Technical report, University of California, Irvine, 2001.

[14] P. Grun, A. Halambi, N. Dutt, and A. Nicolau. RTGEN: An algorithm for automatic generation of reservation tables from architectural descriptions. In *ISSS*, 1999.

[15] J. Hennessy and D. Patterson. *Computer Architecture: A quantitative approach*. Morgan Kaufmann Publishers Inc, San Mateo, CA, 1990.

[16] R. Kessler, M. Hill, and D. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. Technical report, University of Wisconsin, 1991.

[17] K. Lahiri, A Raghunatan, G. Lakshminarayana, and S. Dey. Communication architecture tuners: A methodology for hte deisng of high-performance communication architectures for systems-on-chip. In *DAC*, 2000.

[18] Seapahn Maguerdichian, Milenko Drinic, and Darko Kirovski. Latency-driven design of multi-purpose systems-on-chip. In *DAC*, 2001.

[19] P. Mishra, P. Grun, N. Dutt, and A. Nicolau. Processor-memory co-explotation driven by a memory-aware architecture description language. In *International Conference on VLSI Design*, Bangalore, India, 2001.

[20] Sanjiv Narayan and Daniel D. Gajski. Protocol generation for communication channels. In *DAC*, 1994.

[21] S. Przybylski. Sorting out the new DRAMs. In *Hot Chips Tutorial*, Stanford, CA, 1997.

[22] S. Wuytack, F. Catthoor, G. de Jong, B. Lin, and H. De Man. Flow graph balancing for minimizing the required memory bandwith. In *ISSS*, La Jolla, CA, 1996.