IMPROVING PIPELINED SOFT PROCESSORS WITH MULTITHREADING

Martin Labrecque and J. Gregory Steffan

Department of Electrical and Computer Engineering, University of Toronto
email: {martinl,steffan}@eecg.toronto.edu

ABSTRACT

Designers of FPGA-based systems are increasingly including *soft processors*—processors implemented in programmable logic—in their designs. Any combination of area, clock frequency, performance, and power may be of importance in the choice of a soft processor design to use, motivating *area efficiency* as the best metric with which to compare potential designs. In this paper we demonstrate that 3, 5, and 7-stage pipelined multithreaded soft processors are 33%, 77%, and 106% more area efficient than their single-threaded counterparts, the result of careful tuning of the architecture, ISA, and number of threads.

1. INTRODUCTION

FPGAs that are used to implement a system-on-chip often contain one or more *soft processors*: compared with custom logic, a soft processor is easier to program (using `C` instead of hardware-description language), flexible, and can be used to manage and communicate with other custom accelerators in the design. For soft processors, raw performance is often not crucial—otherwise an off-chip or on-chip (if available) hard processor would be preferable. Other metrics or their combination may instead be more important, such as minimizing the area of the processor, matching the clock frequency of another key component in the same clock domain, handling requests or interrupts within specified time constraints (i.e., real-time), or processing a stream of requests or data at a sufficient rate. Flexibility and control over performance/area trade-offs in the soft processor design space are key, and hence when comparing soft processor designs a summarizing metric that combines area, frequency, and cycle count such as *area efficiency* is most relevant.

1.1. Multithreaded Soft Processors

A promising way to improve area efficiency is through the use of multithreading. Fort *et al.* [1] presented a 4-way multithreaded soft processor design, and demonstrated that it provides a significant area savings over having four soft processors—but with a moderate cost in performance. In particular, they show that a multithreaded soft processor need not have hazard detection logic nor forwarding lines, so long as the number of threads matches the number of pipeline stages such that all instructions concurrently executing in different stages are independent. Researchers in the CUSTARD project [2] have also developed a similar pipelined 4-stage 4-way multithreaded soft processor.

This work is supported by NSERC, NATEQ, and Altera.

Rather than comparing with multiple soft processors, in this paper we show that a multithreaded soft processor can be better than one that is single-threaded. In particular we make four main contributions by demonstrating: (i) that multithreaded soft processors are more area-efficient and are capable of a better sustained instructions-per-cycle (IPC) than single-threaded soft processors; (ii) that these benefits increase with the number of pipeline stages (at least up to and including 7-stage pipelines); (iii) that careful optimization of any unpipelined multi-cycle paths in the original soft processor is important, and (iv) that careful selection of certain ISA features, the number of registers, and the number of threads are key to maximizing area-efficiency.

2. SOFT PROCESSOR INFRASTRUCTURE

In this section we briefly describe our infrastructure for designing and measuring soft processors, including the SPREE system for generating single-threaded pipelined soft processors, our methodology for comparing soft processor designs, our compilation infrastructure, and the benchmark applications we study.

SPREE: We use the SPREE system [3] to generate a wide range of soft processor architectures. SPREE takes as input ISA and datapath descriptions and produces RTL which is synthesized, mapped, placed, and routed by `Quartus 5.0` [4] using the default optimization settings. The generated processors target Altera Stratix FPGAs, and we currently synthesize for a `EP1S40F780C5` device—a mid-sized device in the family with the fastest speed grade. We determine the area and clock frequency of each soft processor design using the arithmetic mean across 10 seeds (which produce different initial placements before placement and routing) to improve our approximation of the true mean. For each benchmark, the soft processor RTL design is simulated using `Modelsim 6.0b` [5] to (i) obtain the total number of execution cycles, and (ii) to generate a trace which is validated for correctness against the corresponding execution by an emulator (MINT [6]).

Measurement: For Altera Stratix FPGAs, the basic logic element (LE) is a 4-input lookup table plus a flip-flop—hence we report the area of these processors in *equivalent*

LEs, a number that additionally accounts for the consumed silicon area of any hardware blocks (e.g. multiplication or block-memory units). For the processor clock rate, we report the maximum frequency supported by the critical path of the processor design. To combine area, frequency, and cycle count to evaluate an optimization, we use a metric of *area efficiency*, in million instructions per second (MIPS) per thousand equivalent LEs. It is important to have such a summarizing metric since a system designer may be most concerned with soft processor area in some cases, or frequency or wallclock-time performance in others. Finally, we obtain dynamic power metrics for our benchmarks using Quartus' Power Play tool and report nano-Joules per instruction (nJ/instr), discounting the power consumed by I/O pins.

Single-Threaded Processors: The single-threaded processors that we compare with are pipelined with 3 stages (`pipe3`), 5 stages (`pipe5`), and 7 stages (`pipe7`). The 2 and 6 stage pipelines were previously found to be uninteresting [3], hence we study the 3, 5, and 7 stage pipelines for even spacing. All three processors have hazard detection logic and forwarding lines for both operands. The 3-stage pipeline implements shift operations using the multiplier, and is the most area-efficient processor generated by SPREE [3] (at 1256 equiv. LEs, 78.3 MHz). The 5-stage pipeline also has a multiplier-based shifter, and implements a compromise between area efficiency and maximum operating frequency (at 1365 equiv. LEs, 86.79 MHz). The 7-stage pipeline has a barrel shifter, is the largest processor, and has the highest frequency (at 1639 equiv. LEs, 100.59 MHz). `Pipe3` and `pipe5` both take one extra cycle for shift and multiply instructions, and `pipe3` requires an extra cycle for loads from memory.

Compilation: Our compiler infrastructure is based on modified versions of `gcc` 4.0.2, `Binutils` 2.16, and `Newlib` 1.14.0 that target variations of the 32-bits MIPS I [7] ISA; for example, we can trade support for Hi/Lo registers with 3-operand multiplies, enable or disable branch delay slots, and vary the number of architected registers used. Integer division is implemented in software.

Benchmarking: We evaluate our soft processors using the 10 embedded benchmark applications described in Table 1, which are divided into 4 categories: dominated by loads (`L`), shifts (`S`), multiplies (`M`) or by none of the above (other, `O`).[1] By selecting benchmarks from each category, we also create 3 multiprogrammed mixes that each execute until the completion of the shortest application in the mix (applications are chosen from the left to right of the mix until all threads available on the processor are filled). We measure multithreaded processors using (i) multiple copies of the same program executing as separate threads (with separate data memory), and (ii) using the multiprogrammed mixes.

[1]The benchmarks chosen are a subset of those used in previous work [11] since for now we require both data and instructions to each fit in separate single MegaRAMs in the FPGA.

Table 1. Benchmark applications evaluated.

Source	Benchmark	Modified	Dyn. Instr. Counts	Category
MiBench [8]	BITCNTS	di	26,175	L
XiRisc [9]	BUBBLE_SORT	-	1,824	L
	CRC	-	14,353	S
	DES	-	1,516	S
	FFT*	-	1,901	M
	FIR*	-	822	M
	QUANT*	-	2,342	S
	IQUANT*	-	1,896	M
	VLC	-	17,860	L
RATES [10]	GOL	di	129,750	O

Mix	T0	T1	T2	T3	T4	T5	T6
1	FFT	QUANT	BUBBLE_SORT	GOL	FIR	CRC	VLC
2	FIR	CRC	VLC	GOL	IQUANT	DES	BITCNTS
3	IQUANT	DES	BITCNTS	GOL	FFT	QUANT	BUBBLE_SORT

* Contains multiply
di Reduced data input set and number of iterations
Categories: dominated by (L)oads, (S)hifts, (M)ultiplies, (O)ther

3. MULTITHREADING A SOFT PROCESSOR

Commercially-available soft processors such as Altera's NIOS II and Xilinx's Microblaze are both single-threaded, in-order, and pipelined, as are our SPREE processors (described in the previous section). Such processors require hazard detection logic and forwarding lines for correctness and good performance. These processors can be multithreaded with minimal extra complexity by adding support for instructions from multiple independent threads to be executing in each of the pipeline stages of the processor—an easy way to do this is to have as many threads as there are pipeline stages. This approach is known as *Fine-Grained Multithreading* (FGMT), and is also the approach adopted by Fort *et al.* [1] and the CUSTARD project [2].[2]

In this section we evaluate several SPREE processors of varying pipeline depth that support fine-grained multithreading. Since each pipe stage executes an instruction from an independent thread, these processors no longer require hazard detection logic nor forwarding lines—which as we show can provide improvements in both area and frequency. Focusing on our base ISA (MIPS), we also found that load and branch delay slots are undesirable, which makes intuitive sense since the dependences they hide are already hidden by instructions from other threads—hence we have removed them from the modified version of the ISA that we evaluate.

To support multithreading, the main challenge is to replicate the hardware that stores state for a thread: in particular, each thread needs access to independent architected registers and memory. In contrast with ASICs, for FPGAs the relative latency of on-chip memory vs logic latency grows very slowly with the size of the memory—hence FPGAs are amenable to implementing the replicated storage required

[2]The CUSTARD group also investigated *Block Multi-Threading* where threads are switched only at long-latency events, but found this approach to be inferior to FGMT.

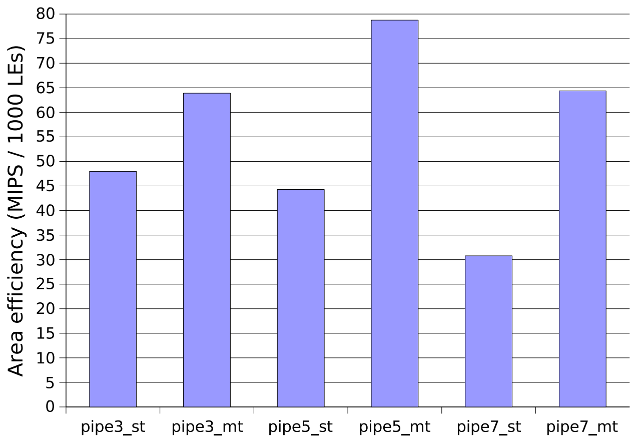


Fig. 1. Area efficiency of single-threaded (`st`) and multithreaded (`mt`) processors with varying pipeline depths, where the number of threads is equal to the number of stages for the multithreaded processors. Results are the mean across all single benchmarks (i.e., not the mixes).

for multithreading. We provide replicated program counters which are selected in a round-robin fashion each cycle. Rather than physically replicating the register file, which would require the addition of costly multiplexers, we index different ranges of a shared physical register file by appropriately shifting the register numbers. We implement this register file using block memories available on Altera Stratix devices; in particular, we could use either `M4K`s (4096 bits capacity, 32 bits width) or `M512`s (512 bits capacity, 16 bits width). We choose `M4K`s because (i) they more naturally support the required 32-bit register width; and (ii) we can implement the desired register file using a smaller total number of block memories, which minimizes the amount of costly multiplexing logic required.

We must also carefully provide separation of instruction and data memory as needed. For this paper we support only on-chip memory—we are currently working on support for caches and off-chip memory. Similar to the register file, we provide only one physical instruction memory and one physical data memory, but map to different ranges of those memories as needed. In particular, every thread is always allocated a unique range of data memory. When we execute multiple copies of a single program then threads share a range of instruction memory,[3] otherwise instruction memory ranges are unique as well.

Figure 1 shows the mean area efficiency, in MIPS per 1000 equivalent LEs, across all single benchmarks from Table 1 (i.e., we do not yet consider the multiprogrammed mixes). We measure single-threaded (`st`) and multithreaded (`mt`) processors with varying pipeline depths, and for the multithreaded processors the number of threads is equal to the number of pipeline stages. The area efficiency of the 3, 5, and 7-stage pipelined processors is respectively 33%, 77% and 106% greater than each of their single-threaded counterparts. The 5-stage pipeline has the maximum area efficiency, as it benefits the most from the combination of optimizations we describe next. The 3 and 7-stage pipeline have similar area efficiencies but offer different trade-offs in IPC, thread count, area, frequency, and power.

[3]Since each thread needs to initialize its global pointer and stack pointer differently (in software), we create a unique initialization routine for each thread, but otherwise they share the same instruction memory range.

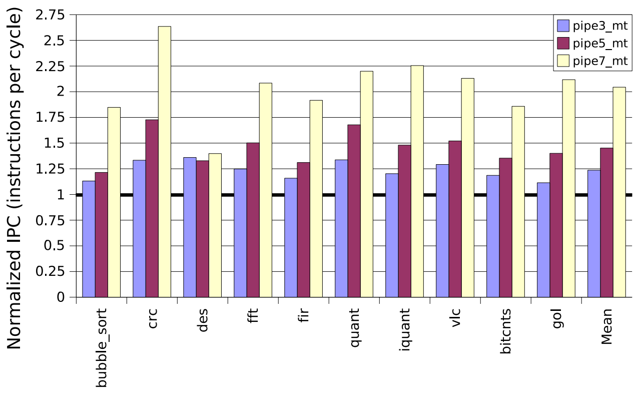


Fig. 2. IPC gain of multithreaded over single-threaded processors.

Figure 2 shows the improvement in instructions-per-cycle (IPC) of multithreaded processors over single-threaded processors. For the 3, 5, and 7-stage pipelines IPC improves by 24%, 45% and 104% respectively. These benefits are partly due to the interleaving of independent instructions in the multithreaded processors which reduce or eliminate the inefficiencies of the single-threaded processors such as unused delay slots, data hazards, and mispredicted branches (our single-threaded processors predict branches are "not-taken"). We have shown that multithreading offers compelling improvements in area-efficiency and IPC over single-threaded processors. In the sections that follow we describe the techniques we used to achieve these gains.

4. TUNING THE ARCHITECTURE

In this section, we identify two architectural features of multithreaded processors that differ significantly from their single-threaded version and hence must be carefully tuned: the choice of Hi/Lo registers versus 3-operand multiplies, and the organization of multicycle paths.

Optimizing the Support for Multiplication: By default, in the MIPS ISA the 64-bit result of 32-bit multiplication is stored into two special 32-bit registers called Hi and Lo—the benefit of these being that multicycle multiplication need not have a write-back path into the regular

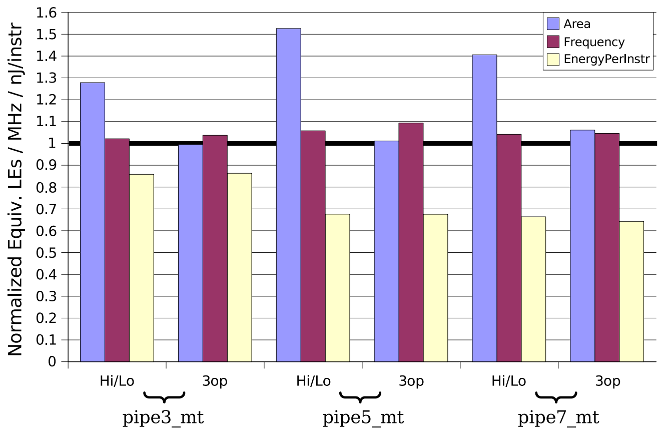
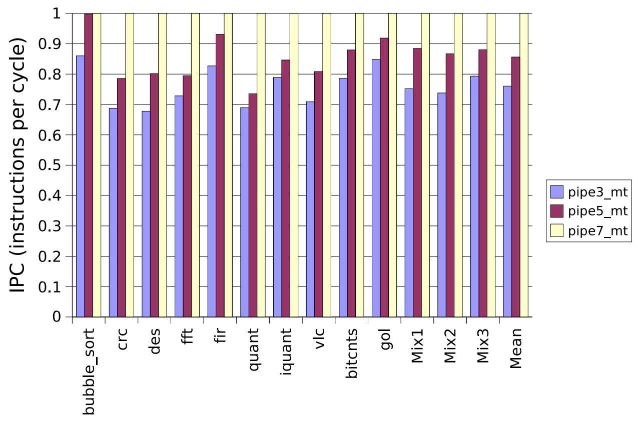
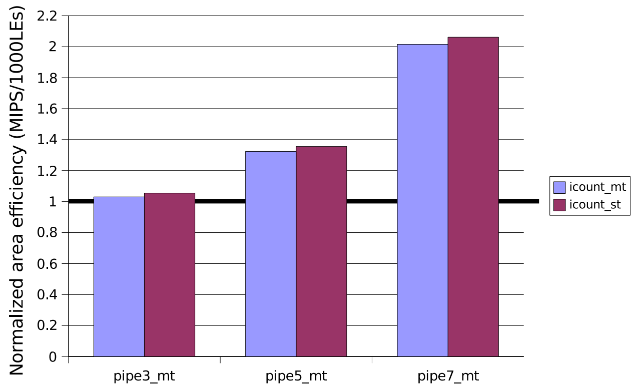


Fig. 3. Hi/Lo registers vs 3-operand multiplies for various pipeline depths, normalized to the corresponding single-threaded processor.



(a) Raw IPC. The 7-stage pipeline has an IPC of 1 because it never stalls.



(b) Area efficiency normalized to the single-threaded processors computed with the instruction count on the multithreaded-processors (`icount_mt`) and with the scaled instruction count on the single-threaded processor (`icount_st`).

Fig. 4. IPC and area efficiency for the baseline multithreaded processors.

register file, allowing higher-frequency designs. Hence for a multithreaded implementation of a MIPS processor we must also replicate the Hi and Lo registers. Another alternative is to modify MIPS to support two 3-operand multiply instructions, which target the regular register file and compute the upper or lower 32-bit result independently. We previously demonstrated that Hi/Lo registers result in better frequency than 3-operand multiplies but at the cost of extra area and instruction count, and are a better choice for more deeply-pipelined single-threaded processors [11]. In this paper we re-investigate this option in the context of multithreaded soft processors. Figure 3 shows the impact on area, frequency, and energy-per-instruction with Hi/Lo registers or 3-operand multiplies, for multithreaded processors of varying pipeline stages each relative to the corresponding single-threaded processor. We observe that Hi/Lo registers require significantly more area than 3-operand multiplies due to their replicated storage but more importantly to the increased multiplexing required to route them.

Since the frequency benefits of Hi/Lo registers are no longer significant, and 3-operand multiplies also have significantly reduced energy-per-instruction, we chose support for 3-operand multiplies as our default in all multithreaded experiments in this paper. In Figure 4(a) we show the raw IPC of the multithreaded processors on all benchmarks as well as the multiprogrammed mixes, demonstrating two key things: (i) that our results for replicated copies of the individual benchmarks are similar to those of multiprogrammed mixes; and (ii) that stalls for the multithreaded 7-stage pipeline have been completely eliminated (since it achieves an IPC of 1). For the three-stage pipeline, the most area efficient for single-threaded processors, the baseline multithreaded processor is only 5% more area efficient than the single-threaded processor (see Figure 4(b)). Area and frequency of our multithreaded processors are similar to those of the single-threaded processors, hence the majority of these gains (36% and 106% for the 5 and 7-stage pipelines) are related to reduction in stalls due to various hazards in the single-threaded designs. Figure 4(b) shows that the reduction of instructions due to the removal of delay slots and 3-operand multiplies also contributes by 3% on average to the final area efficiency that utilizes the scaled instruction count of single-threaded processors to compare a constant amount of work. Comparing with Figure 1, we see that single-threaded soft processors favor short pipelines while multithreaded soft processors favor deep pipelines.

Optimizing Multicycle Paths: Our 3 and 5-stage processors must both stall for certain instructions (such as shifts and multiplies), which we call *unpipelined multicycle paths* [3]. It is important to optimize these paths, since otherwise such stalls will impact all other threads in a multithreaded processor. Fort et al. [1] address this challenge by queueing

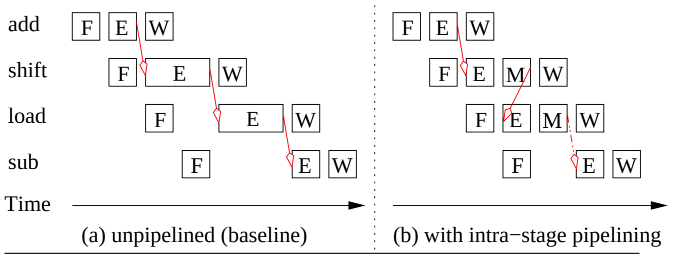


Fig. 5. Example execution showing multicycle paths in the 3-stage pipeline where: (i) shifts and loads require two cycles in the execute stage; and (ii) we assume each instruction has a register dependence on the previous. Assuming a single-threaded processor, forward arrows represent forwarding lines required, while the backward arrow indicates that a stall would be required.

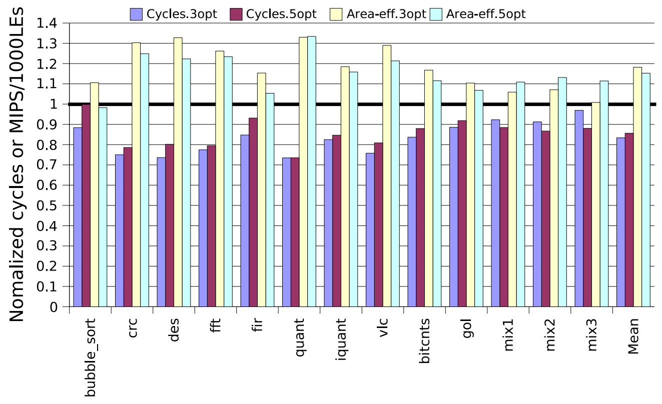


Fig. 6. Impact on both cycle count and area-efficiency of optimizing multicycle paths for the 3 and 5-stage pipeline multithreaded processors, relative to the corresponding baseline multithreaded processors.

requests that stall in a secondary pipeline as deep as the original, allowing other threads to proceed. We instead attempt to eliminate such stalls by modifying the existing processor architecture.

For the single-threaded 3-stage pipeline, multicycle paths were created by inserting registers to divide critical paths, improving frequency by 58% [3]; this division could also have been used to create two pipeline stages such that shifts and multiplies would be pipelined, but this would have created new potential data hazards (see Figure 5), increasing the complexity of hazard detection logic—hence this option was avoided for the single-threaded implementation. In contrast, for a multithreaded processor we can pipeline such paths without concern for data hazards (since consecutive instructions are from independent threads)—hence we do so for the 3-stage processor. The single-threaded 5-stage pipeline also contains multicycle paths. However, we found that for a multithreaded processor that pipelining these paths was not worth the cost, and instead opted to revert the multicycle paths to be single-cycle at a cost of reduced frequency but improved instruction rate. Consequently, eliminating these multicycle paths results in an IPC of 1 for the 5-stage multithreaded processor. The 7-stage processor has no such paths, hence we do not consider it further here.

Figure 6 shows the impact on both cycle count and area-efficiency of optimizing multicycle paths for the 3 and 5-stage pipeline multithreaded processors, relative to the corresponding baseline multithreaded processors. First, our optimizations reduced area for both processors (by 1% for the 3-stage, and by 4% for the 5-stage); however, frequency is also reduced for both processors (by 3% for the 3-stage and by 5% for the 5-stage). Fortunately, in all cases cycle count is reduced significantly, improving the IPC by 24% and 45% for the 3-stage and 5-stage processors over the corresponding single-threaded processors. Overall, this technique alone improves area-efficiency by 18% and 15% for the 3 and 5-stage processors over their multithreaded baseline.

Focusing on the multiprogrammed mixes, we see that the cycle count savings is less pronounced: when executing multiple copies of a single program it is much more likely that consecutive instructions will require the same multicycle path, resulting in avoided stalls with the use of pipelining (as shown in Figure 5(b)); for multiprogrammed workloads such consecutive instructions are less likely. Hence for multiprogrammed mixes we achieve only 5% and 12% improvements in area-efficiency for the 3 and 5-stage pipelines.

5. REDUCING THREAD STATE

In this section, we investigate two techniques for improving the area-efficiency of multithreaded soft processors by reducing thread state.

Reducing the Register File: In previous work [11] we demonstrated that 8 of the 32 architected registers (`s0-s7`) could be avoided by the compiler (such that programs do not target them at all) with only a minor impact on performance for most applications. Since our multithreaded processors have a single physical register file, we can potentially significantly reduce the total size of the register file by similarly removing these registers for each thread. Since our register file is composed of `M4K` block memories, we found that this optimization only makes sense for our 5-stage pipeline: only for that processor does the storage saved by removing registers allow us to save entire `M4K` blocks. In particular, if we remove 7 of 32 registers per thread then the entire resulting 25-register register file fits in a single `M4K` block (since 25 registers $\times$ 5 threads $\times$ 32 bits $<$ 4096 bits).[4] In fact, since our register file is actually duplicated to provide

[4]However, rather than simply shifting, we must now add an offset to register numbers to properly index the physical register file.

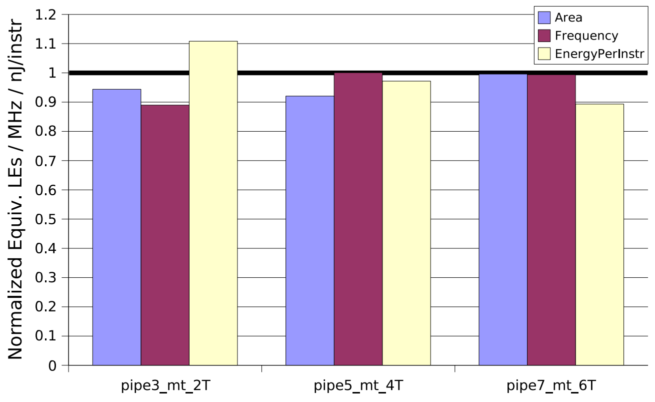


Fig. 7. Impact of having one thread less than the pipeline depth, normalized to processors having the number of threads equal to the pipeline depth.

enough read and write ports, this optimization allows us to use two M4Ks instead of four. For our 5-stage multithreaded processor this optimization allows us to save 5% area and improve frequency by 3%, but increases cycle count by 3% on average due to increased register pressure.

Reducing the Number of Threads: Multithreaded soft processors proposed to date have supported a number of threads equal to the number of pipeline stages [1,2]. For systems where long multicycle stalls are possible (such as with high latency off-chip memory), supporting a larger number of threads than pipeline stages may be interesting. However, for our work which so far assumes on-chip memory, it may also be beneficial to have fewer threads than pipeline stages: we actually require a minimum number of threads such that the longest possible dependence between stages is hidden, which for the processors we study in this paper requires one less thread than there are pipeline stages. This reduction by one thread may be beneficial since it will reduce the latency of individual tasks, result in the same overall IPC, and reduce the area by the support for one thread context. Figure 7 shows the impact on our CAD metrics of subtracting one thread from the baseline multithreaded implementation. Area is reduced for the 3-stage pipeline, but frequency also drops significantly because the computation of branch targets becomes a critical path. In contrast, for the 5 and 7-stage pipelines we achieve area and power savings respectively, while frequency is nearly unchanged. Overall this possibility gives more flexibility to designers in choosing the number of threads to support for a given pipeline, with potential area or power benefits. When combined with eliminating multicycle paths, reducing the number of threads by one for the 5-stage pipeline improves area-efficiency by 25%, making it 77% more area-efficient than its single-threaded counterpart with 78 MIPS/1000 LEs.

6. CONCLUSIONS

We have shown that, relative to single-threaded 3, 5, and 7-stage pipelined processors, multithreading can improve overall IPC by 24%, 45%, and 104% respectively, and area-efficiency by 33%, 77%, and 106%. In particular, we demonstrated that (i) intra-stage pipelining is undesirable for single threaded processors but can provide significant increases in area-efficiency for multithreaded processors; (ii) optimizing unpipelined multicycle paths is key to gaining area-efficiency; (iii) for multithreaded soft processors that 3-operand multiplies are preferable over Hi/Lo registers such as in MIPS; (iv) reducing the registers used can potentially reduce the number of memory blocks used and save area; (v) having one thread less than the number of pipeline stages can give more flexibility to designers while potentially saving area or power. In summary, this work shows that there are significant benefits to a multithreaded soft processor design over single-threaded, and gives system designers a strong incentive to program with independent threads.

7. REFERENCES

[1] B. Fort, D. Capalija, Z. G. Vranesic, and S. D. Brown, "A multithreaded soft processor for SoPC area reduction," in *Proc. of FCCM '06*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 131–142.

[2] R. Dimond, O. Mencer, and W. Luk, "Application-specific customisation of multi-threaded soft processors," *IEE Proceedings—Computers and Digital Techniques*, vol. 153, no. 3, pp. 173– 180, May 2006.

[3] P. Yiannacouras, J. G. Steffan, and J. Rose, "Exploration and customization of FPGA-based soft processors," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, February 2007.

[4] Altera Corporation, "Quartus II," San Jose, CA, USA, Altera.

[5] Mentor Graphics Corp., "Modelsim SE," http://www.model.com, Mentor Graphics, 2004.

[6] J. Veenstra and R. Fowler, "MINT: a front end for efficient simulation of shared-memory multiprocessors," in *Proc. of MASCOTS '94*, NC, USA, January 1994, pp. 201–207.

[7] S. A. Przybylski, T. R. Gross, J. L. Hennessy, N. P. Jouppi, and C. Rowen, "Organization and VLSI implementation of MIPS," Stanford University, CA, USA, Tech. Rep., 1984.

[8] M. G. et al., " MiBench: A free, commercially representative embedded benchmark suite," in *Proc. of WWC '01*, December 2001.

[9] F. Campi, R. Canegallo, and R. Guerrieri, "IP-reusable 32-bit VLIW RISC core," in *Proc. of the 27th European Solid-State Circuits Conf*, 2001, pp. 456–459.

[10] L. Shannon and P. Chow, "Standardizing the performance assessment of reconfigurable processor architectures," in *Proc. of FCCM '03*, 2003, pp. 282–283.

[11] M. Labrecque, P. Yiannacouras, and J. G. Steffan, "Custom code generation for soft processors," in *Proc. of RAAW '06*, Florida, US, December 2006.