Application-Specific Customization and Scalability of Soft Multiprocessors

Deepak Unnikrishnan, Jia Zhao, and Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003

Abstract

Although soft microprocessors are widely used in FPGAs, limited work has been performed regarding how to automatically and efficiently generate soft multiprocessors. In this paper, an automated parallel compilation environment for multiple soft processors which incorporates parallel compilation and inter-processor communication structures is described. A total of eight previously-developed parallel processing benchmarks have been automatically mapped to a varying number of synthesized soft microprocessors in commercial FPGAs. The new automated infrastructure allows for an evaluation of area, performance, and power tradeoffs for a range of architectural choices. Experiments show that our soft-multiprocessor systems consisting of up to 16 processors can offer up to 5x improvement in application performance against their uniprocessor counterparts.

1. Introduction

Over the past few years, soft microprocessors have become integral parts of FPGA-based systems-on-a-programmable chip (SoPC). Increased FPGA area has driven interest in the combination of multiple concurrently-executing soft processors implemented on the same FPGA substrate. For many applications, multiple soft processors provide a flexible and programmable platform for fast application mapping without the need for intensive RTL design. These implementations may be used for a variety of end purposes ranging from initial hardware prototyping to end product design implementation.

Most research on soft multiprocessors has focused on the development of automated synthesis tools for small numbers of processors [11][12] and the investigation of interprocessor communication topologies [6][8][13] Although these projects demonstrated the potential of soft multiprocessors and their usefulness, a comprehensive evaluation of the combined impact of multiprocessor synthesis, interconnect topology choice, and scalability has not been performed for a substantial collection of multiprocessor benchmarks. An understanding of the interaction of these design choices on area, performance, and energy consumption is critical for successful soft multiprocessor implementation.

In this evaluation, the customization and scalability of soft multiprocessor designs are assessed for a sizable number of stream-based parallel computing benchmarks. The StreamIt parallelizing compiler [7] provides a front-end which can automatically map high-level software descriptions of stream applications to multiple processors. Individual soft processors are customized with the SPREE synthesis infrastructure [15]. To determine the effect of soft multiprocessor customization and scaling, a collection of architectural optimizations are considered including interconnection network topology optimization, soft processor pipeline depth variation, inter-processor communication buffer sizing, and unused instruction removal for individual soft processors. Experimental results are determined via synthesis to Altera Stratix II and Stratix III devices. Performance results generated via simulation are verified by mapping designs containing up to sixteen processors to a Stratix III device located on an Altera DE3 board [16].

2. Background

Our work builds on previous research in soft uni- and multiprocessor design and implementation. This previous research encompasses synthesis systems for soft multiprocessors, interconnection techniques for soft multiprocessors, and architectural optimization for individual soft processors. Unlike the experiments described in the paper, these previous efforts have primarily evaluated design area, performance, and energy impacts in isolation (e.g. interconnect-only, processor-only) without considering the underlying tradeoffs in complete system design and synthesis.

Partitioning systems which automatically map stream-based applications described at a high level to multiple soft processors have been considered in several contexts. Initial work by Ravindran et al. [12] focused on the use of an integer linear programming formulation to map tasks to multiple processors while achieving acceptable throughput. A subsequent clustering and packing approach [1] for soft multiprocessor synthesis targeted M-JPEG mapping. This mapping technique considers the assignment of tasks to processors connected via FIFOs in a point-to-point fashion. Both design latency and throughput constraints are considered during synthesis for up to seven processors. A recent soft multiprocessor environment [11] iteratively assigns tasks to a fixed number of processors to balance computation. An evaluation of an M-JPEG application for up to eight processors is considered for both point-to-point and crossbar topologies. Although these synthesis systems provide initial analysis, conclusions regarding appropriate inter-processor topology and mapping effectiveness for a range of automatically-mapped applications are difficult to ascertain. Additionally, these previous projects do not consider processor-specific optimizations and scalable numbers of processors connected in mesh topologies.

A series of recent studies have examined appropriate on-chip interconnect approaches for multiprocessors implemented in FPGAs. Saldana et al. [13] considered a range of inter-processor topologies, such as meshes, hypercubes, star, and fully-connected topologies for multiprocessors containing up to 64 nodes. The study concluded that all topologies, except fully-connected and star, could be easily synthesized to FPGAs. Although interesting, this study did not consider the communication patterns of applications mapped to multiple processors or their need for synchronization. Other network-on-chip (NoC) on FPGA studies [6] concluded that NoCs can significantly outperform on-chip buses and provide system scalability. Kapre et al. [8] observed that time-switched and packet-switched butterfly fat trees can be effectively mapped to FPGAs. Although not comprehensive, several experiments [2], which manually mapped sorting applications to hypercube networks, observed considerable application speedup. In general, none of these previous FPGA-based NoC studies considered a range of applications automatically mapped to a large number (e.g. > 10) of soft processors.

The optimization of soft microprocessors for area, performance, and energy consumption has been an active area of experimentation for several years. Yiannacouras et al. [15] examined the effect of optimizations such as shifter implementation, pipelining, and instruction set subsetting on soft microprocessor performance. It was found that the features of the optimal processor architecture for each application often vary greatly. More recently, researchers have examined the benefit of soft microprocessor multithreading to improve application performance and energy savings. Dimond et al. [3][4] examined the use of multi-threading, custom instruction coding and instruction scheduling as techniques to maintain high throughput while minimizing processor area and energy.

Studies by Labrecque and Steffan [9] and Fort et al. [5] considered processor pipeline length variation and custom functional units as techniques to promote increased multithreading. The former study was later extended [10] to consider a performance comparison between a single multi-threaded soft microprocessor, multiple multi-threaded soft processors and multiple single-threaded soft processors.

3. Soft Multiprocessor Details and Tradeoffs

The explored optimizations are driven by multiprocessor architectures which can be customized through the use of various parameters. These parameters allow for the customization of both the individual soft microprocessors via pipeline depth determination and instruction set subsettting and the interconnection network via automatic topology generation and communication buffer sizing. The main characteristics of the processor and interconnection components used in this work include the following details.

Soft processor architecture: The 32-bit soft processors used in our evaluation are created from the SPREE soft processor generator [15]. The base RISC architecture used by this generator supports a subset of the standard MIPS instruction set architecture (ISA). Datapath pipelines of multiple stages are supported. The three-stage pipelines consist of fetch/decode, execute and write back stages. Four-stage pipelines extend three-stage pipelines by splitting fetch/decode into two separate stages. Five-stage pipelines extend the four-stage pipelines by including an additional execute stage. The soft processor architecture currently does not support caches, although this is not a significant issue for stream-based applications which do not exhibit temporal data locality. Our generated processors also do not support off-chip memories. The stream algorithms operate on test data generated internally within the multiprocessor system. Although support for off-chip memories will be a nice enhancement, we leave this as future work. Dynamic branch prediction, exceptions, and floating point operations are also not supported.

Soft multiprocessor interconnection: Customizable soft processors in our generated multiprocessors are interconnected with unidirectional FIFO buffers. FIFO empty/full status is determined simultaneously with processor FIFO read/writes operations. Unsuccessful transactions are repeated. FIFOs are located at memory mapped processor locations (i.e. each processor is able to address FIFOs as though they are memory locations through load/store instructions in a single cycle). Soft multiprocessors can be configured in either mesh or direct point-to-point topologies. In **meshes**, each processor is assigned to a specific (x, y) location. Inter-processor connections between non-adjacent processors are made as a series of inter-processor hops. In this case, each processor uses at most eight unidirectional FIFOs for North, South, East, and West communication. For each hop, an intermediate processor must read the data from one FIFO and write it to another.

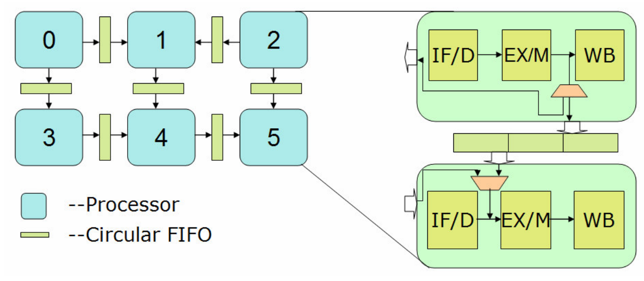
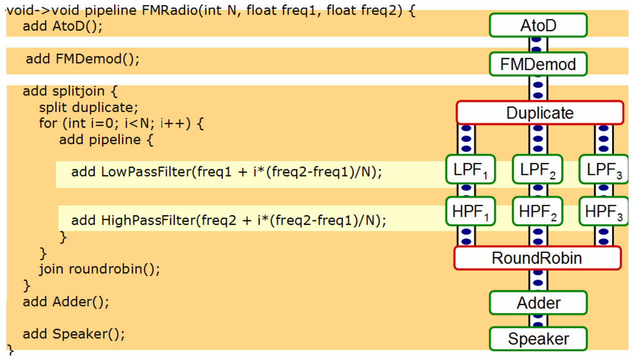


Figure 1. Inter-processor interconnection scheme

As an example of mesh interconnection, Figure 1 illustrates the software FM Radio application [7] parallelized over six processors. In this example, each processor consists of a three-stage in-order pipeline, register file and local on-chip instruction/data memories. The FIFO buffers are tightly integrated into the bypass paths of the processor pipeline using memory mapped input/output ports. In contrast to mesh topologies, **point-to-point topologies** involve direct FIFO connections between producers and consumers of data, irrespective of topology. This configuration indicates that some processors may have numerous input FIFOs while others may have only one.

It is important to note that our synthesized soft multiprocessor architectures differ from current ASIC implementations of multicore stream processors, such as RAW [14]. In RAW multiprocessors, each RISC processor is augmented with a pipelined crossbar switch which is used as an inter-processor network-on-chip component. The crossbar is configured on a cycle-by-cycle basis based on a pre-compiled schedule. This approach has the benefit of providing a common communication substrate for many applications mapped to the same chip at the cost of increased hardware resources. The per-application synthesis of our soft processors eliminates the need for this overhead.

Application language: Supported applications are written in StreamIt [7], a stream-based high-level language. StreamIt was initially created for mesh-based tiled multiprocessor architectures, such as RAW, that have predictable computation and communication schedules. The language and associated compiler effectively isolate the details of the multiprocessor architecture from the application designer. The StreamIt language represents applications as a hierarchical series of functional operations called filters. Each filter consists of initialization and steady state routines. Filters exchange data with each other using push(), pop() and peek() methods via FIFOs. Figure 2 (taken from [7]) illustrates simplified software FM Radio application written in StreamIt language. As shown in the figure, StreamIt exposes the natural parallelism inherent in an application. Each application program consists of a hierarchical composition of many stream structures such as pipelines, splitters (duplicate) and joiners (roundrobin).



```
void->void pipeline FMRadio(int N, float freq1, float freq2) {
  add AtoD();

  add FMDemod();

  add splitjoin {
    split duplicate;
    for (int i=0; i<N; i++) {
       add pipeline {

         add LowPassFilter(freq1 + i*(freq2-freq1)/N);

         add HighPassFilter(freq2 + i*(freq2-freq1)/N);
       }
    }
    join roundrobin();
  }
  add Adder();

  add Speaker();
}
```

Figure 2. Software FM Radio in StreamIt [7]

Independent pipelined, parallel streams which diverge at a splitter and converge at a joiner form a split-join. Splitters send a copy of each data item into each parallel stream. A joiner then combines the results of the streams in a prescribed fashion.

4. Soft Multiprocessor Synthesis Flow

As stated in Section 1, a significant differentiating factor between this work and previous soft multiprocessor projects is the use of fully automatic compilation and use of a substantially-sized, unmodified benchmark set to evaluate both soft multiprocessor computation and interconnect tradeoffs. Existing tools (StreamIt, SPREE) are used along with new supporting tools to create a novel flow. The framework allows users to adjust a variety of multiprocessor system parameters such as the number of soft processors, the interconnect topology, and the number of pipeline stages for each processor. The framework also offers choices for application specific customization at the microarchitectural and application level.

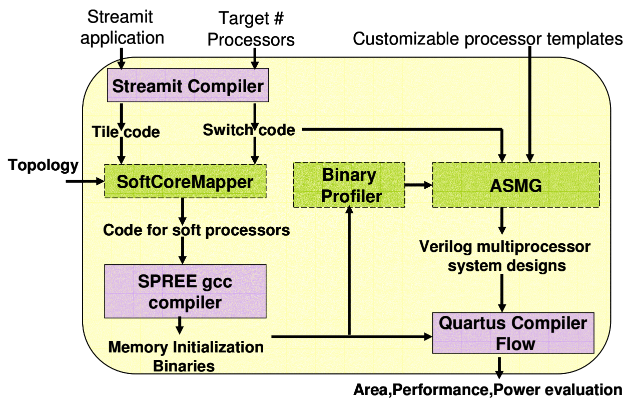


Figure 3. Soft multiprocessor synthesis framework

The main steps in the soft multiprocessor generator appear in Figure 3. Details of the five main steps are included in the following subsections.

The **StreamIt** compiler, which was developed to support the RAW architecture, has been optimized to map stream-based applications to multiple processors. Although the StreamIt compiler contains many passes (the interested reader is directed to [7] for more details), the two StreamIt algorithms most relevant to soft multiprocessor generation are partitioning and scheduling. The role of partitioning is to assign filters so that each processor has roughly the same amount of operations, thus ensuring balanced operational throughput. To support balancing, StreamIt partitioning algorithms attempt to form a number of clusters that is equal to the number of processors. Partitioning offers an opportunity to split large filters into pieces (fission) or group filters together (fusion). Two specific partitioning algorithms are used. Greedy partitioning clusters filters based on their size (e.g. the number of operations) until the number of clusters equals the number of target processors. Dynamic programming-based partitioning clusters filters in adjacent pipeline stages together to form larger filters. The assignment of filters/clusters to specific processors in a mesh configuration takes place via a simulated annealing algorithm.

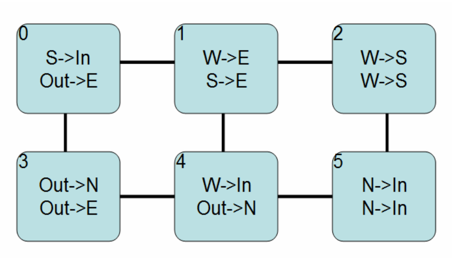


Figure 4. Communication for a mesh topology

Although the original StreamIt stream scheduling algorithm for meshes can be reused for the new mesh-based topology, the scheduler is changed to support point-to-point topologies. For example, consider the mesh-based communication pattern shown in Figure 4. Under steady state conditions, Processor 3 produces two data values. The first value is routed north (N) to Processor 0 and the second one is routed east (E) to processor 4. Processors 0 and 4 process the incoming data and forward the results through Processors 1 and 2 to Processor 5. In this example, it takes multiple cycles for the data values produced by Processors 0 and 4 to be transferred to their final destinations since the data must hop through intermediate processors.

A modified schedule is shown in Figure 5 (note processor numbers are used rather than N, S, E, and W). The new schedule is used to coordinate point-to-point inter-processor communication. To derive the modified switch schedule, a data flow graph is generated from the mesh-style switch schedule produced by StreamIt. In this graph, nodes are represented by processors and edges are represented by switch operations that transfer the data. A depth-first traversal is performed in the graph from each data source to all the data's destinations. Any inter-tile data hop edge discovered during this traversal is eliminated. Finally, edges are inserted in the graph between data sources and destinations. The updated data flow graph is then retimed to account for the removed hops in the point-to-point topology.

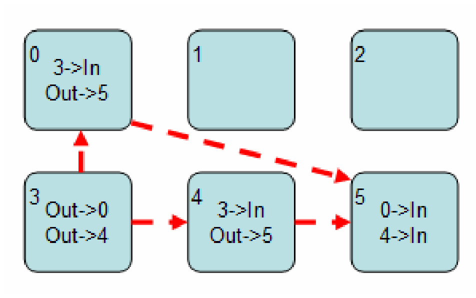


Figure 5. Communication for point-to-point

The **SoftCoreMapper** modifies filters to include inter-processor read/write primitives. In the case of clustered filters, some inter-filter communication takes place via soft microprocessor registers for filters assigned to the same processor. Communication between processors takes place via FIFOs. SoftCoreMapper parses StreamIt output to identify parts of the application where data exchange occurs and inserts appropriate FIFO operations. Synchronization primitives are included for each read/write operation.

The **SPREE** soft processor generator creates an internal representation for the processors based on the parameters noted in Section 3 (e.g. pipeline depth, multiplier usage). The generator automatically ensures appropriate coverage for instructions in the ISA in creating the data and control paths. Communication FIFOs and supporting interface logic are also instantiated during processor creation. SPREE also provides a modified MIPS gcc compiler to generate application binaries for the customized processors.

The **binary profiler** extracts application specific instruction set usage from disassembled binaries so that unused processor data and control path resources can be removed. The **application specific multiprocessor generator (ASMG)** modifies the internal representation of each processor and associated communication based on the results of the binary profiler. The instruction set of each processor is minimized to suit the assigned tasks. Information regarding inter-processor communication, individual processor instruction set usage, and pipeline depth is used to generate the complete Verilog model of the multiprocessor system. The MIPS compiler has not been modified to target the soft processors. Standard Quartus FPGA compile serves as a final step.

5. Experimental Results

Like other stream processing evaluations [7], throughput and overall execution time are assessed. Execution time is obtained by multiplying the cycles per output determined with Modelsim-Altera 6.1g with the maximum design frequency reported by Quartus v8.0. All designs were compiled with a timing constraint of 150 MHz. Altera's PowerPlay power analyzer was used to determine core dynamic power results.

A set of eight StreamIt benchmarks [7], were parallelized over multiprocessor systems consisting of 6, 9 and 16 processors using our automated synthesis flow. The benchmarks consist of a mix of sorting (*Bitonic*), signal processing (*FMRadio, Equalizer, Autocor, Lattice, Filterbank, FFT*), and security (*DES*) applications. Since the generated SPREE multiprocessor designs do not support floating point arithmetic, the benchmarks were executed using equivalent integer operations. All the designs were mapped to a 90nm Stratix II EP2S180 device. Designs consisting of 16 processors were implemented and executed on a 65nm EP3SL150 Stratix III device on a DE3 board to verify functionality.

5.1 Interconnect topology variation

In this experiment, the run time performance of a set of four applications for mesh and direct point-to-point style network topologies is evaluated. Application speedups which consider both changes in clock cycles and design frequency are shown in Figure 6 for three-stage soft processors with complete instruction sets. Overall, point-to-point interconnect outperforms a mesh-style network for all applications by a factor of between 1.1x and 2x. Point-to-point topologies gain significant cycle speedups due to reduced synchronization overhead from the elimination of network hops. Point-to-point topologies consumed 28.6% less cycles when compared to mesh-style topologies on average. Interestingly, point-to-point topologies also give slightly better performance in terms of design frequency.

For a 16 processor system, the point-to-point topology shows an average 2% improvement in design frequency. This frequency improvement results from the removal of unnecessary input/output FIFO ports. In a mesh-style topology, many processors need close to four ports as these nodes perform data forwarding in addition to computation. The improvement is observed even though processors with large data fan-outs (sources) and fan-ins (sinks) in point-to-point topologies typically require more than four ports. For example, in a mesh-style topology for a 16 processor FM Radio application, the average port usage per processor is approximately 3, while for a point-to-point topology, the average port usage per processor is approximately 2. The processors executing splitter and joiner filters in the point-to-point topology for this application requires 11 and 9 ports, respectively. For smaller designs, like AutoCor, cycles per output increases or remains unchanged when parallelized over larger multiprocessor systems since increased communication costs dominate over the reduced computation costs.

The clock frequency differences between individual mesh and point-to-point implementations of each design are less than 2.3 MHz. Overall, implementation frequencies range between 118.0 and 128.7 MHz. In all designs, the critical path is located within the three-stage processor logic. Thus, the addition of point-to-point links does not degrade the maximum design frequency significantly, although the addition of more point-to-point links may make the FPGA more difficult to route. The number of point-to-point links scales linearly with processor count in most designs. The effectiveness of our point-to-point results validates previous topology research for FPGAs [13], which did not consider specific applications.

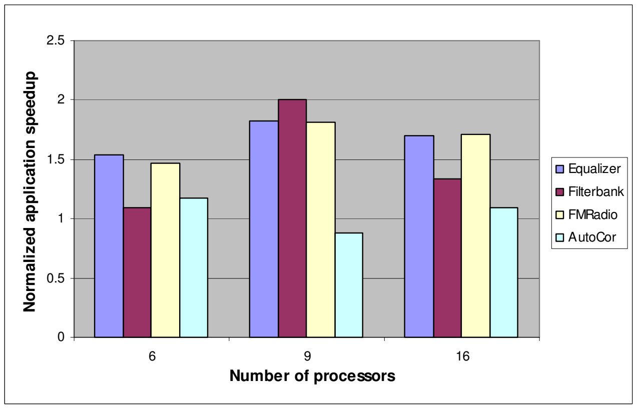


Figure 6. Point-to-point speedups vs. mesh

5.2 Customization of pipeline depth

The choice of microarchitectural pipeline depth of each processor influences the overall throughput of the application. The impact of 3, 4 and 5 stage pipelining on application performance is studied. Deepening individual processor pipelines from three to four stages can give substantial performance improvements of 22% on average at a 9.6% increase in area. Figure 7 shows the relative execution time per output for six stream benchmarks mapped over 16 processors. The four-stage pipeline multiprocessor systems generally give better performance than their three-stage and five-stage counterparts. The critical paths of the multiprocessor systems for all designs are within the individual processors. In three-stage pipelines, the critical path is located between the register file and memory write-back logic through the branch predictor. For four- and five-stage pipelines, the critical path is between the register file and memory write-back logic through the integer multiplier.

The relative performance improvement of the four-stage pipelines results from improved per-processor performance. On average, the maximum design frequency improves by 26% from 118 MHz to 149 MHz as a transition from three- to four-stage pipelines is made. However, the maximum design frequency remains largely unchanged when the pipeline depth is increased to five since the critical path remains between register file and memory write-back logic through the integer multiplier. As more stages are added to the pipeline, an increase in the cycles per output is observed for all the applications. When compared to three-stage pipeline multiprocessor systems, the cycles per output increases by 5% for four-stage systems and by 14% for five stage systems. The trends are consistent for 6 and 9 processor design cases.

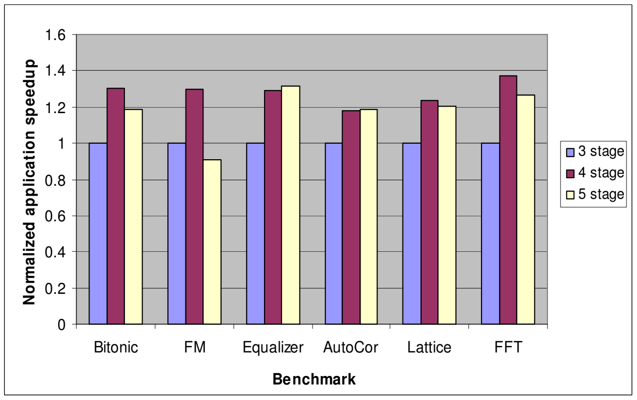


Figure 7. Impact of pipeline stage count on application performance for 16 processor systems

The increase in cycles can be attributed to two factors. First, the processors generated by the SPREE framework use interlocking to resolve data hazards. As pipeline depth increases, it becomes increasingly difficult for the compiler to support independent instructions within the interlocking window, which introduces more stalls. SPREE uses a simple static *branch not taken* prediction scheme [15]. In general, branch mispredictions can be costly in deeper pipelines. Also, it can be difficult to support branch delay slot instructions in deeper pipelines, causing more stalls. Stalls due to branch mispredictions and data hazards in individual processor pipelines can ripple across multiple processors in communication-intensive stream applications.

5.3 Customization of communication buffer depth

Stream applications are often communication-intensive since they consist of a pipeline of tasks. In many cases, communication overhead must be amortized to achieve effective performance. Figure 8 shows the variation of normalized application speedups with varying FIFO sizes for five benchmarks mapped to 9 processors using previously-discussed topology and processor pipeline preferences.

For large applications, we observe that the cycle reduction (e.g. throughput) increased once a critical FIFO size is reached. For example, for Bitonic sort, the application speedup improved by over 20% when FIFO size was increased from 8 to 16 words. Smaller applications, such as AutoCor and Lattice, benefit little from an increase in buffer sizes due to limited inter-processor communication. In general, well-matched communication buffers prevents communication stalls without wasting system resources.

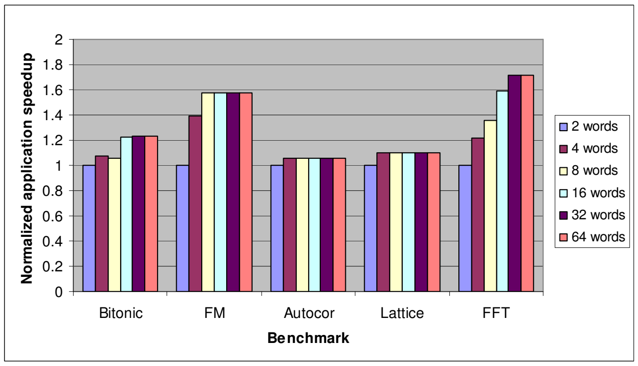


Figure 8. Impact of interconnect buffer sizing on application cycles per output

Each soft-multiprocessor system consists of customizable processors which communicate using simple FIFO buffers. In previous work [11], communication controllers (CC) were used to interconnect processors. Each CC requires 468 four-input LUTs and about 128 flip flops for four word storage. In contrast, our synthesis results indicate that each FIFO requires only 11 LUTs, 72 registers and 128 memory bits, a small fraction of available FPGA resources.

5.4 Soft-multiprocessor ISA subsetting

In general, soft microprocessors use only a portion of their ISA for filter implementation. Figure 9 shows the average instruction set usage for the eight benchmarks mapped over a 16 processor system. Most applications, except DES, use less than fifty percent of the instructions supported by the instruction set. Smaller applications, such as Lattice, consume only about 26% of the available instructions. For a given application, the use of instructions per processor in the multiprocessor system is highly variable. For example, the instruction usage of each processor in a sixteen processor system for software FM Radio application varies between 20% and 50%.

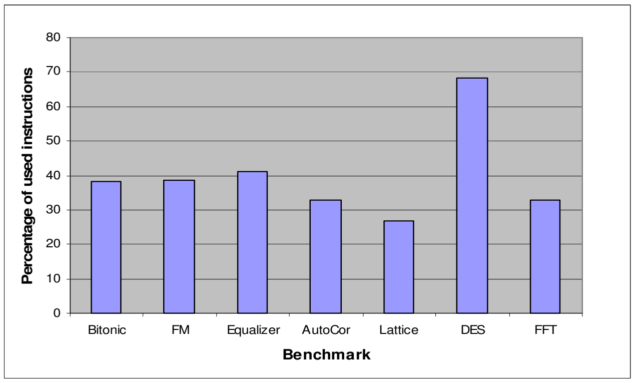


Figure 9. Average instruction set usage per design

On average, instruction set customization yielded a 27% percent improvement in area for the seven multiprocessor designs. A modest 4.2% improvement in maximum design frequency was also observed for the customized designs. The majority of the area savings were obtained in the decode logic and control circuitry in each processor. On average, the power consumption of subsetted designs consistently decreased by about 30% for 6, 9 and 16 processor designs.

5.5 Application scalability

Figure 10 shows the application speedup for the set of eight benchmarks normalized to a single soft core processor system for the parameters described in previous subsections. The cycles per output and maximum design frequency in MHz are given in Table 1. The performance of larger applications such as DES, Bitonic and Filterbank improves by about a factor of 5x when parallelized over sixteen processors. The speedup improvement is primarily attributed to the amount of coarse-grained task-level parallelism inherent in these applications. The performance of smaller benchmarks, such as Autocor and Lattice, degrades when parallelized over multiple processors. The performance degradation is due to increased communication overhead in larger multiprocessor systems. This effect is also seen as Fliterbank scales from 9 to 16 processors. As seen in Table 1, the maximum frequency of all the designs degrades when more soft processors were embedded on the FPGA substrate. On average, an 11% frequency degradation is observed when all applications are mapped to 16 processors. The critical paths in these designs are within the processors, between the register file and memory through the branch predictor.

Figure 11 shows the dynamic core power consumption at 50 MHz for 1, 4, 9 and 16 processor designs for two benchmarks. A single processor design consumes about 60-100 mW of dynamic power. The dynamic power consumption scales up linearly when the number of processors is increased from one to four. The power consumption for 9 and 16 processor designs for Bitonic sort shows mostly linear growth. In larger designs, each processor switches fewer times on average to produce the same number of outputs. However, increased communication and synchronization power costs increase the overall dynamic power.

5.6 Combined impact of customizations

In this section, the combined impact of all the optimizations is considered. The application speedup of four benchmarks under their best case and worst case configurations are considered for 16 processors. The best case configuration is the choice of microarchitectural pipeline depth, interconnection topology and instruction set that gives the best application performance in absolute execution time. The worst case configuration uses the multiprocessor parameters that give the worst case application performance. Figure 12 shows the normalized application speedup of the best case configurations of four benchmarks against their worst case configurations for each optimization and in total. On average, the performance of

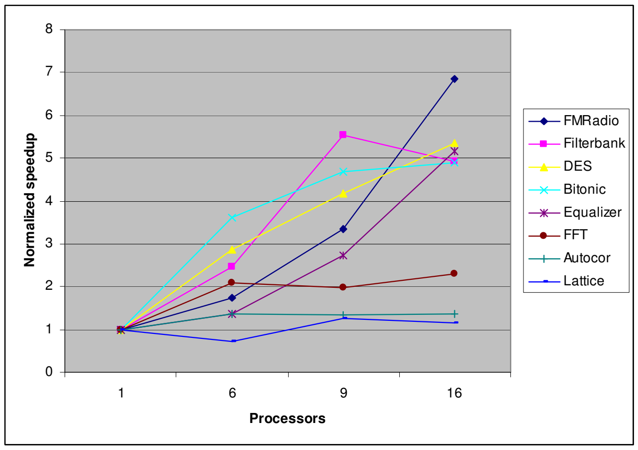


Figure 10. Application speedup scaling

Table 1. Clock cycle counts and frequency (MHz)

	Processors			
Bench-mark	1	6	9	16
	Cycle/Freq	Cycle/Freq	Cycle/Freq	Cycle/Freq
FMR	17728/131	9816/127	4930/122	2392/121
FB	7986/131	3021/123	1339/122	1503/121
Bitonic	13511/131	3628/127	2883/131	2470/118
DES	69094/131	23338/127	16452/130	11527/117
Eq	13862/131	9812/127	4765/123	2475/121
FFT	137/131	64/127	63/121	54/119
Autocor	306/131	211/123	214/122	208/121
Lattice	55/131	75/130	40/121	43/122

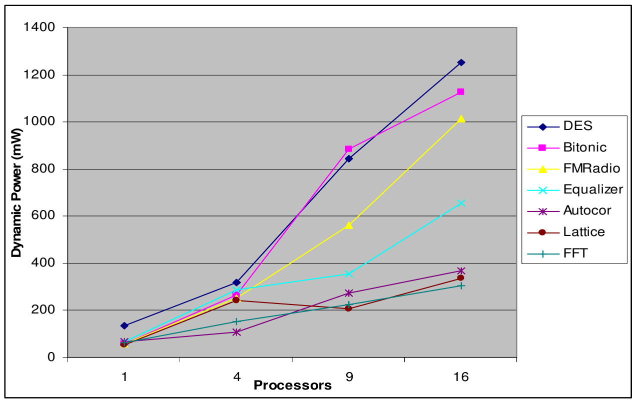


Figure 11. Power consumption scaling

applications improves by a factor of 2.1x when all the customizations are applied on the soft-multiprocessor system. The primary factors contributing to the overall application speedup are the choice of the pipeline stage depth and the choice of the interconnection topology. Although instruction subsetting saves considerable area, it contributes only 4% improvement to the overall application speedup. Our results indicate that a judicious choice of interconnection topologies and microarchitectural features can give significant performance and area benefits in soft-multiprocessor systems. Previously [15], it was determined that a single SPREE soft processor demonstrates an 11% speedup over an Altera NIOS II/s processor. Our results add to this improvement.

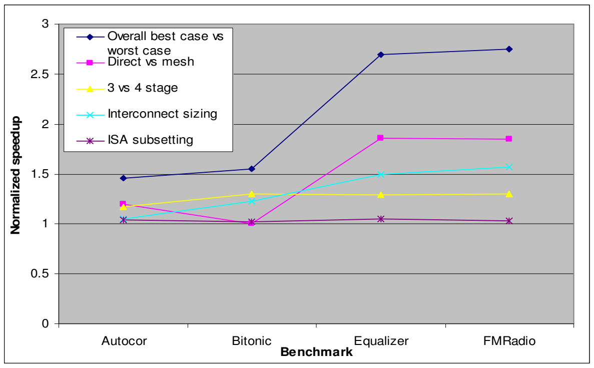


Figure 12. Combined impact of soft-multiprocessor optimizations for 16 processors

6. Conclusion and Future Work

An automatic parallel compilation and synthesis environment for soft multiprocessors has been presented. Our results indicate larger soft multiprocessor systems benefit from point-to-point interconnection topologies rather than more common meshes and microarchitectural optimizations such as instruction subsetting, inter-processor buffer sizing and pipeline depth variation yield significant performance and area benefits. In the future, we plan to improve the performance of the soft multiprocessor designs by more aggressive pipelining and better branch prediction.

Acknowledgments: This work was funded by a grant from Altera Corporation. We acknowledge the efforts of Nicolas Vincent in the early stages of this project.

7. References

[1] J. Cong, G. Han and W. Jiang, "Synthesis of an application-specific soft multiprocessor system", *ACM/SIGDA Int. Symposium on Field Programmable Gate Arrays*, Feb. 2007, pp. 99-107.

[2] J. P. Derutin, et al., "Design of a scalable network of communicating soft processors on FPGA", *International Workshop on Computer Architecture for Machine Perception and Sensing,* Sept. 2006, pp. 184-189.

[3] R. Dimond, O. Mencer and W. Luk, "Combining instruction coding and scheduling to optimize energy in system-on-FPGA", *IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2006, pp. 175-184.

[4] R. Dimond, O. Mencer, and W. Luk, "CUSTARD: A customizable, threaded FPGA soft processor and tools," *IEEE Int. Conference on Field Programmable Logic and Applications*, Aug. 2005, pp. 1-6.

[5] B. Fort, D. Capalija, Z. Vranesic, and S. Brown, "A multithreaded soft processor for SoPC area reduction", *IEEE Symposium on Field-Programmable Custom Computing Machines,* Apr. 2006, pp. 131-142.

[6] H. Freitas, D. Colombo, F. Kastensmidt, and P. Navaux, "Evaluating network-on-chip for homogeneous embedded processors in FPGAs", *IEEE Int. Symposium on Circuits and Systems*, May 2007, pp. 3776-3779.

[7] M. I. Gordon, et al., "A Stream Compiler for Communication Exposed Architectures," *In International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002, pp. 291-303.

[8] N. Kapre, et al., "Packet switched vs. time multiplexed FPGA overlay networks", *IEEE Symposium on Field-Programmable Custom Computing Machines,* Apr. 2006, pp. 205-216.

[9] M. Labrecque and J. G. Steffan, "Improving pipelined soft processors with multithreading", *Int. Conference on Field Programmable Logic and Applications*, Aug. 2007, pp. 210-215.

[10] M. Labrecque, P. Yiannacouras, and J. G. Steffan, "Scaling soft processor systems", *IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2008, pp. 195-205.

[11] H. Nikolov, T. Stefanov, and E. Deprettere, "Systematic and automated multiprocessor system design, programming, and implementation, *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems,* vol. 27, no. 3, Mar. 2008, pp. 542-555.

[12] K. Ravindran, N. Satish, Y. Jin, and K. Keutzer, "An FPGA based multiprocessor system for IPv4 packet forwarding", *Int. Conference on Field Programmable Logic and Applications*, Aug. 2005, pp. 487-492.

[13] M. Saldana, et al., "Routability of network topologies," *IEEE Transactions on VLSI Systems*, vol. 15, no. 8, Aug. 2007, pp. 948-951.

[14] M. B. Taylor, et al., "Evaluation of the RAW Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams," *International Symposium on Computer Architecture*, June 2004, pp. 2-13.

[15] P. Yiannacouras, J. G. Steffan, and J. Rose, "Application-specific customization of soft processor microarchitecture", *ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, Feb. 2006, pp. 201-210.

[16] Altera Corporation, Development and Education 3 (DE3) board manual 2008