AFRL-VS-PS-
TR-2004-1024

The Raw Fabric: A Technology for Rapid Embedded System Customization

Anant Agarwal, Saman Amarasinghe

Lab. For Computer Science, NE43-624
Massachusetts Institute of Technology
Cambridge, MA 02139

01 June 2004

Final Report

AIR FORCE RESEARCH LABORATORY
Space Vehicles Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776

DTIC COPY

AFRL-VS-PS-TR-2004-1024

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data, does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report has been reviewed by the Public Affairs Office and is releasable to the National Technical Information Services (NTIS). At NTIS, it will be available to the general public, including foreign nationals.

If you change your address, wish to be removed from this mailing list, or your organization no longer employs the addressee, please notify AFRL/VSSE, 3550 Aberdeen Ave SE, Kirtland AFB, NM 87117-5776.

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

This report has been approved for publication.

//signed//

JIM LYKE
Project Manager

//signed//

KIRT S. MOSER, DR-IV
Chief, Spacecraft Technology Division

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-06-2004	Final	01-06-2001 To 31-05-2003

4. TITLE AND SUBTITLE
The Raw Fabric: A Technology for Rapid Embedded System Customization

5a. CONTRACT NUMBER
F29601-01-2-0166

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62712E

6. AUTHOR(S)
Anant Agarwal and Saman Amarasinghe

5d. PROJECT NUMBER
DARP

5e. TASK NUMBER
SC

5f. WORK UNIT NUMBER
AG

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lab. For Computer Science,
NE43-624
Massachusetts Institute of Technology
Cambridge, MA 02139

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/VSSE
3550 Aberdeen Avenue SE
Kirtland AFB, NM 87117-5776

10. SPONSOR/MONITOR'S ACRONYM(S)
DARPA
AFRL

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-VS-PS-TR-2004-1024

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report is the final technical report for the project: The Raw Fabric: A Technology for Rapid Embedded-System Customization. The Raw fabric is a universal computational substrate suitable for signal processing and embedded applications. The key innovation behind Raw fabrics is the ability for software to customize chip-level communication channels in an application-specific manner, thereby enabling the construction of mission-specific embedded systems cost-effectively. Raw fabrics offer the promise of orders of magnitude improvements for embedded applications when compared to microprocessor-based systems. These improvements are in performance, power, and size, and will allow system customization to be measured in hours instead of years. Raw Fabrics comprise single Raw chips with on-chip customizable interconnect, and board-level systems containing many Raw chips. Our project has built a Raw chip prototype and a handheld computer system based on Raw. Our results demonstrate that Raw performs at or close to the level of the best specialized machine for many application classes. When compared to a Pentium-III implemented in the same technology, Raw displays one to two orders of magnitude more performance for stream applications while performing within a factor of two for sequential-desktop applications.

15. SUBJECT TERMS
Raw Fabric, Rapid Embedded System Customization, Microprocessor-Based Systems, Raw Chips

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jim Lyke
Unclassified	Unclassified	Unclassified	Unlimited	30	19b. TELEPHONE NUMBER *(include area code)* 505-846-5812

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

TABLE OF CONTENTS

1. Abstract 1

2. Overview of the Project 1

3. Summary of Accomplishments and Systems Built 3

4. Accomplishments and Progress 4

   4.1 The Raw Chip 4

   4.2 The Raw Handheld Board 5

   4.3 The Multi-Chip Raw Fabric 6

   4.4 The RawCC Compiler and the Stream Compiler 6

   4.5 The Raw OS 8

   4.6 Applications, Experimentation and Elevation 8

   4.7 Support in Standardizing a Morphware Stable Interface 17

5 Conclusions and Recommendations for Future Work 18

6 Key Publications 19

LIST OF FIGURES

1 Photo of the Raw chip 5

2 Photo of the Raw prototype motherboard. The board is 13 inches by 13 inches 5

LIST OF TAB LES

1 Sources of speedup for Raw over P3 (As configured in Table 3). 9

2 Functional unit timings on a single Raw tile and on a P3. Commonly executed instructions appear first. FP operations are single precision.
10

3 Memory system data for Raw tile and P3. 11

4 Raw power consumption at 425 MHz, 25˚ C 12

5 Breakdown of the end-to-end latency (in cycles) for a one-word message on Raw's static network. 12

6 Performance of sequential programs on Raw and on P3. 13

7 Speedup of the ILP benchmarks relative to the single-tile Raw, from two to 16 tiles. 14

8 StreamIt performance results. 16

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

1 Abstract

This report is the final technical report for the project: The Raw Fabric: A Technology for Rapid Embedded-System Customization. The Raw fabric is a universal computational substrate suitable for signal processing and embedded applications. The key innovation behind Raw fabrics is the ability for software to customize chip-level communication channels in an application-specific manner, thereby enabling the construction of mission-specific embedded systems cost-effectively. Raw fabrics offer the promise of orders of magnitude improvements for embedded applications when compared to microprocessor-based systems. These improvements are in performance, power, and size, and will allow system customization to be measured in hours instead of years. Raw Fabrics comprise single Raw chips with on-chip customizable interconnect, and board-level systems containing many Raw chips. Our project has built a Raw chip prototype and a handheld computer system based on Raw. Our results demonstrate that Raw performs at or close to the level of the best specialized machine for many application classes. When compared to a Pentium-III implemented in the same technology, Raw displays one to two orders of magnitude more performance for stream applications, while performing within a factor of two for sequential-desktop applications.

2 Overview of the Project

We begin the report by providing a technical overview of the project.

The Raw fabric [1] is a universal computational substrate that is suitable for signal processing and embedded applications. The key innovation behind raw fabrics is the ability for software to customize chip-level communication channels in an application-specific manner. Raw Fabrics comprise single chip Raw systems with on-chip customizable interconnect, and board-level systems that comprise one to many Raw chips.

Raw Fabrics address a major problem both with extant special purpose hardware systems and general purpose machines. First, modern supercomputers, built from state-of-the-

art COTS microprocessors, have failed to eliminate the need for specialized hardware in the signal processing and embedded domains. Although supercomputing systems have the highest computational power, their inability to cost- effectively utilize this power for solving signal processing problems have led to the proliferation of ASICs, FPGAs, DSPs and full-custom hardware. Second, the need for special purpose hardware is even more acute in the embedded application domain, where efficient utilization of area, weight, and power is paramount. Unfortunately, the enormous cost and lengthy time-to-deployment of special-purpose hardware systems significantly reduces their appeal.

The Raw fabric draws its design motivation from both the strengths and weaknesses of using custom hardware for signal processing and embedded applications. There are three main benefits to developing custom hardware: First, when processing a stream of data, the ability to customize pipeline stages provides an order-of-magnitude performance improvement over using a fixed pipeline when the energy budget is fixed. Second, custom hardware is able to efficiently orchestrate direct data movement between pipeline stages. In contrast, using a fixed memory hierarchy with caches is very inefficient in handling certain access patterns, such as stream data. Third, a custom design can tailor its resources to match both the level and the granularity of the available parallelism in the application. This approach is more efficient than using a processor supporting a fixed amount of parallelism. A fourth, and minor advantage of custom hardware, is the ability to efficiently meet the granularity of data required by the application by customizing the size of registers, data paths, and ALUs.

Despite these advantages, custom hardware has a series of shortcomings. One of the biggest drawbacks in using custom hardware is its inflexibility. The inability to change the applications that run on a given hardware platform dramatically reduces their cost-effectiveness. Although FPGAs were partially successful in addressing this problem, seamless reconfiguration during continuous operation is yet to be achieved. More importantly, the inability of these devices to present an abstraction of unlimited resources renders the task of mapping programs to these devices incredibly difficult. Because ASICs are application-specific and cannot be applied to multiple problems, designing a custom ASIC for an algorithm

can only afford a fraction of the development cost of designing a microprocessor. Therefore, it is not feasible to produce an ASIC with the same clock speed as a microprocessor of the same generation. Furthermore, it becomes prohibitively expensive to design a custom ASIC for each new process generation, while porting an application to a later version of a microprocessor is relatively simple.

Our project consisted of four major components, which together provide a complete polymorphous computing environment. The four components are the Raw Processor, the Raw Fabric, the Raw Compiler, and the Raw Operating System. As described shortly, in each of the components, we resolved many open research issues and technical challenges.

In summary, this project designed and built a flexible and scalable computation fabric that can be morphed into solving many embedded applications in an energy, area, and time-efficient manner. A major component of this research was the Raw processor. The Raw processor is a simple tiled architecture with an innovative communication subsystem and is an ideal building block for larger computation fabrics. As such, the Raw fabric is an early proof-of-concept prototype of the general polymorphous computing architecture concept.

In the project, we investigated many novel micro- architectural features, compiler algorithms, and operating system components. Each of these are central to a successful polymorphous computing environment.

The project also completed the design of a multi-Raw-chip fabric. The project also implemented an optimized compiler and operating system for the Raw environment.

The project developed several prototype Raw systems that are now in use at two DARPA sites including ISI and ATL at Lockheed Martin. Additional boards are in the process of being tested and they will go to several more of our DARPA collaborators.

3 Summary of Accomplishments and Systems Built

The following are the major components of our system that were implemented in our project.

1. Completion of fabrication of a Raw chip with an embedded on-chip customizable fabric
2. Completion of a single-board Raw system (referred to as the "handheld system") for single-chip embedded applications, including
   (a) Completion of the implementation of a handheld communicator system for standalone general purpose computation
   (b) design of an embedded wireless processor
   (c) design of an embedded networking fabric and control plane
   (d) Implemented a 32-node acoustic beamformer. This is being extended to 1K nodes.
3. The design of a multiple-Raw chip fabric system.
4. Design of a PCI interface
5. Optimization of a stream language and compiler
6. A beamformer microphone array design

4 Accomplishments and Progress

This section provides specific details of our accomplishments.

We built a working prototype of a Polymorphous Computing Architecture platform based on the Raw infrastructure. The following sections discuss the components in detail.

4.1 The Raw Chip

The Raw processor was a major system that we built. We implemented the prototype Raw microprocessor in the SA-27E ASIC flow, which uses IBM's CMOS 7SF, a 180nm, 6-layer copper process. We received 120 chips from IBM in October of 2002. We are pleased to report that there were no bugs in first silicon.

Figure 1 shows a micro-photograph of the Raw die. The 16-tile geometry of the chip can be clearly made out.

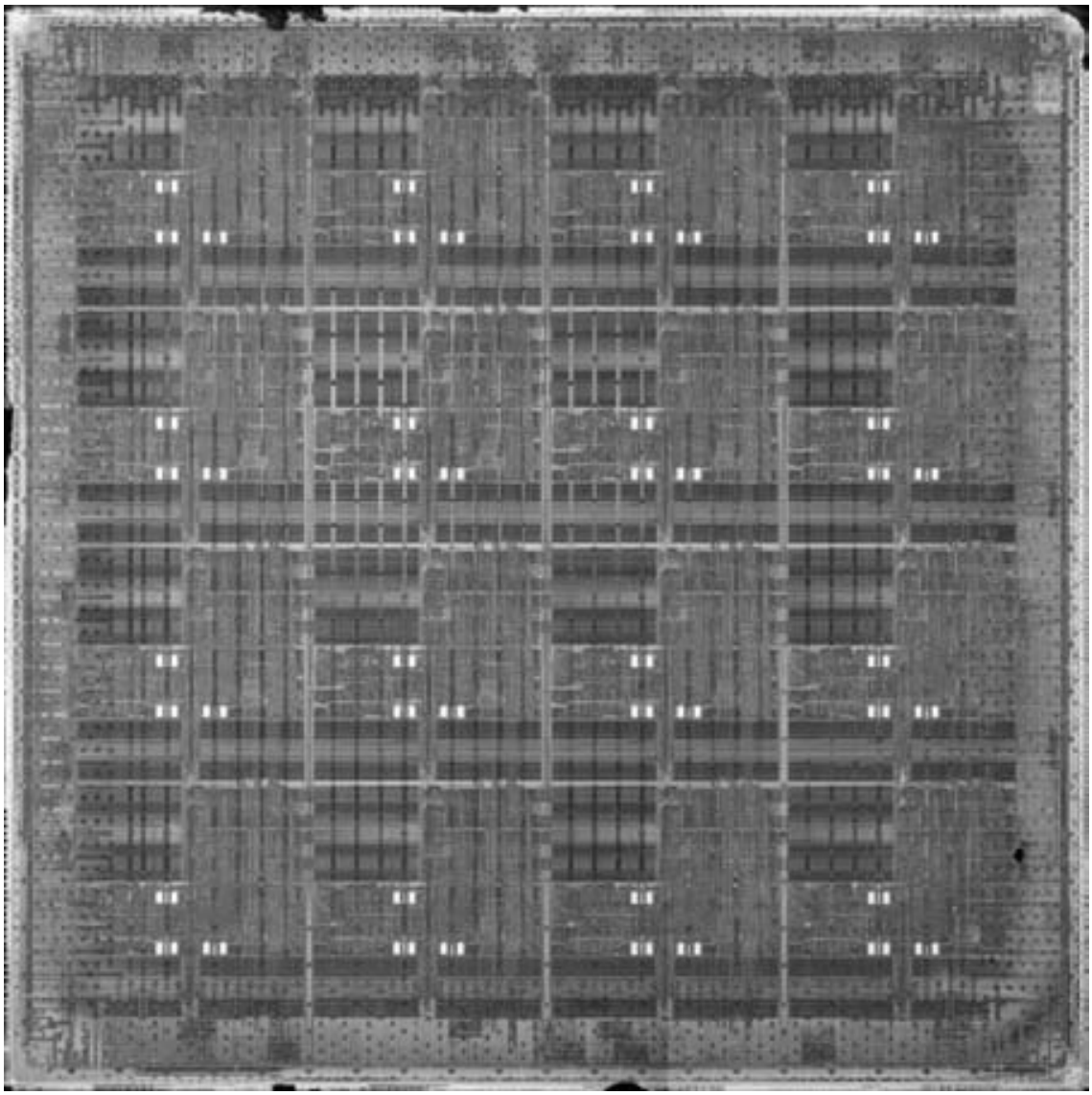

Figure 1: Photo of the Raw chip.

4.2 The Raw Handheld Board

We validated the Raw processor on a prototype mother board called the Raw handheld board. We built several such boards along with our collaborators at ISI. Boards are in use at ISI and

ATL (Lockheed Martin). Each board contains the Raw chip, SDRAM chips, I/O interfaces and interface FPGAs.

Figure 2 shows a photograph of the Raw motherboard.

We have also designed an embedded wireless processor, an embedded networking fabric and control plane, and a 32-node acoustic beamformer (to be extended to 1K nodes in the following year).

We also implemented a high-speed USB interface for the motherboard. This allows the board to be connected to any laptop of PC with a USB2 interface.

We are nearing completion of the PCI interface, which will allow the Raw motherboard to function as a standalone system and provide even higher speed I/O.

4.3 The Multi-Chip Raw Fabric

We designed the Raw fabric board and the Raw fabric I/O board as well (along with our collaborators at ISI). The fabric board contains 4 Raw chips and can be connected in a mesh along with other fabric boards. The I/O board plugs into the periphery of the fabric board mesh and provides I/O, memory and other expansion functions.

In the next phase we will implement and test these boards and build a fabric system containing 64 Raw chips.

4.4 The RawCC Compiler and the Stream Compiler

We built RawCC, which takes sequential C or FORTRAN programs and compiles them on to the Raw fabric. We developed the analysis necessary to extract ILP (instruction-level parallelism) out of sequential programs. Thus, programs written in the SUIF (Stanford University Intermediate Form) supported languages of FORTRAN, C, are able to use our compiler.

We also built a complete steaming compiler and language called Streamit. The language

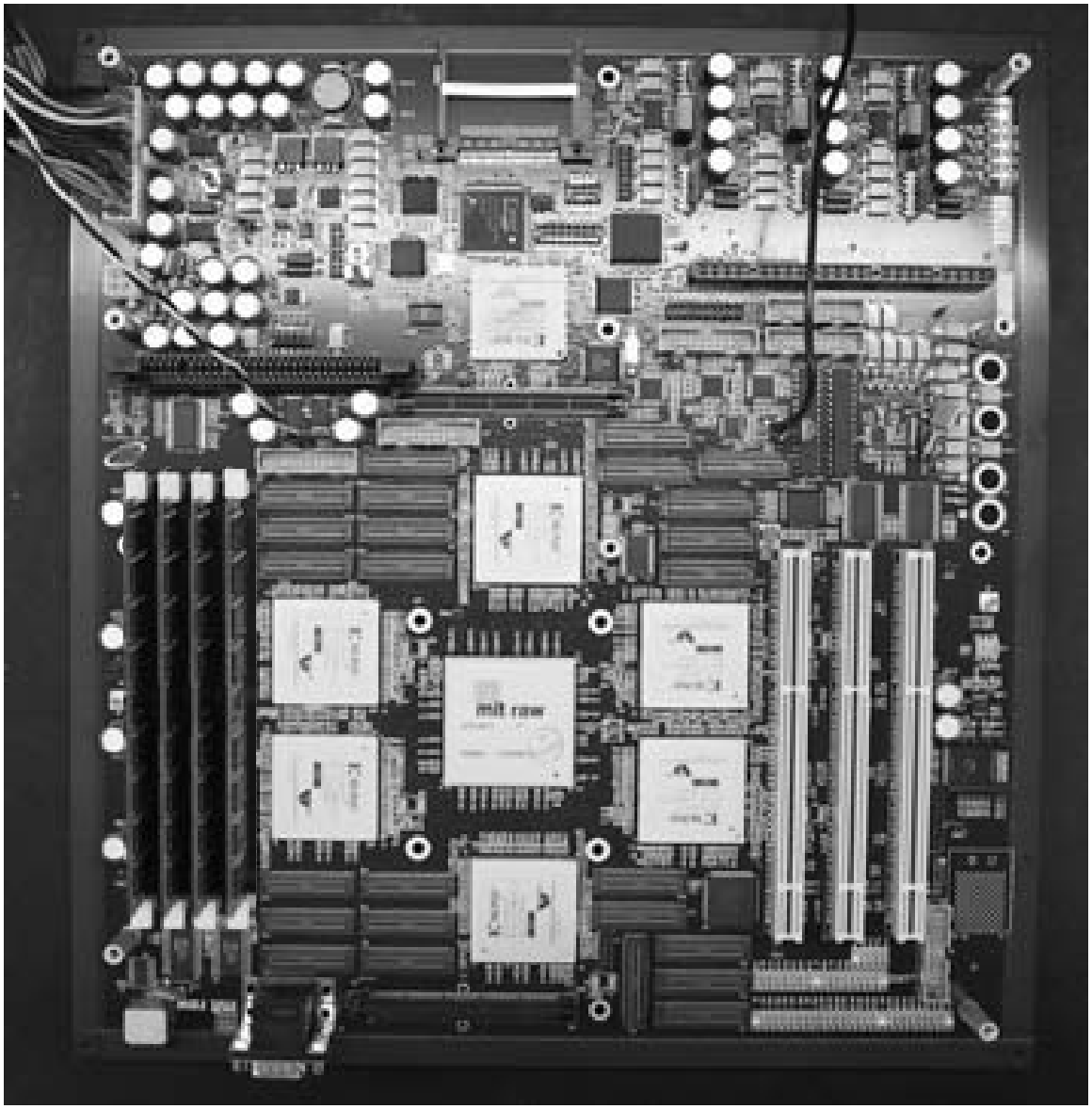

Figure 2: Photo of the Raw prototype motherboard. The board is 13 inches by 13 inches.

syntax is like that of Java and allows the user to express stream programs effectively. Streamit and the compiler have been distributed to the DARPA community.

4.5 The Raw OS

We developed a prototype host-based operating system for the Raw fabric. Our initial fabric operating system includes features needed to support I/O devices and OS services expected by applications. We developed and deployed a nano-kernel for each tile. A set of POSIX commands have been implemented using a few dedicated OS tiles and cooperating nano-kernels. We have distributed this OS to the DARPA community.

We also developed RawGDB a debugger for Raw. RawGDB has also been released to external users.

We also implemented the compiler and runtime system needed for the software supported instruction cache and SUDS [2] (software undo system) systems.

4.6 Applications, Experimentation and Evaluation

We performed a substantial amount of experimentation of applications using the real Raw system. We also validated our simulator against the real hardware and conducted more experiments. We used our working RawCC compiler, the stream compiler and Raw OS for this task. ISI and Lincoln Labs have also developed several PCA applications on Raw.

Specifically, here are some highlights. The domains we examined include ILP computation, and stream and embedded computation. The performance of Raw in these individual areas are presented as comparison to a reference 600 MHz Pentium III, because the Pentium III was manufactured using the same 180 nm technology as Raw.

We note that Raw achieves greater than 16x speedup (versus a single tile) for several applications (listed in 6). Table 1 discusses the various factors that helped Raw.

Tables 2 and 3 show functional unit timings and memory system characteristics for both

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

Table 1: Sources of speedup for Raw over P3 (as configured in Table 3).

systems, respectively. Table 4 shows Raw's measured power consumption. Table 5 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 performance for ILP.

Operation	Latency		Throughput	
	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 2: Functional unit timings on a single Raw tile and on a P3. 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)	425 MHz	100 MHz
DRAM Access Width	8 bytes	8 bytes
L1 D cache size	32K	16K
L1 I cache size	32K	16K
L1 miss latency	54 cycles	7 cycles
L1 fill width	4 bytes	32 bytes
L1 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 3: Memory system data for Raw tile and P3.

	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 4: 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 5: Breakdown of the end-to-end latency (in cycles) for a one-word message on Raw's static network.

Much like a VLIW (very long instruction word) architecture, Raw relies on the compiler to find and exploit ILP. We now examine how well Raw is able to support ILP. For this evaluation, we select a range of benchmarks that encompasses a wide spectrum of program types and degree of ILP. For some of the irregular integer benchmarks that Rawcc is not mature enough to orchestrate, we compile and execute them on one tile to get a conservative worst case bound on their performance on Raw. Table 6 presents the performance of these benchmarks on RawPC and on the P3.

Of the benchmarks in our study, Raw is able to outperform the P3 for all the scientific benchmarks and several irregular applications. Of these, about half have speedups in the 2-3 range, but the other half have more promising speedups in the 4-7 range. At the other end of the spectrum for the integer applications run on a single Raw tile, our sampling of

applications showed that a Raw tile is roughly a factor of 2 slower.

Benchmark	Source	# Raw Tiles	Cycles on Raw	Speedup vs P3 by Cycles	Speedup vs P3 by Time
Dense-Matrix Scientific Applications					
Swim	Spec95	16	58M	4.0	2.8
Tomcatv	Spec92	16	3.2M	1.9	1.4
Btrix	Nasa7:Spec92	16	4.6M	5.5	3.9
Cholesky	Nasa7:Spec92	16	5.5M	2.9	2.5
Mxm	Nasa7:Spec92	16	2.0M	3.5	2.5
Vpenta	Nasa7:Spec92	16	2.5M	10.3	7.3
Jacobi	Raw benchmark suite	16	150K	6.4	4.5
Life	Raw benchmark suite	16	4.0M	7.4	5.2
Sparse-Matrix/Integer Applications					
Fpppp-kernel	Spec92	16	150K	11.2	7.9
SHA	Perl Oasis	16	920K	1.9	1.3
Unstructured	CHAOS	16	15M	1.1	0.75
Adpcm	Mediabench	1	20M	0.85	0.60
GSM	Mediabench	1	310M	0.57	0.40
175.vpr	Spec 2000	1	2.9B	0.71	0.51
300.twolf	Spec 2000	1	2.3B	0.56	0.40

Table 6: Performance of sequential programs on Raw and on a P3.

Table 7 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 1), but several of the dense matrix benchmarks benefit from increased cache capacity with parallel access as well (which explains the super-linear speedups). In addition, Fpppp-kernel benefits from increased register capacity, which leads to fewer spills.

	Number of tiles			
Benchmark	**2**	**4**	**8**	**16**
Dense-Matrix Scientific Applications				
Swim	1.4	2.2	4.3	7.9
Tomcatv	1.6	3.0	4.2	4.8
Btrix	1.6	4.5	10.3	21.8
Cholesky	1.9	3.7	6.2	6.1
Mxm	1.3	3.7	5.7	7.0
Vpenta	1.6	4.9	11.3	22.0
Jacobi	1.7	4.2	8.2	16.5
Life	0.8	2.0	4.9	10.5
Sparse-Matrix/Irregular Applications				
SHA	1.1	1.8	1.9	2.3
Fpppp-kernel	1.4	3.3	6.5	7.4
Unstructured	1.2	2.0	2.1	2.0

Table 7: Speedup of the ILP benchmarks relative to the single-tile Raw, from two to 16 tiles.

Next, 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.

The following results are for programs written in StreamIt, a high level stream language, and automatically compiled to Raw. 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. We have developed a Raw backend for the StreamIt compiler, which includes fully automatic load balancing, graph layout, communication scheduling, and routing.

We evaluate the performance of RawPC on several StreamIt benchmarks, which represent large and pervasive DSP applications. Table 8 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 5 of the 7 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 9 shows detailed speedups relative to StreamIt code running on a 1-tile Raw configuration.

Benchmark	Cycles Per Output on Raw	Speedup vs P3	
		by Cycles	by Time
Beamformer	2675	6.6	4.6
Bitonic Sort	11	5.7	4.0
FFT	22	2.7	1.9
Filterbank	305	9.5	6.7
FIR	59	7.7	5.4
FMRadio	2610	9.6	6.8
Matrix Mult	183	5.4	3.8

Table 8: StreamIt performance results.

Benchmark	StreamIt on P3	StreamIt on n Raw tiles				
		1	2	4	8	16
Beamformer	2.6	1.0	3.7	4.0	8.1	17
Bitonic Sort	1.2	1.0	1.9	3.4	5.3	6.9
FFT	1.0	1.0	1.3	1.7	2.4	2.7
Filterbank	0.72	1.0	1.3	1.3	3.4	6.9
FIR	3.4	1.0	2.3	5.6	12	26
FMRadio	1.2	1.0	1.0	1.1	4.4	12
Matrix Mult	1.1	1.0	2.0	2.9	2.8	5.7

Table 9: 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 9 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 other applications, the scaling is slightly inhibited for small configurations. This is because 1) IMEM constraints prevent an unrolling optimization for small tile sizes (Beamformer, FM, Matrix Mult) and 2) there is some constant overhead that is amortized in larger configurations.

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 9. 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. 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.

4.7 Support in Standardizing a Morphware Stable Interface

Our project has played an active role in the Morphware Forum.

Specifically, we have taken a leadership role in specifying the Streaming Virtual Machine (SVM) which will provide a common interface for high-level compilation tools to target all PCA architectures. In order to achieve high performance for streaming applications, the SVM

embraces a separation of control and data-intensive code, explicit communication via streams, and explicit memory management for streaming data. It also adopts a novel two-stage compilation strategy whereby high-level tools input a description of the target architecture (using the PCA Machine Model) and compile to a version of the SVM that is parameterized for that architecture. Reaching consensus on this interface has involved detailed design discussions with many of the PCA teams, including Stanford, Raytheon, Georgia Tech, USC and the University of Texas. Along with Reservoir Labs, we have coordinated the design process and have produced a stable specification that is serving as a cornerstone of the Morphware toolchain. We have also expressed the Raw architecture in terms of the Machine Model, and are actively engaged with the evaluation of the Reservoir High-Level Compiler that targets the SVM.

5 Conclusions and Recommendations for Future Work

This report described the architecture and implementation of the Raw prototype. Raw's exposed ISA (instruction set architecture) 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.

Our effort has pointed to several future directions that are worth exploring: (1) evaluating the performance for much larger numbers of tiles and a wider set of programs, (2) generalizing on-chip operand networks so that they support other forms of parallelism exploited by microprocessors such as stream parallelism and thread parallelism, (3) complete designs and evaluation of both dynamic and compile-time schemes for operation/operand assignment and scheduling, (4) mechanisms for fast exception handling and context switching, (5) a thorough analysis of the tradeoffs between commit point, exception handling capability, and network latency, (6) low energy tiled processors and scalar operand networks, and (7) tiled architectures with support for bit-level processing.

6 Key Publications

Our major publications are these: [3], [1], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18].

References

[1] Michael Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Jae-Wook Lee, Paul Johnson, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen Matt Frank, Saman Amarasinghe, and Anant Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. *IEEE Micro*, pages 25–35, March/April 2002.

[2] Matthew Frank, C. Andras Moritz, Benjamin Greenwald, Saman Amarasinghe, and Anant Agarwal. SUDS: Primitive Mechanisms for Memory Dependence Speculation. Technical report, M.I.T., January 6 1999.

[3] C. Andras Moritz, Donald Yeung, and Anant Agarwal. SimpleFit: A Framework for Analyzing Design Tradeoffs in Raw Architectures. *IEEE Transactions on Parallel and Distributed Systems*, July 2001.

[4] Volker Strumpen, Henry Hoffmann, and Anant Agarwal. A Stream Algorithm for the SVD. Technical Memo MIT-LCS-TM-641, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, October 2003.

[5] Henry Hoffmann, Volker Strumpen, and Anant Agarwal. Stream Algorithms and Architecture. Technical Memo MIT-LCS-TM-636, Laboratory for Computer Science, Massachusetts Institute of Technology, March 2003.

[6] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fataneh Ghodrat, Benjamin Greenwald, Henry Hoffmann, Paul Johnson, Walter Lee, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Saman Amarasinghe, and Anant Agarwal. A 16-Issue Multiple-Program-Counter Microprocessor with Point-To-Point Scalar Operand Network. In *Proceedings of the IEEE International Solid-State Circuits Conference*, 2003.

[7] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2003.

[8] Jason Kim, Michael Bedford Taylor, Jason Miller, and David Wentzlaff. Energy Characterization of a Tiled Architecture Processor with On-Chip Networks. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2003.

[9] Diego Puppin, Mark Stephenson, Saman Amarasinghe, Una-May O'Reilly, and Martin C. Martin. Adapting convergent scheduling using machine learning. In *Languages and Compilers for Parallel Computing*, College Station, TX, October 2003.

[10] Walter Lee, Diego Puppin, Shane Michael Swenson, and Saman Amarasinghe. Convergent scheduling. In *International Symposium on Microarchitecture*, Istanbul, Turkey, November 2002.

[11] Mark Stephenson, Johnathan Babb, and Saman Amarasinghe. Bitwidth analysis with application to silicon compilation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Vancouver, British Columbia, June 2000.

[12] Rajeev Barua, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Compiler support for scalable and efficient memory systems. *IEEE Transactions on Computers*, 50(11), November 2001.

[13] Gleb A. Chuvpilo, David Wentzlaff, and Saman Amarasinghe. Gigabit ip routing on raw. In *IEEE HPCA Workshop on Network Processors*, pages 2–9, Cambridge, Massachusetts, February 2002.

[14] William Thies, Michal Karczmarek, and Saman Amarasinghe. Streamit: A language for streaming applications. In *International Conference on Compiler Construction*, Grenoble, France, April 2002.

[15] Michael Gordon, William Thies, Michal Karczmarek, Jasper Lin, Ali S. Meli, Chris Leger, Andrew A. Lamb, Jeremy Wong, Henry Hoffman, David Z. Maze, and Saman Amarasinghe. A stream compiler for communication-exposed architectures. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA USA, October 2002.

[16] Michal Karczmarek, William Thies, and Saman Amarasinghe. Phased scheduling of stream programs. In *Languages, Compilers, and Tools for Embedded Systems*, San Diego, CA, June 2003.

[17] Andrew A. Lamb, William Thies, and Saman Amarasinghe. Linear analysis and optimization of stream programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Diego, CA, June 2003.

[18] Gleb A. Chuvpilo and Saman Amarasinghe. High-bandwidth packet switching on the raw general-purpose architecture. In *International Conference on Parallel Processing*, Kaohsiung, Taiwan, Republic of China, October 2003.

DISTRIBUTION LIST

DTIC/OCP
8725 John J. Kingman Rd, Suite 0944
Ft Belvoir, VA 22060-6218 1 cy

AFRL/VSIL
Kirtland AFB, NM 87117-5776 1 cy

AFRL/VSIH
Kirtland AFB, NM 87117-5776 1 cy

Lab. For Computer Science, NE43-624
Massachusetts Institute of Technology
Cambridge, MA 02193 1 cy

Official Record Copy
AFRL/VSSV/Jim Lyke 1 cy