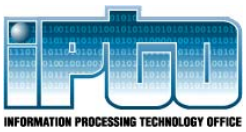
AFRL-IF-WP-TR-2004-1514

TERA-OP RELIABLE INTELLIGENTLY ADAPTIVE PROCESSING SYSTEM (TRIPS)

Stephen W. Keckler, Doug Burger, Michael Dahlin, Lizy John, Calvin Lin, Kathryn McKinley, Tom Keller, and Kevin Nowka

The University of Texas at Austin
Department of Computer Science
1 University Station C0500
Austin, TX 78712-0233



APRIL 2004

Final Report for 19 June 2001 – 31 March 2004

Approved for public release; distribution is unlimited

STINFO FINAL REPORT

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE OH 45433-7334

NOTICE

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 US 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 IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

/s/
AL SCARPELLI
Project Engineer/Team Leader
Embedded Info Sys Engineering Branch
Information Technology Division

/s/
JAMES S. WILLIAMSON, Chief
Embedded Info Sys Engineering Branch
Information Technology Division
Information Directorate

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

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
April 2004	Final	06/19/2001 – 03/31/2004

4. TITLE AND SUBTITLE
TERA-OP RELIABLE INTELLIGENTLY ADAPTIVE PROCESSING SYSTEM (TRIPS)

5a. CONTRACT NUMBER
F33615-01-C-1892

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62712E

6. AUTHOR(S)
Stephen W. Keckler, Doug Burger, Michael Dahlin, Lizy John, Calvin Lin, Kathryn McKinley, Tom Keller, and Kevin Nowka

5d. PROJECT NUMBER
L169

5e. TASK NUMBER
18

5f. WORK UNIT NUMBER
92

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

The University of Texas at Austin
Department of Computer Sciences
1 University Station C0500
Austin, TX 78712-0233

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Information Directorate
Air Force Research Laboratory
Air Force Materiel Command
Wright-Patterson AFB, OH 45433-7334

DARPA/IPTO
3701 Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/IFTA

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-IF-WP-TR-2004-1514

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The TRIPS project proposes and evaluates technology for scalable and adaptive computer systems. The TRIPS processor and on-chip memory architectures are designed to handle both the increasing wire delays and power constraints of near-future integrated circuit fabrication technology. Combined with the new TRIPS compiler, the results of detailed architectural models show that the TRIPS system can achieve performance improvements by up to an order of magnitude over that of conventional architectures (at comparable clock rates) on applications ranging from signal processing to threaded server workloads. TRIPS innovations also include low-power circuits, such as latches and digital phase-locked loops, that will be required for future high-performance polymorphous chips such as TRIPS. Static power analysis tools were also developed to better estimate and balance power consumption in multiple modes of operation. The results of the proof-of-concept phase of TRIPS have shown substantial scientific promise, justifying construction of a prototype in phase 2. The basic parameters of the prototype are outlined in this report as well.

15. SUBJECT TERMS
scalable architectures, polymorphous computing, adaptive hardware and software, low-power circuits

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Al Scarpelli
Unclassified	Unclassified	Unclassified	SAR	34	19b. TELEPHONE NUMBER *(Include Area Code)* (937) 255-6548 x3603

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

Table of Contents

List of Figures

Acknowledgement

The authors gratefully acknowledge the support of DARPA/IPTO and the Air Force Research Laboratory under contract F33615-01-C-1892.

1 Executive Summary

Polymorphous Computing Architectures (PCAs), architectures that can adapt to application and environmental requirements, are an attractive option for deployment in high-performance embedded systems. Current solutions include multiple computer systems, with each one tailored to a particular anticipated task or class of tasks. To date, however, no such polymorphous systems exist. It was the goal of phase 1 of the TRIPS project to develop proof-of-concept hardware and software technology targetted at polymorphous systems. Successful development and evaluation of such technology would justify further investment to fabricate and measure prototype and software systems.

The TRIPS project has developed and evaluated two major architectural advances each sharing two common characteristics: (1) scalable with future deep-submicron fabrication technology, and (2) adaptable to application demands. The Grid Processor architecture is a scalable, tile-oriented processor architecture which accounts for on-chip wire delay and overall power consumption as first-order principles. We have demonstrated through detailed design studies and simulation that this architecture, combined with our proof-of-concept compiler, surpasses the capabilities of conventional microprocessor architectures by factors of up to 14. We have also developed the non-uniform cache architecture (NUCA) which follows similar design principles, and surpasses the performance of conventional on-chip cache architectures by 50%.

Polymorphism in TRIPS is obtained through the grid processor computation substrate that is suitable for multiple granularities of parallelism. In addition, certain components of the TRIPS hardware are configurable. Thus signal processing codes with ample amounts of data-level parallelism and threaded codes with multiple cooperating tasks can each run well on TRIPS. Experiments show that it is competitive with special purpose architectures on such application domains as graphics and signal processing, and is more performance/area efficient than chip-multiprocessors.

The TRIPS project has developed additional technology in support of the design and evaluation of polymorphous systems. The Mambo simulator was augmented with a better user interface and extended to support the development environment for a polymorphous operating system. This tool is actively being used by researchers within the IBM Austin Research Laboratory (ARL) on other DARPA-supported projects. Multiple applications from diverse application classes were ported to the TRIPS architecture for ultimate evaluation on a prototype. To ameliorate future power restrictions on polymorphous chip designs, low power circuits and tools were developed and integrated into IBM's ASIC chip development environment.

Finally, team members at UT-Austin and ARL have defined the implementation path and preliminary design of the TRIPS prototype and system to be constructed in phase 2 of the program. TRIPS technology has already had influence beyond the scope of this project; the influence is expected to increase with the fabrication of the prototype in phase 2.

2 Introduction

The Tera-op Reliable and Intelligently adaptive Processing System (TRIPS) was conceived to solve many of the fundamental problems that are arising as semiconductor processes advance. Left unaddressed, these problems will inhibit the effective use of commercial components in military systems [2, 10]. Mission-critical applications will benefit from the advanced polymorphous capabilities of the TRIPS system, which include numerous malleable hardware components for ultra-flexible application mapping, resource oblivious scheduling for rapid application prototyping and deployment, environmental adaptivity for resilience in hostile environments, and dynamic performance adaptation for changing and unpredictable field constraints. The TRIPS system is intended to provide a single interface to applications that will allow them to run on a wide spectrum of SWEPT (Size, Weight, Energy, Power, and Time) implementations, including lightweight, battery-operated field systems, resilient but powerful base station systems, and ultra-high-performance back-room systems. Most important, TRIPS is intended to be extremely scalable with technology, and addresses many of the long-term problems that will emerge as devices scale down to 50 nanometers and below.

Phase 1 of the TRIPS project was intended to develop proof-of-concept technologies that meet the scalability and adaptivity goals described above. To that end, the TRIPS team has developed novel processor microarchitectures and on-chip memory system architectures that are both technology scalable and amenable to different types of applications. The processor core employs an explicit data graph execution (EDGE) instruction set that enables different types of programs to be efficiently mapped to a spatial computation substrate. We have constructed a new compiler for the architecture based on the University of Massachusetts Scale compiler that employs new hyperblock generation heuristics and novel instruction placement algorithms. We have built multiple simulators to help measure the performance and drive the final design of the architecture.

We have also built a higher-level modeling tool, the full-system simulator Mambo, which serves as a platform for investigating morphing and adaptivity in modern applications. We have examined a large range of applications including server, signal processing, graphics, desktop, and network processing workloads to determine their varying characteristics. We have used many of these same benchmarks to evaluate the adaptivity of the TRIPS architecture.

In support of infrastructure for future morphable systems, the TRIPS team has developed a range of strategies and tools that will reduce power consumption. We have designed new low-power latches and dynamic voltage and frequency islands, and have examined the feasibility of low power digital phase-locked loops (PLLs) to replace the more power hungry analog versions. We have also developed tools to examine static power dissipation to identify and mitigate high-leakage circuits and regions. While these circuits will not be used for initial TRIPS prototypes, they will be required for scalable and adaptive systems in the next generation.

Finally, we have been preparing for phase 2 of the TRIPS project, in which we will construct and measure a TRIPS hardware and software prototype. The UT and IBM members of the TRIPS team have spent considerable time examining tools, methodology, and ASIC technologies to determine what the TRIPS prototype system should look like, and how it should be built. Section 7 of this document describes the expected parameters of the TRIPS prototype chip and board.

3 Scalable Architectures

The first goal of the TRIPS project was to develop several new technologies for scalable processor and memory system architectures. We have invented innovative architectures that address scalability in the processor core, level-2 cache, and memory disambiguation hardware. In addition to the hardware innovations, we also have developed a compilation strategy for the novel processor architecture and have made substantial progress in the implementation of the compiler. Compiler implementation will be completed in phase 2 of the project.

3.1 Explicit Data Graph Execution (EDGE) Architectures

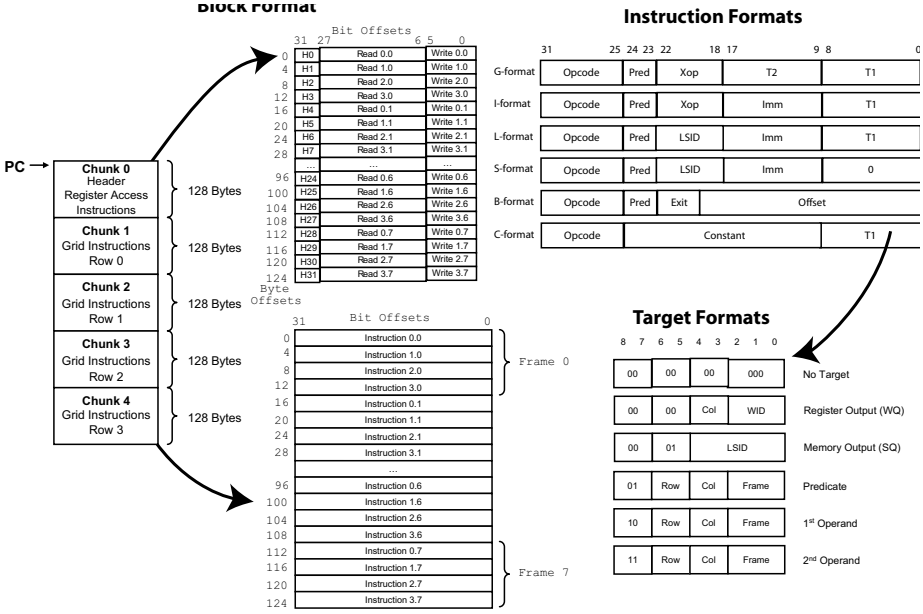
We developed the TRIPS architecture to address the wire-delay and power scaling challenges of conventional microprocessors. The architecture employs an Explicit Data Graph Execution (EDGE) instruction set which has two main characteristics: (1) a program is partitioned into atomically executed blocks of instructions which have a single entry-point and no internal branching, and (2) instructions within a block execute in dataflow order according to their true data dependences. Logically, the TRIPS chip fetches and executes individual blocks sequentially, but as an optimization the chip speculatively fetches and executes multiple subsequent blocks concurrently with the non-speculative block. In the TRIPS chip prototype, this method opens up an instruction window of up to 1024 instructions, providing the opportunity to exploit concurrency from different regions of the executing program.

3.1.1 TRIPS Instruction Set Architecture

We have defined not just the class of EDGE architectures, but we have also designed an instruction set for the prototype to be implemented in phase 2. Figure 1 shows the current definition of the TRIPS prototype's instruction set architecture (ISA). Each block contains a 128-byte block header, which specifies the number of block outputs, as well as the registers that are read from and written to by the block. The rest of the block consists of four 128-byte instruction "chunks," each of which services 32 32-bit instructions to a row. Each row contains 4 nodes with 8 instructions per node.

The ISA provides the six instruction formats shown in Figure 1, which differ based on instruction function and number of distinct instruction targets. The major difference between the TRIPS EDGE instructions and conventional RISC instructions is that each instruction contains no source operand specifiers, but instead contains the name(s) of its consumer instructions. These consumer names are called *targets*; one or two per value-producing instruction are provided. If an instruction has more targets in a block than can be specified in the instruction format, extra instructions must be inserted to forward the value to the additional targets, resulting in a software fan-out tree. This "push" model permits the compiler to schedule dataflow-like execution within a block on the underlying substrate. Communication among different blocks must still occur through registers. As shown in Figure 1, targets in the TRIPS ISA consist of nine bits. Seven of the bits specify the target instruction number, which corresponds either to a physical reservation station in the execution array, or as a block output to one of the 128 architectural registers. Physical reservation stations are identified using row, column, and frame coordinates. The frame represents a specific buffer location at the ALU indicated by $< row, column >$. A 4×4 TRIPS processor with 128 reservation stations (one reservation station for each instruction in the block) requires 8 frames. The remaining two bits specify whether each target is an architectural register or is in the block, and if it is in the block, whether it is the left, right, or predicate operand of the consuming instruction.

In addition to the standard immediate and extended opcode fields, there are several other non-standard fields in the instruction set. Each instruction contains a two-bit predicate field, which specifies whether the instruction should execute as soon as its operands arrive, or whether it should wait for a predicate operand to arrive before firing. In the latter case, the instruction fires only if the arriving predicate matches its predicate type (true or false). Having binary predicate values reduces the number of overhead instructions needed to invert predicates. Finally, memory and branch instructions have



Instructions

Load and Store Instructions		
LB	Load Byte	L
LH	Load Halfword	L
LW	Load Word	L
LD	Load Doubleword	L
SB	Store Byte	S
SH	Store Halfword	S
SW	Store Word	S
SD	Store Doubleword	S
Integer Arithmetic Instructions		
ADD	Add	G
ADDI	Add Immediate	I
SUB	Subtract	G
SUBI	Subtract Immediate	I
MUL	Multiply	G
MULI	Multiply Immediate	I
DIVS	Divide Signed	G
DIVSI	Divide Signed Immediate	I
DIVU	Divide Unsigned	G
DIVUI	Divide Unsigned Immediate	I
Integer Logical Instructions		
AND	Bitwise AND	G
ANDI	Bitwise AND Immediate	I
OR	Bitwise OR	G
ORI	Bitwise OR Immediate	I
XOR	Bitwise XOR	G
XORI	Bitwise XOR Immediate	I

Integer Shift Instructions		
SLL	Shift Left Logical	G
SLLI	Shift Left Logical Immediate	I
SRL	Shift Right Logical	G
SRLI	Shift Right Logical Immediate	I
SRA	Shift Right Arithmetic	G
SRAI	Shift Right Arithmetic Immediate	I
Integer Extend Instructions		
EXTSB	Extend Signed Byte	G
EXTSH	Extend Signed Halfword	G
EXTSW	Extend Signed Word	G
EXTUB	Extend Unsigned Byte	G
EXTUH	Extend Unsigned Halfword	G
EXTUW	Extend Unsigned Word	G
Integer Test Instructions		
TEQ	Test EQ	G
TEQI	Test EQ Immediate	I
TLT	Test LT	G
TLTI	Test LT Immediate	I
TLE	Test LE	G
TLEI	Test LE Immediate	I
TLTU	Test LT Unsigned	G
TLTUI	Test LT Unsigned Immediate	I
TLEU	Test LE Unsigned	G
TLEUI	Test LE Unsigned Immediate	I

Floating-Point Arithmetic Instructions		
FADD	FP Add	G
FSUB	FP Substract	G
FMUL	FP Multiply	G
FDIV	FP Divide	G
Floating-Point Test Instructions		
FEQ	FP Test EQ	G
FLT	FP Test LT	G
FLE	FP Test LE	G
Floating-Point Conversion Instructions		
FITOD	Convert Integer to Double FP	G
FDTOI	Convert Double FP to Integer	G
FSTOD	Convert Single FP to Double FP	G
FDTOS	Convert Double FP to Single FP	G
Control Flow Instructions		
BR	Branch	B
BRO	Branch with Offset	B
CALL	Call	B
CALLO	Call with Offset	B
RET	Return	B
SCALL	System Call	B
Miscellaneous Instructions		
NULL	Nullify Output	G
MOV	Move	G
MOVI	Move Immediate	I
GENS	Generate Signed Constant	C
GENU	Generate Unsigned Constant	C
APP	Append Constant	C
NOP	No Operations	C

Figure 1: The TRIPS Instruction Set Architecture

special tags; loads and stores have tags (LSIDs) to specify their program order to the memory system, and branches contain tags to assist the block-level branch predictor.

3.1.2 TRIPS Microarchitecture

Figure 2a shows a diagram of an example TRIPS chip architecture. The diagram shows four polymorphous, 16-wide out-of-order issue cores, 80 64KB memory tiles connected by a routed network, and a set of distributed memory controllers with channels to external memory. A scaled down prototype chip will be built in phase 2 using a 130nm process and is targeted for completion in 2005. More details on the prototype chip are found in Section 7 of this report. Due to the partitioned structures and short point-to-point wiring connections, the architecture is scalable to larger dimensions and higher clock rates than will be achieved in the first prototype.

Figure 2b shows an expanded view of a TRIPS core and the primary memory system. The TRIPS core is an example of the Grid Processor family of designs [19], which are typically composed of an array of homogeneous execution nodes, each containing an integer ALU, a floating point unit, a set of reservation stations, and router connections at the input and output. Each reservation station has storage for an instruction, two source operands, and a predicate field. When a reservation station contains a valid instruction and a pair of valid operands, the node can select the instruction for execution. After execution, the node can forward the result to any of the operand slots in local or remote reservation stations within the ALU array. The nodes are directly connected to their nearest neighbors, but the routing network can deliver results to any node in the array. Figure 2c shows an expanded view of the ALU with the local reservation stations (frames). Note that an instruction block only uses a subset of the frames, while subsequent groups of frames can be filled with speculative blocks by the hardware speculation engine.

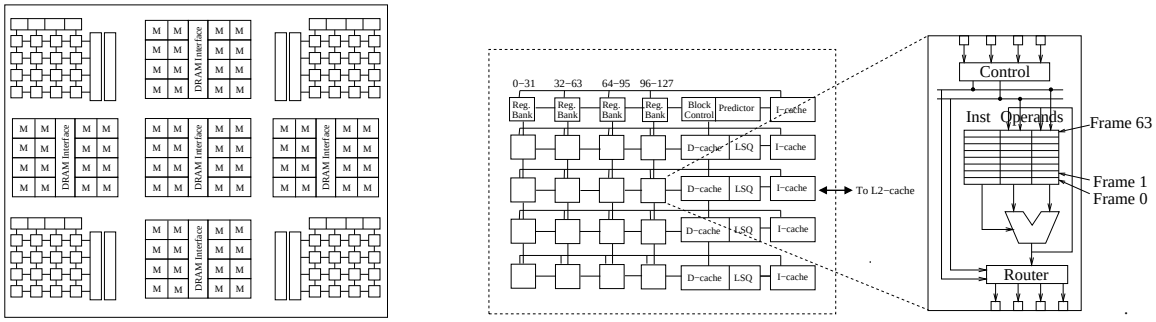
The banked instruction cache on the right side of Figure 2b couples one bank per row, with an additional instruction cache bank to hold block headers, which include the encoded instructions that move values from registers for injection into the ALU array. The banked register file above the ALU array holds a portion of the architectural state. Also to the right of the execution nodes are a set of banked, cache-line interleaved level-1 data caches, which can be accessed by any ALU through the local ALU routing network. In the upper right-hand corner of the processor is the block control logic tile that is responsible for sequencing block execution and selecting the next block. The control tile also contains the L1 instruction cache tags. The back side of the L1 caches are connected to secondary memory tiles through the two-dimensional, switched on-chip interconnection network (OCN). The OCN provides a robust and scalable connection to a large number of memory tiles, using less wiring than conventional, dedicated point-to-point channels among these components.

3.1.3 Performance Estimates

To evaluate the TRIPS architecture, we developed a high-level configurable simulator which allows modulation of many architectural parameters. For our early experiments, we adapted the Trimaran compiler infrastructure [11] to generate TRIPS code. Our performance results indicate that the 16-ALU TRIPS architecture can achieve instructions-per-clock (IPC) of 1-12 on the SPEC serial codes used to evaluate modern workstations. These results exceed those capable by conventional processors by factors of 1.4 to 8. Full results of this work can be found in [26].

3.2 Compiling for EDGE Architectures

During phase 1, we initiated a compiler development effort for the TRIPS architecture. We initially retargeted the Scale compiler from the University of Massachusetts [3] and have designed TRIPS specifics mappings and optimization. To date, we have added loop-unrolling heuristics in the front end, and retargeted the back end to the TRIPS EDGE ISA. We developed novel scheduling algorithms to map instruction blocks to the ALU array, to minimize communication distances,



(a) TRIPS Chip (b) TRIPS Core (c) Execution Node

Figure 2: TRIPS Architecture Overview

yet provide adequate load balancing. The scheduler is a separate tool that operates on the TRIPS Intermediate Language (TIL) produced by the Scale compiler. We also adapted the gnu binary utilities to implement an assembler and linker. We also added error checking into the assembler to ensure that a block is not too large, and does not read or write too many registers or memory locations.

While this implementation is ongoing and will be completed in phase 2, we have obtained results that demonstrate the capabilities of our scheduling algorithms. Early scheduling of instructions on the critical path through a block, and recomputing the critical path on the fly during scheduling, provides a 19% improvement over a baseline scheduling algorithm. Accounting for load balancing of instructions across the grid provides an additional 12% improvement in performance. Finally, taking into account communication latencies between not just ALUs but also ALUs, register files, and memory banks, produces an additional 9%, for a grand total of 45%. This is within 26% of an optimistic upper bound on performance.

3.3 Technology Scalable Non-Uniform Caching (NUCA)

In our examination of technology scaling in the memory system, we determined that the design of large caches in future technologies must be substantially different than those of today. In future technologies, large on-chip caches with a single, discrete hit latency will be undesirable, due to increasing global wire delays across the chip [2, 18]. Data residing in the part of a large cache close to the processor could be accessed much faster than data that reside physically farther from the processor. For example, the closest bank in a 16-megabyte, on-chip L2 cache built in a 50nm process technology could be accessed in 4 cycles, while an access to the farthest bank might take 47 cycles. The bulk of the access time will involve routing to and from the banks, not the bank accesses themselves.

3.3.1 Cache Architectures

Figure 3 shows the types of organizations that we explored, listing the number of banks and the average access times, assuming 16MB caches modeled with a 50nm technology. The numbers superimposed on the cache banks show the latency in cycles of a single contentionless request, derived from a modified version of the Cacti [30] cache modeling tool. The average loaded access times shown below are derived from performance simulations that use the unloaded latency as the access time but which include port and channel contention.

We call a traditional cache a Uniform Cache Architecture (`UCA`), shown in Figure 3a. Figure 3b shows a traditional multi-level cache (L2 and L3), called `ML-UCA`. Figure 3c shows an aggressively banked cache, which supports non-uniform

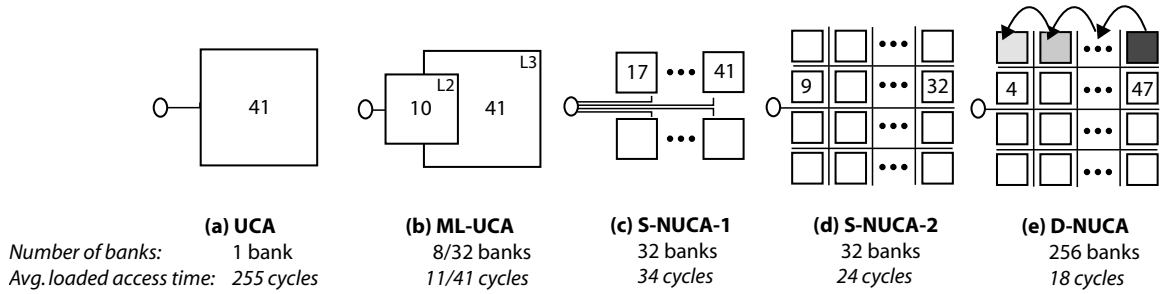


Figure 3: Level-2 Cache Architectures

access to the different banks without the inclusion overhead of `ML-UCA`. The mapping of data into banks is predetermined, based on the block index, and thus can reside in only one bank of the cache. Each bank uses a private, two-way, pipelined transmission channel to service requests. We call this statically mapped, non-uniform cache `S-NUCA-1`.

When the delay to route a signal across a cache is significant, increasing the number of banks can improve performance. A large bank can be subdivided into smaller banks, some of which will be closer to the cache controller, and hence faster than those farther from the cache controller. The original, larger bank was necessarily accessed at the speed of the farthest, and hence slowest, sub-bank. Increasing the number of banks, however, can increase wire and decoder area overhead. Private per-bank channels, used in `S-NUCA-1`, heavily restrict the number of banks that can be implemented, since the per-bank channel wires adds significant area overhead to the cache if the number of banks is large. To circumvent that limitation, we propose a static NUCA design that uses a two-dimensional switched network instead of private per-bank channels, permitting a larger number of smaller, faster banks. This organization, called `S-NUCA-2`, is shown in Figure 3d.

Even with an aggressive multi-banked design, performance may still be improved by exploiting the fact that accessing closer banks is faster than accessing farther banks. By permitting data to be mapped to many banks within the cache, and to migrate among them, a cache can be automatically managed in such a way that most requests are serviced by the fastest banks. Using the switched network, data can be gradually promoted to faster banks as they are frequently used. This promotion is enabled by spreading sets across multiple banks, where each bank forms one way of a set. Thus, cache lines in closer ways can be accessed faster than lines in farther ways. We call this dynamic non-uniform scheme `D-NUCA`, which is shown in Figure 3e.

3.3.2 Results

Our results show that, for multi-megabyte level-two caches, an adaptive, dynamic NUCA design achieves 1.5 times the IPC of a Uniform Cache Architecture of any size, outperforms the best static NUCA scheme by 11%, outperforms the best three-level hierarchy–while using less silicon area–by 13%, and comes within 13% of an ideal, minimal hit latency solution.

3.4 Scalable Memory Disambiguation Technology

A third area of modern processor architectures that is facing scalability challenges is hardware memory disambiguation, typically implemented using a load/store queue (LSQ) [29]. As processors increase their issue rates, more loads and stores must be in flight, which in turn requires larger load/store queues that must support more simultaneous accesses. Current designs are not scalable. We have proposed and evaluated a scheme for lightweight approximate hashing in hardware with structures called Bloom filters that provide several opportunities for improvements to LSQ designs.

We examined two types of filtering schemes using Bloom filters: *search filtering*, which uses hashing to reduce both the number of lookups to the LSQ and the number of entries that must be searched, and *state filtering*, in which the number of entries kept in the LSQs is reduced by coupling address predictors and Bloom filters, permitting smaller queues. We evaluated these techniques for LSQs indexed by both instruction age and the instruction's effective address, and for both centralized and physically partitioned LSQs. We showed that search filtering avoids up to 98% of the associative LSQ searches, providing significant power savings and keeping LSQ searches to under one high-frequency clock cycle. We also showed that with state filtering, the load queue can be eliminated altogether with only minor reductions in performance for small instruction window machines. These results show tremendous promise in enabling scaling of memory disambiguation hardware.

4 System-level Polymorphism

The second goal of the TRIPS project was to develop hardware and software mechanisms that allow the system run different types of applications well at different points in time. We have developed hardware mechanisms that allow the baseline TRIPS architecture to be configured to exploit thread-level and data-level parallelism, in addition to instruction-level parallelism, as described in Section 3. We have also developed cooperative hardware/software techniques that provide a method of adaptively setting the cache prefetch parameters to balance the demands of memory latency and memory bandwidth. Finally, we have designed a set of morphware interfaces for the threaded virtual machine (TVM) being used in the PCA Morphware Forum.

4.1 Adaptivity of the TRIPS Architecture

We defined three major morphs, each of which corresponds to a class of applications. The D-morph is suited to instruction-level parallelism, the S-morph is tailored to streaming or data-level parallelism, and the T-morph is designed to exploit thread-level parallelism on a single processor core.

4.1.1 Configurable Resources

To support these major morphs, we designed a set of polymorphous hardware resources which can be configured differently for different types of applications. These resources include the following:

- Reservation stations: As shown in Figure 2c, each execution node contains a set of reservation stations. These reservation stations are managed differently by each mode to support efficient execution of alternate forms of parallelism. The D-morph maps one non-speculative instruction block of instructions into a subset of the reservation stations, and maps multiple speculative blocks in the remaining ones. The T-morph partitions the reservations stations into up to 4 groups, with each group dedicated to a thread. Each thread may have one non-speculative block and multiple speculative blocks mapped into the reservation stations. The S-morph maps larger blocks of instructions into larger partitions of reservation stations, and supports multiple non-speculative blocks by exploiting hardware loop unrolling.
- Register file banks: Although the programming model of each execution mode sees essentially the same number of architecturally visible registers, the hardware substrate provides many more registers. The extra copies can be used in different ways, such as for speculation or multithreading, depending on the mode of operation. In the D-morph, these additional registers hold speculative state for the speculative instruction blocks executing concurrently with the non-speculative instruction block. In the T-morph, some of these registers hold non-speculative state for the multiple simultaneously executing threads.
- Block sequencing controls: The block sequencing controls determine when a block has completed execution, when a block should be deallocated from the frame space, and which block should be loaded next into the free frame space. To implement different modes of operation, a range of policies can govern these actions. The deallocation logic may be configured to allow a block to execute more than once, as is useful in streaming applications in which the same inner loop is applied to multiple data elements. The next block selector can be configured to limit the speculation, or to prioritize between multiple concurrently executing threads, which is useful for multithreaded parallel programs.
- Memory tiles: The TRIPS Memory tiles (Figure 2a) can be configured to behave as NUCA style L2 cache banks [14], scratchpad memory, or synchronization buffers for producer/consumer communication. The D-morph and T-morph configure all of the secondary memory tiles as a NUCA cache. The S-morph configures the memory tiles closest

to the processor as scratchpad storage and uses higher-bandwidth channels to stream the data into the processor, producing the effect of a stream register file.

4.1.2 Results

We evaluated these configurable mechanisms using the TRIPS simulator described above. We used different types of applications to examine each morph. The D-morph evaluation used serial SPEC2000 applications. The T-morph workloads were composed of multiple SPEC applications running simultaneously. The S-morph applications consisted of signal processing kernels. Our results show that all three modes achieve the goal of high performance on their respective application domain. The D-morph sustains 1–12 IPC (average of 4.4) on serial codes. The T-morph achieves aggregate IPC on highly serial programs of 4.7, 7.4, and 9.8 for two, four, and eight threads, respectively. The S-morph executes as many as 12 arithmetic instructions per clock (AIPC) on a 16-ALU core, and an average of 23 on an 8x8 core. Note that AIPC differs from IPC in that AIPC does not include the non-arithmetic memory and control instructions. This is the first such performance analysis of a processor architecture composed of coarse-grained configurable resources.

4.2 Cooperative Prefetching

Despite large caches, main-memory access latencies still cause significant performance losses in many applications. Numerous hardware and software prefetching schemes have been proposed to tolerate these latencies. Software prefetching typically provides better prefetch accuracy than hardware, but is limited by prefetch instruction overheads and the compiler's limited ability to schedule prefetches sufficiently far in advance to cover level-two cache miss latencies. Hardware prefetching can be effective at hiding these large latencies, but generates many useless prefetches and consumes considerable memory bandwidth. We proposed and evaluated a cooperative hardware-software prefetching scheme called Guided Region Prefetching (GRP), which uses compiler-generated hints encoded in load instructions to regulate an aggressive hardware prefetching engine [33]. We compare GRP against a sophisticated pure hardware stride prefetcher and a scheduled region prefetching (SRP) engine. SRP and GRP show the best performance, with respective 22% and 21% gains over no prefetching, but SRP incurs 180% extra memory traffic—nearly tripling bandwidth requirements. GRP achieves performance close to SRP, but with a mere eighth of the extra prefetching traffic, a 23% increase over no prefetching. The GRP hardware-software collaboration thus combines the accuracy of compiler-based program analysis with the performance potential of aggressive hardware prefetching, bringing the performance gap versus a perfect L2 cache under 20%.

4.3 Morphware Interfaces and Infrastructure

As a part of our work in collaboration with the PCA Morphware Forum [4], we have designed a user-level virtual machine (UVM) interface to polymorphous architectures that enable resource allocation, arbitration, and management in a multitasking system [7]. This interface differs slightly from the threaded virtual machine hardware abstraction layer (TVM-HAL) interface developed by our colleagues at Stanford [8] in that it does not assume complete access to all machine resources. As a result, it provides interfaces that allow schedulers to react when resources are taken away, and it provides interfaces for mapping memory when other processes may also be doing the same thing. Of course, if a process is lucky enough to run in an environment where it is the only process on the machine, that is fine too. Thus, applications written to this interface should be portable to both "thin" run-time systems that dedicate an entire machine to one process and "thicker" operating systems that multiplex a machine across processes as well as "thin kernel/fat library" systems that provide minimal multiplexing mechanisms while leaving policy decisions to application libraries.

Note that although this UVM interface will typically be built over the TVM-HAL interface, this UVM interface is still quite low level. Generally, applications programmers will not access these functions directly. Instead, these functions will

be accessed by libraries that provide higher-level functionality. For example, a user level scheduler library might run above the low-level scheduler activations interfaces included in the UVM. Thus, for portability, we anticipate UVM run-time systems and libraries being built on the TVM-HAL and exporting the interfaces defined here. Then, user level libraries are built on these UVM interfaces, and user level programs run over these library interfaces. We documented our interface design in a Morphware Forum report [7] and anticipate building a prototype of the interface during phase 2 of the TRIPS project.

5 Application Development and Adaptive Software

5.1 Processor Simulation and Commercial Workloads

IBM provided a commercial workload and PowerPC simulator to UT for the purpose of investigating the role of morphing in commercial workloads. The commercial benchmark was ported to the simulator and several additions were made to the simulator to accommodate the benchmark. In addition, the K42 operating system was adapted for the simulator to investigate the role of morphing in operating systems.

5.1.1 Mambo Simulator

Mambo, an IBM tool for simulating PowerPC-based systems, was used by UT for the purpose of analyzing commercial workloads and evaluating and studying the K42 operating system. Instrumented versions of Linux and K42 run on the simulator and provide performance information for workloads running on the simulator. IBM prepared distributions of Mambo and K42 for UT at regular intervals. They were installed at the UT site and continuously updated and supported.

Simulating a substantial commercial benchmark can cause the simulator to simulate many billions of instructions. This results in very long simulation times. To make the simulator more useful, a checkpointing feature was added so that simulations could be stopped, saved to disk, and restarted later. In addition, a new regression testing infrastructure was added to Mambo so that tests written in an arbitrary or ad hoc manner could be run against the simulator as it was developed. This new regression testing was necessary to validate the checkpointing feature, which did not fit into the regression model previously used.

5.1.2 Port of TPC-W to Mambo

An industry standard commercial workload for web transactions—TPC-W, the transactional web electronic commerce benchmark from the transaction processing council (TPC)—was ported to the Mambo simulator to enable the analysis of morphing for commercial workloads. In order to support the efficient execution of this workload on Mambo, the disk and network simulation was modified to have extremely fast functional models by using the disk and network of the host system. For example, when the simulated system accessed a file in the simulated OS, the simulator would supply the file by reading it from the disk on the host system. In addition, Mambo was modified to provide efficient execution of the idle loop of the operating system. This allows the simulator to "fast-forward" between periods of activity in the simulated operating system. For example, it provided a 10x speedup in the boot time of K42 run on Mambo, mainly due to avoiding simulation of the idle loop when the OS is waiting for startup files to be loaded from a network or disk.

Modifications were made to the Linux kernel running on Mambo so that computing resources could be accounted for on a process by process basis. This contributes to our understanding of complicated workloads such as TPC-W by measuring its individual component processes. The Linux OS was instrumented to tell Mambo which process is currently running. The "emitter" functions in Mambo that take simulation traces were modified to collect simulation events on a per-process basis. Previously, a statistic such as the number of cache misses or number of instructions executed was only known for the total workload. It is now possible for these same statistics to be known for each of the processes (web server, application scripts, and database) that comprise the TPC-W benchmark.

5.2 Application Studies

We have performed two types of workload studies. The first is a characterization of signal processing and server workloads to determine the processor and memory requirements. In addition, we have evaluated the PCA ground moving target indicator (GMTI) application from Lincoln Laboratories on the TRIPS architecture.

5.2.1 IRT

We fully characterized the behavior of the 3-D radar processing kernels delivered to the PCA community by Lincoln Laboratories as a prelude to the Integrated Radar Tracker (IRT) application [15]. We analyzed this PCA radar-processing benchmark algorithmically in order to derive formulas for computational complexity based on the data set parameters. We also simulated the components of the application on three data set sizes, using a validated timing accurate simulator modeling an Alpha 21264 processor system. We measured such characteristics as execution time, cache behavior, instruction throughput, memory bandwidth requirements, and instruction mixes. As a whole, the benchmark had very low cache miss rates, excellent branch prediction, and maintained nearly constant IPC, regardless of data set size [20].

5.2.2 Server Workload Characterization

We examined twelve operating system and server intensive workloads to study their amenability to morphing and adaption. The applications included such diverse workloads as a database server, a chat room, decision support, compilation, and file system access. We found wide variance in the behavior, not only across the different applications, but also within different phases of the application. For example in `jess`, a Java expert system shell based on the NASA CLIPS expert system shell, we found variation in processor and memory system demands of up to a factor of three. We also found and examined multiple methods of exploiting these variances in conventional processor architectures to reduce power consumption [16]. We first examined on-line monitoring techniques that observe the recent resource demands and turn on or off more execution units as necessary. We also examined an approach that uses the statically defined application/operating system boundaries as points to morph so that the system uses fewer execution resources. We found that this second approach is able to reduce the energy/delay product by a factor of three, while the best monitoring approach achieved only a factor of two.

5.2.3 GMTI

We characterized in detail the GMTI phase of the Lincoln Laboratories integrated radar tracking application. We performed a function level profile analysis and found that the execution time is dominated by FFT, IFFT, and matrix multiply functions. We ported the application to the TRIPS compilation and simulation framework. We discovered that the early stages of the application, which include the bulk of the matrix computations, benefit most from the parallelism provided by the TRIPS processor. Almost all of the application exhibited good locality and was not sensitive to L1 cache capacity. We believe that 64KB is more than adequate. The adaptive (and irregular) stages, such as adaptive beam forming (ABF) and space-time adaptive processing (STAP), showed the highest miss rates, at only 7%. In the secondary memory system, a larger L2 cache capacity of 1MB benefits ABF, but not STAP.

6 Design Technology

6.1 Circuit Studies

Future production TRIPS machines will be very large, high-performance VLSI System-on-Chip (SoC) processors, with extreme degrees of dynamic reconfigurability in support of morphing. Innovations in VLSI chip and circuit design will be required to realize TRIPS machines as a result of these characteristics, along with the escalating difficulties in controlling active and leakage power in high-performance CMOS chips. IBM has developed a number of CMOS circuit technologies and design approaches addressing performance/power-efficiency improvements that will be key to a successful realization of the TRIPS architecture.

6.1.1 Active Performance and Power Management Techniques

Active power consumption has become a limiting factor in high-performance VLSI designs. Today's designs employ various active-power-reduction mechanisms, such as clock gating, because continuous operation of all devices in large chips violates power-density and cooling limits. In the production TRIPS timeframe these problems will be more acute, as these implementations will have high transistor densities and high total transistor counts. Accordingly, we have developed and evaluated several methods for improving the active power consumption of CMOS circuits. These methods will help to provide the dynamic performance and power management capabilities required by eventual production implementations of TRIPS.

Latches strongly affect the performance and robustness of synchronous VLSI systems, and because of their large numbers, they also often dissipate a substantial fraction of the total power. Accordingly, a family of optimized, non-scannable latches has been designed for use in the IBM Cu-11 fabrication technology. These latches, which have been integrated into the IBM bulk-CMOS technology design flow, provide significant performance and power benefits for those portions of the logic in which scan-testability is not required. These latch designs have been fully validated and are in use in several IBM product chips.

Dynamic voltage and frequency islands are promising techniques for managing power/performance. They are particularly applicable to TRIPS due to the large scale of the architecture and the desire to operate various sections of the design at different operating points. Substantial research is required to develop and refine these methods for use in high-performance designs. We have investigated several of the core capabilities needed to realize dynamic islands in large-scale, high-performance designs.

New methods for on-chip clock generation are needed to support frequency islands in large designs. We have performed initial studies of the feasibility of digital PLLs (DPLLs) as replacements for analog PLLs in systems where certain known jitter levels can be tolerated. Our proposed structure is the same as an analog PLL, but with a digitally-controlled oscillator replacing the voltage-controlled oscillator. We completed a Matlab/Simulink model for the proposed DPLL and demonstrated basic functionality, including locking. The DPLL has the potential of significantly smaller area and lower power, as well as ease-of-design advantages, over analog equivalents.

Voltage islands will require the use of on-chip embeddable voltage down-converters, since pin constraints prohibit supplying multiple power supplies from off chip. We have designed and prototyped several linear-regulator-based voltage down-converters, along with a methodology for taking advantage of the resulting reduced supply voltage, to improve the power consumption of non-performance-critical circuitry. This methodology is compatible with the existing IBM ASIC/SoC tools flow, and these techniques have been shown to provide up to a 25% reduction in the power consumption of the digital logic in a clock-and-data-recovery function.

Dynamic frequency islands enable various cores or functional units to operate at the minimum speeds necessary to satisfy

performance requirements, rather than operating them all at unnecessarily high speeds and/or at fixed frequency ratios. As a step towards realizing this capability in a synchronous environment, an interface mechanism to efficiently transfer data between bus occupants with different fundamental frequency capabilities has been developed. This technique allows a fast bus master to efficiently communicate with inherently slower intellectual property (IP) cores by scaling to a common frequency, but only for the duration of a data transfer. In addition to providing a power-efficient means to respond to workload variability, this method avoids having to redesign non-performance-critical IP cores to match the highest-performance cores.

6.1.2 Static Power Reduction

Leakage is becoming a very significant power concern in advanced CMOS processes. Subthreshold leakage has become a design issue over the past several years, and in the near future gate leakage will grow to be a significant contributor to total power dissipation. Subthreshold and gate leakage both increase exponentially with device scaling. Leakage will need to be carefully managed in the design of large chips, such as production TRIPS implementations, to remain within allowable power envelopes. As a result, IBM has focused effort on the development and assessment of techniques to improve the leakage power consumption of CMOS circuits in future technologies.

Advanced technologies support multiple transistor threshold voltages. We have conducted analyses of the effects of the assignment of transistors with different threshold voltages for a variety of logic circuits. Substantial reductions in leakage have been demonstrated with multiple-threshold design techniques and through transistor topology manipulations. Simulations of these techniques on representative circuits have achieved up to an 80% reduction in gate-leakage power and a 20% reduction in subthreshold power. We have studied "active-well" techniques to dynamically modify the transistor threshold voltages. Our results show that the leakage power can thereby be varied by approximately two orders of magnitude. A test chip demonstrating these techniques and enabling evaluation of their effectiveness in terms of performance, leakage, and active power has been developed at IBM. The active-well technique was shown to be effective for extending the operating supply voltage of a design and for reducing static leakage power when systems are idle. Alternative methods of affecting the subthreshold power, active power, and operating frequency were developed as well, and are currently under evaluation.

6.2 Soft Error Rate Analysis

We also examined the effect of technology scaling and microarchitectural trends on the rate of soft errors in CMOS memory and logic circuits [32]. We developed and validated an end-to-end model that enables us to compute the soft error rates (SER) for existing and future microprocessor-style designs. The model captures the effects of two important masking phenomena, electrical masking and latching-window masking, which inhibit soft errors in combinational logic. We quantified the SER due to high-energy neutrons in SRAM cells, latches, and logic circuits for feature sizes from 600nm to 50nm and clock periods from 16 to 6 fan-out-of-4 inverter delays. Our model predicts that the SER per chip of logic circuits will increase nine orders of magnitude from the years 1992 to 2011 and at that point will be comparable to the SER per chip of unprotected memory elements. Our results emphasize that computer system designers must address the risks of soft errors in logic circuits for future designs.

7 TRIPS Prototyping

7.1 Study on the High-Level Design Tools in ASIC Design Flow

IBM completed a study of design tools for the TRIPS processor implementation using the IBM ASIC design flow. IBM also implemented a new high-level design language which enables fast prototyping and implementation of a processor like the one proposed for TRIPS.

7.1.1 ASIC Design Flow

The IBM ASIC design flow consists of many design tools. Figure 4 shows the major tools used in the IBM ASIC methodology. As an experiment, we used those design tools and implemented a component of the TRIPS processor. Since one of the goals of this exercise was to introduce UT staff to the IBM tools, a UT student worked with an IBM employee on this task.

We synthesized a prototype design of a router component in the TRIPS grid processor. This component was chosen because UT already had a Verilog specification written and the prototype TRIPS processor will use a large number of instances of this component. We went through the IBM ASIC synthesis and design flow shown in the Figure 4. The component was implemented using a 120nm CMOS technology. It uses 119476 ASIC logic cells, and has an area of 3.725 x 3.337mm^2. The target clock frequency was 300MHz. We have since revised the router implementation to use memory structures which will enable a smaller and faster design.

This exercise provided us feedback on the existing Verilog description. Some parts of the Verilog specification of the router component were not suitable for high-level synthesis. For example, Verilog timing units were ignored by the Portals compiler. The IBM ASIC flow can use other synthesis tools like Synopsys, but high-level synthesis tools should be used from the very early stage of design entry in order to make sure Verilog designs are correctly synthesized. We also found synthesis problems in the FIFO design. The TRIPS processor uses many input and output FIFO components. The FIFO synthesized from the original Verilog description uses a huge number of multiplexers and latches. This design requires a large clock gaining tree to drive many latches, increasing the die area. We also have a concern that the power consumption could be very high. One better implementation is a ring FIFO implemented using an array. This synthesis exercise has provided us a much better understanding of the actual synthesis flow that will be used for the TRIPS processor.

7.1.2 Hardware Meta Language

As the second subtask, we studied the high-level description of the polymorphic processor. We conducted this study by implementing a prototype of a high-level description language, which we call HML (Hardware Meta Language). Our aim is to create a HML that is suitable as a high-level description language for polymorphous systems such as TRIPS. We summarize the specific goals as follows:

1. Reuse of parameterized components and design patterns. The TRIPS processor has many repetitive structures. Similar components are reused a number of times with slight modifications. For example, FIFO's are used many places. The language should support the reuse of the same component with various parameters instantiated.
2. Simulation Capability. Quick prototyping is very useful for experimenting different configurations. For instance, going from 9x9 grids to 16x16 grids. An efficient simulation capability is essential to improve the design productivity of an advanced micro-architecture like TRIPS.

Tool Name	Design Task
Portals™ Compiler	High-level Synthesis
Booledozer™	Logic Synthesis, Design for Test Synthesis
ClockPro™	Clock Tree Design
Einstimer™	Static Timing Analysis
ChipBench™	Floor Planning
CPlace™	Global Routing
TheGuide™	Methodology Organizer

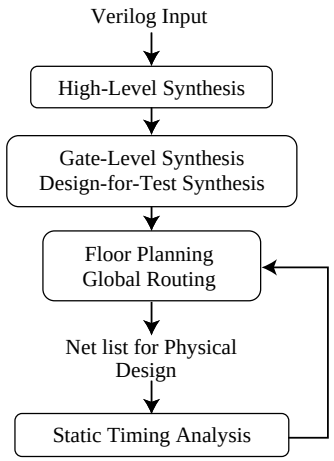


Figure 4: IBM ASIC Tools and Synthesis Flow

3. Compilation of HML to other languages, such as Verilog, VHDL, and netlist languages like IBM's VIM. This allows synthesis of the processor from the high-level design directly, without rewriting it in a hardware description language like Verilog.

HML is based on the "ML" family of MetaLanguage programming languages. It extends the concepts of ML into the domain of hardware. One of the many strengths of the ML family is its polymorphic type system. It allows the programmer to combine different computational elements together to form a new program component. In a sense, the programmer can write a template of a program and customize it as needed by instantiating that template. We may say that it is similar to the template system of the C++ programming language, but ML's polymorphic type system is more powerful as it allows the programmer to plug-in functions or functions of functions to the template. HML applies this idea to hardware description languages in order to allow reuse of components in hardware. To demonstrate the reuse advantages of HML we offer the following example. We first define a simple (transparent) latch:

```
let trips_latch clk din = forward q in if clk then din else delay(false,q)
```

The definition says that `latch` is a function of two arguments, `clk` and `din`. Its result is the (forward declared) value `q` which is defined to be `din` if `clk` is true, and a (one cycle) delayed version of `q` if `clk` is false. We then define a master-slave flip-flop from two instances of the latch just defined:

```
let msff clk din =
  let mstr = trips_latch clk din in
  let slv  = trips_latch (not clk) mstr in
  {q=slv}
```

The function `shiftreg` is a template for different types of shift registers. It is a generic higher-order function that connects sub-components in a linear array. We will use it to construct a simple shift register, and a shift register with a

parallel load capability.

```
let shiftreg n din dout eltf = forward a in array_init n (fun i -> if
i = n-1 then eltf din else eltf (dout (a[i+1])))
```

The definition says that `shiftreg` is a function with four arguments:

- `n` specifies the number of elements in the shift register.
- `eltf` is a function to be used to create the shiftregister elements.
- `din` is supplied to the 0th instantiation of `eltf`.
- `dout` is a function that is used to extract the output of each element.

The body of `shiftreg` creates an array of n applications of `eltf`. The argument of application `n-1` is simply the value of `din`. For each other array index `(i)`, the input is provided by applying `eltf` to the output of the element at index `(i+1)`.

The generic function `shiftreg` can now be used to create two different FIFO elements. The first is a simple FIFO, the second has a parallel load capability. In both cases, we simply supply functions for instantiating and extracting the output from each element to the shiftreg function.

```
let fifo n clk din = shiftreg n din (fun x -> x.q) (msff clk)
```

In this first case of a simple FIFO, the FIFO itself takes three arguments: the depth of the FIFO `(n)`, a clock, and the data input signal. An `n` stage FIFO is then created by calling `shiftreg`. Each FIFO element will be created by calling `msff` with the supplied `clk`. The output from each element is the `q` component of the `msff` output structure.

```
let fifo_pload clk load pdin sdin =
  let reg din = msff clk (if load then pdin[i] else din) in
  shiftreg (length pdin) sdin (fun x -> x.q) reg in
```

In the second case of a parallel load FIFO, the FIFO takes two additional arguments: `load` and `pdin`. The width argument `n` is no longer needed because the width can be determined from the width of the parallel load data. Each FIFO element will be created by calling (instantiating) the locally defined function `din`, which multiplexes the input to a master-slave flip-flop between the parallel load data and the serial input data based upon the value of the `load` signal. This concept of plugging in sub-components can be seen in Figure 5. In this diagram, `shiftreg` is a higher-order function defining a template of shift registers. Replacing dashed boxes with simple flip-flops creates a simple FIFO, `fifo`, while replacing them with a multiplexed flip-flop creates a loadable FIFO queue, `fifo_pload`.

A prototype of the HML simulator is implemented using a functional language OCaml. The language type-checker and compiler have not been fully implemented yet. We see that this type of high-level language, once it is fully implemented, has a high potential to allow efficient prototyping of a TRIPS-like processor.

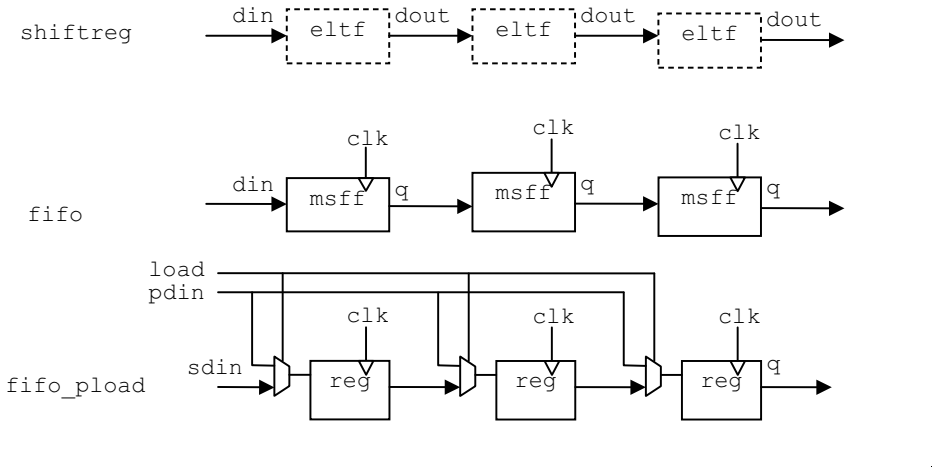


Figure 5: Example of HML Using FIFOs and FIFO queues

7.2 Feasibility Studies of TRIPS Prototype

In addition to developing proof-of-concept models and simulators during phase 1, we also performed a feasibility study of a TRIPS prototype system. Based on our analysis of available fabrication technologies we determined the 130nm technology was the most suitable. We expect the die size to be $350mm^2$, include approximately 250 million transistors, and achieve a rather conservative clock rate of 500MHz. Based on our area estimates of the TRIPS components, the chip itself will include two TRIPS processor cores (each with 16 execution tiles as described), 32 on-chip memory tiles, two SDRAM controllers, and four ports that extend the off-chip network out to the pins. Figure 6 shows an overview of the expected TRIPS chip prototype with anticipated specifications.

The off-chip extensions will allow the construction of systems composed of multiple TRIPS chips on a board. The first board implementation will include four TRIPS chips and an embedded on-board SoC (system on a chip), such as the IBM 440GP, to run the management tasks. Figure 7 details the parameters of the TRIPS system design. We have decided to select IBM as our chip fabrication partner, and USC-ISI as the board fabrication partner. We will engage with both of these organizations during phase 2.

CHIP-2-CHIP:

C
Protocol: OCN extension
64bit data path each direction
4 channels: N/S/E/W
2 GB/s each direction on each channel

Memory:

S
DDR SDRAM, PC2100 DIMMs likely
4 channels w/ page interleave
Synopsys memory controller MacroCell
2 GB/s each channel

Control Processor (CP) Interface:

G
Slave side of generic memory interface
Source interrupts to get attention
Runs like asynchronous memory
Includes CP command handler

JTAG:

J
Protocol: IEEE 1149
Includes scan intercept TAP controller
Used for test and early low-level debug

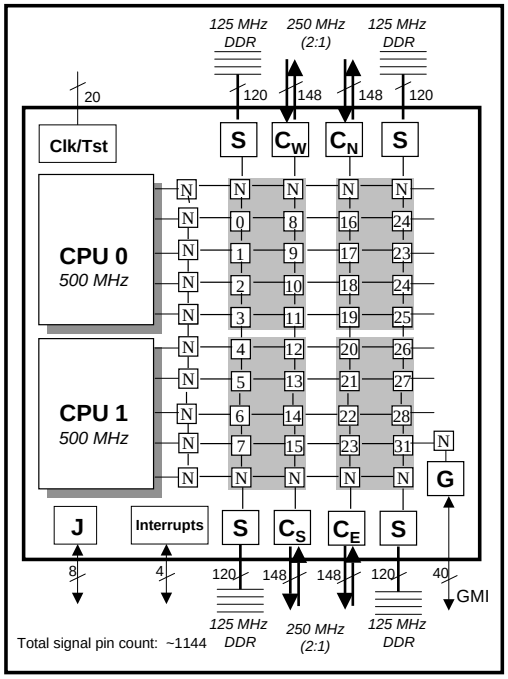


Figure 6: Overview of the TRIPS Prototype Chip

Chip-Chip and board-board:
Packet extension of on-chip network
64bit data path each direction
4 board connectors

Memory:
DDR SDRAM, PC2100 DIMMs
Up to 2 GB/port (32 GB/board)
Separate SDRAM for PPC chip

FPGA External Interface:
Xilinx Virtex-II Pro, FF896 pkg
Optionally includes a PPC core

JTAG: 5 IEEE 1149 JTAG ports
1 for each TRIPS & 1 for PPC

Host PC Connection: 10/100 Base/T E'net

Board Characteristics:
16" x 16" (est.)
No sockets!

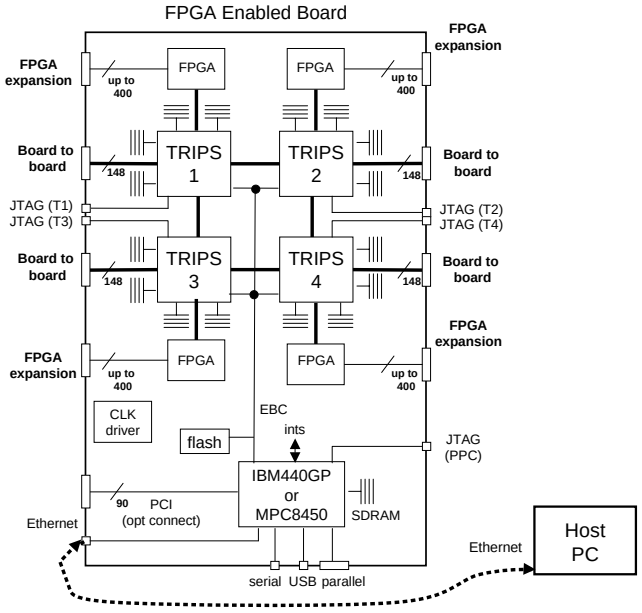


Figure 7: Overview of the TRIPS Prototype System

8 Recommendations

The TRIPS project has made substantial strides across a wide spectrum of areas. We have developed technology scalable and adaptive architectures, and demonstrated their capabilities on a diverse range of workloads. We have also developed the expertise and infrastructure necessary to transition the proof-of-concept designs into real prototypes in phase 2.

9 References

[1] E. Acar, A. Devgan, R. Rao, Y. Liu, H. Su, S. Nassif, and J. Burns. Leakage and leakage sensitivity computation for combinational circuits. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 96–99, August 2003.

[2] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger. Clock rate vs. IPC: The end of the road for conventional microprocessors. In *27th Annual International Symposium on Computer Architecture (ISCA)*, pages 248–259, June 2000.

[3] Architecture and Language Implementation Group, University of Massachusetts, Amherst. Scale compiler infrastructure. 2002.

[4] D. P. Campbell, D. M. Cottel, R. R. Judd, and M. A. Richards. Introduction to morphware: Software architecture for polymorphous computing architectures. December 2003.

[5] J. Carballo, J. Burns, S. Yoo, I. Vo, and V. R. Norman. A semi-custom voltage-island technique and its application to high-speed serial links. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 60–65, August 2003.

[6] V. Chandra, G. Carpenter, and J. Burns. Dynamically optimized synchronous communication for low power system on chip designs. In *21st International Conference on Computer Design (ICCD)*, pages 134–139, October 2003.

[7] M. D. Dahlin. User-level TVM (UVM) interface specification. Technical report, Department of Computer Sciences, The University of Texas at Austin, February 2004.

[8] L. Hammond. TVM-HAL specification. May 2004.

[9] H. Hanson, M. Hrishikesh, V. Agarwal, S. Keckler, and D. Burger. Static energy reduction techniques for microprocessor caches. *IEEE Transactions on VLSI Systems*, 11(3):303–313, June 2003.

[10] M. Hrishikesh, K. Farkas, N. P. Jouppi, D. Burger, S. W. Keckler, and P. Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In *29th International Symposium on Computer Architecture (ISCA)*, pages 14–24, May 2002.

[11] V. Kathail, M. Schlansker, and B. Rau. Hpl-pd architecture specification: Version 1.1. Technical Report HPL-93-80(R.1), Hewlett-Packard Laboratories, February 2000.

[12] S. Keckler, D. Burger, C. Moore, R. Nagarajan, K. Sankaralingam, V. Agarwal, M. Hrishikesh, N. Ranganathan, and P. Shivakumar. A wire-delay scalable microprocessor architecture for high performance systems. In *International Solid-State Circuits Conference (ISSCC)*, pages 1068–1069, February 2003.

[13] C. Kim, D. Burger, and S. Keckler. Nonuniform cache architectures for wire-delay dominated on-chip caches. *IEEE Micro*, 23(6):99–107, November/December 2003.

[14] C. Kim, D. Burger, and S. W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches. In *10th International Conference on Architectural Support for Programming Languages and Operatins Systems (ASPLOS)*, pages 211–222, October 2002.

[15] J. Lebak, J. McMahon, and M. Arakawa. Polymorphous Computing Architecture (PCA) Application Benchmark 1: Three-Dimensional Radar Data Processing. November 2001.

[16] T. Li and L. John. Routine based os-aware microprocessor resource adaptation for run-time operating system power saving. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 241–246, August 2003.

[17] Y. Luo, J. Rubio, L. John, P. Seshadri, and A. Mericas. Benchmarking internet servers on superscalar machines. *IEEE Computer*, 36(2):34–40, February 2003.

[18] D. Matzke. Will physical scalability sabotage performance gains? *IEEE Computer*, 30(9):37–39, September 1997.

[19] R. Nagarajan, K. Sankaralingam, D. Burger, and S. W. Keckler. A Design Space Evaluation of Grid Processor Architectures. In *34th International Symposium on Microarchitecture (MICRO)*, pages 40–51, December 2001.

[20] J. Rahe and S. W. Keckler. Analysis of polymorphous computing architecture (PCA) radar-processing benchmark. Technical Report TR-2003-41, Department of Computer Sciences, The University of Texas at Austin, 2003.

[21] R. Rao, J. Burns, and R. Brown. Circuit techniques for gate and sub-threshold leakage minimization for future CMOS technologies. In *29th European Solid-State Circuits Conference (ESSCIRC)*, pages 313–316, September 2003.

[22] R. Rao, J. Burns, A. Devgan, and R. Brown. Efficient techniques for gate leakage estimation. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 100–103, August 2003.

[23] R. Rao, F. Liu, J. Burns, and R. Brown. A heuristic to determine low leakage sleep state vectors for CMOS combinational circuits. In *International Conference on Computer Aided Design (ICCAD)*, pages 689–692, November 2003.

[24] K. Sankaralingam, S. W. Keckler, W. R. Mark, and D. Burger. Universal Mechanisms for Data-Parallel Architectures. In *36th International Symposium on Microarchitecture (MICRO)*, pages 303–314, December 2003.

[25] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. Keckler, and C. Moore. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. *IEEE Micro*, 23(6):46–51, November/December 2003.

[26] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, R. G. McDonald, and C. R. Moore. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. *IEEE Transactions on Architecture and Code Optimization*, 1(1):62–93, March 2004.

[27] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. In *International Symposium on Computer Architecture (ISCA)*, pages 422–433, June 2003.

[28] K. Sankaralingam, V. Singh, S. Keckler, and D. Burger. Routed inter-alu networks for ILP scalability and performance. In *International Conference on Computer Design (ICCD)*, pages 170–177, October 2003.

[29] S. Sethumadhavan, R. Desikan, D. Burger, C. R. Moore, and S. W. Keckler. Scalable memory disambiguation for high ILP processors. In *36th International Symposium on Microarchitecture (MICRO)*, pages 399–410, December 2003.

[30] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. Technical report, Compaq Computer Corporation, August 2001.

[31] P. Shivakumar, S. Keckler, C. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *International Conference on Computer Design (ICCD)*, pages 481–488, October 2003.

[32] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *International Conference on Dependable Systems and Networks (DSN)*, pages 389–398, June 2002.

[33] Z. Wang, G. N. Santhanakrishnan, D. Burger, S. Reinhardt, K. S. McKinley, and C. C. Weems. Guided region prefetching: A cooperative hardware/software approach. In *International Symposium on Computer Architecture (ISCA)*, pages 388–398, June 2003.