Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture

Karthikeyan Sankaralingam Ramadass Nagarajan Haiming Liu Changkyu Kim
Jaehyuk Huh Doug Burger Stephen W. Keckler Charles R. Moore

The University of Texas at Austin

November 23, 2003

1 Introduction

This paper describes a partitioned architecture that can scale to arbitrarily wide issue, from 4 to 16 to 64-wide out-of-order issue and beyond, particularly for future wire-dominated technologies. This TRIPS architecture is based on a new, post-RISC instruction-set architecture paradigm called EDGE, for *explicit dataflow graph execution*, first realized in the Grid Processor Architecture [7] family of architectures. Coupled with an adaptive memory system, this architecture can scale both with increased on-chip wire delays and to larger on-chip memory capacities than is conventionally possible. In this paper, we discuss how to subdivide these large TRIPS uniprocessor cores to exploit many types of parallelism, including instruction, thread, and data-level parallelism. The TRIPS architecture can thus serve as a single solution to many diverse application domains.

In the last decade, general purpose programmable processors have proliferated into increasingly diverse application domains, producing distinct markets for desktop, network, server, scientific, graphics, and digital signal processors. While clearly providing application-specific performance improvements, these processors perform poorly on applications outside of their intended domain, primarily because they are tuned to exploit specific types and granularities of parallelism, and to some extent due to instruction set specialization. Emerging applications with heterogeneous computational requirements, such as image recognition and tracking or video databases, can use multiple domain-specific processors for high performance. Such heterogeneous systems suffer from two problems: reduced economies of scale compared to a single general purpose design and design-time freezing of the processor mix and composition; should the expected workload composition change at runtime, processors will be overworked or idle.

In the TRIPS approach, a single processor core and memory system is used to support all granularities of parallelism and workloads, using an architectural capability termed *polymorphism*. Programs with serial execution chains use the entire processor core, and when thread-level or data-level parallelism is available, it is logically partitioned into finer-grained execution units. We show that this partitioning strategy yields high performance on applications with different granularities of parallelism, whereas the alternative of synthesizing a powerful coarse-grained processor from a collection of fine-grained processors has proven difficult.

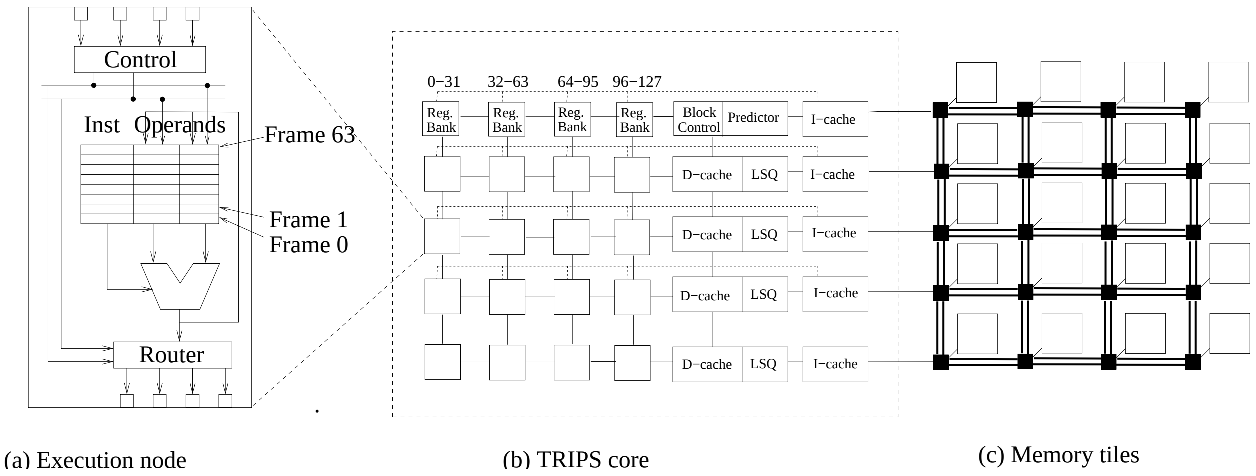


Figure 1: TRIPS architecture overview.

Thus, the industry trend of building larger numbers of less powerful processors on a single die may prove problematic. While this paper provides an overview of polymorphism in TRIPS, greater detail is presented in [9].

2 The TRIPS Architecture

The TRIPS architecture, shown in Figure 1, is partitioned and modular in both the processor and memory systems. A TRIPS processor core is composed of an array of homogeneous execution nodes connected with an operand network, each containing an integer ALU, an FPU, a lightweight router, and a set of reservation stations which can store an instruction and two operands. Banked instruction caches, data caches, and register files are placed around the execution array. The back-side of the L1 caches are connected to secondary memory tiles through the chip-wide two-dimensional interconnection network. The TRIPS team is implementing a prototype TRIPS chip with two cores and 32x64KB array of memory tiles in a 130nm ASIC process, scheduled for fabrication in 2005.

The TRIPS architecture is *block oriented* and builds on prior proposals for block-structured ISAs [2]. Programs are partitioned into large blocks of instructions with a single entry point, no internal loops, and possibly multiple exit points as found in hyperblocks [5]. Blocks commit atomically and interrupts are *block precise*, meaning that they are handled only at block boundaries. The compiler statically schedules each block of instructions onto the computational engine specifying inter-instruction dependences explicitly using an EDGE (explicit dataflow graph execution) instruction set. Instructions within a block are executed in dataflow order, and operands are routed through the operand network from producer to consumer. Block inputs and outputs are read from and written to the register file, similar to the semantics of a single instruction in a conventional ISA.

3 Polymorphism

The TRIPS architecture contains three types of resources: (1) fixed resources which operate in the same manner in every mode, like the execution node, (2) specialized resources which are not used in every mode, such as the branch predictor, and (3) polymorphous resources which are reconfigured for different modes of execution. Unlike a reconfigurable architecture, a polymorphous architecture alters the behavior of coarse grain components, rather than building new functions from primitive logic operators. The TRIPS processor contains four major polymorphous resources. The *frame space* is composed of a subset of the reservation stations across all of the execution units, and can be configured to support varying degrees of speculation. The physical *register file banks* are configured to hold the state of speculative or non-speculative threads. The *block sequencing control*, which handles block completion, deallocation, and assignment of frames adjusts the degree of speculation depending on the mode of execution. Finally, the *memory tiles* can be configured to behave as NUCA style L2 cache banks [4], scratch-pad memory or synchronization buffers for producer/consumer communication. In addition, the memory tiles closest to each processor present a special high bandwidth interface that further optimizes their use as stream register files [3].

3.1 D-morph: Instruction-Level Parallelism

The *desktop morph* (D-morph) uses polymorphism to run single-threaded codes by exploiting instruction-level parallelism (ILP). In this configuration the polymorphous frame space is treated as a large, distributed, instruction issue window, holding both non-speculative and speculative instructions from a serial program. Instructions are executed out-of-order without the need for an associative issue window of a conventional processor. The TRIPS instruction window is a three-dimensional scheduling region constructed using the ALU array and the multiple instruction slots (frames) at each ALU. The compiler schedules hyperblocks into an architectural frame (A-frame), which is a logical subset of this space.

Since the instruction window size is much larger than the average hyperblock size, speculative hyperblocks are mapped to empty A-frames. This next-hyperblock prediction is made using a highly tuned tournament exit predictor which predicts the branch likely to exit the hyperblock [9]. Figure 2a depicts the mapping and simultaneous execution of a non-speculative and a speculative hyperblock that communicate via the register file. Hyperblocks are partitioned and stored in an interleaved fashion across the different I-cache banks, thereby providing high-bandwidth instruction fetch. The memory system provides a high-bandwidth, low-latency data cache, and enforces sequential memory semantics. The secondary memory system in the D-morph configures the polymorphous memory tiles as a non-uniform cache access (NUCA) array [4].

3.2 T-morph: Thread-Level Parallelism

The *threaded morph* (T-morph) provides higher processor utilization by mapping multiple threads of control onto a single TRIPS core and exploiting thread level parallelism (TLP), in a fashion similar to simultaneous multithreading [10]. The T-morph differs from the D-morph primarily in the management of the frame space. The block sequencing control partitions the frame space and assigns a fixed subset to different threads. Each thread then uses its A-frames for one non-speculative and several speculative blocks. The polymorphous register file banks are reconfigured to hold the architectural state of multiple threads and

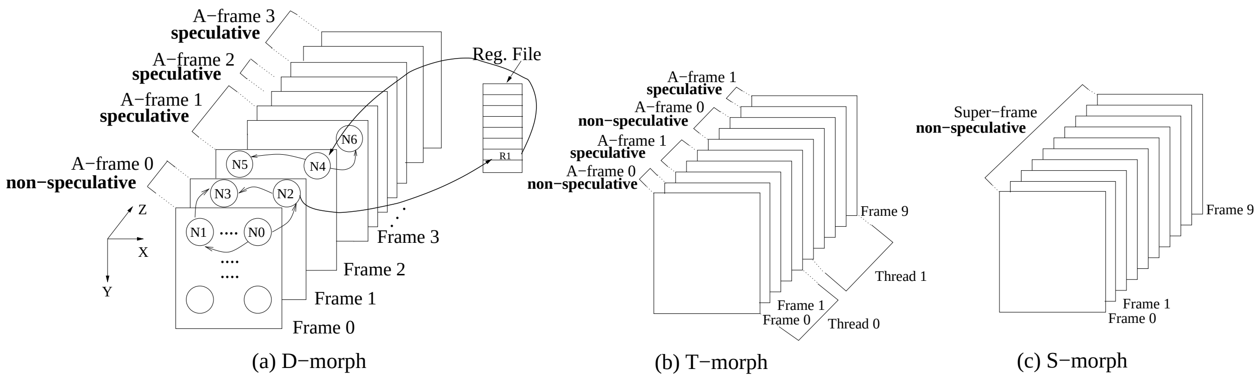


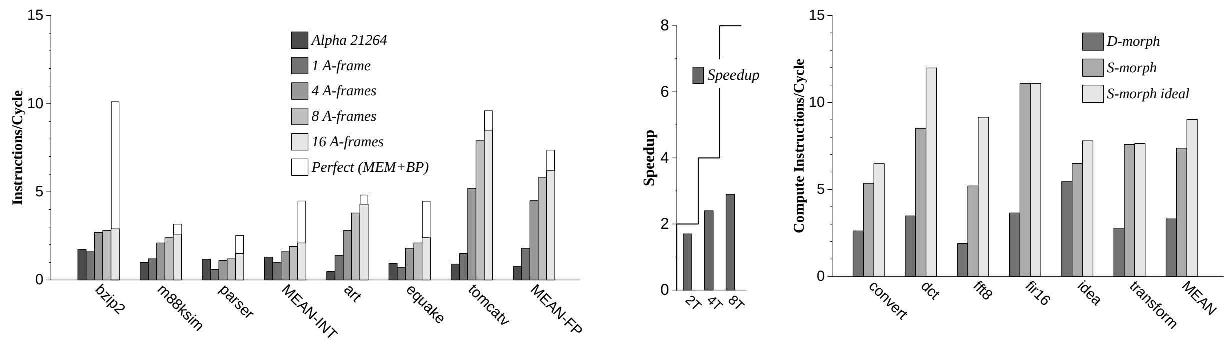
Figure 2: Frame space management.

their non-speculative and speculative blocks. In addition to configuring the polymorphous resources, the T-morph maintains dedicated per-thread program counters, commit buffers, block control, and global history shift registers in the exit predictor to reduce thread-induced mispredictions.

3.3 S-morph: Data-Level Parallelism

The *streaming morph* (S-morph) uses polymorphsim to support applications with data level parallelism (DLP). DLP applications include many streaming media and scientific applications, and are characterized by predictable loop-based control flow with large iteration counts, large data sets, regular access patterns, poor locality but tolerance to memory latency, and high computation intensity with tens to hundreds of arithmetic operations performed per element loaded from memory. The S-morph was heavily influenced by the Imagine architecture [3] and uses the Imagine execution model in which a set of stream kernels are sequenced by a control thread.

Since the control flow of these programs is highly predictable, the S-morph fuses multiple polymorphous frames to make a single *super A-frame*. Inner-loops of a streaming application are unrolled to fill the reservation stations within these super A-frames. To reduce the power and instruction fetch bandwidth overhead of repeated fetching of the same code block across inner-loop iterations, the S-morph employs *mapping reuse*, in which the block sequencing control loads a block into the reservation stations once and uses it multiple times. The S-morph implements a stream register file (SRF), similar to that of Imagine, using a subset of the polymorphous memory tiles. Implementing and configuring a memory tile as an SRF includes turning off tag checks to allow direct data array access, augmenting the cache line replacement state machine to include DMA-like capabilities (strided and scatter/gather accesses), and allowing burst access across higher bandwidth channels from the neighboring tiles and the TRIPS core. The remaining memory tiles are configured as a conventional L2-cache to implement a conventional memory hierarchy for random memory accesses.



(a) D-morph (b) T-morph (c) S-morph

Figure 3: TRIPS processor performance in different morphs.

4 Performance

Figure 3a shows D-morph performance (measured in IPC) and the effect of speculation depth as the number of A-frames is increased from 1 to 16. The D-morph consistently achieves high IPCs when compared to conventional processor cores (represented by the Alpha 21264) and to a point, shows benefits from increasing the speculative depth. Performance saturates at between 8 and 16 A-frames (with 7 or 15 being speculative) for these benchmarks. Adding frames beyond the saturation point provides no performance improvement because the frame space is underutilized due to either low hyperblock predictability or a lack of program ILP.

Figure 3b shows the average speedup of executing two, four, and eight threads concurrently in the T-morph mode, compared to running them sequentially. Of course, perfect speedup is unattainable due to instruction fetch, arithmetic, and memory resource conflicts. The results show, adding non-speculative threads improves performance, saturating at about 4 threads. While each thread runs more slowly than in isolation, due to resource conflicts and reduced speculation depth (due frame space sharing), overall throughput improves.

Figure 3c compares the S-morph configuration to the D-morph. DLP codes benefit greatly from the increased data bandwidth for regular memory accesses that are mapped to the SRF banks, and from the increased instruction fetch bandwidth provided by mapping reuse. The bars labeled *ideal* indicate that performance on several of the benchmarks is limited by memory bandwidth constraints. The TRIPS S-morph compares favorably to the Tarantula architecture, which consists of special purpose vector cores integrated with conventional ILP processors [1]. A 32-ALU TRIPS S-morph configuration sustains 15 compute ops/cycle compared to between 10 and 20 for a 32-ALU Tarantula processor.

5 Conclusions and Future Directions

Polymorphous systems such as TRIPS enable a single set of processing and storage elements to be configured for multiple application domains. Unlike prior configurable systems that aggregate small primitive components into larger processors, TRIPS starts with a large, technology-scalable core that can be logically subdivided to support ILP, TLP, and DLP. TRIPS implements polymorphism using a small set of mechanisms that alter the behavior of the reservation stations and memory tiles. The results show that by using polymorphism on a common execution substrate, TRIPS can sustain high performance across applications with different forms of concurrency. Other researchers have proposed and evaluated similar mechanisms for reconfigurable memories [6]. We have extended our work by examining a greater range of data level parallel applications and architectures [8]. Regardless of the underlying polymorphous architecture, the design of the interfaces between the software and the polymorphous hardware present challenges and opportunities to system designers. We are continuing to explore these design issues in the course of our development of the TRIPS prototype system.

Acknowledgments

This research is supported by DARPA under contract F33615-01-C-1892, NSF grants EIA-9985991, CCR-9985109, and CCR-9984336, and grants from the Alfred P. Sloan Foundation, the Peter O'Donnell Foundation, IBM, and the Intel Research Council.

References

[1] R. Espasa, F. Ardanaz, J. Emer, S. Felix, J. Gago, R. Gramunt, I. Hernandez, T. Juan, G. Lowney, M. Mattina, and A. Seznec. Tarantula: A Vector Extension to the Alpha Architecture. In *Proceedings ISCA-29*, pages 281–292, May 2002.

[2] E. Hao, P. Chang, M. Evers, and Y. Patt. Increasing the Instruction Fetch Rate via Block-structured Instruction Set Architectures. In *Proceedings MICRO-29*, pages 191–200, December 1996.

[3] B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, and A. Chang. Imagine: Media processing with streams. *IEEE Micro*, 21(2):35–46, March/April 2001.

[4] C. Kim, D. Burger, and S. W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches. In *Proceedings ASPLOS-10*, pages 211–222, October 2002.

[5] S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, and R. A. Bringmann. Effective Compiler Support for Predicated Execution Using the Hyperblock. In *Proceedings MICRO-25*, pages 45–54, 1992.

[6] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz. Smart memories: A modular reconfigurable architecture. In *Proceedings ISCA-27*, pages 161–171, June 2000.

[7] R. Nagarajan, K. Sankaralingam, D. Burger, and S. W. Keckler. A Design Space Evaluation of Grid Processor Architectures. In *Proceedings MICRO-34*, pages 40–51, December 2001.

[8] K. Sankaralingam, S. W. Keckler, W. R. Mark, and D. Burger. Universal Mechanisms for Data-Parallel Architectures. In *Proceedings MICRO-36*, Dec. 2003.

[9] 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 *Proceedings ISCA-30*, pages 422–433, June 2003.

[10] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In *Proceedings ISCA-22*, pages 392–403, June 1995.

Author Info

- Karthikeyan Sankaralingam is a PhD student in Computer Sciences at UT-Austin.
- Ramadass Nagarajan is a PhD student in Computer Sciences at UT-Austin.
- Haiming Liu is a PhD student in Computer Sciences at UT-Austin.
- Changkyu Kim is a PhD student in Computer Sciences at UT-Austin.

- Jaehyuk Huh is a PhD student in Computer Sciences at UT-Austin.
- Doug Burger is an Assistant Professor of Computer Sciences at UT-Austin. Dr. Burger has a PhD in computer science from the University of Wisconsin, is an Alfred P. Sloan Foundation fellow and a member of IEEE and ACM.
- Stephen W. Keckler is an Assistant Professor of Computer Sciences at UT-Austin. Dr. Keckler has a PhD in computer science from MIT, is an Alfred P. Sloan Foundation fellow and a member of IEEE and ACM.
- Charles R. Moore is a senior research fellow at UT-Austin. Previously, he was the chief engineer on IBM's Power4 and PowerPC 601 microprocessors.

Direct questions and comments about this article to Stephen W. Keckler, Department of Computer Sciences, The University of Texas at Austin, 1 University Station C0500, Austin, TX, 78712; skeckler@cs.utexas.edu.