A Quantitative Comparison of Reconfigurable, Tiled, and Conventional Architectures on Bit-level Computation

David Wentzlaff and Anant Agarwal
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
{wentzlaf, agarwal}@cag.csail.mit.edu

MIT-LCS-TR-944, April 2004

Abstract

General purpose computing architectures are being called on to work on a more diverse application mix every day. This has been fueled by the need for reduced time to market and economies of scale that are the hallmarks of software on general purpose microprocessors. As this application mix expands, application domains such as bit-level computation, which has primarily been the domain of ASICs and FPGAs, will need to be effectively handled by general purpose hardware. Examples of bit-level applications include Ethernet framing, forward error correction encoding/decoding, and efficient state machine implementation.

In this paper we compare how differing computational structures such as ASICs, FPGAs, tiled architectures, and superscalar microprocessors are able to compete on bit-level communication applications. A quantitative comparison in terms of absolute performance and performance per area will be presented. These results show that although modest gains (2-3x) in absolute performance can be achieved when using FPGAs versus tuned microprocessor implementations, it is the significantly larger gains (2-3 orders of magnitude) that can be achieved in performance per area that will motivate work on supporting bit-level computation in a general purpose fashion in the future.

1 Introduction

Recent trends in computer systems have been to move applications that were previously only implemented in hardware into software on microprocessors. This has been motivated by several factors. Firstly, microprocessor performance has been steadily increasing over time. This has allowed more and more applications that previously could only be done in ASICs and special purpose hardware, due to their large computation requirements, to be done in software on microprocessors. Also, added advantages such as decreased development time, ease of programming, the ability to change the computation in the field, and the economies of scale due to the reuse of the same microprocessor for many applications have influenced this change.

If we believe that this trend will continue, then in the future we will have one computational fabric that will need to do the work that is currently done by all of the chips inside of a modern computer. Thus we will need to pull all of the computation that is currently being done inside of helper chips onto our microprocessors. We have already seen this being done in current computer systems with the advent of all-software modems and software radios.

Two consequences follow from the desire to implement all parts of a computer system in one computational fabric. First, the computational requirements of this one computational fabric are now much higher. Second, the mix of computation that it will be doing is significantly different from applications that current day microprocessors are optimized for. Thus if we want to build future architectures that can handle this new application mix, we need to develop architectural mechanisms that efficiently handle conventional applications, SpecInt and SpecFP, multimedia applications, which have been the focus of significant research recently, and the before mentioned applications which we will call software circuits.

In modern computer systems most of the helper chips are there to communicate with different devices and mediums. Examples include sound cards, Ethernet cards, wireless communication cards, memory controllers and I/O protocols such as SCSI and Firewire. This research work will focus on the subset of software circuits for communication systems, examples being Ethernet cards (802.3) and wireless communication cards (802.11a, 802.11b). Communication systems are chosen as a starting point for this research for two reasons. One, it is a significant fraction of the software circuits domain. Secondly, if communication bandwidth is to continue to grow as is foreseen, the computation needed to handle it will become a significant portion of our future processing power. This is mostly due to the fact that communication bandwidth is on a steep exponentially increasing curve. Accordingly, this research will further focus on bit-level computation contained in communication processing. Fine grain bit-level computation is an interesting sub-area of communications processing, because unlike much of the rest of communications processing, it is not easily parallelizable on word oriented systems because very fine grain, bit-level, communication is needed. Examples of this type of computation include error-correcting codes, convolutional codes, framers, and source coding.

In this paper we investigate how bit-level computation in communication applications maps to differing architectures. Our methodology is to make very carefully optimized implementations on many differing architectures and then study the applications in terms of absolute *performance* and *performance per area*. The architectures examined include an IBM ASIC flow (SA-27E), a Xilinx FPGA (Virtex II), the Pentium 3, the Pentium 4, and the tiled Raw architecture.

Through this work we found some surprising results. To our surprise FGPAs did not have as large of an absolute performance gain versus microprocessors as we had expected, especially considering that the bit-level application mix heavily favored reconfigurable architectures. Rather, FPGAs only had a 2-3x performance improvement versus a microprocessor implementation. This was due largely to the high clock rates of microprocessors and that using lookup tables in a high speed microprocessor does a good job of emulating combinational logic. What we did find is that the real reason to implement bit-level communication processing in FPGAs or ASICs is their large area wins. For our application mix we found that ASICs provide 5-6 orders of magnitude and FPGAs provide 2-3 orders of magnitude better performance per area than software in a microprocessor.

The rest of this paper is organized as follows. Section 2 describes the applications that we explored and how they are used in the world. Section 3 describes the target architectures that we mapped the applications to, the tools used, and assumptions made. We then go on to present our results in Section 4 and analyze them in Section 5. Section 6 relates this paper to previous work. Finally we outline future directions and conclude in Sections 7 and 8.

2 Applications

In this section we describe the applications that were used in this study. One of the guiding principles that we followed when choosing applications was to pick applications that are actually used and not simply contrived applications. To accomplish this these applications are components of current day communication systems.

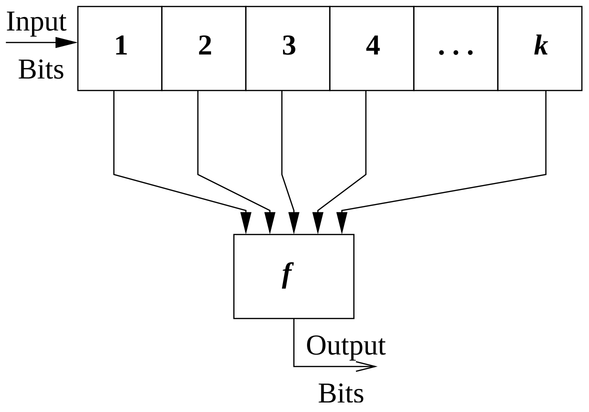

Figure 1. Generalized Convolutional Encoder

2.1 802.11a Convolutional Encoder

The first application that we studied was the convolutional encoder in 802.11a. IEEE 802.11a is the wireless Ethernet standard [13] used in the 5GHz. Industrial, Scientific and Medical (ISM) band. In this band, 802.11a provides for transmission of information up to 54Mbps. Other wireless standards that have similar convolutional encoders to that of 802.11a include IEEE 802.11b [14], the current WiFi standard and most widely used wireless data networking standard, and Bluetooth [2], a popular standard for short distance wireless data transmission.

Convolutional encoders are a forward error correction scheme which unlike block codes do not translate a fixed sized block into another fixed sized block. Rather convolutional encoders work on a bit at a time and contain storage. The basic structure of a convolutional encoder has k storage elements chained together. The input bits are shifted into these storage elements. The older data which is still stored in the storage elements shift over as the new data is added. The shift amount s can be one (typical) or more than one. The output is computed as a function of the state elements. A new output is computed whenever new data is shifted in. In convolutional encoders, multiple functions are many times computed simultaneously to add redundancy. Thus multiple output bits can be generated per input bit. Figure 1 shows a generalized convolutional encoder. The boxes with numbers in them are storage elements that shift over by s bits every encoding cycle. The function box, denoted with f, can actually represent multiple differing functions.

This application study uses the default convolutional encoder that 802.11a uses in poor channel quality. It is a rate 1/2 convolutional encoder and contains seven storage elements. 802.11a has differing encoders for different channel quality with rates of 1/2, 3/4, and 2/3. The shift amount for this convolutional encoder is one ($s = 1$). Figure 2 is a block level diagram of the studied encoder. This encoder

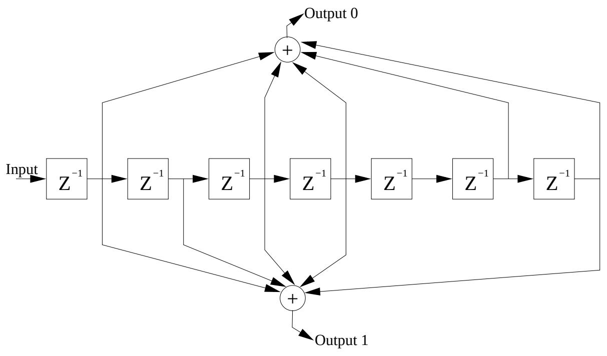

Figure 2. 802.11a Rate 1/2 Convolutional Encoder

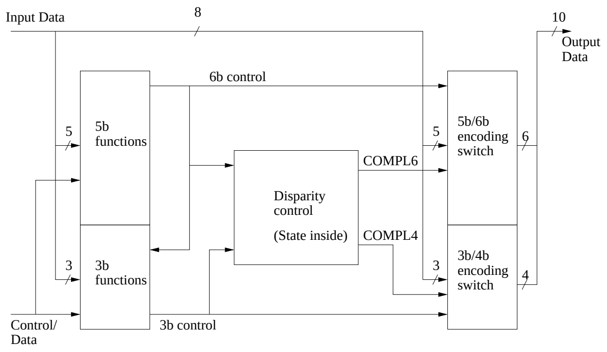

Figure 3. Overview of the 8b/10b encoder taken from [24]

has two outputs with differing tap locations, and uses XOR as the function it computes. The generator polynomials used are $g_0 = 133_8$ and $g_1 = 171_8$.

2.2 8b/10b Block Encoder

The second application we explore is IBM's 8b/10b block encoder. 8b/10b encoding is a byte oriented binary transmission code which translates 8 bits at a time into 10-bit codewords. This particular block encoder was designed by Widmer and Franszek and is described in [24] and patented in [8]. This encoding scheme has some nice features such as being DC balanced, detection of single bit errors, clock recovery, addition of control words and commas, and ease of implementation in hardware.

One may wonder why 8b/10b encoding is important. It is important because it is a widely used line encoder for both fiber-optic and wired applications. Most notably it is used to encode data right before it is transmitted in fiber optic Gigabit Ethernet [15] and 10 Gigabit Ethernet physical layers. Also, because it is used in such high speed applications, it is a performance critical application.

This 8b/10b encoder is a partitioned code, meaning it is made up of two smaller encoders, a 5b/6b and a 3b/4b encoder. This partitioning can be seen in Figure 3. To achieve the DC balanced nature of this code, the Disparity control box contains one bit of state which is the running disparity of the code. It is this state which lets the code change its output codewords such that the overall parity of the line is never more than ± 3 bits, and the parity at sub-word boundaries is always either $+1$ or -1. The coder changes its output codewords by simply complementing the individual subwords in accordance with the COMPL6 and COMPL4 signals. All of the blocks in Figure 3 with the exception of the Disparity control simply contain feed forward combinational logic therefore this design is somewhat pipelinable. But due to the tight parity feedback calculation it is not inherently parallelizable otherwise. Tables are provided in [24] which show the functions implemented in these blocks along with a more minimal combinational implementation.

3 Experimental Setup

One of the greatest challenges of this project was simply finding a way to objectively compare very disparate computing platforms which range from a blank slate of silicon all the way up to a state of the art out-of-order superscalar microprocessor. The first step in objectively comparing architectures is to choose targets that very nearly reflect each other in terms of semiconductor process. While it is possible to scale between different semiconductor feature sizes, not everything simply scales. Characteristics that do not scale include wire delay and the proportions of gate sizes. Thus a nominal feature size of 0.15μm. drawn was chosen. All of the targets chosen with the exception of the Pentium 4 (0.11μm.) are fabricated with this drawn feature size. Those architectures that were not fabricated on 0.15μm. were scaled to 0.15μm. by linearly scaling performance and quadratically scaling area. No attempt was made to take into account the operational voltage differences. Table 1 shows the process parameters for the differing targets used in this study.

Complicating this experiment more, the different targets use differing programming models that vary from Verilog to 'C'. To combat this, the best model was chosen for each architecture and optimized programs were written for each architecture. Also the best algorithm was picked for each programming model and architecture pair. We chose this approach because we favored careful, detailed implementations of a few applications on many architectures over quick implementations of many applications on less platforms.

Following is a description of the different target architectures that were investigated. The programming environment and tools used are also described.

Target	Foundry	Process Generation (μm.)	L$_{drawn}$ (μm.)	L$_{effective}$ (μm.)	Nominal Voltage	Layers of Metal	Type of Metal
IBM SA-27E ASIC	IBM	0.18	0.15	0.11	1.8	6	Al & Cu
Xilinx Virtex II	UMC	0.18	0.15	0.12	1.5	8	Al
Intel Pentium 4 Northwood 2.2GHz.	Intel	0.13	0.11	0.07	1.75	6	Cu
Intel Pentium 3 Coppermine 993MHz.	Intel	0.18	0.15	0.10	1.7	6	Al
Raw 425MHz.	IBM	0.18	0.15	0.11	1.8	6	Al & Cu

Table 1. Summary of Semiconductor Process Specifications

3.1 IBM SA-27E

The IBM SA-27E process is an ASIC flow that is a 6 layer copper metal process with a drawn transistor size of 0.15μm. [12] This is considered to be the best case architecture for this study. While this standard cell approach is not as optimal as a full-custom realization of the circuits, the trends found here should be characteristic of those if the applications in this study were implemented in full-custom logic. The programming model for this architecture is synthesizable behavioral Verilog and the designs were synthesized with Synopsys's Design Compiler II using IBM's technology libraries. Performance and area results are generated with Synopsys's tools and the IBM technology libraries in the form of area and performance reports.

3.2 Xilinx Virtex II

This target is Xilinx's Virtex II, [25] a Field Programmable Gate Array (FPGA). This is a LUT based reconfigurable fabric. The tool flow starts by using the same Verilog source as was used in the ASIC flow. The Verilog was then synthesized with Synopsys's FPGA Compiler II. Then the design was processed by Xilinx's backend tools to generate bit files, performance reports, and area reports. One interesting problem that came up with this flow is that Xilinx's tools provide area in terms of Slices (what CLBs have been renamed to) but that does not provide for an easy area comparison outside of Xilinx's FPGA families. To calculate accurate area numbers for this target, a Xilinx XC2V-40 was purchased and cannibalized to figure out total die area which then provided an easy way to calculate the area of a Slice in mm^2. Because the Slice size calculation is based off of the entire die size of a Xilinx XC2V-40, this measurement is likely an overestimate due to the I/O pads, I/O drivers, and block RAMs being included in a Slice's area estimate. The area calculated for this target is only the area used by the application and not the entire die size. This models a designer's ability to choose an appropriately sized FPGA to fit an application.

3.3 Pentium

Intel's Pentium 4 and Pentium 3 were used in this study. The programming model used for this target was 'C' compiled with gcc 2.95.3 with optimization level of -O3 and the -finline option turned on. Also, on the Pentium 3, the -funroll-all-loops option was also turned on, while this yielded lower performance on a Pentium 4 where it was not used. To gather the speed at which the applications run, the time stamp counter (TSC) was used. The TSC is a 64-bit counter on Intel processors (above a Pentium) which monotonically increases on every clock cycle. With this counter, accurate cycle counts can be determined for execution of a particular piece of code. To prevent memory hierarchy from unduly hurting performance, all source and result arrays were touched to make sure they were within the level of cache being tested before the test's timing commenced. To calculate the overall speed, the tests were run over many iterations of the loop and the overall time as per the TSC was divided by the iterations completed normalized to the clock speed to come up with the resultant performance. To calculate the area size, the overall chip area was used.

3.4 Raw Microprocessor

The Raw microprocessor is a tiled computer architecture, designed in the Computer Architecture Group at MIT [21], used for this study. The trends found for Raw should also be characteristic of other tiled and parallel wire exposed architectures such as Smart Memories [17], Grid [19], ILDP [16], and M3T [22]. The Raw processor contains 16 replicated tiles in a 4x4 mesh. Tiles are connected to each of their four nearest neighbors via register mapped communication by two sets of static network interconnect and two sets of dynamic network interconnect. Many of the ideas in Raw were inspired by FPGAs [1], especially the manner in which the wires connecting tiles are reconfigurable though the static network. Each tile contains a main processor which is roughly equivalent to a MIPS R4000 processor. The Raw processor was fabricated by IBM, runs at 425MHz. and is in daily use at MIT for research.

The applications in this study for Raw were written in 'C' and assembly and were hand parallelized with communication happening over the static network for peak performance. In this paper we will discuss Raw in two different versions. One version contains the "rotate left and mask" instruction, denoted by the opcode 'rlm', and the "rotate left and mask with insert" instruction, denoted by the opcode 'rlmi', while the other version of Raw does not use these instructions. 'rlm' and 'rlmi' are very similar to insert and extract operations found on other architectures, and allow for rotation, masking, and insertion into another register all in one cycle. These instructions form a crude level of fast bit manipulation which helped significantly on the presented applications. While the Raw processor hardware has both of these instructions in it, we split the discussion into two cases because these two instructions are not in the standard MIPS ISA and the results would have been misleading to not discuss how this crude level of bit manipulation improved performance.

Four versions of the convolutional encoder were designed for this target. Three of the designs used only one tile and one used all 16 tiles. The lookup table design was improved through the use of the rlm/rlmi instruction while the distributed 802.11a encoder did not benefit from this instruction, but instead used spatial pipelining to gain parallel performance. For the 8b/10b encoder, three implementations on Raw were designed. Two of the implementations use a lookup table, with and without rlm/rlmi. And the distributed implementation uses six tiles and requires the use of rlm/rlmi. It was only through the use of these two instructions that this application was able to be parallelized. The reason for this is that the 8b/10b encoder has a tight feedback path through the use of the disparity state from the previously calculated word. Thus without the use of the rlm/rlmi instructions, a load was needed in the feedback path which had high enough instruction latency that it negated the benefits from parallelizing the application. To calculate the area used for each implementation, the number of tiles used by the implementation was multiplied by the area used by one tile. Thus the measured area for single tile and six tile implementations was $1/16$ and $3/8$ of the whole chip area respectively. To collect accurate performance information, applications were simulated on Raw's cycle accurate simulator 'btl' and cycle counts were gathered. The 'btl' simulator is cycle accurate and has been verified against the real Raw processor silicon. For each of the tests, the input data was streamed from off-chip via the static network and the results were streamed off via the static network.

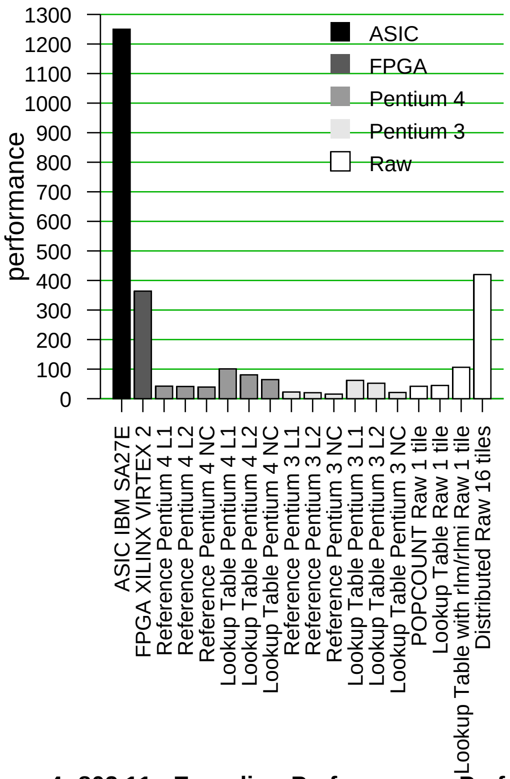

Figure 4. 802.11a Encoding Performance. Performance is measured as the rate (in MHz.) at which the encoder produces one bit of output.

4 Results

4.1 802.11a Convolutional Encoder

Table 2 shows the results of the convolutional encoder running on the differing targets. The area and performance columns are both measured metrics while the performance per area column is a derived metric. Figure 4 shows the absolute, normalized to 0.15μm., performance. This metric is tagged with the units of MHz. which is the rate at which this application produces one bit of output. The trailing letters on the Pentium identifiers denote which cache the encoding matrices fit in. "L1" represents that all of data fits in the L1 cache, "L2" the L2 cache, and "NC" represents no cache, or the data set size is larger than the cache size. These differing levels of memory hierarchy were exercised by varying the data set size on the Pentiums, while on the other architectures more direct means were used for the input and output of data which did not require the use of the memory hierarchy.

Figure 4 shows trends that one would expect. The ASIC implementation provides the highest performance at 1.25GHz. The FPGA and parallelized Raw implementation are the second fastest at approximately 3 times slower. Of

Target	Implementation	Area (mm^2) (Normalized to 0.15μm. L$_{drawn}$)	Performance (MHz.) (Normalized)	Performance per Area (MHz./mm^2.)
IBM SA-27E ASIC		0.0016670976	1250	749806
Xilinx Virtex II		0.77	364	472.7
Intel Pentium 4 Northwood 2.2GHz. Normalized to 1.613GHz.	*reference* L1	271.49	42.45	0.1564
	reference L2	271.49	41.37	0.1524
	reference NC	271.49	39.35	0.1449
	lookup table L1	271.49	100.8	0.3714
	lookup table L2	271.49	80.67	0.2971
	lookup table NC	271.49	64.53	0.2377
Intel Pentium 3 Coppermine 993MHz.	*reference* L1	106	22.56	0.2128
	reference L2	106	20.26	0.1911
	reference NC	106	15.52	0.1464
	lookup table L1	106	62.06	0.5855
	lookup table L2	106	52.26	0.4930
	lookup table NC	106	21.0	0.1981
Raw 1 tile 425 MHz.	*POPCOUNT*	16	42	2.625
	lookup table	16	44.74	2.796
	lookup table with rlm/rlmi	16	106.3	6.641
Raw 16 tiles	*distributed*	256	425	1.641

Table 2. 802.11a Convolutional Encoder Results

the microprocessors without bit manipulation support, the Pentium 4 shows the fastest non-parallel implementation. The Raw processor provides interesting results. It is significantly slower than the Pentium 4 using a single tile without rlm/rlmi, which is to be expected considering that the Raw processor runs at 425MHz. versus 2GHz. and is only a single issue in-order processor. But by using Raw's parallel architecture a parallel mapping can be made which provides 10x the performance of one tile when using all 16 tiles. Note that this shows sub-linear performance scaling of this application. Also, the Raw single tile implementation with rlm/rlmi sports similar performance to that of the Pentium 4. These performance numbers show for a real world application how much more adept an ASIC is than an FPGA and how much more adept an FPGA is than a conventional processor at bit-level computation.

Figure 5 shows a much more interesting metric. It plots the performance per area for all of the differing targets. This metric measures how efficiently each target uses the silicon area for convolutional encoding. One would want to use a metric like this if an application was fully parallel and you wanted to build the most efficient design for a given silicon area. Also, this metric gives an idea of how to compare architectures when it comes down to area comparison and helps answer the age old question of quantitatively how much better is an FPGA or ASIC compared to architecture ”X”. As can be seen from Figure 5, the ASIC is 5-6 orders of magnitude better than processor implementations on this bit-level application. FPGAs are 2-3 orders of magnitude more efficient than any of the tested processors. With respect to the processors, they are all within an order of magnitude of each other. It is interesting to see that the Pentium 4 is less efficient than Pentium 3 for performance per area while it does have better peak performance. This is because the area used by the Pentium 4 is proportionally more than the gained performance when compared to a Pentium 3. Likewise, because of Raw's simpler data-path, its grain size more closely matches the application's grain size and thus it gets a smaller area punishment. Lastly, it is interesting to note that the parallelized 16-tile *distributed* Raw version has lower performance per area than the single tile Raw implementations. This is due to the sub-linear scaling of this application. While this implementation is able to realize 10x the performance of the single tile implementation without rlm/rlmi, it uses 16x the area and thus if absolute performance is not a concern a single tile implementation is more efficient with respect to area.

4.2 8b/10b Block Encoder

The trends of the 802.11a encoder are also shared with the second application. Table 3 contains full results for all implementations. Figure 6 shows the absolute encoding performance. Note, that one cannot easily compare this to Figure 4 because the units are different. The performance metric used in Figure 6 is the rate at which 10-bit output blocks

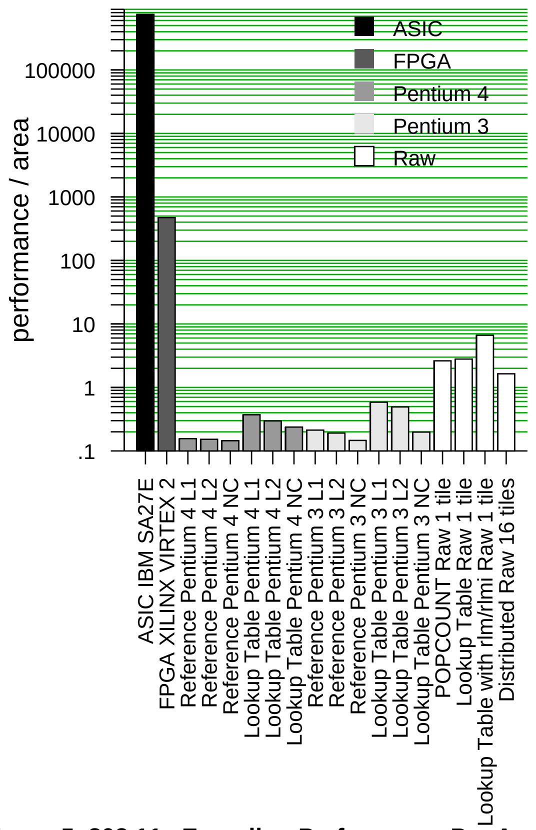

Figure 5. 802.11a Encoding Performance Per Area (MHz./mm^2.)

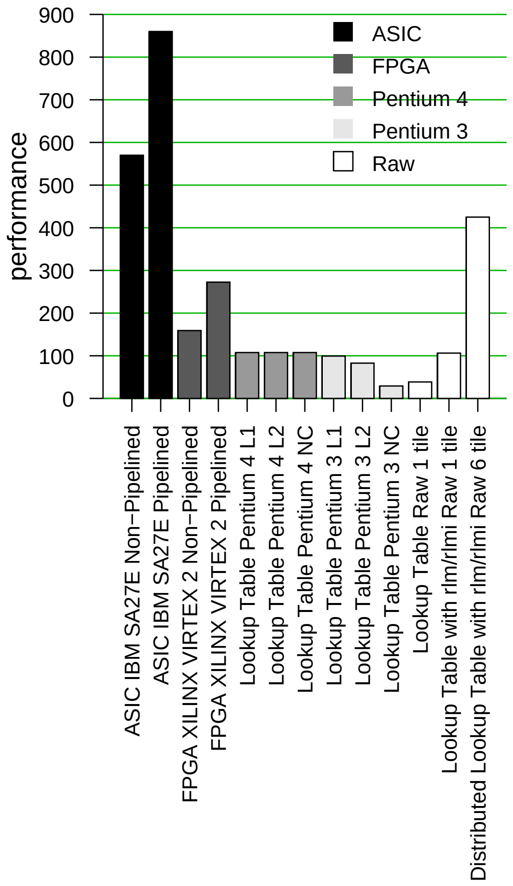

Figure 6. 8b/10b Encoding Performance. Performance is measured as the rate (in MHz.) at which the encoder produces a 10-bit word of output.

are produced, while Figure 4 is the rate at which bits are produced for a totally different application. As can be seen from the performance chart, the *pipelined* ASIC implementation is approximately 3x faster than the FGPA implementation, and the FPGA is 3x faster than a software implementation. As is to be expected due to clock speed, the Pentium 4 is faster than the Pentium 3 which is faster than a single Raw tile in absolute performance.

Figure 7 shows the performance per area for 8b/10b encoding. These results corroborate the results for the convolutional encoder. The ASIC provides 5 orders of magnitude better area efficiency than a microprocessor. Also, an FPGA has two orders of magnitude better area efficiency than a software implementation on a conventional processor. Likewise the efficiency of the processors parallel the grain size of the differing architectures. One interesting thing that is not totally intuitive about Figure 7 is how pipelining this application has differing effects on different targets. In performance per area, pipelining the ASIC implementation is a net win as can be seen from the first two bars. This means that the added performance outpaced the added area of extra pipeline flip-flops. But the opposite story is true for the Xilinx Virtex II. While a 1.7x performance gain was achieved by pipelining this application, the area for this application increased by a factor of 2.14. This result shows off the relative difference between flip-flop costs of these two targets. In an FPGA, because of the relative sparseness of flip-flops and the larger flip flops due to the added reconfiguration complexities, the cost of pipelining an application is much higher than in an ASIC where the flip-flops cost less in both direct area, and that they restrict the placement of the circuit far less than in an FPGA.

5 Analysis

When one looks at the results of this paper, there are a couple of quantitative Rules of Thumb that present themselves with respect to bit-level processing.

1. ASICs provide a 2-3x absolute performance improvement over an FPGA implementation.

2. FPGAs provide a 2-3x absolute performance improvement over a microprocessor implementation.

Target	Implementation	Area (mm^2) (Normalized to 0.15μm. L_{drawn})	Performance (MHz.) (Normalized)	Performance per Area (MHz./mm^2.)
IBM SA-27E ASIC	*non-pipelined*	.005117952	570	111372.67
	pipelined	.00756464032	860	113695.98
Xilinx Virtex II	*non-pipelined*	1.4706	159.109	108.1933
	pipelined	3.1514	272.554	86.4866
Intel Pentium 4 Northwood 2.2GHz. Normalized to 1.613GHz.	*lookup table* L1	271.5	107.6	0.3962
	lookup table L2	271.4	107.6	0.3962
	lookup table NC	271.4	107.6	0.3962
Intel Pentium 3 Coppermine 993MHz.	*lookup table* L1	106.0	99.30	0.9367
	lookup table L2	106.0	82.75	0.7807
	lookup table NC	106.0	29.20	0.2755
Raw 1 tile 425 MHz.	*lookup table*	16	38.64	2.415
	lookup table with rlm/rlmi	16	106.3	6.641
Raw 6 tile 425 MHz.	*distributed lookup table with rlm/rlmi*	96	425	4.427

Table 3. 8b/10b Encoder Results

3. ASICs provide 5-6 orders of magnitude better performance per area than software implementation on a microprocessor.

4. FPGAs provide 2-3 orders of magnitude better performance per area than software implementation on a microprocessor.

5. Parallel implementations on Tiled architectures yield competitive absolute performance to that of FPGAs but use at least an order of magnitude more area to do so.

These Rules of Thumb at first may be relatively surprising. The first question that people may wonder is why is the FPGA only 2-3 times faster than a microprocessor? And why is the absolute performance of an ASIC only 4-9 times that of a microprocessor? Many people may think that the absolute performance of ASICs and FPGAs should be higher than that shown here. There are two reasons that the performance difference is not larger. One, when using a microprocessor as a lookup table it does a surprisingly good job of running bit-level applications. The use of a lookup table on microprocessors is essentially an emulation layer that uses a microprocessor as a very high speed lookup table. Second, the hardware implementations used in this study are base implementations done not to bloat the area of the designs. Thus it is believed that higher performance implementations could be made, but this would be at the cost of complexity and silicon area. But one must keep in mind that there are diminishing performance returns when a design is made more complex for the purpose of speed.

Another question that these results inspire is why is there a larger performance per area gain than simply word size when comparing an ASIC to a microprocessor? This can be reposed as asking why is using a 32-bit processor as a one-bit processor more than 32 times area inefficient? There are several factors at work here. One reason why smaller grain size applications can actually have super-linear performance speed up is that the relative cost of communication is cheaper. An example of this can be seen in the Raw processor which is a 32-bit processor. If this processor was shrunk to a 16-bit processor, we will assume roughly half the area, the distance that is needed to be traversed to communicate with the nearest other tile will not simply stay constant. Rather, the distance that is traversed will decrease to roughly 70% [1] of the 32-bit example's distance. Secondly, both ASIC and FPGA technology gain performance increases due to datapath and control specialization. Why build complicated general purpose structures when all you need is something small and specific? Specialization can increase clock rate by having the custom dataflow needed to match the computation and it also uses less area by simply not needing all of the complexity of a microprocessor such as instruction fetch, register renaming, reorder buffers, etc. The control can be reduced to either hardwired control or small state machines.

One problem with the performance per area metric that should be acknowledged, is the fact that it does not properly

[1] This can easily be calculated if you know that the area is decreasing by $1/2$ this corresponds to scaling in each direction by $\sqrt{1/2} = 0.707$.

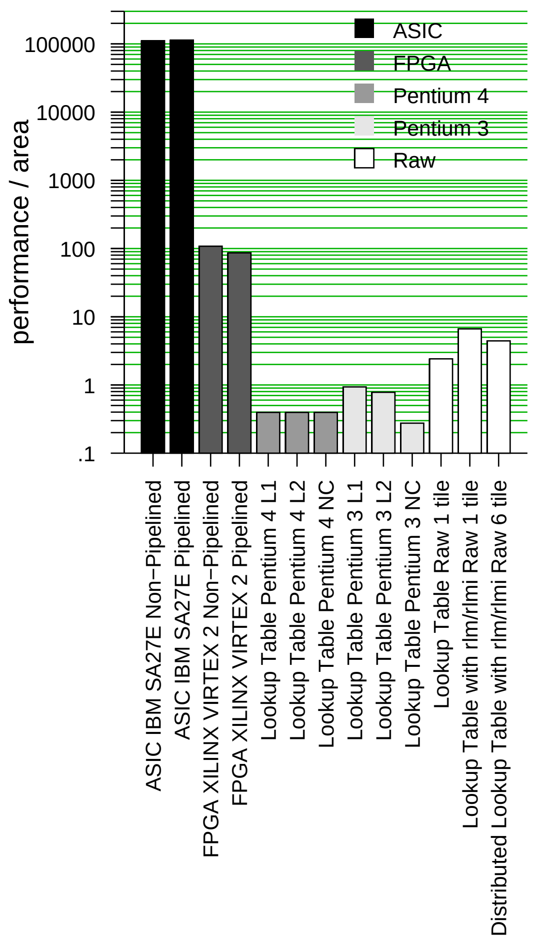

Figure 7. 8b/10b Encoding Performance Per Area (MHz./mm^2.)

credit the ability of a microprocessor to time multiplex its hardware area. In a microprocessor, instructions are stored in a very dense instruction memory or possibly in an even denser main memory store, DRAM. It is via this time multiplexed use of the hardware area resources that the performance per area efficiency goes up. This is difficult to quantize though because it requires total knowledge of all of the applications that are ever going to be run on the computational target. But, if all that is going to be run on a target, in a small period of time, is one application, then the performance per area metric is a worthwhile metric.

Finally, the most important thing that these results point to is the fact that computational grain size is very important. If an architecture only has one grain size, emulation of a differing grain size can in many cases be emulated relatively efficiently with respect to performance. An example of this is how a processor can creatively use a lookup table to emulate random logic. But, when it comes down to efficient area utilization, proper grain size is critical.

6 Related Work

Several reconfigurable architectures have investigated bit-level computation. Garp [5, 6], PRISC [20], and Chimaera [11] have investigated mixing reconfigurable logic with microprocessor cores. The PipeRench processor [10] and MATRIX [18] investigated how to make hybrid microprocessor-reconfigurable architectures. Lastly research into how to make improved FPGA architectures have studied mappings of bit-level applications to reconfigurable architectures. Examples include RaPiD [7], DPGA [3] and the High-Speed, Hierarchical Synchronous Reconfigurable Array [23].

Comparisons such as this study, but with less target architectures, have been carried out by Babb et al. in the Raw benchmark suite [1]. Also a large number of applications have been mapped to FPGAs in the Splash [9] and Splash 2 [4] projects.

7 Future Work

In the future we would like to investigate several more applications in the bit-level application domain. This would be done with the hope of gaining more insight into these applications. Specifically we would like to be able to quantify how much each factor, specialization, parallelization, and wire length, effects performance and performance per area. The next application that we want to implement is a finite state machine as this is a bit-level computation that is used in almost every system.

Also we would like to move this work into studying power wins that can be gained from fine grain computational fabrics. We think that the trends found with the performance per area metric will be consistent with performance per energy wins. Unfortunately, the tools to explore power make it difficult to make objective comparisons.

The results in this paper are very promising and we believe that by using some form of hybrid architecture that mixes microprocessors and reconfigurable logic, that bit-level and general purpose computation can be supported in a general purpose way. This new general purpose architecture would reduce the area inefficiency that microprocessors exhibit on bit-level computation. One possible architecture that can be imagined is to take a modern tiled architecture such as Raw and add to each tile a small tightly integrated array of sequencable lookup tables (LUT). These would be able to work like an FPGA fabric and implement bit-level computation. Also, by adding a LUT array to each tile, this architecture would be able to scale the amount of reconfigurable logic.

8 Conclusion

This work has studied how different architectures are able to handle bit-level communication processing. To study this, two characteristic applications were selected and experimentally mapped onto common computational structures of differing grain size including microprocessors, tiled architectures, FPGAs, and ASICs. From these results it can be concluded that for these applications, fine grain computational fabrics (FPGAs) can provide a 2-3x absolute performance improvement over a best case microprocessor in the same fabrication process. And more importantly a fine grain computational fabric is able to provide 2-3 orders of magnitude better performance per area than software on a microprocessor.

From these results we conclude that it is usually possible to use one grain size to run an application with a smaller grain size with not too large of an absolute performance degradation through some emulation mechanism. In this case this emulation mechanism is the ability to use a microprocessor's cache as a lookup table to substitute for custom logic. But, unfortunately these forms of grain size mismatch cause large, multiple orders of magnitude, inefficiencies when it comes to area utilization. Thus we need to study integration of fine grain computational structures with architectures that have larger grain size such as word-based microprocessors to be able to support both bit-level and general purpose computation in an area efficient manner.

Acknowledgments

The first author would like to thank Michael B. Taylor for optimizations to some of the Raw versions of the presented applications, and Ian Bratt for his input to this paper. This work is funded by DARPA, the National Science Foundation, and MIT's Project Oxygen.

References

[1] J. Babb, M. Frank, V. Lee, E. Waingold, R. Barua, M. Taylor, J. Kim, S. Devabhaktuni, and A. Agarwal. The raw benchmark suite: Computation structures for general purpose computing. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 134–143, Apr. 1997.

[2] Bluetooth Special Interest Group. *Specification of the Bluetooth System: Core*, 1.1 edition, Feb. 2001.

[3] M. Bolotski, A. DeHon, and T. Knight. Unifying FPGAs and SIMD arrays. In *Proceedings of the International Workshop on Field-Programmable Gate Arrays*, Feb. 1994. MIT Transit Note Number 95.

[4] D. A. Buell, J. M. Arnold, and W. J. Kleinfelder. *Splash 2: FPGAs in a Custom Computing Machine*. IEEE Computer Society Press, 1996.

[5] T. J. Callahan, J. R. Hauser, and J. Wawrzynek. The garp architecture and c compiler. *IEEE Computer*, 33(4):62–69, Apr. 2000.

[6] T. J. Callahan and J. Wawrzynek. Adapting software pipelining for reconfigurable computing. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, San Jose, CA, 2000. ACM.

[7] C. Ebeling, D. C. Cronquist, P. Franklin, J. Secosky, and S. G. Berg. Mapping applications to the RaPiD configurable architecture. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 106–115, Apr. 1997.

[8] P. A. Franaszek and A. X. Widmer. Byte oriented DC balanced (0,4) 8b/10b partitioned block transmission code. US Patent, Dec. 1984. US Patent Number 4,486,739.

[9] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti. Building and using a highly parallel programmable logic array. *IEEE Computer*, 24(1):81–89, Jan. 1991.

[10] S. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer. PipeRench: A coprocessor for streaming multimedia acceleration. In *Proceedings of the International Symposium on Computer Architecture*, pages 28–39, 1999.

[11] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao. The Chimaera reconfigurable functional unit. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 87–96, Apr. 1997.

[12] IBM. *ASIC SA-27E Databook*, 2000.

[13] IEEE. *IEEE Standard 802.11a-1999 Supplement to IEEE standard for Information Technology- telecommunications And Information exchange Between Systems-Local And Metropolitan Area Networks-specific Requirements-part 11: Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications*, 1999.

[14] IEEE. *IEEE Standard 802.11b-1999 Supplement to IEEE standard for Information Technology- telecommunications And Information exchange Between Systems-Local And Metropolitan Area Networks-specific Requirements-part 11: Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications*, 1999.

[15] IEEE. *IEEE Standard 802.3-2000 IEEE standard for Information Technology- telecommunications And Information exchange Between Systems-Local And Metropolitan Area Networks-specific Requirements-part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, 2000.

[16] H.-S. Kim and J. E. Smith. An instruction set architecture and microarchitecture for instruction level distributed processing. In *Proceedings of the International Symposium on Computer Architecture*, May 2002.

[17] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz. Smart memories: A modular reconfigurable architecture. In *Proceedings of the International Symposium on Computer Architecture*, pages 161–171, June 2000.

[18] E. Mirsky and A. DeHon. MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 157–166, Apr. 1996.

[19] R. Nagarajan, K. Sankaralingam, D. Burger, and S. W. Keckler. A design space evaluation of Grid processor architectures. In *Proceedings of the International Symposium on Microarchitecture*, pages 40–51, Dec. 2001.

[20] R. Razdan and M. D. Smith. A high-performance microarchitecture with hardware-programmable functional units. In *Proceedings of the International Symposium on Microarchitecture*, pages 172–80, Nov. 1994.

[21] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, J.-W. Lee, P. Johnson, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The Raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, Mar. 2002.

[22] J. Torrellas. Morphable multithreaded memory tiles (M3T). DARPA Polymorphos Computing Architectures PI Meeting, 2001.

[23] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhanu, V. George, J. Wawrzynek, and A. DeHon. Hsra: High-speed, hierarchical synchronous reconfigurable array. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 125–134, Feb. 1999.

[24] A. X. Widmer and P. A. Franaszek. A DC-balanced, partitioned-block, 8b/10b transmission code. *IBM Journal of Research and Development*, 27(5):440–451, Sept. 1983.

[25] Xilinx Corporation. *Virtex-II Data Sheet*, Nov. 2001.