Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams

Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff,
Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota,
Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank,
Saman Amarasinghe, and Anant Agarwal
CSAIL, Massachusetts Institute of Technology

ABSTRACT

This paper evaluates the Raw microprocessor. Raw addresses the challenge of building a general-purpose architecture that performs well on a larger class of stream and embedded computing applications than existing microprocessors, while still running existing ILP-based sequential programs with reasonable performance in the face of increasing wire delays. Raw approaches this challenge by implementing plenty of on-chip resources – including logic, wires, and pins – in a tiled arrangement, and exposing them through a new ISA, so that the software can take advantage of these resources for parallel applications. Raw supports both ILP and streams by routing operands between architecturally-exposed functional units over a point-to-point scalar operand network. This network offers low latency for scalar data transport. Raw manages the effect of wire delays by exposing the interconnect and using software to orchestrate both scalar and stream data transport.

We have implemented a prototype Raw microprocessor in IBM's 180 nm, 6-layer copper, CMOS 7SF standard-cell ASIC process. We have also implemented ILP and stream compilers. Our evaluation attempts to determine the extent to which Raw succeeds in meeting its goal of serving as a more versatile, general-purpose processor. Central to achieving this goal is Raw's ability to exploit all forms of parallelism, including ILP, DLP, TLP, and Stream parallelism. Specifically, we evaluate the performance of Raw on a diverse set of codes including traditional sequential programs, streaming applications, server workloads and bit-level embedded computation. Our experimental methodology makes use of a cycle-accurate simulator validated against our real hardware. Compared to a 180 nm Pentium-III, using commodity PC memory system components, Raw performs within a factor of 2x for sequential applications with a very low degree of ILP, about 2x to 9x better for higher levels of ILP, and 10x-100x better when highly parallel applications are coded in a stream language or optimized by hand. The paper also proposes a new versatility metric and uses it to discuss the generality of Raw.

1. INTRODUCTION

Fast moving VLSI technology will soon offer billions of transistors, massive chip-level wire bandwidth for local interconnect, and a modestly larger number of pins. However, there is growing evidence that wire delays become relatively more significant with shrinking feature sizes and clock speeds [6, 29, 1]. Processors need to convert the abundant chip-level resources into application performance, while mitigating the negative effects of wire delays.

The advance of technology also expands the number of applications that are implementable in VLSI. These applications include sequential programs that can run on today's ILP-based microprocessors, as well as highly parallel algorithms that are currently implemented directly using application-specific integrated circuits (ASICs). One such circuit is found in the Nvidia Ti 4600 graphics accelerator, which executes hundreds of parallel operations per cycle at 300 MHz. Many of these parallel ASICs implement algorithms with computational demands that are far beyond the capabilities of today's general-purpose microprocessors.

The Raw project addresses the challenge of whether a future general-purpose microprocessor architecture could be built that runs a greater subset of these ASIC applications while still running the same existing ILP-based sequential applications with reasonable performance in the face of increasing wire delays. To obtain some intuition on how to approach this challenge, we conducted an early study [4, 48] on the factors responsible for the significantly better performance of application-specific VLSI chips. We concluded that there were four main factors: specialization; exploitation of parallel resources (gates, wires and pins); management of wires and wire delay; and management of pins.

1. **Specialization:** ASICs specialize each "operation" at the gate level. In both the VLSI circuit and microprocessor context, an operation roughly corresponds to the unit of work that can be done in one cycle. A VLSI circuit forms operations by combinational logic paths, or "operators", between flip-flops. A microprocessor, on the other hand, has an instruction set that defines the operations that can be performed. Specialized operators, for example, for implementing an incompatible floating point operation, or implementing a linear feedback shift register, can yield an order of magnitude performance improvement over an extant general purpose processor that may require many instructions to perform the same one-cycle operation as the VLSI hardware. As an example, customized Tensilica ASIC processors take advantage of specialization by augmenting a general purpose processor core with specialized instructions for specific applications.

2. **Exploitation of Parallel Resources:** ASICs further exploit plentiful silicon area to implement enough operators and communications channels to sustain a tremendous number of parallel operations in each clock cycle. Applications that merit direct digital VLSI circuit implementations typically exhibit massive, operation-level parallelism. While an aggressive VLIW implementation like Intel's Itanium II [32] executes six instructions per cycle, graphics accelerators may perform hundreds or thousands of word-level operations per cycle. Because they operate on very small word operands, logic emulation circuits such as Xilinx II Pro FPGAs can perform hundreds of thousands of operations each cycle. The recent addition of MMX and SSE-style multigranular instructions that operate on multiple subwords simultaneously marks an effort to improve the efficiency of microprocessors by exploiting additional parallelism available due to smaller word sizes.

The Itanium II die photo reveals that less than two percent of the die area is dedicated to its 6-way issue integer execution core.

Clearly, the ALU area is not a significant constraint on the execution width of a modern-day wide-issue microprocessor. On the other hand, the presence of many physical execution units is a minimum prerequisite to the exploitation of the same massive parallelism that ASICs are able to exploit.

3. **Management of Wires and Wire Delay:** ASIC designers can place and wire communicating operations in ways that minimize wire delay, minimize latency, and maximize bandwidth. In contrast, it is now well known that the delay of the interconnect inside traditional microprocessors limits scalability [36, 1, 15, 38, 45]. Itanium II's 6-way integer execution unit presents evidence for this – it spends over half of its critical path in the bypass paths of the ALUs. ASIC designers manage wire delay inherent in large distributed arrays of function units in multiple steps. First, they place close together operations that need to communicate frequently. Second, when high bandwidth is needed, they create multiple customized communication channels. Finally, they introduce pipeline registers between distant operators, thereby converting propagation delay into pipeline latency. By doing so, the designer acknowledges the inherent tradeoff between parallelism and latency: leveraging more resources requires signals to travel greater distances. The Alpha 21264 is an example of a microprocessor that acknowledges this tradeoff on a small scale: it incurs a one-cycle latency for signals to travel between its two integer clusters.

4. **Management of Pins:** ASICs customize the usage of their pins. Rather than being bottlenecked by a cache-oriented multi-level hierarchical memory system (and subsequently by a generic PCI-style I/O system), ASICs utilize their pins in ways that fit the applications at hand, maximizing realizable I/O bandwidth or minimizing latency. This efficiency applies not just when an ASIC accesses external DRAMs, but also in the way that it connects to high-bandwidth input devices like wide-word analog-to-digital converters, CCDs, and sensor arrays. There are currently few easy ways to arrange for these devices to stream data into a general-purpose microprocessor in a high-bandwidth way, especially since DRAM must almost always be used as an intermediate buffer. In some senses, microprocessors strive to minimize, rather than maximize, the usage of pin resources, by hiding them through a hierarchy of caches.

Our goal was to build a microprocessor that could leverage the above four factors, and yet implement the gamut of general-purpose features that we expect in a microprocessor such as functional unit virtualization, unpredictable interrupts, instruction virtualization, and data caching. The processor also needed to exploit ILP in sequential programs, as well as space and time multiplex (i.e., context switch) threads of control. Furthermore, these multiple threads of control should be able to communicate and coordinate in an efficient fashion.

This paper evaluates the Raw microprocessor and discusses our success in achieving these goals. Raw takes the following approach to leveraging the four factors behind the success of ASICs.

1. Raw implements the most common operations needed by ILP or stream applications in specialized hardware mechanisms. Most of the primitive mechanisms are exposed to software through a new ISA. These mechanisms include the usual integer and floating point operations, specialized bit manipulation operations, scalar operand routing between adjacent function units, operand bypass between function units, registers and I/O queues, and data cache access (i.e., data load with tag check).

2. Raw implements a large number of these operators which exploit the copious VLSI resources – including gates, wires and pins – and exposes them through a new ISA, such that the software can take advantage of them for both ILP and highly parallel applications.

3. Raw manages the effect of wire delays by exposing the wiring channel operators to the software, so that the software can account for latencies by orchestrating both scalar and stream data transport. By orchestrating operand flow on the interconnect, Raw can also create customized communications patterns. Taken together, the wiring channel operators provide the abstraction of a scalar operand network [45] that offers very low latency for scalar data transport and enables the exploitation of ILP.

4. Raw software manages the pins for cache data fetches and for specialized stream interfaces to DRAM or I/O devices.

We have implemented a prototype Raw microprocessor in the SA-27E ASIC flow, which uses IBM's CMOS 7SF, an 180nm, 6-layer copper process. We received 120 chips from IBM in October of 2002. We have built a prototype Raw motherboard containing a single Raw chip, SDRAM chips, I/O interfaces and interface FPGAs. We have also implemented ILP and stream compilers for sequential programs and stream applications respectively. Our development tools include a validated, cycle-accurate simulator, an RTL-level simulator, and a logic emulator.

Our evaluation attempts to determine the extent to which Raw succeeds in meeting its goal of serving as a more versatile, general-purpose processor. Specifically, we evaluate the performance of Raw on applications drawn from several classes of computation including ILP, streams and vectors, server, and bit-level embedded computation. Our initial results are very encouraging (see Figure 3 for a quick sampling of our results). For each of the application classes, we find that Raw is able to perform at or close to the level of the best-in-class machine – i.e., the best specialized machine for the given application class. For example, for sequential applications with varying degrees of ILP, the performance of Raw ranges from 0.5x to 9x to that of a Pentium III (P3) processor implemented in the same technology generation as Raw. For streaming computations, Raw's performance is 10x to 100x better than the P3, and comparable to that of specialized stream and vector engines like Imagine [17] and VIRAM [21].

We present a metric called versatility that folds into a single scalar number the performance of an architecture over many application classes, and show that Raw's is seven times better than that for the P3. Our future work will expand the number of applications in each class and see if this trend continues to hold.

The rest of this paper is organized as follows. Section 2 provides an overview of the Raw architecture and its mechanisms for specialization, exploitation of parallel resources, orchestration of wires, and management of pins. Section 3 describes the implementation of Raw, and Section 4 provides detailed results. Section 5 introduces our versatility metric and analyzes the results. Section 6 follows with a detailed discussion of related work, and Section 7 concludes the paper.

2. ARCHITECTURE OVERVIEW

This section provides a brief overview of the Raw architecture. A more detailed discussion of the architecture is available elsewhere [43, 44, 45].

Tiled Architecture The Raw architecture supports an ISA that provides a parallel interface to the gate, pin, and wiring resources of the chip through suitable high level abstractions. As illustrated in Figure 1, the Raw processor exposes the copious gate resources of the chip by dividing the usable silicon area into an array of 16 identical, programmable tiles. A tile contains an 8-stage in-order single-issue MIPS-style processing pipeline, a 4-stage single-precision pipelined

FPU, a 32 KB data cache, two types of communication routers – static and dynamic, and 32KB and 64KB of software-managed instruction caches for the processing pipeline and static router respectively. Each tile is sized so that the amount of time for a signal to travel through a small amount of logic and across the tile is one clock cycle. We expect that the Raw processors of the future will have hundreds or even thousands of tiles.

On-chip Networks The tiles are interconnected by four 32-bit full-duplex on-chip networks, consisting of over 12,500 wires (see Figure 1). Two of the networks are static (routes are specified at compile time) and two are dynamic (routes are specified at run time). Each tile is connected only to its four neighbors. Every wire is registered at the input to its destination tile. This means that *the longest wire in the system is no greater than the length or width of a tile*. This property ensures high clock speeds, and the continued scalability of the architecture.

The design of Raw's on-chip interconnect and its interface with the processing pipeline are its key innovative features. These on-chip networks are exposed to the software through the Raw ISA, thereby giving the programmer or compiler the ability to directly program the wiring resources of the processor, and to carefully orchestrate the transfer of data values between the computational portions of the tiles – much like the routing in an ASIC. Effectively, the wire delay is exposed to the user as network hops. To go from corner to corner of the processor takes 6 hops, which corresponds to approximately six cycles of wire delay. To minimize the latency of inter-tile scalar data transport, which is critical for ILP, the on-chip networks are not only register-mapped but also integrated directly into the bypass paths of the processor pipeline.

Raw's on-chip interconnects belong to the class of scalar operand networks [45], which lend an interesting way of looking at modern day processors. The register file used to be the central communication mechanism between functional units in a processor. Starting with the first pipelined processors, the bypass network has become largely responsible for the communication of active values, and the register file is more of a checkpointing facility for inactive values. The Raw networks (the static networks in particular) are in one sense 2-D bypass networks serving as bridges between the bypass networks of separate tiles.

The static router in each tile contains a 64KB software-managed instruction cache and a pair of routing crossbars. Compiler generated routing instructions are 64 bits and encode a small command (e.g., conditional branch with/without decrement) and several routes – one for each crossbar output. These single-cycle routing instructions are one example of Raw's use of specialization. Because the router program memory is cached, there is no practical architectural limit on the number of simultaneous communication patterns that can be supported in a computation. This feature, coupled with the extremely low latency and low occupancy of in-order inter-tile ALU-to-ALU operand delivery (3 cycles nearest neighbor) distinguishes Raw from prior systolic or message passing systems [3, 12, 23].

Raw's two dynamic networks support cache misses, interrupts, dynamic messages, and other asynchronous events. The two networks use dimension-ordered routing and are structurally identical. One network, the memory network, follows a deadlock-avoidance strategy to avoid end-point deadlock. It is used in a restricted manner by trusted clients such as data caches, DMA and I/O. The second network, the general network, is used by untrusted clients, and relies on a deadlock recovery strategy [23].

Raw supports context switches. On a context switch, the contents of the processor registers and the general and static networks on a subset of the Raw chip occupied by the process (possibly including multiple tiles) are saved off, and the process and its network data can be restored at any time to a new offset on the Raw grid.

Direct I/O Interfaces On the edges of the network, the network channels are multiplexed down onto the pins of the chip to form flexible I/O ports that can be used for DRAM accesses or external device I/O. In order to toggle a pin, the user programs one of the on-chip networks to route a value off the side of the array. Our 1657 pin CCGA (ceramic column-grid array) package provides us with fourteen full-duplex, 32-bit I/O ports. Raw implementations with fewer pins are made possible via logical channels (as is already the case for two out of the sixteen logical ports), or simply by bonding out only a subset of the ports.

The static and dynamics networks, the data cache of the compute processors, and the external DRAMs connected to the I/O ports comprise Raw's memory system. The memory network is used for cache-based memory traffic, while the static and general dynamic networks are used for stream-based memory traffic. Memory intensive domains can have up to 14 full-duplex full-bandwidth DRAM memory banks to be placed on the 14 I/O ports of the chip. Minimal embedded Raw systems may eliminate DRAM altogether – they may use a single boot ROM so that Raw can execute out of the on-chip memory. In addition to transferring data directly to the tiles, off-chip devices connected to the I/O ports can route through the on-chip networks to other devices in order to perform glueless DMA and peer-to-peer communication.[1]

ISA Analogs to Physical Resources By creating first class architectural analogs to the physical chip resources, Raw attempts to minimize the ISA gap – that is, the gap between the resources that a VLSI chip has available and the amount of resources that are usable by software. Unlike conventional ISAs, Raw exposes the quantity of all three underlying physical resources (gates, wires and pins) to the ISA. Furthermore, it does this in a manner that is backwards-compatible – the instruction set does not change with varying degrees of resources.

Physical Entity	Raw ISA analog	Conventional ISA Analog
Gates	Tiles, new instructions	New instructions
Wires, Wire delay	Routes, Network Hops	none
Pins	I/O ports	none

Table 1: How Raw converts increasing quantities of physical entities into ISA entities

Table 1 contrasts the way the Raw ISA and conventional ISAs expose physical resources to the programmer. Because the Raw ISA has more direct interfaces, Raw processors can have more functional units, and have more flexible and more efficient pin utilization. High-end Raw processors will typically have more pins, because the architecture is better at turning pin count into performance and functionality. Finally, Raw processors are more predictable and have higher frequencies because of the explicit exposure of wire delay.

This approach, exposure, makes Raw scalable. Creating subsequent, more powerful, generations of the processor is straightforward: we simply stamp out as many tiles and I/O ports as the silicon die and package allow. The design has no centralized resources, no global buses, and no structures that get larger as the tile or pin count increases. Finally, the longest wire, the design complexity, and the verification complexity are all independent of transistor count.

3. IMPLEMENTATION

The Raw chip is a 16-tile prototype implemented in IBM's 180 nm 1.8V 6-layer CMOS 7SF SA-27E copper process. Although the Raw array is only 16 mm x 16 mm, we used an 18.2 mm x 18.2

[1] In fact, we are building a 4x4 IP packet router using a single Raw chip and its peer-to-peer capability.

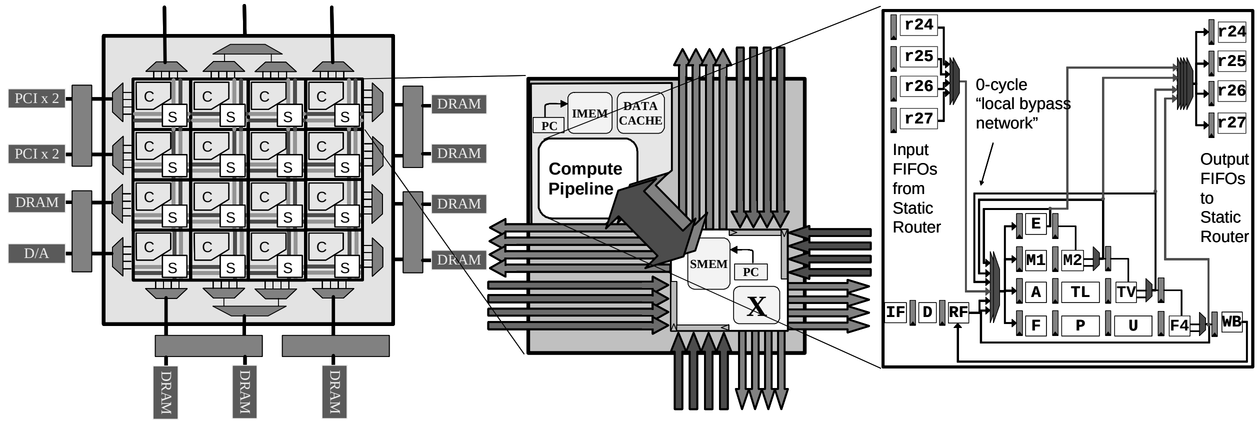

Figure 1: The Raw microprocessor comprises 16 tiles. Each tile has a compute processor, routers, network wires, and instruction and data memories.

mm die to allow us to use the high pin-count package. The 1657 pin ceramic column grid array package (CCGA) provides us with 1080 high speed transceiver logic (HSTL) I/O pins. Our measurements indicate that the chip core averages 18.2 watts at 425MHz. We quiesce unused functional units and memories and tri-state unused data I/O pins. We targeted a 225 MHz worst-case frequency in our design, which is competitive with other 180 nm lithography ASIC processors, like VIRAM, Imagine, and Tensilica's Xtensa series. The nominal running frequency is typically higher – the Raw chip core, running at room temperature, reaches 425MHz at 1.8V, and 500 MHz at 2.2V. This compares favorably to IBM-implemented microprocessors in the same process; the PowerPC 405GP runs at 266-400 MHz, while the follow-on PowerPC 440GP reaches 400-500 MHz.

We pipelined our processor aggressively and treated control paths very conservatively in order to ensure that we would not have to spend significant periods closing timing in the backend. Despite these efforts, wire delay inside a tile was still large enough that we were forced to create an infrastructure to place the cells in the timing and congestion critical data paths. More details on the Raw implementation are available in [44].

A difficult challenge for us was to resist the temptations of making the absolutely highest performance, highest frequency tile processor, and instead to concentrate on the research aspects of the project, such as the design of Raw's scalar operand network. As one can infer from Section 5, moving from a one-way issue compute processor to a two-way issue compute processor would have likely improved our performance on low-ILP applications. Our estimates indicate that such a compute processor would have easily fit in the remaining white space of the tile. The frequency impact of transitioning from 1-issue to 2-issue is generally held to be small.

With our collaborators at ISI-East, we have designed a prototype motherboard (shown in Figure 2) around the Raw chip that we use to explore a number of applications with extreme computation and I/O requirements. A larger system, consisting of 64 Raw chips, connected to form a virtual 1024 tile Raw processor, is also being fabricated in conjunction with ISI-East.

4. RESULTS

This section presents measurement and experimental results of the Raw microprocessor. We begin by explaining our experimental methodology. Then we present some basic hardware statistics. The remainder of the section focuses on evaluating how well Raw supports a range of programming models and application types. The domains we examine include ILP computation, stream and embedded computation, server workloads, and bit-level computation. The performance of Raw in these individual areas are presented as comparison to a reference 600 MHz Pentium III.

Factor responsible	Max. Speedup
Tile parallelism (Exploitation of Gates)	16x
Load/store elimination (Management of Wires)	4x
Streaming mode vs cache thrashing (Management of Wires)	15x
Streaming I/O bandwidth (Management of Pins)	60x
Increased cache/register size (Exploitation of Gates)	∼2x
Bit Manipulation Instructions (Specialization)	3x

Table 2: Sources of speedup for Raw over P3.

We note that Raw achieves greater than 16x speedup (either versus a Pentium or versus a single tile) for several applications because of compounding or additive effects from several factors listed in Table 2. The following is a brief discussion of these effects.

1. When all 16 tiles can be used, the speedup can be 16-fold.

2. If a, b, and c are variables in memory, then an operation of the form $c = a + b$ in a load-store RISC architecture will require a minimum of 4 operations – two loads, one add, and one store. Stream architectures such as Raw can accomplish the operation in a single operation (for a speedup of 4x) because processor can issue bulk data stream requests and then process data directly from the network without going through the cache.

3. When vector lengths exceed the cache size, streaming data from off-chip DRAM directly into the ALU achieves 7.5x the throughput of cache accesses (each cache miss transports 8 words in 60 cycles, while streaming can achieve one word per cycle). The streaming effect is even more powerful with strided requests that use only part of a full cache line. In this case, streaming throughput is 15 times greater than going through the cache.

4. Raw has 60x the I/O bandwidth of the P3. Furthermore, Raw's direct programmatic interface to the pins enables efficient utilization.

5. When multiple tiles are used in a computation, the effective number of registers and cache lines is increased, allowing a greater working set to be accessed without penalty. We approximate the potential speedup from this effect as 2-fold.

6. Finally, specialized bit manipulation instructions can optimize table lookups, shifts, and logical operations. We estimate the potential speedup from this effect as 3-fold.

4.1 Experimental methodology

Validated Simulator The evaluation for this paper makes use of a validated cycle-accurate simulator of the Raw chip. Using the validated simulator as opposed to actual hardware allows us to better normalize differences with a reference system, e.g., DRAM memory latency, and instruction cache configuration. It also allows us to

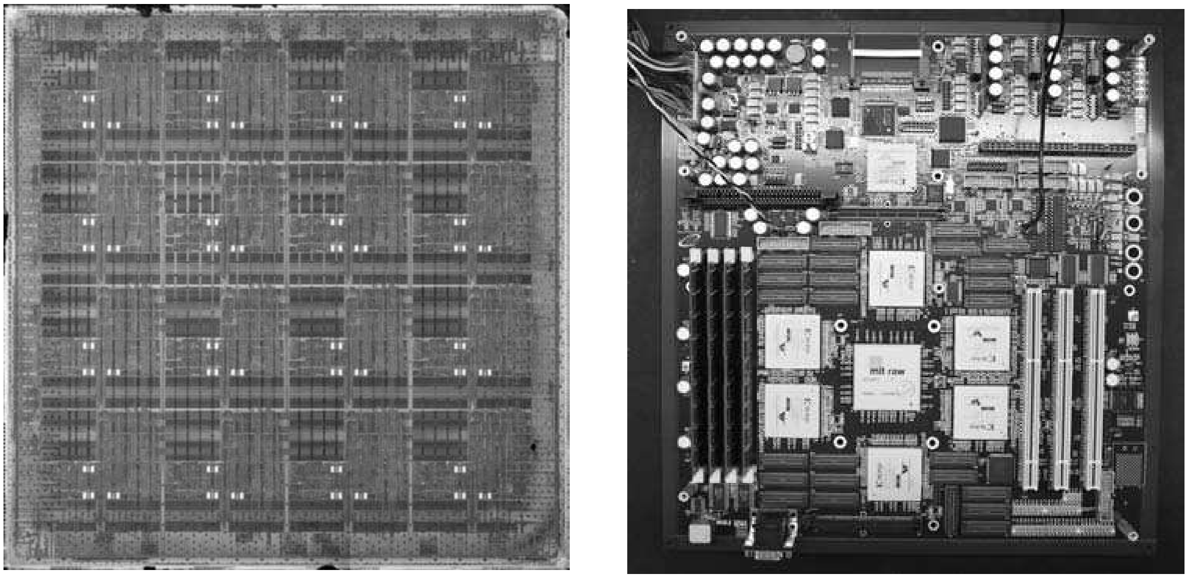

Figure 2: Photos of the Raw chip and Raw prototype motherboard respectively.

explore alternative motherboard configurations. We verified that the simulator and the gate-level RTL netlist have *exactly* the same timing and data values for all 200,000 lines of our hand-written assembly test suite, as well as for a number of C applications and randomly generated tests. Every stall signal, register file write, SRAM write, on-chip network wire, cache state machine transition, interrupt signal, and chip signal pin matches in value on every cycle between the two. This gate-level RTL netlist was then shipped to IBM for manufacturing. Upon receipt of the chip, we compared a subset of the tests on the actual hardware to verify that the chip was manufactured according to spec.

Selection of a Reference Processor In our evaluation, we realized the importance of tying our performance numbers to an existing commercial system. For fairness, this comparison system must be implemented in a process that uses the same lithography generation, 180 nm. Furthermore, the reference processor needs to be measured at a similar point in its lifecycle, i.e., as close to first silicon as possible. This is because most commercial systems are speedpath or process tuned after first silicon is created [7]. For instance, the 180nm P3 initial production silicon was released at 500-733 MHz and gradually was tuned until it reached a final production silicon frequency of 1 GHz. The first silicon value for the P3 is not publicly known. However, the frequencies of first-silicon and initial production silicon have been known to differ by as much as 2x.

The P3 is especially ideal for comparison with Raw because it is in common use, because its fabrication process is well documented, and because the common-case functional unit latencies are almost identical. The back ends of the processors share a similar level of pipelining, which means that relative cycle-counts carry some significance. Conventional VLSI wisdom suggests that, when normalized for process, Raw's single-ported L1 data cache should have approximately the same area and delay as the P3's two-ported L1 data cache of half the size. For sequential codes with working sets that fit in the L1 caches, the cycle counts should be quite similar. And given that the fortunes of Intel have rested (and continue to rest, with the Pentium-M reincarnation) upon this architecture for almost ten years, there is reason to believe that the implementation is a good one. In fact, the P3, upon release in 4Q'99, had the highest SpecInt95 value of any processor [13].

Itanium and Pentium 4 (P4) came as close seconds to our final choice. Our decision to avoid them came from our need to match the lifecycle of the reference system to Raw's. Intel's market pressures cause it to delay the release of new processors such as P4 or Itanium until they have been tuned enough to compete with the existing Pentium product line. Consequently, when these processors are released, they may be closer to final-silicon than first-silicon. For example, it is documented in the press that Itanium I was delayed for two years between first-silicon announcement and initial production silicon availability. Finally, the implementation complexity of Raw is more similar to the P3 than P4 or Itanium.

Comparison of Silicon Implementations Table 3 compares the two chips and their fabrication processes, IBM's CMOS 7SF [27, 37] and Intel's P858 [51]. CMOS 7SF has denser SRAM cells and less interconnect resistivity, due to copper metalization. P858, on the other hand, attempts to compensate for aluminum metalization by using a lower-k dielectric, SiOF, and by carefully varying the aspect ratios of the wires.

Parameter	**Raw (IBM ASIC)**	**P3 (Intel)**
Lithography Generation	180 nm	180 nm
Process Name	CMOS 7SF (SA-27E)	P858
Metal Layers	Cu 6	Al 6
Dielectric Material	SiO_2	SiOF
Oxide Thickness (T_{ox})	3.5 nm	3.0 nm
SRAM Cell Size	4.8 μm^2	5.6 μm^2
Dielectric *k*	4.1	3.55
Ring Oscillator Stage (FO1)	23 ps	11 ps
Dynamic Logic, Custom Macros (SRAMs, RFs)	no	yes
Speedpath Tuning since First Silicon	no	yes
Initial Frequency	425 MHz	500-733 MHz
Die Area[2]	331 mm^2	106 mm^2
Signal Pins	$\sim$ 1100	$\sim$ 190
Vdd used	1.8 V	1.65 V
Nominal Process Vdd	1.8 V	1.5 V

Table 3: Comparison of Implementation Parameters for Raw and P3-Coppermine.

The Ring Oscillator metric measures the delay of a fanout-of-1 (FO1) inverter. It has been suggested that an approximate FO4 delay can be found by multiplying the FO1 delay by 3 [15]. Thus, P858 gates appear to be significantly (2.1x) faster than the CMOS

7SF gates. This is to be expected, as IBM terms CMOS 7SF a "value" process. IBM's non-ASIC, high-performance, 180 nm process, CMOS 8S, is competitive with P858 [8], and has ring oscillator delays of 11 ps and better. Furthermore, production 180 nm P3's have their voltages set 10% higher than the nominal process voltage, which typically improves frequency by 10% or more.

A recent book, [7], lists a number of limitations that ASIC processor implementations face versus full-custom implementations. We mention some applicable ones here. First, because the ASIC flow predetermines aspects of a chip, basic overheads are relatively high in comparison to full-custom designs. Two of Raw's largest overheads were the mandatory scan flip-flops (18%), and clock skew and jitter (13%). Second, ASIC flows tend to produce logic that is significantly less dense than corresponding custom flows. Third, ASIC flows prevent use of custom or dynamic logic, except for a limited menu (up to 2 read ports and 2 write ports) of fixed pipeline-depth register files and SRAMs, which are machine generated. A 40-80% improvement in frequency often is attributed to the use of dynamic logic. Process and speedpath tuning account for 35%. Finally, speed-binning yields approximately 20%.

We compensate only for the last two factors in this paper. We selected a 600 MHz P3 as the reference system. The 600 MHz P3, released prior to process tuning, and after limited speedpath tuning, is solidly in the middle of the P3 initial production frequency range, presumably representing an average-speedbin part.

We believe that a Raw implementation with the same engineering effort and process technology as the Intel P3 would be smaller and significantly faster. However, we make no attempt to normalize for these factors.

Normalization Details With the selection of a reference CPU implementation comes a selection of an enclosing computer. We used a pair of 600 MHz Dell Precision 410's to run our reference benchmarks. We outfitted these machines with identical 100 MHz 2-2-2 PC100 256 MB DRAMs, and wrote several microbenchmarks to verify that the memory system timings matched.

To compare the Raw and Dell systems more equally, we used the Raw simulator's extension language to implement a cycle-matched PC100 DRAM model and a chipset[4]. This model has the same wall-clock latency and bandwidth as the Dell 410. However, since Raw runs at a slower frequency than the P3, the latency, measured in cycles, is less. We use the term **RawPC** to describe a simulation which uses 8 PC100 DRAMs, occupying 4 ports on the left hand side of the chip, and 4 on the right hand side.

Because Raw is also designed for streaming applications, we also wanted to measure applications that use the full pin bandwidth of the chip. In this case, we use a simulation of CL2 PC 3500 DDR DRAM, which provides enough bandwidth to saturate both directions of a Raw port. In this case, we use 16 PC 3500 DRAMs, attached to all 16 logical ports on the chip, in conjunction with a memory controller, implemented in the chipset, that supports a number of stream requests. A Raw tile can send a message over the general dynamic network to the chipset to initiate large bulk transfers from the DRAMs into and out of the static network. Simple interleaving and striding is supported, subject to the underlying access and timing constraints of the DRAM. We call this configuration **RawStreams**.

The placement of a DRAM on a Raw port does not exclude the use of other devices on that port – the chipsets have a simple demultiplexing mechanism that allows multiple devices to connect to a single port.

Operation	Latency		Throughput	
	1 Raw Tile	P3	Raw	P3
ALU	1	1	1	1
Load (hit)	3	3	1	1
Store (hit)	-	-	1	1
FP Add	4	3	1	1
FP Mul	4	5	1	1/2
Mul	2	4	1	1
Div	42	26	1	1
FP Div	10	18	1/10	1/18
SSE FP 4-Add	-	4	-	1/2
SSE FP 4-Mul	-	5	-	1/2
SSE FP 4-Div	-	36	-	1/36

Table 4: Functional unit timings. Commonly executed instructions appear first. FP operations are single precision.

	1 Raw Tile	P3
CPU Frequency	425 MHz	600 MHz
Sustained Issue Width	1 in-order	3 out-of-order
Mispredict Penalty	3	10-15
DRAM Freq (RawPC)	100 MHz	100 MHz
DRAM Freq (RawStreams)	2 x 213 MHz	-
DRAM Access Width	8 bytes	8 bytes
L1 D cache size	32K	16K
L1 D cache ports	1	2
L1 I cache size	32K	16K
L1 miss latency	54 cycles	7 cycles
L1 fill width	4 bytes	32 bytes
L1 / L2 line sizes	32 bytes	32 bytes
L1 associativities	2-way	4-way
L2 size	-	256K
L2 associativity	-	8-way
L2 miss latency	-	79 cycles
L2 fill width	-	8 bytes

Table 5: Memory system data.

Except where otherwise noted, we used gcc 3.3 -O3 to compile C and Fortran code for both Raw[5] and the P3[6]. For programs that do C or Fortran stdio calls, we use newlib 1.9.0 for both Raw and the P3. Finally, to eliminate the impact of disparate file and operating systems, the results of I/O system calls for the Spec benchmarks were captured and embedded into the binaries as static data using [42].

We performed one final normalization. Our preliminary Raw software-managed instruction-caching system has not been optimized, which made it difficult to compare the two systems. To enable comparisons with the P3, we augmented the cycle-accurate simulator so that it employs conventional 2-way associative hardware instruction caching. These instruction caches are modelled cycle-by-cycle in the same manner as the rest of the hardware. Like the data caches, they service misses over the memory dynamic network. Resource contention between the caches is modeled accordingly.

4.2 Basic data

Tables 4 and 5 show functional unit timings and memory system characteristics for both systems, respectively. Table 6 shows Raw's measured power consumption [19]. Table 7 lists a breakdown of the end-to-end message latency on Raw's scalar operand network. The low 3-cycle inter-tile ALU-to-ALU latency and zero cycle send and receive occupancies are critical for obtaining good ILP performance.

4.3 ILP Computation

This section examines how well Raw is able to support conventional sequential applications. Typically, the only form of parallelism available in these applications is ILP-level parallelism. For

[2] Note that despite the area penalty for an ASIC implementation, it is almost certain that the Raw processor is a bigger design than the P3. Our evaluation does not aim to make a cost-normalized comparison, but rather seeks to demonstrate the scalability of our approach for future microprocessor designs.

[4] The support chips typically used to interface a processor to its memory system and I/O peripherals.

[5] The Raw gcc backend, based on the MIPS backend, targets a single tile's compute and network resources.

[6] For P3, we added -march=pentium3 -mfpmath=sse

	Core	Pins
Idle - Full Chip	9.6 W	0.02 W
Average - Per Active Tile	0.54 W	-
Average - Per Active Port	-	0.2 W
Average - Full Chip	18.2 W	2.8 W

Table 6: Raw power consumption at 425 MHz, 25^o C

	Latency
Sending Processor Occupancy	0
Latency to Network Input	1
Latency per hop	1
Latency from Network Output to ALU	1
Receiving Processor Occupancy	0

Table 7: Breakdown of the end-to-end latency (in cycles) for a one-word message on Raw's static network.

this evaluation, we select a range of benchmarks that encompasses a wide spectrum of program types and degree of ILP.

Much like a VLIW architecture, Raw is designed to rely on the compiler to find and exploit ILP. We have developed Rawcc [5, 24, 25] to explore these compilation issues. Rawcc takes sequential C or Fortran programs and orchestrates them across the Raw tiles in two steps. First, Rawcc distributes the data and code across the tiles to attempt to balance the tradeoff between locality and parallelism. Then, it schedules the computation and communication to maximize parallelism and minimize communication stalls.

Rawcc was developed as a prototyping environment for exploring compilation techniques for Raw. As such, unmodified Spec applications stretch its robustness. We are working on improving the robustness of Rawcc. Additionally, we have made progress on a follow-on parallelizing compiler which has more focus on robustness and code quality. The speedups attained in Table 8 shows the potential of automatic parallelization and ILP exploitation on Raw. Of the benchmarks compiled by Rawcc, Raw is able to outperform the P3 for all the scientific benchmarks and several irregular applications.

Benchmark	Source	# Raw Tiles	Cycles on Raw	Speedup vs P3	
				Cycles	Time
Dense-Matrix Scientific Applications					
Swim	Spec95	16	14.5M	4.0	2.9
Tomcatv	Nasa7:Spec92	16	2.05M	1.9	1.3
Btrix	Nasa7:Spec92	16	516K	6.1	4.3
Cholesky	Nasa7:Spec92	16	3.09M	2.4	1.7
Mxm	Nasa7:Spec92	16	247K	2.0	1.4
Vpenta	Nasa7:Spec92	16	272K	9.1	6.4
Jacobi	Raw bench. suite	16	40.6K	6.9	4.9
Life	Raw bench. suite	16	332K	4.1	2.9
Sparse-Matrix/Integer/Irregular Applications					
SHA	Perl Oasis	16	768K	1.8	1.3
AES Decode	FIPS-197	16	292K	1.3	0.96
Fpppp-kernel	Nasa7:Spec92	16	169K	4.8	3.4
Unstructured	CHAOS	16	5.81M	1.4	1.0

Table 8: Performance of sequential programs on Raw and on a P3.

Table 9 shows the speedups achieved by Rawcc as the number of tiles varies from two to 16. The speedups are compared to performance of a single Raw tile. Overall, the source of speedups comes primarily from tile parallelism (see Table 2), but several of the dense matrix benchmarks benefit from increased cache capacity as well (which explains the super-linear speedups). In addition, Fpppp-kernel benefits from increased register capacity, which leads to fewer spills.

For completeness, we also compiled a selection of the Spec2000 benchmarks with gcc for a single tile, and ran them using the MinneSPEC's [20] *LgRed* data sets to reduce the length of simulations. The results, shown in Table 10, represent a lower bound for the performance of those codes on Raw, as they only use 1/16 of the resources on the Raw chip. The numbers are quite surprising; on average; the simple in-order Raw tile with no L2 cache is only

Benchmark	Number of tiles				
	1	2	4	8	16
Dense-Matrix Scientific Applications					
Swim	1.0	1.1	2.4	4.7	9.0
Tomcatv	1.0	1.3	3.0	5.3	8.2
Btrix	1.0	1.7	5.5	15.1	33.4
Cholesky	1.0	1.8	4.8	9.0	10.3
Mxm	1.0	1.4	4.6	6.6	8.3
Vpenta	1.0	2.1	7.6	20.8	41.8
Jacobi	1.0	2.6	6.1	13.2	22.6
Life	1.0	1.0	2.4	5.9	12.6
Sparse-Matrix/Integer/Irregular Applications					
SHA	1.0	1.5	1.2	1.6	2.1
AES Decode	1.0	1.5	2.5	3.2	3.4
Fpppp-kernel	1.0	0.9	1.8	3.7	6.9
Unstructured	1.0	1.8	3.2	3.5	3.1

Table 9: Speedup of the ILP benchmarks relative to single-tile Raw.

1.4x slower by cycles and 2x slower by time than the full P3. This suggests that in the event that the parallelism in these applications is too small to be exploited across Raw tiles, a simple two-way Raw compute processor might be sufficient to make the performance difference easily be hidden by other aspects of the system.

Benchmark	Source	# Raw Tiles	Cycles on Raw	Speedup vs P3	
				Cycles	Time
172.mgrid	SPECfp	1	.240B	0.97	0.69
173.applu	SPECfp	1	.324B	0.92	0.65
177.mesa	SPECfp	1	2.40B	0.74	0.53
183.equake	SPECfp	1	.866B	0.97	0.69
188.ammp	SPECfp	1	7.16B	0.65	0.46
301.apsi	SPECfp	1	1.05B	0.55	0.39
175.vpr	SPECint	1	2.52B	0.69	0.49
181.mcf	SPECint	1	4.31B	0.46	0.33
197.parser	SPECint	1	6.23B	0.68	0.48
256.bzip2	SPECint	1	3.10B	0.66	0.47
300.twolf	SPECint	1	1.96B	0.57	0.41

Table 10: Performance of SPEC2000 programs on one tile on Raw.

4.4 Stream computation

We present performance of stream computations for Raw. Stream computations arise naturally out of real-time I/O applications as well as from embedded applications. The data sets for these applications are often large and may even be a continuous stream in real-time, which makes them unsuitable for traditional cache based memory systems. Raw provides a more natural support for stream based computation by allowing data to be fetched efficiently through a register mapped, software orchestrated network.

We present two sets of results. First we show the performance of programs written in StreamIt, a high level stream language, and automatically compiled to Raw. Then, we show the performance of some hand written applications.

4.4.1 StreamIt

StreamIt is a high-level, architecture-independent language for high-performance streaming applications. StreamIt contains language constructs that improve programmer productivity for streaming, including hierarchical structured streams, graph parameterization, and circular buffer management; these constructs also expose information to the compiler and enable novel optimizations [47]. We have developed a Raw backend for the StreamIt compiler, which includes fully automatic load balancing, graph layout, communication scheduling, and routing [11].

We evaluate the performance of RawPC on several StreamIt benchmarks, which represent large and pervasive DSP applications. Table 11 summarizes the performance of 16 Raw tiles vs. a P3. For both architectures, we use StreamIt versions of the benchmarks; we do not compare to hand-coded C on the P3 because StreamIt performs at least 1-2X better for 4 of the 6 applications (this is due to aggressive unrolling and constant propagation in the StreamIt compiler). The comparison reflects two distinct influences: 1) the scaling of Raw performance as the number of tiles increases, and 2) the performance of a Raw tile vs. a P3 for the same StreamIt code. To distinguish between these influences, Table 12 shows detailed speedups relative to StreamIt code running on a 1-tile Raw configuration.

Benchmark	Cycles Per Output on Raw	Speedup vs P3	
		Cycles	Time
Beamformer	2074.5	7.3	5.2
Bitonic Sort	11.6	4.9	3.5
FFT	16.4	6.7	4.8
Filterbank	305.6	15.4	10.9
FIR	51.0	11.6	8.2
FMRadio	2614.0	9.0	6.4

Table 11: StreamIt performance results.

Benchmark	StreamIt on P3	StreamIt on n Raw tiles				
		1	2	4	8	16
Beamformer	3.0	1.0	4.1	4.5	5.2	21.8
Bitonic Sort	1.3	1.0	1.9	3.4	4.7	6.3
FFT	1.1	1.0	1.6	3.5	4.8	7.3
Filterbank	1.5	1.0	3.3	3.3	11.0	23.4
FIR	2.6	1.0	2.3	5.5	12.9	30.1
FMRadio	1.2	1.0	1.0	1.2	4.0	10.9

Table 12: Speedup (in cycles) of StreamIt benchmarks relative to a 1-tile Raw configuration. From left, the columns indicate the StreamIt version on a P3, and on Raw configurations with one to 16 tiles.

The primary result illustrated by Table 12 is that StreamIt applications scale effectively for increasing sizes of the Raw configuration. For FIR, FFT, and Bitonic, the scaling is approximately linear across all tile sizes (FIR is actually super-linear due to decreasing register pressure in larger configurations). For Beamformer, Filterbank, and FMRadio, the scaling is slightly inhibited for small configurations. This is because 1) these applications are larger, and IMEM constraints prevent an unrolling optimization for small tile sizes, and 2) they have more data parallelism, yielding speedups for large configurations but inhibiting small configurations due to a constant control overhead.

The second influence is the performance of a P3 vs. a single Raw tile on the same StreamIt code, as illustrated by the second column in Table 12. In most cases, performance is comparable. The P3 performs better in two cases because it can exploit ILP: Beamformer has independent real/imaginary updates in the inner loop, and FIR is a fully unrolled multiply-accumulate operation. In other cases, ILP is obscured by circular buffer accesses and control dependences.

In all, StreamIt applications benefit from Raw's exploitation of parallel resources and management of wires (see Table 19 for summary). The abundant parallelism and regular communication patterns in stream programs are an ideal match for the parallelism and tightly orchestrated communication on Raw. As stream programs often require high bandwidth, register-mapped communication serves to avoid costly memory accesses. Also, autonomous streaming components can manage their local state in Raw's distributed data caches and register banks, thereby improving locality. These aspects are key to the scalability demonstrated in the StreamIt benchmarks.

4.4.2 Hand written stream applications

ISI East, the MIT Oxygen Team, and MIT CAG have hand-written a wide range of stream based applications to take advantage of Raw as an embedded processor. This section presents the results. These include a set of linear algebra routines implemented as Stream Algorithms, the STREAM benchmark, and several other embedded applications including a real-time 1020-node acoustic beamformer. The benchmarks are typically written in C and compiled with gcc, with inline assembly for a subset of inner loops. Some of the simpler benchmarks like the STREAM benchmark and the FIR were small enough that coding entirely in assembly was most expedient.

Stream Algorithms Table 13 presents the performance of a set of linear algebra algorithms on RawPC versus the P3.

Benchmark	Problem Size	MFlops on Raw	Speedup vs P3	
			Cycles	Time
Matrix Multiplication	256 x 256	6310	8.6	6.3
LU factorization	256 x 256	4300	12.9	9.2
Triangular solver	256 x 256	4910	12.2	8.6
QR factorization	256 x 256	5170	18.0	12.8
Convolution	256 x 16	4610	9.1	6.5

Table 13: Performance of linear algebra routines.

The Raw implementations are coded as Stream Algorithms [16], which emphasize computational efficiency in space and time and are designed specifically to take advantage of tiled microarchitectures like Raw. They have three key features. First, stream algorithms operate directly on data from the interconnect and achieve an asymptotically optimal 100 % compute efficiency for large numbers of tiles. Second, stream algorithms use no more than a small, bounded amount of storage on each processing element. Third, data are streamed through the compute fabric from and to peripheral memories.

With the exception of Convolution, we compare against the P3 running single precision Lapack (Linear Algebra Package). We use clapack version 3.0 [2] and a tuned BLAS implementation, ATLAS [50], version 3.4.2. We disassembled the ATLAS library to verify that it uses P3 SSE extensions appropriately to achieve high performance. Since Lapack does not provide a convolution, we compare against the Intel Integrated Performance Primitives (IPP).

As can be seen in Table 13, Raw performs significantly better than the P3 on these applications even with optimized P3 SSE code. Raw's better performance is due to load/store elimination (see Table 2), and the use of parallel resources. Stream Algorithms operate directly on values from the network and avoid loads and stores, thereby achieving higher utilization of parallel resources than the blocked code on the P3.

STREAM benchmark The STREAM benchmark was created by John McCalpin to measure sustainable memory bandwidth and the corresponding computation rate for vector kernels [30]. Its performance has been documented on thousands of machines, ranging from PCs and desktops to MPPs and other supercomputers.

Problem Size	Bandwidth (GB/s)			Raw/P3
	P3	Raw	NEC SX-7	
Copy	.567	47.6	35.1	84
Scale	.514	47.3	34.8	92
Add	.645	35.6	35.3	55
Scale & Add	.616	35.5	35.3	59

Table 14: Performance (by time) of STREAM benchmark.

We hand-coded an implementation of STREAM on RawStreams. We also tweaked the P3 version to use single precision SSE floating point, improving its performance. The Raw implementation employs 14 tiles and streams data between 14 processors and 14 memory ports through the static network. Table 14 displays the results. Raw is 55x-92x better than the P3. The table also includes the performance of STREAM on NEC SX-7 Supercomputer, which has the highest reported STREAM performance of any single-chip processor. Note that Raw surpasses that performance. This extreme single-chip performance is achieved by taking advantage of three Raw architectural features: its ample pin bandwidth, the ability to precisely route data values in and out of DRAMs with minimal overhead, and a careful match between floating point and DRAM bandwidth.

Other stream-based applications Table 15 presents the performance of some hand written stream applications on Raw. We are developing a real time 1020 microphone Acoustic Beamformer which will use the Raw system for processing. On this application, Raw runs 16 instantiations of the code and the microphones are striped in a data parallel manner across the array. Raw's software exposed I/O is also much more efficient than getting the stream data from DRAM in the case of the P3. Inputting and outputting data from DRAM is the best case for the P3. The P3 results would be much worse in an actual system where the data would come over a PCI bus. For the FIR, we compared to the Intel IPP. Results for Corner Turn, Beam Steering, and CSLC are discussed in the previously published [41].

Benchmark	Machine Config.	Cycles on Raw	Speedup vs P3	
			Cycles	Time
Acoustic Beamforming	RawStreams	7.83M	9.7	6.9
512-pt Radix-2 FFT	RawPC	331K	4.6	3.3
16-tap FIR	RawStreams	548K	10.9	7.7
CSLC	RawPC	4.11M	17.0	12.0
Beam Steering	RawStreams	943K	65	46
Corner Turn	RawStreams	147K	245	174

Table 15: Performance of hand written stream applications.

4.5 Server

To measure the performance of Raw on server-like workloads, we conduct the following experiment on RawPC to obtain SpecRate-like metrics. For each of a subset of Spec 2000 applications, we execute an independent copy of it on each of the 16 tiles, and we measure the overall throughput of that workload relative to a single run on the P3.

Table 16 presents the results. Note that the speedup of Raw versus P3 is equivalent to the throughput of Raw relative to P3's throughput. As anticipated, RawPC outperforms the P3 by a large margin, with an average throughput advantage of 10.8x (by cycles) and 7.6x (by time). The key Raw feature that enables this performance is the high pin bandwidth available to off-chip memory. RawPC contains eight separate memory ports to DRAM. This means that even when all 16 tiles are running applications, each memory port and DRAM is only shared among two applications.

Benchmark	Cycles on Raw	Speedup vs P3		Efficiency
		Cycles	Time	
172.mgrid	.240B	15.0	10.6	96%
173.applu	.324B	14.0	9.9	96%
177.mesa	2.40B	11.8	8.4	99%
183.equake	.866B	15.1	10.7	97%
188.ammp	7.16B	9.1	6.5	87%
301.apsi	1.05B	8.5	6.0	96%
175.vpr	2.52B	10.9	7.7	98%
181.mcf	4.31B	5.5	3.9	74%
197.parser	6.23B	10.1	7.2	92%
256.bzip2	3.10B	10.0	7.1	94%
300.twolf	1.96B	8.6	6.1	94%

Table 16: Performance of Raw on server workloads relative to the P3.

Table 16 shows the efficiency of RawPC's memory system for each server workload. Efficiency is the ratio between the actual throughput and the ideal 16x speedup attainable on 16 tiles. Less than the ideal throughput is achieved because of interference among memory requests originating from tiles that share the same DRAM banks and ports. We see that the efficiency is high across all the workloads, with an average of 93%.

4.6 Bit-Level Computation

We measure the performance of RawStreams on two bit-level computations [49]. Table 17 presents the results for the P3, Raw, FPGA, and ASIC implementations. We compare with a Xilinx Virtex-II 3000-5 FPGA, which is built on the same process generation as the Raw chip, and for the ASIC implementations we synthesize to the IBM SA-27E process that the Raw chip is implemented in. For each benchmark, we present three problem sizes: 1024, 16384, and 65536 samples. These problem sizes are selected to fit in the L1, L2, and miss in the cache on the P3, respectively. We use a randomized input sequence in all cases.

	Problem Size	Cycles on Raw	Speedup vs P3			
			Raw		FPGA	ASIC
			Cycles	Time	Time	Time
802.11a ConvEnc	1024 bits	1048	11.0	7.8	6.8	24
	16408 bits	16408	18.0	12.7	11	38
	65536 bits	65560	32.8	23.2	20	68
8b/10b Encoder	1024 bytes	1054	8.2	5.8	3.9	12
	16408 bytes	16444	11.8	8.3	5.4	17
	65536 bytes	65695	19.9	14.1	9.1	29

Table 17: Performance of two bit-level applications: 802.11a Convolutional Encoder and 8b/10b Encoder. The hand coded Raw implementations are compared to reference sequential implementations on the P3.

On these two applications, Raw is able to excel by exploiting fine-grain pipeline parallelism. To do this, the computations were spatially mapped across multiple tiles. Both applications benefited by more than 2x from Raw's specialized bit-level manipulation instructions, which reduce the latency of critical feedback loops. Another factor in Raw's high performance on these applications is Raw's exposed streaming I/O. This I/O model is in sharp contrast to having to move data though the cache hierarchy on a P3.

We also present in Table 18 results for the operation on 16 parallel input streams. This is to simulate a possible workload that a base-station communications chip may need to complete by encoding 16 simultaneous connections. For this throughput test, a more area efficient implementation was used on Raw. This implementation has lower peak performance, but by instantiating 16 instances, a higher throughput per area is achieved.

Benchmark	Problem Size	Cycles on Raw	Speedup vs P3	
			Cycles	Time
802.11a ConvEnc	16*64 bits	259	45	32
	16*1024 bits	4138	71	51
	16*4096 bits	16549	130	92
8b/10b Encoder	16*64 bytes	257	34	24
	16*1024 bytes	4097	47	33
	16*4096 bytes	16385	80	56

Table 18: Performance of two bit-level applications for 16 streams: 802.11a Convolutional Encoder and 8b/10b Encoder. This test simulates a possible workload for a base-station which processes multiple communication streams.

5. ANALYSIS

Sections 4.3 through 4.6 presented performance results for Raw for several application classes and showed that Raw's performance was within a factor of 2x of the P3 for low-ILP applications, 2x-9x better than the P3 for high-ILP applications, and 10-100x better for stream or embedded computations. Table 19 summarizes the primary features that are responsible for performance improvements on Raw.

In this section, we compare the performance of Raw to other machines that have been designed specifically with streams or embedded computation in mind. We also attempt to explore quantitatively the degree to which Raw succeeds in being a more versatile general-purpose processor. To do so, we selected a representative subset of applications from each of our computational classes, and obtained performance results for Raw, P3 and machines especially suited for each of those applications. We note that these results are exploratory in nature and not meant to be taken as any sort of proof of Raw's versatility, rather as an early indication of the possibilities.

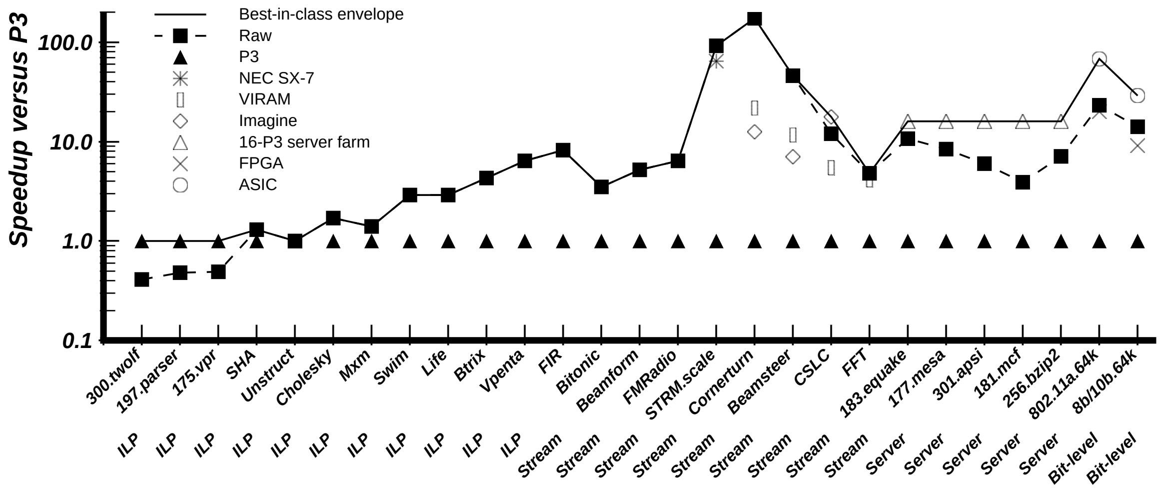

Figure 3: Performance of various architectures over several applications classes. Each point in the graph represents the speedup of an architecture over the P3 for a given application. The best-in-class envelope connects the best speedups for each application. The dashed line connects the speedups of Raw for all the applications. A versatile architecture has speedups close to the best-in-class envelope for all application classes. Imagine and VIRAM results are obtained from [41] and [34]. Bit-level results for FPGA and ASIC implementations are obtained from [49].

Figure 3 summarizes these results. We can make several observations from the figure. First, it is easy to see that the P3 does well relative to Raw for applications with low degrees of ILP, while the opposite is true for applications with higher degrees of ILP, such as Vpenta. For streams and vectors, the performance of Raw is comparable to that of stream and vector architectures like VIRAM and Imagine. All three outperform the P3 by factors of 10x to 100x. Raw, using the RawStreams configuration, beats the highest reported single-chip STREAM memory bandwidth champion, the NEC SX-7 Supercomputer, and is 55x-92x better than the P3. Essential to Raw's performance on this benchmark is the ample pin bandwidth, the ability to precisely route data values in and out of DRAM with minimal overhead, and a careful match between floating point and DRAM bandwidth.

Category	Benchmarks	S	R	W	P
ILP	Swim, Tomcatv, Btrix, Cholesky, Vpenta, Mxm, Life, Jacobi, Fpppp-kernel, SHA, AES Encode, Unstructured, 172.mgrid, 173.applu, 177.mesa, 183.equake, 188.ammp, 301.apsi, 175.vpr, 181.mcf, 197.parser, 256.bzip2, 300.twolf	X	X	X	
Stream:StreamIt	Beamformer, Bitonic Sort, FFT, Filterbank, FIR, FMRadio	X	X	X	
Stream:Stream Algo.	Mxm, LU fact., Triang. solver, QR fact., Conv.	X	X	X	
Stream:STREAM	Copy, Scale, Add, Scale & Add			X	X
Stream:Other	Acoustic Beamforming, FIR, FFT, Beam Steering	X	X	X	
	Corner Turn			X	X
	CSLC	X	X		
Server	172.mgrid, 173.applu, 177.mesa, 183.equake, 188.ammp, 301.apsi, 175.vpr, 181.mcf, 197.parser, 256.bzip2, 300.twolf		X		X
Bit-Level	802.11a ConvEnc, 8b/10b Encoder	X	X	X	

Table 19: Raw feature utilization table. S = Specialization. R = Exploiting Parallel Resources. W = Management of Wire Delays. P = Management of Pins.

We chose a server farm with 16 P3s as our best-in-class server system. Notice that a single-chip Raw system comes within a factor of three of this server farm for most applications. (Note that this is only a pure performance comparison, and we have not attempted to normalize for cost.) We chose FPGAs and ASICs as the best in class for our embedded bit-level applications. Raw's performance is comparable to that of an FPGA for these applications, and is a factor of 2x to 3x off from an ASIC. (Again, note that we are only comparing performance – ASICs use significantly lower area and power than Raw [49].) Raw performs well on these applications for the same reasons that FPGAs and ASICs do – namely, a careful orchestration of the wiring or communication patterns.

Thus far, our analysis of Raw's flexibility has been qualitative. Given the recent interest in flexible, versatile or polymorphic architectures such as Tarantula [9], Scale [22], Grid [33], and SmartMemories [28], which attempt to perform well over a wider range of applications than extant general purpose processors, it is intriguing to search for a metric that can capture the notion of versatility. We would like to offer up a candidate and use it to evaluate the versatility of Raw quantitatively. In a manner similar to the computation of SpecRates, *we define the versatility of a machine M as the geometric mean over all applications of the ratio of machine M's speedup for a given application relative to the speedup of the best machine for that application.*[7]

For the application set graphed in Figure 3, Raw's versatility is 0.72, while that of the P3 is 0.14. The P3's relatively poor performance on stream benchmarks hurts its versatility. Although Raw's 0.72 number is relatively good, even our small sample of applications highlights two clear areas which merit additional work in the design of polymorphic processors. One is for embedded bit-level designs, where ASICs perform 2x-3x better than Raw for our small application set. Certainly there are countless other applications for which ASICs outstrip Raw by much higher factors. Perhaps the addition of small amounts of bit-level programmable logic à la PipeRench [10] or Garp [14] can bridge the gap.

Computation with low levels of ILP is another area for further research. We will refer to Figure 4 to discuss this in more detail. The figure plots the speedups *(in cycles)* of Raw and a P3 with respect to execution on a single Raw tile. The applications are listed on the x-axis and sorted roughly in the order of increasing ILP. The figure indicates that Raw is able to convert ILP into performance when ILP exists in reasonable quantities. This indicates the scala-

[7] Since we are taking ratios, the individual machine speedups can be computed relative to any one machine, since the effect of that machine cancels out. Accordingly, the speedups in Figure 3 are expressed relative to the P3 without loss of generality. Further, like SpecRates' rather arbitrary choice of the Sun Ultra5 as a normalizing machine, the notion of versatility can be generalized to future machines by choosing equally arbitrarily the best-in-class machines graphed in Figure 3 as our reference set for all time. Thus, since the best-in-class machines are fixed, the versatilities of future machines can become greater than 1.0.

bility of Raw's scalar operand network. The performance of Raw is lower than the P3 by about 33 percent for applications with low degrees of ILP for several reasons. First, the three leftmost applications in Figure 4 were run on a single tile. We hope to continue tuning our compiler infrastructure and do better on some of these applications. Second, a near-term commercial implementation of a Raw-like processor might likely use a two-way superscalar in place of our single-issue processor which would be able to match the P3 for integer applications with low ILP. (See [31] for details on grain-size tradeoffs in Raw processors).

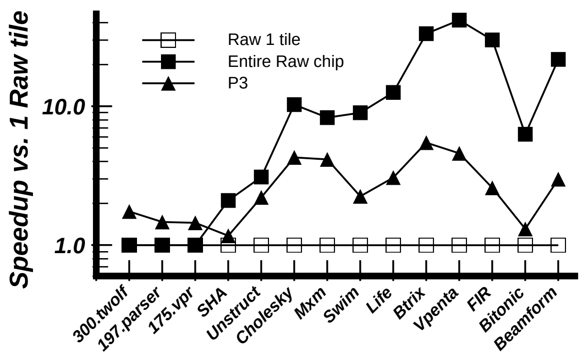

Figure 4: Speedup (in cycles) achieved by Raw and the P3 over executing on a single Raw tile.

6. RELATED WORK

Raw distinguishes itself from others by being a modeless architecture and supporting all forms of parallelism, including ILP, DLP, TLP and streams. Several projects have attempted to exploit specific forms of parallelism. These include systolic (iWarp [12]), vector (VIRAM [21]), stream (Imagine [17]), shared-memory (DASH [26]), and message passing (J machine [35]). These machines, however, were not designed for ILP. In contrast, Raw was designed to exploit ILP effectively in addition to these other forms of parallelism. ILP presents a hard challenge for these machines because it requires that the architecture be able to transport scalar operands between logic units with very low latency, even when there are a large number of highly irregular communication patterns. A recent paper, [45], employs a 5-tuple to characterize the cost of sending operands between function units in a number of architectures. Table 7 lists the components of this 5-tuple in order. Qualitatively, larger 5-tuple values represent proportionally more expensive operand transport costs. The large values in the network 5-tuples for iWarp $< 1, 6, 5, 0, 1 >$, shared memory $< 1, 18, 2, 14, 1 >$, and message passing $< 3, 7, 1, 1, 12 >$, compared to the low numbers in the 5-tuples of machines that can exploit ILP, e.g., superscalar $< 0, 0, 0, 0, 0 >$, Raw $< 0, 1, 1, 1, 0 >$, Grid $< 0, 0, 1/2, 0, 0 >$, and ILDP $< 0, 1, 0, 1, 0 >$ quantitatively demonstrate the difference. The low 5-tuple of Raw's scalar operand network compared to that of iWarp enables Raw to exploit diverse forms of parallelism, and is a direct consequence of the integration of the interconnect into Raw's pipeline and Raw's early pipeline commit point. We will further discuss the comparison with iWarp here, but see [45] for more details on comparing networks for ILP.

Raw supports statically orchestrated communication like iWarp or NuMesh [39]. iWarp and NuMesh support a small number of fixed communication patterns, and can switch between these patterns quickly. However, establishing a new pattern is more expensive. Raw supports statically orchestrated communication by using a programmable switch which issues an instruction each cycle. The instruction specifies the routes through the switch during that cycle. Because the switch program memory in Raw is large, and virtualized through caching, there is no practical architectural limit on the number of simultaneous communication patterns that can be supported in a computation. This virtualization becomes particularly important for supporting ILP, because switch programs become as large or even larger than the compute programs.

Processors like Grid [33] and ILDP [18] are targeted specifically for ILP and propose to use low latency scalar operand networks. Raw shares in their ILP philosophy, and implements a static-transport, point-to-point scalar operand network, while Grid uses a dynamic-transport, point-to-point network, and ILDP uses a broadcast based dynamic-transport network. Both Raw and Grid perform compile time instruction assignment to compute nodes, while ILDP uses dynamic assignment of instruction groups. Raw uses compile-time operand matching, while Grid uses dynamic associative operand matching queues, and ILDP's dynamic scheme uses full-empty bits on distributed register files. Accordingly, using the *AsTrO* categorization (Assignment, Transport, Ordering) from [46], Raw, Grid and ILDP can be classified as SSS, SDD, and DDD architectures respectively, where S stands for static and D for dynamic. Both the Grid and ILDP designs project lower network 5-tuples than Raw, but the final numbers should be forthcoming as their implementations mature. Taken together, Grid, ILDP and Raw represent three distinct points in the scalar operand network design space, ranging from the more compile-time oriented approach as in Raw, to the dynamic approach as in ILDP.

Raw took inspiration from the Multiscalar processor [40], which uses a separate one-dimensional network to forward register values between ALUs. Raw generalizes the basic idea, and supports a two-dimensional programmable mesh network both to forward operands and for other forms of communication.

Both Raw and SmartMemories [28] share the philosophy of an exposed communication architecture, and represent two design points in the space of tiled architectures that can support multiple forms of parallelism. Raw uses homogeneous, programmable static and dynamic mesh networks, while SmartMemories uses programmable static communication within a local collection of nodes, and a dynamic network between these collections of nodes. The node granularities are also different in the two machines. Perhaps the most significant architectural difference, however, is that Raw (like Scale [22]) is *modeless*, while SmartMemories and Grid have modes for different application domains. Another architecture that represents a natural extreme point in modes is Tarantula [9], which implements two distinct types of processing units for ILP and vectors. Raw's research focus is on discovering and implementing a minimal set of primitive mechanisms (e.g., scalar operand network) useful for all forms of parallelism, while the modes approach implements special mechanisms for each form of parallelism. We believe the modeless approach is more area efficient and significantly less complex. Looked at another way, for a given area, machines with modes must demonstrate quantitatively better versatility numbers than modeless machines to justify their increased complexity. Much like for GUIs in the late 70's, we believe the issue of modes versus modeless for versatile processors is likely to be a controversial topic of debate in the forthcoming years.

Finally, like VIRAM and Imagine, Raw supports vector and stream computations, but does so very differently. Both VIRAM and Imagine sport large memories or stream register files on one side of the chip connected via a crossbar interconnect to multiple, deep compute pipelines on the other. The computational model is one that extracts data streams from memory, pipes them through the compute pipelines, and then deposits them back in memory. In contrast, Raw implements many co-located smaller memories and compute elements, interconnected by a mesh network. The Raw computational model is more ASIC-like in that it streams data through the pins and on-chip network to the ALUs, continues through the network to more ALUs, and finally through the network to the pins. Raw's ALUs also can store data temporarily in the local memories if necessary. We believe the lower latencies of the memories in Raw, together with the tight integration of the on-chip network with the compute pipelines, make Raw more suitable for ILP.

7. CONCLUSION

This paper describes the architecture and implementation of the Raw microprocessor. Raw's exposed ISA allows parallel applications to exploit all of the chip resources, including gates, wires and pins. Raw supports ILP by scheduling operands over a scalar operand network that offers very low latency for scalar data transport. Raw's compiler manages the effect of wire delays by orchestrating both scalar and stream data transport. The Raw processor demonstrates that existing architectural abstractions like interrupts, caches, and context-switching can continue to be supported in this environment, even as applications take advantage of the low-latency scalar operand network and the large number of ALUs.

Our results demonstrate that the Raw processor performs at or close to the level of the best specialized machine for each application class. When compared to a Pentium III, Raw displays one to two orders of magnitude more performance for stream applications, while performing within a factor of two for low-ILP applications. It is our hope that the Raw research will provide insight for architects who are looking for ways to build versatile processors that leverage the vast silicon resources while mitigating the considerable wire delays that loom on the horizon.

Acknowledgments We thank our StreamIt collaborators, specifically M. Gordon, J. Lin, and B. Thies for the StreamIt backend and the corresponding section of this paper. We are grateful to our collaborators from ISI East including C. Chen, S. Crago, M. French, L. Wang and J. Suh for developing the Raw motherboard, firmware components, and several applications. T. Konstantakopoulos, L. Jakab, F. Ghodrat, M. Seneski, A. Saraswat, R. Barua, A. Ma, J. Babb, M. Stephenson, S. Larsen, V. Sarkar, and several others too numerous to list also contributed to the success of Raw. The Raw chip was fabricated in cooperation with IBM. Raw is funded by Darpa, NSF, ITRI and the Oxygen Alliance.

REFERENCES

[1] V. Agarwal, et al. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. *2000 ISCA*, pp. 248–259.

[2] E. Anderson, et al. LAPACK: A Portable Linear Algebra Library for High-Performance Computers. *1990 ICS*, pp. 2–11.

[3] M. Annaratone, et al. The Warp Computer: Architecture, Implementation and Performance. *IEEE Transactions on Computers 36*, 12 (December 1987), pp. 1523–1538.

[4] J. Babb, et al. The RAW Benchmark Suite: Computation Structures for General Purpose Computing. *1997 FCCM*, pp. 134–143.

[5] R. Barua, et al. Maps: A Compiler-Managed Memory System for Raw Machines. *1999 ISCA*, pp. 4–15.

[6] M. Bohr. Interconnect Scaling - The Real Limiter to High Performance ULSI. *1995 IEDM*, pp. 241–244.

[7] D. Chinnery, et al. *Closing the Gap Between ASIC & Custom*. Kluwer Academic Publishers, 2002.

[8] K. Diefendorff. Intel Raises the Ante With P858. *Microprocessor Report* (January 1999), pp. 22–25.

[9] R. Espasa, et al. Tarantula: A Vector Extension to the Alpha Architecture. *2002 ISCA*, pp. 281–292.

[10] S. Goldstein, et al. PipeRench: A Coprocessor for Streaming Multimedia Acceleration. *1999 ISCA*, pp. 28–39.

[11] M. I. Gordon, et al. A Stream Compiler for Communication-Exposed Architectures. *2002 ASPLOS*, pp. 291–303.

[12] T. Gross, et al. *iWarp, Anatomy of a Parallel Computing System*. The MIT Press, Cambridge, MA, 1998.

[13] L. Gwennap. Coppermine Outruns Athlon. *Microprocessor Report* (October 1999), p. 1.

[14] J. R. Hauser, et al. Garp: A MIPS Processor with Reconfigurable Coprocessor. *1997 FCCM*, pp. 12–21.

[15] R. Ho, et al. The Future of Wires. *Proceedings of the IEEE 89*, 4 (April 2001), pp. 490–504.

[16] H. Hoffmann, et al. Stream Algorithms and Architecture. Technical Memo MIT-LCS-TM-636, LCS, MIT, 2003.

[17] U. Kapasi, et al. The Imagine Stream Processor. *2002 ICCD*, pp. 282–288.

[18] H.-S. Kim, et al. An ISA and Microarchitecture for Instruction Level Distributed Processing. *2002 ISCA*, pp. 71–81.

[19] J. Kim, et al. Energy Characterization of a Tiled Architecture Processor with On-Chip Networks. *2003 ISLPED*, pp. 424–427.

[20] A. KleinOsowski, et al. MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research. *Computer Architecture Letters 1* (June 2002).

[21] C. Kozyrakis, et al. A New Direction for Computer Architecture Research. *IEEE Computer 30*, 9 (September 1997), pp. 24–32.

[22] R. Krashinsky, et al. The Vector-Thread Architecture. 2004 ISCA.

[23] J. Kubiatowicz. *Integrated Shared-Memory and Message-Passing Communication in the Alewife Multiprocessor*. PhD thesis, MIT, 1998.

[24] W. Lee, et al. Space-Time Scheduling of Instruction-Level Parallelism on a Raw Machine. *1998 ASPLOS*, pp. 46–54.

[25] W. Lee, et al. Convergent Scheduling. *2002 MICRO*, pp. 111–122.

[26] D. Lenoski, et al. The Stanford DASH Multiprocessor. *IEEE Computer 25*, 3 (March 1992), pp. 63–79.

[27] R. Mahnkopf, et al. System on a Chip Technology Platform for .18 micron Digital, Mixed Signal & eDRAM applications. *1999 IEDM*, pp. 849–852.

[28] K. Mai, et al. Smart Memories: A Modular Reconfigurable Architecture. *2000 ISCA*, pp. 161–171.

[29] D. Matzke. Will Physical Scalability Sabotage Performance Gains? *IEEE Computer 30*, 9 (September 1997), pp. 37–39.

[30] J. McCalpin. STREAM: Sustainable Memory Bandwidth in High Perf. Computers. http://www.cs.virginia.edu/stream.

[31] C. A. Moritz, et al. SimpleFit: A Framework for Analyzing Design Tradeoffs in Raw Architectures. *IEEE Transactions on Parallel and Distributed Systems* (July 2001), pp. 730–742.

[32] S. Naffziger, et al. The Implementation of the Next-Generation 64b Itanium Microprocessor. *2002 ISSCC*, pp. 344–345, 472.

[33] R. Nagarajan, et al. A Design Space Evaluation of Grid Processor Architectures. *2001 MICRO*, pp. 40–51.

[34] M. Narayanan, et al. Generating Permutation Instructions from a High-Level Description. TR UCB-CS-03-1287, UC Berkeley, 2003.

[35] M. Noakes, et al. The J-Machine Multicomputer: An Architectural Evaluation. *1993 ISCA*, pp. 224–235.

[36] S. Palacharla. *Complexity-Effective Superscalar Processors*. PhD thesis, University of Wisconsin–Madison, 1998.

[37] N. Rovedo, et al. Introducing IBM's First Copper Wiring Foundry Technology: Design, Development, and Qualification of CMOS 7SF, a .18 micron Dual-Oxide Technology for SRAM, ASICs, and Embedded DRAM. *Q4 2000 IBM MicroNews*, pp. 34–38.

[38] J. Sanchez, et al. Modulo Scheduling for a Fully-Distributed Clustered VLIW Architecture. *2000 MICRO*, pp. 124–133.

[39] D. Shoemaker, et al. NuMesh: An Architecture Optimized for Scheduled Communication. *Journal of Supercomputing 10*, 3 (1996), pp. 285–302.

[40] G. Sohi, et al. Multiscalar Processors. *1995 ISCA*, pp. 414–425.

[41] J. Suh, et al. A Performance Analysis of PIM, Stream Processing, and Tiled Processing on Memory-Intensive Signal Processing Kernels. *2003 ISCA*, pp. 410–419.

[42] M. B. Taylor. Deionizer: A Tool For Capturing And Embedding I/O Calls. Technical Memo, CSAIL/Laboratory for Computer Science, MIT, 2004. http://cag.csail.mit.edu/∼mtaylor/deionizer.html.

[43] M. B. Taylor. The Raw Processor Specification. Technical Memo, CSAIL/Laboratory for Computer Science, MIT, 2004.

[44] M. B. Taylor, et al. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. *IEEE Micro* (Mar 2002), pp. 25–35.

[45] M. B. Taylor, et al. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures. *2003 HPCA*, pp. 341–353.

[46] M. B. Taylor, et al. Scalar Operand Networks: Design, Implementation, and Analysis. Technical Memo, CSAIL/LCS, MIT, 2004.

[47] W. Thies, et al. StreamIt: A Language for Streaming Applications. *2002 Compiler Construction*, pp. 179–196.

[48] E. Waingold, et al. Baring It All to Software: Raw Machines. *IEEE Computer 30*, 9 (September 1997), pp. 86–93.

[49] D. Wentzlaff. Architectural Implications of Bit-level Computation in Communication Applications. Master's thesis, LCS, MIT, 2002.

[50] R. Whaley, et al. Automated Empirical Optimizations of Software and the ATLAS Project. *Parallel Computing 27*, 1–2 (2001), pp. 3–35.

[51] S. Yang, et al. A High Performance 180 nm Generation Logic Technology. *1998 IEDM*, pp. 197–200.