A Characterization of High Performance DSP Kernels on the TRIPS Architecture

Kevin B. Bush Mark Gebhart Doug Burger Stephen W. Keckler

Computer Architecture and Technology Laboratory
Department of Computer Sciences
The University of Texas at Austin
cart@cs.utexas.edu - www.cs.utexas.edu/users/cart

Abstract

Diminishing performance gains in conventional architectures are fueling novel designs which more effectively extract parallelism and have the potential to change the nature of architectural bottlenecks. Consequently, workload characterization is of a growing importance in the design of modern high performance computing architectures. However, the accurate performance evaluation necessary for workload characterization can be prohibitively constrained by immature compilers.

In this paper, we present a workload characterization for High Performance Digital Signal Processing (HP-DSP) applications on the TRIPS architecture. Included is a bottleneck analysis of this novel next-generation architecture and a discussion of our evaluation methodology. Using a combination of hand and machine optimization techniques we succesfuly characterize the workload of the TRIPS architecture on HP-DSP applications under the constraint of a developing compiler. This detailed performance characterization illustrates the potential of HP-DSP applications to successfully map to highly concurrent hardware and discusses bottlenecks unique to the TRIPS architecture.

1 Introduction

By understanding a systems strengths and bottlenecks a compiler can be designed to better extract performance. Current technology trends signal a paradigm shift towards architectures that better extract concurrency which present both new challenges for optimizing compilers and an increased emphasis on architectural specific workload understanding. However, as growing architectural complexity widens the gap between prototype and mature compiler, so does grow the demand for accurate workload characterizations.

Using eight benchmarks of the Polymorphous Computer Architecture (PCA) C Kernel benchmark suite, we have performed a workload characterization and bottleneck analysis of the TRIPS architecture on HP-DSP applications. The PCA C kernel benchmark suite was developed by MIT Lincoln Laboratories in conjunction with the DARPA PCA program as part of an effort to develop next generation architectures for HP-DSP applications, of which this suite of kernels is representative [4]. The kernels are designed to test various aspects of a system with a mix and memory and computationally bound algorithms.

By utilizing a combination of hand and machine optimization techniques, we were able to successfully evaluate the performance of the TRIPS architecture on HP-DSP applications under the constraint of an immature compiler. As a metric for comparison, an aggressive out-of-order general purpose microprocessor, the Alpha 21264, was benchmarked with a mature, industry developed compiler. Our results include a static vs dynamic latency cost of key operations on the critical path of these benchmarks on the TRIPS architecture.

The rest of this paper is organized as follows: Section 2 provides a brief overview of the TRIPS and Alpha architectures, and the LL Kernel benchmark suite. Section 3 details the performance of the TRIPS compiler as well as characterizes each benchmark. Section 4 discusses our methodology for addressing the immature compiler. Section 5 is an analysis of the architectural bottlenecks of the TRIPS architecture and Section 6 concludes with a quantitative analysis of our results.

2 Background

2.1 TRIPS

The TRIPS architecture is the first implementation of an Explicit Data Graph Execution (EDGE) ISA, which offers a non-conventional solution to the emerging difficulties of achieving high performance while maintaining power efficiency [2]. The EDGE ISA uses a limited data-flow execution model in which instructions are statically assigned by the compiler to execution tiles and are dynamically executed as soon as their operands are available. By relying on instruction level communication, the architecture removes the need to communicate intermediate results through a global register file and

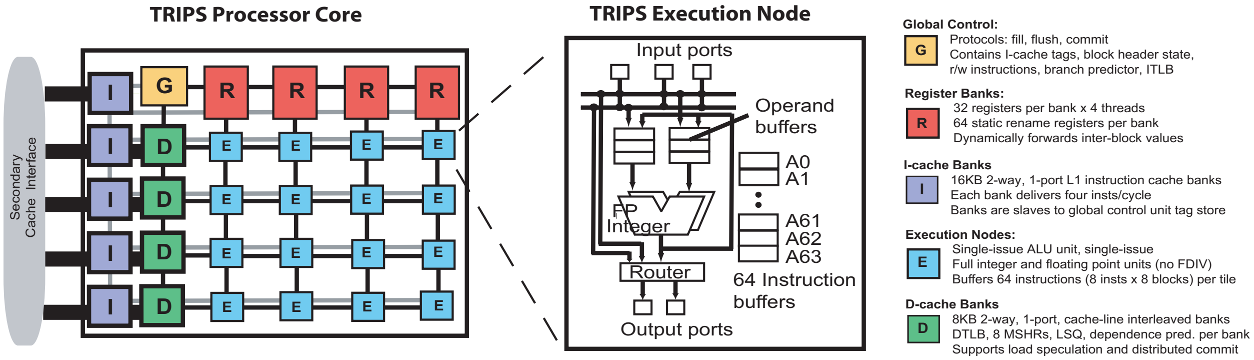
TRIPS Processor Core
Secondary Cache Interface
I
G
R
D
E
TRIPS Execution Node
Input ports
Operand buffers
A0
A1
A61
A62
A63
FP
Integer
Router
64 Instruction buffers
Output ports
Global Control:
G
Protocols: fill, flush, commit
Contains I-cache tags, block header state,
r/w instructions, branch predictor, ITLB
Register Banks:
R
32 registers per bank x 4 threads
64 static rename registers per bank
Dynamically forwards inter-block values
I-cache Banks
I
16KB 2-way, 1-port L1 instruction cache banks
Each bank delivers four insts/cycle
Banks are slaves to global control unit tag store
Execution Nodes:
E
Single-issue ALU unit, single-issue
Full integer and floating point units (no FDIV)
Buffers 64 instructions (8 insts x 8 blocks) per tile
D-cache Banks
D
8KB 2-way, 1-port, cache-line interleaved banks
DTLB, 8 MSHRs, LSQ, dependence pred. per bank
Supports load speculation and distributed commit

Figure 1: Overview of the TRIPS Design

supports distributed out of order execution. Rather than a monolithic processing core, computations are carried out on a grid of replicated ALUs allowing for an increased potential to exploit instruction level parallelism. These characteristics give TRIPS the potential to perform well on signal processing algorithms which typically contain a high degree of parallelism.

Figure 1 shows a block diagram of the TRIPS architecture. The prototype design contains 16 execution tiles arranged in a 2-D mesh topology. Each execution tile consists of 1 ALU, input ports, operand buffers, 64 instruction buffers, and routing hardware to control operand flow. Execution tiles process instructions concurrently, with a window size of 1024 instructions. When an operand must be forwarded to multiple consumers on the grid, fanout instructions are used to construct trees that route data. Ideally, instructions are scheduled on neighboring execution tiles to mitigate this routing delay.

Instructions are aggregated into instruction blocks, forming the atomic unit of execution. Blocks are generated and scheduled by the compiler onto the microarchitecture with a limitation of 128 instructions per block. These statically scheduled blocks resemble the basic execution unit of VLIW architectures, however the key difference is that instructions are not required to be independent and are dynamically issued [11]. Additional block constraints include a maximum of 32 register reads and writes to the global register banks and a maximum of 32 memory accesses to memory tiles. The TRIPS prototype supports concurrent execution of up to 8 blocks, with 7 blocks executing speculatively.

Several simplifying assumptions were made in order to produce the TRIPS hardware prototype, some of which had a substantial impact on the overall performance of this benchmark suite. Most importantly the hardware lacks support for floating point division, square root, and 32 bit floating point operations. The solution used in the prototype is to emulate division and square root in software and to convert all floating point values to double precision before performing any floating point operations. The cost to emulate the division and square root was between five to ten times higher than a hardware implementation. To guarantee that all bits of the floating point value are correct, the double value must be converted back and forth to a single after each operation.

TRIPS uses the Scale compiler which is an optimizing research compiler with a C front-end which supports the TRIPS ISA. It performs classic scalar along with TRIPS specific optimizations. Both the O3 and O4 level of optimizations were utilized which vary primarily by hyperblock formation at O4 [9]. While Scale is continually improving, it is still an immature compiler and has not evolved to the level of an industry developed compiler. A cycle accurate simulator, `tsim_proc`, which provides detailed performance information such as branch prediction, cache performance, and network contention was used to simulate the applications.

2.2 Alpha

The Alpha 21264 is an industry developed, 4-wide, super-scaler microprocessor. With an ISA that closely resembles that of the TRIPS architecture and support for aggressive out-of-order and speculative execution, it provides a good metric for comparison [7]. However, the Alpha does have several hardware advantages over the TRIPS prototype such as hardware floating point division and square root functions. The QR and SVD kernels require the square root operation, while QR, FIR, PM, and SVD all require the floating point division operation [4].

To generate the Alpha results, we first compiled each benchmark using the GEM compiler with architectural optimizations turned on (`-O4 -arch ev6`) [1]. We ran these optimized benchmarks on `sim_alpha`, which is a previously developed and validated cycle-accurate Alpha 21264 simulator [5, 6]. This simulator provides very detailed information about a program's execution and allows fine tuning of parameters to minimize difference in the memory systems of the two processors.

2.3 Benchmark Introduction

We have characterized eight benchmarks from the Polymorphous Computer Architecture (PCA) C Kernel benchmark suite which was developed by MIT Lincoln Laboratories in conjunction with the DARPA PCA program. The goal of the PCA program is to develop next-generation architectures for high performance signal processing [4]. The PCA suite of benchmarks contains many operations that are representative of signal processing. The represented algorithms are found in the libraries of many DSP applications such as radar, software defined radio, image analysis, and noise filtering. These kernels were chosen to be representative of a wide spectrum of DSP applications with some focusing on memory operations while others stress the system's computational throughput.

Each kernel contains verification code to ensure that the various optimizations applied still preserved the correctness of the algorithm. The base data type for all of the kernels is either integers or floats. In the original implementation, the kernels read input from data files. In order to avoid clouding the results with file handling, the kernels were modified to read their inputs from statically linked data sets. The core algorithms that the kernels tested were left unchanged.

3 Characterization

This section provides a characterization of each benchmark on the TRIPS architecture and compares these results to those of the Alpha 21264 processor. For each benchmark, we present a statistical summary from cycle accurate simulators. Percent value characteristics are fractions of overall fetched instructions. Some of these entries are TRIPS specific, such as hyperblock counts and percent executed `mov` instructions. In addition, percent floating point conversions provide a useful metric for accounting for prototype simplifications.

Across the benchmarks, several trends are observable. From the TRIPS O3 to O4 optimization levels, hyperblock formation is turned on which combines basic blocks into a single hyperblock. As a result, a reduction in block counts should be observable which will also impact the number of branches that must be predicted. Larger blocks provide more opportunities for operands to bypass the register file but require more `mov` instructions. A significant difference in the number of register accesses is consistently observed across the benchmarks from TRIPS O4 to Alpha, as a result of the direct instruction communication of the TRIPS architecture.

3.1 CPU Bound Kernels

Several of the signal processing kernels relied heavily on repeated element-wise computations to perform operations such as vector add, multiply, and divide. While independent instructions introduce opportunities for concurrency on the replicated resources on the TRIPS architecture, dependent instructions are accelerated by instruction level communication that bypasses the global register file. In the following sections, we evaluate the CPU intensive QR, convolution, and FIR kernels.

3.1.1 Convolution

The convolution kernel performs element-wise complex multiplications using a series of filters on an input vector defined in the frequency domain. The convolution operation is used extensively in DSP, biomedical engineering, and graphics for smoothing, filtering, and image analyses. This kernel uses a loop to select a particular filter and a nested loop to apply the filter to an input vector.

The vast majority of convolution's execution time is spent in a element-wise multiplication where the complex structs - a pack of two single precision floating point values must be loaded and decomposed. Additionally, the TRIPS architecture only supports computation on 64-bit floating point numbers so each of the floating point values must be converted to and from double-precision before and after the multiplication to guarantee single precision accuracy. This results in a lengthy data dependence chain which accounts for the majority of the execution.

The most striking difference between the TRIPS Scale O4 and the Alpha GEM on this benchmark is a 2-fold increase of cycles from TRIPS to Alpha and a 2-fold decrease of instructions. This benchmark is highly parallel and independent unrolled inner loop bodies can execute concurrently.

3.1.2 Finite Impulse Response Filter (FIR)

The Finite Impulse Response (FIR) kernel is a software implementation of a discrete time filter system. It is commonly used in digital signal processing systems to filter out input frequency components while preserving the phase of the

Characteristic	TRIPS -O3	TRIPS -O4	Alpha
Cycles	198,391	187,663	334,479
Fetched Instructions	857,293	857,611	442,573
% Instructions Executed	99.5%	99.6%	98.3%
% Executed `mov`'s	21.0%	21.0%	N/A
% Floating Point Conversions	22.9%	22.9%	N/A
% Useful Floating Point Ops	22.9%	22.9%	44.4%
% Memory Executed	23.7%	23.7%	48.6%
Blocks	9,976	9,803	N/A
Branch Flush PKI	0.20	0.22	.67
Register Accesses per Instruction	0.11	0.11	2.57
L1 I-cache hit rate	95.6%	95.5%	99.7%
L1 D-cache hit rate	96.3%	96.0%	99.5%

Table 1: Characterization of Convolution

input signal. These characteristics make it extremely useful in applications such as digital communication systems, signal conditioning, and radar. The kernel's main operation consists of a base-4 Fast Fourier Transform (FFT), a fast convolution, and a base-4 Inverse Fast Fourier Transform. The FFT and IFFT operations are O($n \log n$) and dominate execution time.

Working on single precision complex data sets, FIR spends over 46% of it's time performing sine and cosine arithmetic. While FIR does exercise floating point divisions, they are infrequent and account for less than 1% of the overall execution time at TRIPS O4. The instruction count increase and block count decrease, seen in Table 2, from O3 to O4 can both be attributed to optimizations associated with hyperblock formation performed at O4.

In FIR we see the Alpha accessing its registers a factor of 10 more than the TRIPS architecture. This demonstrates a benefit of the block-atomic execution model which bypasses the global register file for inter-block temporary values. Since the register file is a shared global resource, it can become a resource bottleneck especially on CPU intensive programs. This approach to reducing register pressure by reducing the number of accesses rather than increasing the size or number of ports of the register file allows the hardware design to scale more effectively.

3.1.3 QR Factorization

QR factorization is a linear algebra operation that factors a matrix into an orthogonal component Q and a triangular component R. This operation is widely used in adaptive systems and signal processing in conjunction with a triangular

Characteristic	TRIPS -O3	TRIPS -O4	Alpha
Cycles	111,342	101,628	50,350
Fetched Instructions	144,456	147,276	59,691
% Instructions Executed	89.4%	88.9%	93.2%
% Executed `mov`'s	20.1%	21.3%	N/A
% Floating Point Conversions	6.3%	6.2%	N/A
% Useful Floating Point Ops	14.6%	14.3%	39.3%
% Memory Executed	13.7%	13.6%	27.8%
Blocks	5742	5294	N/A
Branch Flush PKI	3.52	2.99	6.52
Register Accesses per Instruction	0.28	0.28	2.56
L1 I-cache hit rate	92.4%	90.6%	97.7%
L1 D-cache hit rate	92.2%	92.6%	96.1%

Table 2: Characterization of FIR

solver to approximate over-determined systems, with applications in communication systems, radar, and biomedical engineering. The Fast-Givens algorithm is used, which is characterized by iterations through several loops on disjoint paths composed of fine-grained computations on floating point numbers representing complex data.

The many conditionals within the primary loop of QR adversely affect the i-cache performance on the TRIPS architecture. By using single precision floating point numbers and performing divisions and square roots, QR exercises several prototype simplifications of the TRIPS architecture. As in convolution, each complex number must be fetched from memory, decomposed into the two single precision floating points before being converted into double precision floating points. Also, because the TRIPS architecture does not support floating point division or square root, software routines to do these computations must be used. Table 3 shows a similar decrease in block count, increase in instructions, and increase in `mov`'s as FIR.

Software emulation of floating point division was responsible for 41.6% of the overall execution time. When software emulation of floating point division is disregarded the TRIPS O4 version executes in 77,710 cycles of which another 25% can be attributed to emulating square root. Disregarding emulation of square root and floating point division the TRIPS O4 achieves a 31.2% speedup over Alpha.

Characteristic	TRIPS -O3	TRIPS -O4	Alpha
Cycles	140,700	133,417	84,649
Fetched Instructions	288,131	298,169	95,004
% Instructions Executed	95.1%	93.2%	84.1%
% Executed `mov`'s	22.5%	23.3%	N/A
% Floating Point Conversions	14.0%	13.5%	N/A
% Useful Floating Point Ops	12.3%	11.9%	38.0%
% Memory Executed	12.0%	11.9%	35.3%
Blocks	7,690	6,831	N/A
Branch Flush PKI	1.63	1.58	5.06
Register Accesses per Instruction	0.23	0.23	2.51
L1 I-cache hit rate	92.6%	90.2%	98.4%
L1 D-cache hit rate	93.7%	94.7%	97.6%

Table 3: Characterization of QR

3.2 Memory Bound Kernels

Many HP-DSP applications operate on large data sets and therefore the memory capabilities of a system must be considered. Several kernels made extensive access to large structures such as databases and matrices, which placed a heavy demand on the memory system. The TRIPS architecture employs a banked memory system to provide high memory bandwidth. This allows memory accesses to different banks to be performed concurrently. The following subsections characterize the memory bound CT and DB kernels.

3.2.1 Corner Turn (CT)

The CT kernel performs a matrix transpose on a contiguous block of memory. Matrix transposition is fundamental to linear algebra and is used widely in multimedia, radar, and image analysis applications. By using the corner turn operation, lower dimensional problems can be transformed into higher dimensional problems to exploit the inherent parallelism for significant performance gains. The matrix transpose is implemented with a double loop and operates on a matrix of 50x750 single precision floating point values. Consisting entirely of loads, stores, and address calculations, this algorithm stresses the memory throughput of the system. Because this algorithm traverses in both row and column major order, it presents a challenge for memory systems.

Table 4 shows a slowdown in cycle counts from O3 to O4. This is due to contention in the operand network resulting from poorly scheduled larger blocks. Because larger blocks are more susceptible to dynamic contention a more mature

Characteristic	TRIPS -O3	TRIPS -O4	Alpha
Cycles	150,957	154,457	165,073
Fetched Instructions	720,686	761,492	369,819
% Instructions Executed	99.2%	98.9%	99.4%
% Executed `mov`'s	14.6%	17.2%	N/A
% Floating Point Conversions	10.4%	9.9%	N/A
% Useful Floating Point Ops	0.0%	0.0%	0.0%
% Memory Executed	10.7%	10.2%	22.6%
Blocks	13,264	13,109	N/A
Branch Flush PKI	0.14	0.15	0.67
Register Accesses per Instruction	0.15	0.14	2.49
L1 I-cache hit rate	97.9%	97.8%	99.6%
L1 D-cache hit rate	33.1%	33.6%	87.5%

Table 4: Characterization of CT

scheduler could realize performance gains.

The TRIPS banked memory system provides high memory bandwidth which is crucial for this application. The poor d-cache performance observed on TRIPS for this benchmark is a reflection of the unusually high instruction window which allows many independent loads to the same cache line to issue concurrently resulting in an artificially high miss rate. Under the constraint of an immature compiler, the TRIPS architecture was able to achieve a speedup over Alpha of only 6.4%. The TRIPS prototype has twice the memory bandwidth of Alpha so the maximum expected speedup would be 100%.

3.2.2 Signal Database (DB)

The DB kernel operates on a large database of signals. The kernel repeatedly inserts, searches, and deletes various entries in the database. The database operations are implemented with a series of red-black tree permutations. Because of the large size of the database, the kernel places a large stress on the memory system.

DB's internal data structure is a Red-Black Tree. 50% of the overall execution time can be attributed to Red-Black Tree operations and memory management. In particular, traversing the tree requires many memory accesses to find the next node. At O4, the many small functions can be grouped together through hyperblock formation to see a overall decrease of dynamic block count of 25%. This in turn, causes a 26% reduction in flushes caused by branch mispredictions shown in Table 5.

Characteristic	TRIPS -O3	TRIPS -O4	Alpha
Cycles	258,088	230,867	75,843
Fetched Instructions	127,339	148,599	89,270
% Instructions Executed	90.5%	85.8%	80.0%
% Executed `mov`'s	30.4%	39.0%	N/A
% Floating Point Conversions	2.9%	2.4%	N/A
% Useful Floating Point Ops	1.4%	1.2%	1.7%
% Memory Executed	23.8%	22.1%	39.6%
Blocks	16,365	12,294	N/A
Branch Flush PKI	9.93	7.38	19.22
Register Accesses per Instruction	0.58	0.52	2.29
L1 I-cache hit rate	89.4%	85.1%	98.9%
L1 D-cache hit rate	96.6%	96.7%	97.8%

Table 5: Characterization of DB

In contrast, the Alpha branch flush rate is 2.6 times higher than the TRIPS O4 flush rate. Since flushes are costly, this can have a significant impact on performance. However, the Alpha was able to output the TRIPS O4 compiler by a factor of 3. This is because rather than experiencing a single large bottleneck, we found the DB kernel to have numerous opportunities for small performance gains across many functions, of which the mature GEM compiler, but not the immature TRIPS compiler, was equipped to handle.

3.3 CPU and Memory Bound Kernels

Some DSP operations place both a computational and memory demand on the system. These algorithms are particularly important to analyze for identifying potential bottlenecks in a systems overall performance on more robust applications. The following subsections discuss the CFAR, SVD and PM kernels.

3.3.1 Constant False Alarm Rate Detection (CFAR)

The CFAR kernel searches for randomly placed targets in an environment filled with background noise. This algorithm is used in radar, sonar, and image processing. In radar applications this operation is crucial to removing environment noise. The algorithm loops though a data cube and looks for cells with a power exceeding a threshold relative to their neighbors.

The CFAR kernel uses a doubly nested loop to examine each dimension of the data cube. This kernel performs

Characteristic	TRIPS -O3	TRIPS -O4	Alpha
Cycles	205,131	204,701	130,542
Fetched Instructions	257,845	357,455	210,782
% Instructions Executed	96.9%	97.3%	87.3%
% Executed mov's	20.1%	40.0%	N/A
% Floating Point Conversions	14.5%	10.4%	N/A
% Useful Floating Point Ops	15.5%	11.2%	19.0%
% Memory Executed	15.7%	15.0%	36.2%
Blocks	18,874	14,991	N/A
Branch Flush PKI	2.17	1.34	3.59
Register Accesses per Instruction	0.82	0.71	2.51
L1 I-cache hit rate	95.7%	95.9%	99.5%
L1 D-cache hit rate	97.5%	97.3%	98.8%

Table 6: Characterization of CFAR

limited floating point division and on TRIPS uses software emulation to perform the division, which accounts for 8% of execution. A striking statistic of this benchmark is the large percentage of mov instructions. Table 6 shows that at the O4 optimization level, nearly 40% of all fetched instructions are operand transfers. This demonstrates a significant overhead of the TRIPS architecture.

In contrast to the TRIPS results, the Alpha experienced a relatively low ratio of instructions executed to fetched. This can be attributed to the factor of 2.69 increase of branch flushes over the TRIPS O4 results. This application exhibits less concurrency than the other benchmarks and the Alpha outperforms TRIPS O4 by 57%.

3.3.2 Pattern Matching (PM)

The Pattern Matching (PM) kernel randomly adds noise to a test signal and compares this signal to a library of test patterns to determine what pattern the signal was originally. The metric for comparison is weighted mean square error. The combination of the large library of test patterns and this mathematically intensive algorithm provides a balanced mix between CPU and memory operations. This kernel is representative of the pattern matching needs of many DSP applications including radar and signal identification, where noisy inputs need to be matched to a library of known signals. This kernel is composed of one main loop that uses the weighted mean square error for each test pattern in the library.

On TRIPS 04 the kernel spends approximately 12% of its time performing log-arithmetic and exponential operations.

Characteristic	TRIPS -O3	TRIPS -O4	Alpha
Cycles	319,833	303,184	123,736
Fetched Instructions	281,194	364,375	149,279
% Instructions Executed	95.5%	94.9%	94.7%
% Executed mov's	20.7%	30.0%	N/A
% Floating Point Conversions	9.5%	8.1%	N/A
% Useful Floating Point Ops	14.3%	11.0%	27.0%
% Memory Executed	15.3%	19.2%	27.4%
Blocks	24,675	19,076	N/A
Branch Flush PKI	3.42	1.72	3.93
Register Accesses per Instruction	0.43	0.33	2.59
L1 I-cache hit rate	96.5%	96.2%	99.1%
L1 D-cache hit rate	98.5%	97.5%	98.2%

Table 7: Characterization of PM

Another 11% of its time is devoted to emulating floating point division. On the TRIPS architecture, there is less correlation between instructions and performance because of the block atomic execution model and this is further pronounced at O4 when hyperblock formation is performed. This can be seen in Table 7 where there is an increase in the number of instructions executed from O3 to O4 but a reduction in cycle and block count.

3.3.3 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a linear algebra transformation that is commonly used to eliminate noise from data. There are applications for SVD in image processing, seismology, and tomography. There are several different operations with order of magnitude n^3 that make up the SVD operation. These include QR factorization, bi-diagonalization, diagonalization and matrix multiplication all of which have the potential to map well to concurrent hardware [8].

The SVD main loop is a single loop that traverse each column of a matrix of complex floating point values. The main performance bottleneck for SVD on TRIPS is the lack of hardware square root - software emulation accounts for 43% of execution time. Another 9% of time is devoted to performing software floating point division.

A striking result in the Alpha and TRIPS O4 results is a 5.14 fold increase in instruction counts depicted in Table 8. The instruction increase can be attributed to the following three factors: software floating point division and square root emulation, single precision to double precision conversions, and operand transfer (mov) instructions.

Characteristic	TRIPS -O3	TRIPS -O4	Alpha
Cycles	270,700	249,349	55,601
Fetched Instructions	340,069	354,849	68,989
% Instructions Executed	81.4%	80.1%	78.2%
% Executed `mov`'s	32.5%	34.3%	N/A
% Floating Point Conversions	3.5%	3.4%	N/A
% Useful Floating Point Ops	3.0%	2.9%	17.0%
% Memory Executed	5.5%	5.5%	32.9%
Blocks	13,694	12,129	N/A
Branch Flush PKI	6.92	7.35	14.00
Register Accesses per Instruction	0.38	0.38	2.45
L1 I-cache hit rate	90.5%	86.6%	97.2%
L1 D-cache hit rate	93.5%	94.6%	96.3%

Table 8: Characterization of SVD

4 Optimizations

4.1 Fill Blocks

Filling hyperblocks on the TRIPS architecture is a key optimization goal. When loop unrolling is performed within a block, independent loop bodies can take better advantage of the redundant hardware in the execution grid and execute concurrently. Additionally, by increasing the number of instructions within a block, one can increase the opportunities for fine-grained instruction level parallelism within that block by exposing new independent instructions to the hardware. However, these increased opportunities for parallelism can sometimes adversely affect performance by increasing contention on the operand network.

4.2 Reduce Dynamic Block Count

Analogous to instruction counts in conventional architectures, the number of dynamic hyper-blocks correlates with overall runtime performance in the block-atomic execution model. It is therefore a strategic optimization principle to aggressively construct hyperblocks in a way as to minimize the number of block fetches.

By merging the instructions of two or more smaller blocks into a single large block, one can amortize the overhead of fetching the blocks. By indirectly creating fuller blocks, block merging provides additional performance gains by exposing new opportunities for concurrency to the hardware. However, simple block merging cannot always be performed because of block constraints and function calls. Function calls create artificial block boundaries in a block-atomic execution model because all branches must target block entries. Inlining not only amortizes the large block overhead of these function calls, but it can create new opportunities for subsequent block merging.

4.3 Removing Control Flow

Control dependences naturally create artificial block boundaries. Support for predication on the TRIPS architecture provides a means to convert control dependences into a data dependences which provides many new opportunities for block merging. Also, by removing difficult to predict control dependences the number of branch flushes can be reduced. Even though predication is supported at O3 and used extensively during hyperblock formation at O4, further predication was necessary during hand optimization in light of the immature compiler.

4.4 Results of Hand Optimization

By applying the aforementioned TRIPS specific optimization principles and improving on Scale's conventional optimizations such as loop unrolling, inlining, and constant propagation, we were able to realize an average performance speed up of nearly 35% over O4 with floating point division hardware support simulated. Table 9 shows the results of performing the hand optimizations discussed in the previous subsections. The fdiv column header indicates that for the benchmarks that used floating point division the cost to emulate floating point division was ignored in both the compiled and hand optimized code.

While an increased ratio of useful floating point operations to total instruction counts can be mostly attributed to the significant overall instruction reductions, some benchmarks such as CONV, DB, and FIR show an overall decrease in floating point operations. This can be attributed to the construction of fuller blocks. With larger blocks, a value can be calculated and shared among more instructions without being register allocated or recalculated.

4.5 Simulated Annealing

The TRIPS architecture is one of several recent projects that shift complexity from the hardware to the software. In the TRIPS architecture, the software is responsible for not only translating the high level language into machine code but

Benchmark	Cycle Speedup over -O4 fdiv	Instruction Reduction over -O4 fdiv	%mov		% Useful FP Ops	
			-O4 fdiv	hand	-O4 fdiv	hand
CFAR	21.7%	7.7%	40.0%	34.1%	11.2%	11.2%
CONV	33.2%	17.1%	21.1%	23.6%	22.9%	22.7%
CT	61.9%	63.4%	17.2%	32.7%	0%	0%
DB	18.0%	20.9%	39.0%	37.7%	1.2%	1.3%
FIR	10.1%	10.5%	21.3%	21.9%	14.4%	13.6%
PM	48.9%	19.6%	30.1%	32.8%	12.7%	14.1%
QR	19.3%	25.6%	21.7%	33.5%	17.0%	21.9%
SVD	66.0%	67.6%	34.9%	28.7%	3.1%	12.6%
Average	34.9%	29.1%	28.2%	30.5%	10.3%	12.2%

Table 9: Results of Hand Optimization

also for mapping each instruction to an execution tile in the hardware. Operands are passed between execution tiles via an OPerand Network (OPN) which has a 1 cycle static cost between tiles. An OPN bandwidth limitation of 1 operand per line per cycle creates potential contention situations whenever two or more operands need to be routed on the same line at the same time. In these situations, contending operands which may be on the critical path of a block, must be delayed.

The performance critical task of assigning operations to execution tiles is handled by the scheduler. However, since the scheduler is still in development, one must isolate its effects to accurately evaluate the performance of this architecture. To accomplish this we used a simulated annealing process previously developed to find the best schedule [3]. The simulated annealer permutes the schedule of the most critical block and calculates the resulting critical path to determine if the permutation was effective. This is an iterative process that is both time and resource consuming; however, once completed the resulting schedule isolates the effect of having an immature scheduler for a more accurate performance evaluation.

4.6 Results of Annealing

Table 10 details the absolute cycle counts before and after the simulated annealing process along with the percent improvement. The table also shows the percent of the critical path spent performing operand routing. This is the portion of execution the annealer aims to improve. Those benchmarks in which OPN routing delays were most critical, such as

Kernel	Hand Til Cycles	Annealed Cycles	% Annealer Improvement	% Operand Transfer of Critical Path
CFAR	159,541	149,925	6.0%	40.2%
CONV	125,444	113,830	9.3%	39.5%
CT	58,775	50,325	14.4%	35.7%
DB	189,352	184,285	2.7%	26.0%
FIR	90,253	89,680	0.6%	18.6%
PM	134,844	*	*	*
QR	62,703	62,566	0.2%	27.3%
SVD	77,321	*	*	*
Average	**-**	**-**	**5.5%**	**31.2%**

Table 10: Results of Simulated Annealing ** to appear in final version*

CT and convolution observed the best gains.

In CT, rapid successive memory accesses generate a heavy load on the OPN, shuffling operands to and from the data tiles. In the case of convolution, its lengthy data dependence of floating point multiplications, subtractions and additions inside the inner most loop, make it particularly schedule sensitive because any contention along that path will be propagated along the entire critical path. In contrast, the relatively small amount (7%) of dynamic delay associated with operand transfer offered little opportunity for scheduling improvement ultimately resulting in QR's observed small performance gain.

5 Bottleneck

The TRIPS architecture exhibits a set of performance constraints. When many consuming instructions need to share a single producer's value, a fanout tree is created with `mov` instructions to route the data. These `mov` instructions often lie on and thus lengthen the critical path significantly. On average, these `mov` instructions account for over 33% of the overall dynamic instruction count.

Figure 2 breaks down the instruction mix of each benchmark into performance critical fields such as branch, integer, memory, floating point, and `mov` instructions. The branches account for a relatively few number of instructions because many control dependences are converted to data dependences within a hyperblock. Although these benchmarks operate primarily on floating points, integer operations account for a significant portion of these benchmarks because of the many

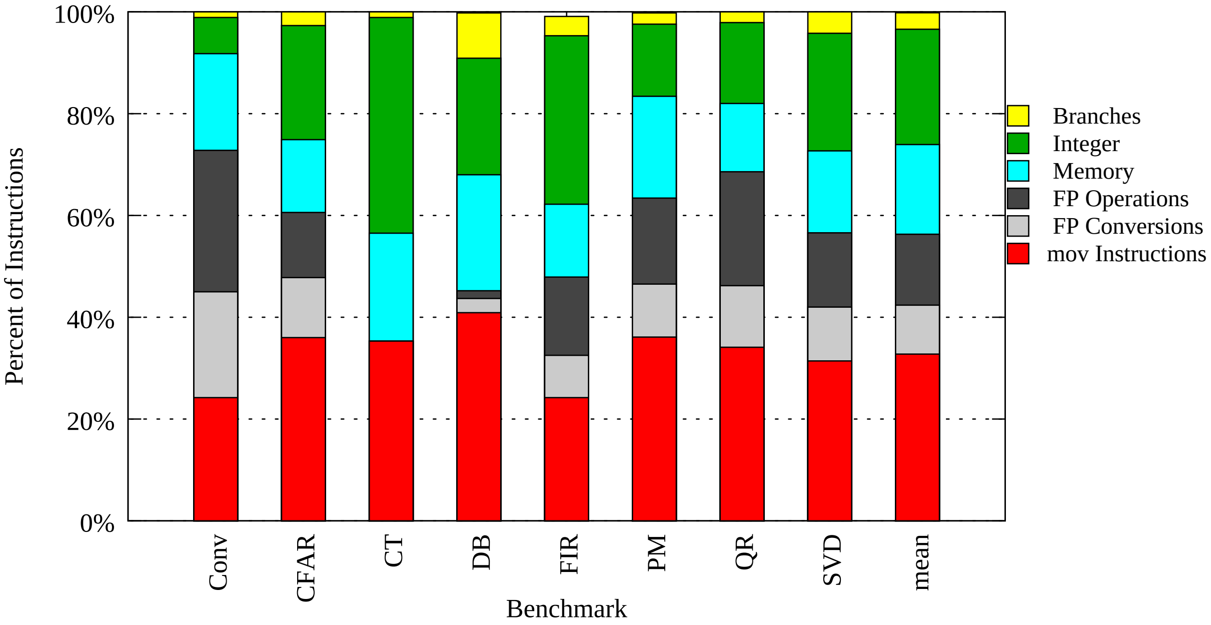

Figure 2: Instruction Mix for Optimized Benchmarks

operations necessary for address calculation. On average, nearly 10% of all instructions are floating point precision conversions which is an artifact of a prototype simplification. Additional prototype simplifications which proved to be performance bottlenecks include lack of floating point division and square root support. These simplifications are addressable, however, with an ISA extension.

Because instructions fire when their operands are ready, it is crucial in TRIPS to consider the dynamic latencies of a program. Operands can be delayed significantly from arriving at their target instruction by OPN contention in addition to cache misses. To accurately express these latencies within the critical path, we employed a previously developed tool, `tsim_critical`, which enumerates the critical path of a program [10]. Figure 3 breaks down the cost associated with each type of operation on the critical path of each benchmark into the expected static and additional dynamic costs. The dynamic block overhead is a function of block-fetch miss-predictions and misses in the 16kB instruction cache. While the critical block schedules have been improved via the simulated annealing process, it is clear from the graph a significant dynamic delay still persist in operand transfer operations. Future work could include an investigation of how to address this problem including increasing OPN bandwidth and employing contention-aware routing logic.

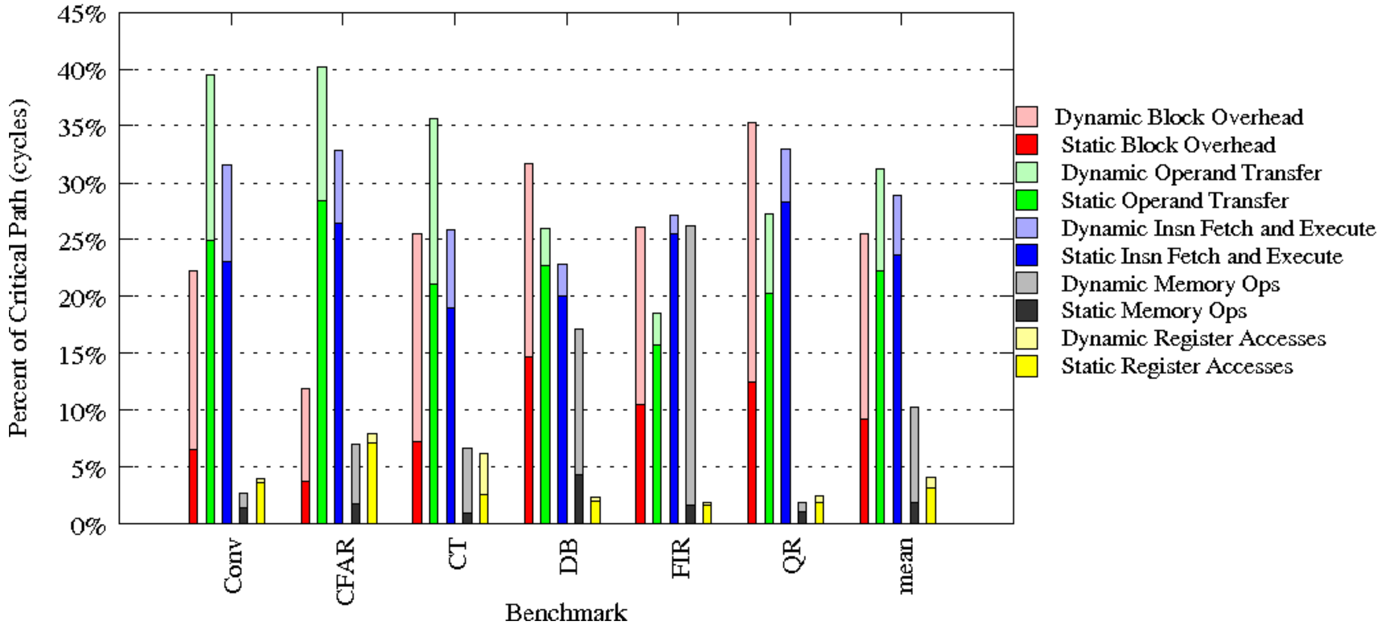

Figure 3: Breakdown of Critical Path

6 Conclusion

In this paper we have presented a detailed characterization and bottleneck analysis of a next-generation processor on High-Performance Digital Signal Processing applications. As a metric for comparison we provided a performance evaluation of these applications on an industry standard Alpha 21264 microprocessor utilizing the highly optimizing GEM compiler. This comparison shows the potential for HP-DSP applications to map successfully to concurrent hardware and highlights the growing importance of an optimizing compiler for future architectures.

The workload characterization necessary for hardware evaluation on highly concurrent hardware can be prohibitively constrained by an immature compiler. By using a combination of hand and machine optimization techniques we were able to mitigate these effects and successfully characterize the TRIPS architecture on HP-DSP applications.

Our analysis highlights several TRIPS specific architectural bottlenecks on HP-DSP applications. In addition to the addressable prototype simplifications, we found two other performance critical bottlenecks unique to the TRIPS architecture. In particular, fanout trees - necessary for routing results to many consumers - account for an average of 32.8% of the overall instruction count. Additionally 31.2% of the critical path is spent performing operand routing, 29% of which can be attributed to network contention. This work successfully identifies TRIPS specific bottlenecks on HP-DSP applications and signals the importance of early stage workload characterization on future highly concurrent architectures.

References

[1] D. Blickstein, P. Craig, C. Davidson, N. Faiman, K. Glossop, R. G. S. Hobbs, and W. Noyce. The GEM Optimizing Compiler System. 4(4):121–136, 1992.

[2] D. Burger, S. W. Keckler, K. S. McKinley, M. Dahlin, L. K. John, C. Lin, C. R. Moore, J. Burrill, R. G. McDonald, and W. Yoder. Scaling to the End of Silicon with EDGE Architectures. *IEEE Computer*, pages 44–55, July 2004.

[3] K. Coons, X. Chen, S. Kushwaha, K. S. McKinley, and D. Burger. A Spatial Path Scheduling Algorithm for EDGE Architectures. In *The Twelth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, CA, October 2006.

[4] DARPA. *Polymorphous Computing Architecture Program, http://www.darpa.mil/ipto/programs/pca*, January 2006.

[5] R. Desikan, D. Burger, S. Keckler, and T. M. Austin. Sim-alpha: a Validated, Execution-Driven Alpha 21264 Simulator. Technical Report TR-01-23, Department of Computer Sciences, The University of Texas at Austin, October 2001.

[6] R. Desikan, D. Burger, and S. W. Keckler. Measuring Experimental Error in Microprocessor Simulation. In *28th International Symposium on Computer Architecture*, July 2001.

[7] R. E. Kessler. The Alpha 21264 Microprocessor. 19(2):24–36, March 1999.

[8] B. A. Maher. Optimization of Singular Value Decomposition. Technical report, University of Texas at Austin, December 2005.

[9] K. S. McKinly, J. Burrill, D. Burger, B. Cahoon, J. Gibson, J. E. B. Moss, A. Smith, Z. Wang, and C. Weems. The Scale Compiler. Technical report, University of Massachusetts, University of Texas, 2005.

[10] R. Nagarajan, X. Chen, R. G. McDonald, D. Burger, and S. W. Keckler. Critical Path Analysis of the TRIPS Architecture. In *IEEE International Symposium on Performance Analysis of Systems and Software*, 2006.

[11] R. Nagarajan, S. K. Kushwaha, D. Burger, K. McKinley, C. Lin, and S. W. Keckler. Static Placement, Dynamic Issue (SPDI) Scheduling for EDGE Architectures. In *International Conference on Compilation Techniques (PACT)*, September 2004.