ALP: Efficient Support for All Levels of Parallelism for Complex Media Applications

RUCHIRA SASANKA, MAN-LAP LI, and SARITA V. ADVE University of Illinois at Urbana-Champaign and YEN-KUANG CHEN and ERIC DEBES Intel Corporation

The real-time execution of contemporary complex media applications requires energy-efficient processing capabilities beyond those of current superscalar processors. We observe that the complexity of contemporary media applications requires support for multiple forms of parallelism, including ILP, TLP, and various forms of DLP, such as subword SIMD, short vectors, and streams. Based on our observations, we propose an architecture, called ALP, that efficiently integrates all of these forms of parallelism with evolutionary changes to the programming model and hardware. The novel part of ALP is a DLP technique called *SIMD vectors and streams* (*SVectors/SStreams*), which is integrated within a conventional superscalar-based CMP/SMT architecture with subword SIMD. This technique lies between subword SIMD and vectors, providing significant benefits over the former at a lower cost than the latter. Our evaluations show that each form of parallelism supported by ALP is important. Specifically, SVectors/SStreams are effective, compared to a system with the other enhancements in ALP. They give speedups of 1.1 to 3.4X and energy-delay product improvements of 1.1 to 5.1X for applications with DLP.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)

General Terms: Design, Performance

Additional Key Words and Phrases: Parallelism, media applications, DLP, TLP, SIMD, vector, multimedia, data-level parallelism

This work is supported in part by a gift from Intel Corp., an equipment donation from AMD, and the National Science Foundation under Grant No.CCR-0209198 and EIA-0224453. Ruchira Sasanka was supported by an Intel graduate fellowship.

Authors' addresses: Ruchira Sasanka, Man-Lap Li, and Sarita V. Adve, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801; email: ruchira.sasanka@intel.com; manlapli@uiuc.edu; sadve@uiuc.edu; Yen-Kuang Chen and Eric Debes, Intel Corporation, Santa Clara, California 95052; email: yen-kuang.chen@intel.com; eric.debes@intel.com.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org. © 2007 ACM 1544-3566/2007/03-ART3 \$5.00 DOI 10.1145/1216544.1216546 http://doi.acm.org/ 10.1145/1216544.1216546

ACM Reference Format:

Sasanka, R., Li, M.-L., Adve, S. V., Chen, Y.-K., and Debes, E. 2007. ALP: Efficient support for all levels of parallelism for complex media applications. ACM Trans. Architec. Code Optim. 4, 1, Article 3 (March 2007), 30 pages. DOI = 10.1145/1216544.1216546 http://doi.acm.org/10.1145/1216544.1216546.

1. INTRODUCTION

Real-time complex media applications such as high-quality and high-resolution video encoding/conferencing/editing, face/image/speech recognition, and image synthesis like ray tracing are becoming increasingly common on general-purpose systems such as desktop, laptop, and handheld computers. General-purpose processors (GPPs) are becoming more popular for these applications because of the growing realization that programmability is important for this application domain as well, as a result of a wide range of multimedia standards and proprietary solutions [Diefendorff and Dubey 1997]. However, real-time execution of such complex media applications needs a considerable amount of processing power that often surpasses the capabilities of current superscalar processors. Further, high-performance processors are often constrained by power and energy consumption, especially in the mobile systems where media applications have become popular.

This paper seeks to develop general-purpose processors that can meet the performance demands of future media applications in an energy-efficient way, *while also continuing to work well on other common workloads* for desktop, laptop, and handheld systems. In particular, we do not consider dedicated solutions such as digital signal processors (DSPs), application specific integrated circuits (ASICs), or media processors for this work.

Fortunately, most media applications have a lot of parallelism that can be exploited for energy-efficient high-performance designs. The conventional wisdom has been that this parallelism is in the form of large amounts of data-level parallelism (DLP). Therefore, many recent architectures have targeted such DLP in various ways, e.g., Imagine [Ahn et al. 2004], SCALE [Krashinsky et al. 2004], VIRAM [Kozyrakis 2002], and CODE [Kozyrakis and Patterson 2003]. Most evaluations of these architectures, however, are based on small kernels, e.g., speech codecs, such as adpcm, color conversion, such as rgb2cmyk, and filters, such as fir. Further, dedicated solutions for large amounts of DLP require a considerable area overhead, without any benefit to applications that do not have such DLP. For example, the Tarantula vector unit has the same area as its scalar core [Espasa et al. 2002], but likely will be underutilized for many applications that run on GPPs.

This paper differs from the above works in one or both of the following important ways. First, to motivate and evaluate our work, we use more complex applications from our recently released ALPBench benchmark suite [Li et al. 2005]. These applications are face recognition, speech recognition, ray tracing, and video encoding and decoding. They cover a wide spectrum of media processing, including image, speech, graphics, and video processing. Second, because of our focus on GPPs, we impose the following constraints and assumptions on

our work: (1) GPPs already exploit some DLP through subword SIMD instructions, such as MMX/SSE (subsequently referred to as SIMD), (2) GPPs already exploit instruction- and thread-level parallelism (ILP and TLP), respectively, through superscalar cores and through chip multiprocessing (CMP) and simultaneous multithreading (SMT), and (3) radical changes and overhead in the hardware and programming model are not acceptable for well-established GPPs. Motivated by the properties of our applications and the above constraints and assumptions, we propose a complete architecture called ALP.

Specifically, we make the following five observations through our study of complex applications.

- *All levels of parallelism.* As reported by others, we also find DLP in the kernels of our applications. However, as also discussed in Li et al. [2005], many large portions of our applications lack DLP and only exhibit ILP and TLP (e.g., Huffman coding in MPEG encode and ray tracing).
- *Small-grain DLP*. Many applications have small-grain DLP (short vectors) because of the use of packed (SIMD) data and new intelligent algorithms used to reduce computation. Packed data reduces the number of elements (words) to be processed. New intelligent algorithms introduce data-dependent control, again reducing the granularity of DLP. For example, older MPEG encoders performed a full-motion search comparing each macroblock from a reference frame to all macroblocks within a surrounding region in a previous frame, exposing a large amount of DLP. Recent advanced algorithms significantly reduce the number of macroblock comparisons by predicting the "best" macroblocks to compare. This prediction is based on the results of prior searches, introducing data-dependent control between macroblock computations and reducing the granularity of DLP.
- *Dense representation and regular access patterns within vectors.* Our applications use dense data structures, such as arrays, which are traversed sequentially or with constant strides, in most cases.
- *Reductions*. DLP computations are often followed by reductions, which are less amenable to conventional DLP techniques (e.g., vectorization), but become significant with the reduced granularity of DLP. For example, when processing blocks (macroblocks) in MPEG using 16B packed words, reductions occur every 8 (16) words of DLP computation.
- *High memory to computation ratio*. DLP loops are often short with little computation per memory access.
- *Multiple forms of DLP*. Our applications exhibit DLP in the form of SIMD, short vectors, long streams, vectors, and streams of SIMD.

To effectively support all levels of parallelism (CALP) exhibited by our applications in the context of current GPP trends, ALP is based on a GPP with CMP, SMT, and SIMD. The most novel part of ALP is a technique called *SIMD* vectors (SVectors) and SIMD streams (SStreams) that support larger amounts of DLP than possible with SIMD. SVectors/SStreams use an evolutionary programming model and can be implemented with modest additional hardware support that is tightly integrated within a modern superscalar pipeline.

The programming model for SVectors lies between SIMD and conventional vectors. SVectors exploit the regular data access patterns that are the hallmark of DLP by providing support for conventional vector *memory* instructions. They differ from conventional vectors in that computation on vector data is performed by existing SIMD instructions. Each architectural SVector register is associated with an internal hardware register that indicates the "current" element of the SVector. A SIMD instruction specifying an SVector register as an operand accesses and autoincrements the current element of that register. Thus, a loop containing a SIMD instruction. SStreams are similar to SVectors except that they may have unbounded length.

Our choice of supporting vector/stream *data*, but not vector/stream *computation*, exploits a significant part of the benefits of vectors/streams for our applications, but without need for dedicated vector/stream compute units. Specifically, ALP largely exploits existing storage and data paths in conventional super-scalar systems and does not need any new special-purpose structures. ALP reconfigures part of the L1 data cache to provide a vector register file when needed (e.g., using reconfigurable cache techniques [Albonesi 1999; Ranganathan et al. 2000]). Data paths between this reconfigured register file and SIMD units already exist, since they are needed to forward data from cache loads into the computation units. These attributes are important given our target is GPPs that have traditionally resisted application-specific special-purpose support.

Our evaluations show that our design decisions in ALP are effective. Relative to a single-thread superscalar without SIMD, for our application suite, ALP achieves aggregate speedups from 5 to 56X, energy reduction from 1.7 to 17.2X, and energy-delay product (EDP) reduction of 8.4 to 970.4X. These results include benefits from a four-way CMP, two-way SMT, SIMD, and SVectors/SStreams. Our detailed results show significant benefits from each of these mechanisms. Specifically, for applications with DLP, adding SVector/SStream support to a system with all the other enhancements in ALP achieves speedups of 1.1 to 3.4X, energy savings of 1.1 to 1.5X, and an EDP improvement of 1.1 to 5.1X (harmonic mean of 1.7X). These benefits are particularly significant given that the system compared already supports ILP, SIMD, and TLP; SVectors/SStreams require a relatively small amount of hardware; the evaluations consider complete applications.

More broadly, our results show that conventional GPPs can be augmented to support complex media applications more efficiently using evolutionary architectural enhancements and simple extensions to the existing programming model.

2. THE ALP PROGRAMMING MODEL

ALP supports conventional threads for TLP. ILP is not exposed to the programmer since ALP uses an out-of-order superscalar core as in current GPPs. The SIMD programming model roughly emulates Intel's MMX/SSE2 with multiple 8, 16, 32, or 64 bit subwords within a 128-bit word and with 8 SIMD logical registers. Most common opcodes are supported, e.g., packed addition, subtraction,

ALP: Efficient Support for All Levels of Parallelism • 5



Fig. 1. An SVR consists of records, a record consists of packed words, and a packed word consists of subwords.

multiplication, absolute difference, average, logical, and pack/unpack operations. SIMD operations use the FP register file and FP units. We next describe the novel SVectors and SStreams programming model.

2.1 SIMD Vectors (SVectors)

SVectors are built on three key enhancements to SIMD support:

2.1.1 *SIMD Vector Registers (SVRs).* These registers hold a sequence of *records*, where each record itself is a sequence of (possibly strided) *packed words* and each one may contain multiple (contiguous) subwords (see Figure 1). Unlike a conventional vector, the records of an SVR can be individually accessed with an index, called the *current record pointer (CRP)*. An SVR is allocated on demand and can have a variable length up to a given maximum.

2.1.2 SVector Allocate (VALLOC) and SVector Load (VLD) Instructions. VALLOC and VLD allocate an SVR. VLD, in addition, loads a (possibly strided) sequence of packed words into the SVR from memory. A slight variation of VAL-LOC, called VALLOCst, allocates an SVR, whose records are flushed to memory as they are written. All of these instructions reset the CRP of the SVR. These are the only special SVector instructions in the ALP ISA.

2.1.3 SIMD Instructions Capable of Accessing SVRs. All computation on SVRs is performed using SIMD instructions, which can directly access an individual record of an SVR. Such an instruction specifies an SVR as an operand, which implicitly accesses the record of the SVR pointed to by its CRP *and also increments the CRP*. Thus, a dynamic sequence of SIMD instructions specifying a given SVR will access successive records of the SVR.

The ALP ISA supports 8 *logical* SVRs, V0–V7, with a record size of 128 bits and subword sizes of 8, 16, 32, and 64 bits. Associated with each logical SVR is an internal SVR descriptor register. This descriptor register stores pertinent information about the SVR, including the CRP. A VLD, VALLOC, or VALLOCst instruction must be used to explicitly allocate an SVR before any SIMD instruction can access it. These vector instructions specify the length of the SVector. The VLD and VALLOCst instructions also specify the organization of the SVector in memory, including the base memory address of the SVector, the stride between two packed words in the SVector, and the size a packed word (hence number of packed words per 128 b record). All of this information is stored in the associated SVR descriptor.

```
// Arrays for vectors V0, V1, and V2
float arr0[length], arr1[length], arr2[length];
for (int i=0; i < length; i++) {
 arr2[i] = k * (arr0[i]+arr1[i]) - 16;
ŀ
                        (a)
float arr0[length], arr1[length], arr2[length];
// num_rec = length * sizeof(float) / 16
                                                 // # of 16B records
// r0 = arr0
                                                 // address of arr0
// r1 = arr1
                                                 // address of arr1
// r2 = arr2
                                                 // address of arr2
// r3 = num_rec
asm {
   /*(1)*/ VLD.float, r3, r0, V0
                                          // loads VO
   /*(2)*/ VLD.float, r3, r1, V1
                                          // loads V1
   /*(3)*/ VALLOCst.float, r3, r2, V2
                                          // allocates V2
for (int i=0: i < num rec: i++) {</pre>
                                                 // SIMD loop
 asm {
   /*(4)*/
             simd_add.float V0, V1, simd_reg0
   /*(5)*/
             simd_mul.float simd_reg0, simd_reg1, simd_reg2
   /*(6)*/
             simd_sub.float simd_reg2, #16, V2
  }
}
                        (b)
```

Fig. 2. (a) C code and (b) SVector assembly enhanced C code, for vector computation V2 = k * (V0 + V1) - 16. k is stored in simd_reg1.

As an example, Figure 2 gives code for a vector computation V2 = k *(V0 + V1) - 16, where V0, V1, and V2 are SVRs, and k is a constant stored in simd_reg1. Part (a) of the figure gives C code for the computation and part (b) gives C code with embedded SVector assembly. The first two assembly instructions, (1) and (2), load the two source SVectors from memory. For this example, the size of a packed word is a float and the stride between packed words is zero. The number of 16 B (128 b) records and the load address are specified in general-purpose registers whereas the stride between packed words is specified in an implicit stride register. The next instruction, VALLOCst, allocates a new SVR for writing V2. The first three assembly instructions implicitly reset the CRP of V0, V1, and V2, respectively. Next, a loop called a SIMD loop is used to traverse the records of the SVectors. This loop contains SIMD instructions that directly read/write the SVRs, accessing the record currently pointed by the corresponding CRP and incrementing this CRP. Each occurrence of instruction (4), therefore, reads from the next record of V0 and V1 and each occurrence of instruction (6) writes to the next record of V2 (and also stores that record to memory, since V2 is allocated with a VALLOCst).

ALP also provides an instruction, ClearCRP, to reset the CRP of the specified SVR, and an instruction, MoveRec, to read a specific SVR record into a SIMD register. ClearCRP is used if an SVR needs to be read again after it has already been traversed once with SIMD instructions, e.g., to reuse a quantization table in MPEG. MoveRec is used to provide random read access into records, e.g., MoveRec V0, #4, simd_reg4 moves record V0[CRP+4] to simd_reg4.

ALP requires that an SVector/SStream be traversed sequentially. If a record needs to be skipped, it must be read and discarded to increment the CRP. Alternatively, it is possible to provide an instruction to increment the CRP by a given number of records; however, our applications do not exhibit such a requirement.

ALP does not support scatter/gather operations on SVectors since our applications do not exhibit memory access patterns that would benefit from such operations.

ALP imposes three implementation-driven ISA restrictions. The first two arise because ALP implements SVRs by reconfiguring part of the L1 data cache to allocate SVR space on demand (Section 3). First, the maximum SVector length allowed in ALP is related to the L1 size and the number of SVRs supported. An SVector length of 32 records (512 B) sufficed for our applications and fit comfortably in our L1 cache (except for FaceRec that uses SStreams). Second, because SVRs are allocated on demand, clearly, an SVR cannot be read unless it is explicitly allocated using a VLD, VALLOC, or VALLOCst. Third, the out-of-order ALP implementation uses conventional renaming to avoid stalls as a result of WAR and WAW hazards even for SVectors. A problem with this is that the renaming occurs at the granularity of the full SVR, at the vector load and allocate instructions. However, the SIMD writes occur at the granularity of individual records. We, therefore, impose a programming restriction that requires a VALLOC instruction before a vector record is overwritten by a SIMD instruction. This instruction indicates to the hardware that a new renamed copy of the vector must be allocated for subsequent SIMD writes of this logical vector. In our applications, writing to SVRs is infrequent, so this restriction has little impact.

2.2 SIMD Stream (SStreams)

An SStream is essentially a long SVector that (1) exceeds the maximum length of an SVR, and (2) must be accessed strictly sequentially. Conventional vector processors would require strip-mining for loops containing such long vectors. Instead, we support two special stream load (SLD) and stream allocate (SALLOC) instructions. These instructions are similar to VLD and VALLOCst, respectively, in that they both allocate the specified SVR. Transparent to the programmer, however, the underlying hardware allocates an SVR size that is smaller than the stream size, and manages it like a FIFO queue—when the program reads a record from the head, it is discarded and a new record is automatically appended to the tail (Section 3.3). An exception incurred by the load of such a record is handled at the instruction that will consume the record (Section 3.3).

Like SVectors, computation on SStreams occurs with SIMD instructions. For instance, to perform the computation in Figure 2 on two streams and produce a resulting stream, we need only change VLD to SLD and VALLOCst to SALLOC. Unlike SVectors, ClearCRP and MoveRec are not supported for streams, since streams are accessed sequentially (to simplify hardware management of the SVR space).

Note that it is possible to replace SStreams with SVectors by strip mining long loops. However, SVectors may not be as effective as SStreams for hiding memory latency (Section 6). This is because SVector loads have to be explicitly scheduled for maximal latency hiding whereas hardware automatically schedules record loads well in advance for SStreams.

2.3 SVectors/SStreams versus SIMD

This section qualitatively describes the performance and energy benefits of SVectors and SStreams over a pure SIMD ISA (e.g., MMX or SSE2). Differences from conventional vectors are discussed in Section 7. Not surprisingly, some of these benefits are similar to those from conventional vectors [Asanovic 1998; Corbal et al. 1999; Hennessy and Patterson 2002]. Section 7 elaborates on the differences between SVectors and conventional vectors.

2.3.1 Performance Benefits

1. Reduced load/store and overhead instructions: SVectors/SStreams reduce instruction count in two ways. First, VLD/SLD and VALLOCst/SALLOC reduce instruction count by replacing multiple loads and stores with one instruction and eliminating the corresponding address arithmetic overhead instructions.

Second, SVRs reduce loads/stores and associated overhead instructions because of increased register locality. The SVRs increase the register space available that can be directly accessed by compute instructions, reducing loads/stores resulting from register spills. For instance, MPGenc and MPGdec repeatedly use quantization/coefficient tables—each table in DCT/IDCT has 32 records. A pure SIMD system repeatedly spills and loads entries of these tables from and into the small number of SIMD registers. With SVRs, these tables are loaded only once and then directly accessed by the SIMD compute instructions for as long as they are needed.

A simple expansion of the SIMD register file is not as effective because (1) it would need a larger instruction width to encode the larger register space and (2) a single large register file is energy inefficient and this price would be paid for *all* SIMD instructions. SVectors mitigate problem (1) by exploiting the regular nature of vector data to access them through an implicit index (the CRP)—this requires encoding only the SVR in the instruction since the CRP is implicit. They mitigate problem (2) by splitting the register space into the smaller (and so more energy efficient) SIMD register file and the larger (less energy efficient) SVR. The more efficient SIMD file stores temporary values from intermediate computation, making the most effective use of that space. The less efficient, larger SVR file primarily stores the large amounts of SVector data directly loaded from memory, reducing pollution of the SIMD file.

2. Increased exposed parallelism and decreased contention from reduced instruction count: The reduction of memory/overhead instruction count in frequently used loops allows more loop iterations to fit in the processor's instruction window. This allows hardware to extract more parallelism and hide latencies of compute instructions. In short loops, instruction

count reduction can be as high as 50%, allowing twice as many compute instructions in flight (e.g., in our face recognition application). Further, the reduction of memory/overhead instructions also reduces contention to critical resources like register files and issue ports.

3. Load latency tolerance: SVectors/SStreams allow more aggressive use of pipelined loads, without limits of register pressure. On an SVector/SStream load, the constituent loads of individual records are pipelined with each other and with the iterations of the corresponding SIMD computation loop. Further, SVector/SStream loads that can be predicted in advance can also be hoisted well before the corresponding SIMD computation loops.

The above benefit from SVector/SStream loads is similar to that from using (hardware or software) prefetching, but is more effective than the latter for the following reasons. First, SVector/SStream loads eliminate many load instructions; prefetching does not have this benefit and software prefetching requires additional instructions for the prefetches and address calculation. Second, SVector/SStream loads only bring the data required, whereas prefetchers bring entire cache lines, potentially polluting the cache. Third, prefetching needs to be carefully scheduled; otherwise, it can evict useful data. Prefetches into separate buffers have been recommended to avoid this problem, but such buffers must be exposed to the cache coherence protocol. Finally, for short vectors such as 16×16 or 8×8 blocks seen in MPEG, there may not be enough time for a hardware prefetcher to effectively learn the pattern [Holliman and Chen 2003]. Section 6 discusses experimental results that show that ALP's benefits exceed well beyond those for prefetching.

- 4. L1 cache space and bandwidth savings because of packed data: SVRs contain packed and aligned data. In contrast, a cache line loaded to L1 using a SIMD load may contain useless data.
- 5. Eliminating record alignment in L1: In many cases, 16-byte SIMD records are not aligned at 16-byte boundaries in memory. SIMD instruction sets like SSE provide special unaligned load instructions to load SIMD data starting at unaligned addresses. Such instructions have higher latency than normal loads. This latency has to be paid each time data is loaded from L1. With SVectors/SStreams, the extra latency for alignment has to be paid only at the time of loading an SVector from L2. Accesses to SVRs do not require any alignment. Further, since this alignment is done in L2 as part of a vector load that is performed in parallel with the computation, it is often possible to remove this additional latency from the critical path.

2.3.2 *Energy Benefits.* SVectors/SStreams provide energy benefits over pure SIMD in the following ways. First, the performance benefits above reduce execution time without a commensurate increase in power, thereby reducing energy. Second, an SVR access is more energy efficient than a usual cache access that it replaces. This is because a load/store requires accessing the TLB and all tag and data arrays in a bank. SVR accesses do not perform TLB and tag accesses at all and access only the cache way where the SVR resides. Finally, it is possible to use the performance benefits of SVectors/SStreams to save even



Fig. 3. Integer, FP/SIMD, and L1 partitions/banks. (a) *Overview*. The integer and FP/SIMD execution units and register files consist of two partitions (upper and lower int or FP/SIMD execution partitions). The L1 cache consists of four banks. Each execution partition connects to all the four L1 banks—two 16-B buses connect the upper partitions and another two connect the lower ones. This enables each of the two SMT threads to perform up to two memory operations per cycle. The shaded regions in the cache banks show SVRs. (b) *Integer execution partitions (32 b wide)*. Integer units in the upper (lower) partition can only read/write the upper (lower) partition of the register file, except for the shaded units which can access both partitions. (c) *FP/SIMD execution partitions (128 b wide)*. Similar to int, only the shaded units can access both register file partitions.

more energy by running at a lower frequency and voltage, but we do not exploit this benefit here.

3. ALP IMPLEMENTATION

3.1 Support for ILP, TLP, and SIMD

ALP's support for ILP, TLP, and SIMD is conventional. As in Section 2, the SIMD implementation is roughly based on that of Intel's MMX/SSE2. Based on a previous study on the best combination of ILP and TLP for multimedia applications [Sasanka et al. 2004] and current GPP trends, ALP implements a CMP with four 4-wide out-of-order cores with two SMT threads per core. Each core has a private L1 instruction cache and a private writethrough L1 data cache. All cores logically share a unified writeback L2 cache. The L1 caches are kept coherent with a writethrough invalidate protocol.

To ensure that the baseline core is energy efficient, almost all processor resources are partitioned and caches are banked. Figure 3 illustrates the partitioning/banking for some resources. When both SMT threads run, each thread has exclusive access to one-half the partitions for most resources (e.g., reorder buffer/retirement logic, load/store queue). Notable exceptions are the caches, TLBs, and a few execution units (Figure 3)—these are physically partitioned, but logically shared among both threads as in a typical SMT design.

The L2 cache is logically shared among all four cores. It is physically divided into four banks (each with four subbanks) connected with a crossbar. Each processor has one L2 bank closest to it called its *home bank*. There is a dedicated connection between a processor's L1 and its home L2 bank.

	Value Per	# of
Parameter	Partition	Partitions
Phy int reg file (32 b)	64 regs, 5R/4W	2
Phy FP/SIMD reg file (128 b)	32 regs, 4R/4W	2
Int issue queue	-	2
# of entries	24	
# of R/W ports	3R/4W	
# of Tag R/W ports	6R/3W	
Max issue width	3	
FP/SIMD issue queue		2
# of entries	24	
# of R/W ports	3R/4W	
# of tag R/W ports	5R/3W	
Max issue width	3	
Load/store queue		2
# of entries	16	
# of R/W ports	2R/2W	
Max issue width	2	
Branch predictor (gselect)	2KB	2
SVector descriptors	12	2
Integer ALUs (32 b)	See Fig. 3	2
FP SIMD units (128 b)	See Fig. 3	2
Int SIMD units (128 b)	See Fig. 3	2
Reorder buffer	32 ent, 2R/2W	4
Retire width	2	
Rename width	4 per thread	2
Max. fetch/decode width	6 (max 4 per three	ead)

Table I. Base Architecture Parameters a

Parameter	Value Per Bank	# Banks
L1 I-cache	8 K, 4-Way, 32-B line, 1 port	2
L1 D-cache	8 K, 2-Way, 32-B line, 1 port	4
(writethrough)		
L2 cache	256 K, 16-Way, 64-B line, 1 port	4
(writeback, unified)		

Bandwidth and Contentionless Latencies @ 500 MHz							
Parameter	Value (cycles @ 500 MHz)						
ALU/int SIMD latency	8 (div-32 b), 2 (mult-32 b), 1 (other)						
FP/FP SIMD latency	12 (div), 4 (other)						
L1 I-cache hit latency	1						
L1 D-cache/SVR hit latency	1						
L2 cache latency	10 (hit), 42 (miss)						
Memory bandwidth	16 GB/s						

 a Note that several parameter values are *per partition or bank*. Section 6.1 reports some sensitivity results.

Table I provides the specific parameters used in our experiments. These choices were made to provide reasonable size/ports and reduced energy/cycle time for each structure. The processor frequency is a relatively low 500-MHz (in 90-nm technology) since ALP is targeted toward energy efficiency. We can also interpret this frequency as a low-frequency setting for a higher-frequency processor with dynamic voltage/frequency scaling. We expect our qualitative

ACM Transactions on Architecture and Code Optimization, Vol. 4, No. 1, Article 3, Publication date: March 2007.

11



Fig. 4. SVRs and SVector descriptor registers. Shaded cache lines contain SVRs, whereas unshaded ones comprise normal cache data. Start line, CRP, and last available record are relative to the start of the cache.

results to hold with a wide range of parameter choices representative of modern superscalar GPPs. Section 6.1 reports some sensitivity results (limited for space reasons).

3.2 Support for SIMD Vectors

3.2.1 *Modifications to the Caches.* SVRs are allocated in the L1 data cache (Figures 3 and 4). Thread 0 allocates even-numbered SVectors in bank 0 and odd-numbered SVectors in bank 2 of the L1. Thread 1 allocates odd and even SVectors in banks 1 and 3, respectively. This allows each thread to access one record from each of two SVectors in a cycle. Although each cache bank has multiple ways, we currently allocate SVRs only in way 0.

Reconfiguring lines of a cache bank into an SVR is quite simple [Albonesi 1999; Ranganathan et al. 2000]. One additional bit (SVR bit) per cache line is needed to indicate that it is part of an SVR. Since the L1 cache is writethrough, reconfiguration of a cache line into part of an SVR simply requires the above bit to be set; no cache scrubbing is required. Further, when an SVR is allocated, it is not necessary to set the SVR bits of all the allocated cache lines simultaneously. Instead, the SVR bit for an allocated cache line is set as it is loaded from the L2. An additional decoder to decode the SVR location is also not necessary, since caches already have such a decoder to decode the cache line address. A multiplexer (or an additional input to an existing one) is necessary to drive the input of the cache line decoder since now there is one additional input (the CRP of a SIMD instruction). The SVector records traveling from an SVR to execution units use the existing forwarding paths used by usual SIMD loads. Thus, the L1 cache requires only minor modifications to support SVRs.

We note that since results of intermediate computations are stored in SIMD registers, SIMD instructions typically do not access SVectors for all three

operands. This reduces the L1 cache bandwidth required to support multiple SIMD instructions per cycle. The two L1 buses per partition already provided (Figure 3) are sufficient to feed two SIMD instructions accessing two SVectors each in a cycle. It should be noted that the use of SIMD registers for temporaries makes it practically possible to allocate SVRs in the L1 cache. A traditional vector ISA requiring all vector instructions to use the vector register file will make it difficult to allocate the vector registers in the L1 cache because of the higher number of register ports required.

The L2 cache requires more support than the L1. SVector loads are sent to the requesting processor's home L2 cache bank. This bank then sends requests for the packed words constituting the SVector to other banks as needed (recall that an SVector load may specify a stride between packed words). Each bank inserts such requests in its wait queue and services the requests in order (in parallel with the other banks). When the data is available, the bank sends it to the home bank (across the crossbar). It should be possible for individual banks to access words starting at any byte location (i.e., to perform unaligned loads). This capability is generally found in caches to access individual words for writing, on architectures that support unaligned accesses. The home bank assembles two 16-B records into a 32-B L1 cache line and sends these to the SVR in the L1. Each L2 bank contains separate buffers for packing records to cache lines. Note that entire L2 cache lines are not transmitted to L1 and only the records required are assembled and sent, thereby saving bandwidth and energy. The connection between the L1 and L2 can support one 32-B cache line (two 16-B records) per cycle.

3.2.2 Modifications to the Rest of the Core. We model an out-of-order pipeline with eight stages: fetch, decode, rename, issue, operand-read, execute, writeback, and retirement. Fetch and execute do not need any modification; decode needs small modifications to decode a handful of new instructions; and operand-read and retire stages need straightforward enhancements to read from/write to SVRs. The following discusses the modifications to rename, issue/scheduling, and retirement, and the handling of speculation and precise exceptions.

3.2.2.1 *Rename Stage*. The rename stage of an SVector load or allocate instruction allocates an SVR and an SVector descriptor corresponding to the destination logical SVR. The logical SVector to physical descriptor mapping is stored in a rename table. The SVector descriptor register (see Figure 4) contains two fields (not programmer visible) in addition to the CRP and those initialized from the SVector instruction discussed in Section 2: (1) Start line specifies the cache index address of the first record of the SVR, and (2) last available record specifies the absolute record number of the last record that has been produced (loaded/written) so far. The last available record and CRP fields store the absolute record number relative to the start of the cache bank.

The allocation process for an SVR is analogous to that for a scalar register, requiring maintaining a free list of available space. However, an SVR requires allocation of a sequence of cache lines. One simple way to achieve this is to logically divide a cache bank into N equal-sized segments, where N is determined

by the number of physical registers that can be allocated in that bank. This fixes the number of SVRs and the start line of each SVR in a cache bank. Now a free bit can be maintained for each such logical segment to indicate whether it is free or allocated.

When a SIMD instruction with an SVector operand is renamed, the CRP field from the corresponding SVector descriptor is read to provide the location of the operand. The CRP is then incremented for the next SIMD instruction to the same SVector. Thus, the CRP is accessed and updated only in the in-order part of the pipeline, avoiding any RAW, WAW, or WAR hazards on it. Similarly, the ClearCRP instruction also performs its CRP update in the rename stage.

3.2.2.2 *Issue and Scheduling.* Only minor changes are necessary for the issue and scheduling logic. For a SIMD instruction that reads an SVR record, the availability of the record is marked in a ready bit as done for a normal register source operand. An SVR record is known to be available if the CRP of the SVector descriptor is less than or equal to the last available record of the same descriptor.

If the required record is not yet available, the SIMD instruction awaits its availability in the issue queue just like other instructions waiting for their operands. When the required record arrives in the SVR, the cache sends a message to the rename stage to update the last available record field for that SVR. At the same time, the cache index plus bank number is passed as an 8-bit tag along a wakeup tag port of the issue queue (along the same tag ports used for passing an 8-bit register identifier when a normal load completes) and compared against the same information carried in the decoded instruction. On the tag match, the waiting SIMD instruction sets its operand ready bit and is ready for issue if both its operands are ready. If an instruction reads from two vectors mapped to the same L1 bank, the instruction has to be stalled in the read-operand stage until both operands are read. However, this condition can be often avoided in the code itself by using vectors that map to different banks (i.e., odd and even vectors map to different banks).

For memory disambiguation and conflict resolution, the load/store queue receives VLD instructions and SIMD instructions that write to an SVector allocated with VALLOCst. Such an instruction may access several possibly strided packed words—we conservatively assume that it accesses the entire memory range from the address of the first to the last packed word for resolving conflicts. Support for detecting conflicts among accesses with different address ranges already exists, e.g., conflicts between regular SIMD loads/stores spanning 16 consecutive bytes and other FP/integer loads spanning fewer bytes.

3.2.2.3 *Retirement.* For SVector load and allocate instructions, the retirement stage frees SVRs similar to the freeing of renamed registers for ordinary instructions, i.e., the physical register that used to map to the destination logical register of the retired instruction is freed. In addition, the SVR bits of the corresponding L1 cache lines are reset. Since the start lines and the maximum number of cache lines for SVRs are predetermined, simple circuitry can be used to reset all SVR bits of an SVR in a cycle. An SVR is also freed when the thread

that created it is killed. ALP also provides a special instruction to explicitly free all SVRs, which can be used when the subsequent code does not use SVectors. As for ordinary stores, storing records of a vector to memory also occurs at retirement.

3.2.2.4 *Speculation*. To rollback modifications to SVR-related resources by mispredicted instructions, ALP uses the conventional combination of renaming and checkpointing. Specifically, on a branch, ALP checkpoints the rename map table for SVectors and the CRP values (analogous to checkpointing integer/FP rename tables).

3.2.2.5 *Precise Exceptions.* Precise exceptions are largely handled through register renaming and in-order retirement, as with current GPPs, with three additions. First, on an exception, the (currently allocated) SVRs need to be saved. Second, exceptions within VLD can be handled as in CODE by allowing the VLD to be restarted with partial completion [Kozyrakis and Patterson 2003]. Third, as discussed for retirement, for SIMD instructions that write to memory, the memory update is done only at retirement, after examining for exceptions. In case a memory write because of such an instruction needs to modify multiple packed words and there is a TLB miss/page fault on one of them, again, partial completion, as in Kozyrakis and Patterson [2003], can be used.

3.3 Support for SIMD Streams

SStreams are implemented similar to SVectors, with the following additional support. For an SStream, an SVR is managed as a FIFO circular buffer with a head and a tail (LastAvailRec). An SStream load (SLD) initially brings data to fill an entire SVR (up to 40 records in the current implementation). When a SIMD instruction reading from (or writing to) the head of an SStream retires, the retirement logic checks if the record involved is at the end of an L1 cache line. In that case, the cache line is evicted from the SVR and a load request is sent to the L2 to bring in the next records that need to be appended to the tail (or a store request is sent to write the records at the head). Since new records are appended to the tail before they are referenced (at the head), the processor can usually read records without stalling.

Since an SVR cannot hold an entire SStream, an SStream load (SLD) is allowed to retire before the entire SStream is loaded. If the load of a subsequent record later incurs an exception, the record number is stored in a special exception field in the corresponding SVector descriptor. The exception is taken at the next instruction that refers to the record.

4. APPLICATIONS AND PARALLELISM

We used five complex media applications available in the ALPBench benchmark suite [Li et al. 2005]: MSSG MPEG-2 encoder and decoder [MPEG Software Simulation Group 1994] (MPGenc and MPGdec), Tachyon ray tracer [Stone 2003] (RayTrace), Sphinx-3 speech recognizer [Reddy et al. 2001] (SpeechRec), and CSU face recognizer [Beveridge and Draper 2003] (FaceRec).

	MPGenc	MPGdec	SpeechRec	FaceRec
Sub-word size	1 B, 2 B	1 B, 2 B	4 B (float)	8 B (double)
% SIMD instr.	24	24	17	66
Computation/memory	1.70	1.84	1.43	1.00
DLP granularity	1,8,16	1,4,8	10	9750
(in 128-b words)				(stream)
% Reductions	36	10	28	50
Source lines in C code	8412	10480	15351	11601

Table II. DLP Characteristics of Applications^a

^aThe last row gives the number of lines of C code (without comments).

The applications are modified by hand to extract TLP and DLP. TLP is exploited using POSIX threads. The threads usually share read-only data, requiring little additional synchronization. For DLP, the applications include ALP's SIMD instructions. In addition, we extended this SIMD support with SVector/SStream instructions. MMX style hand-coding is prevalent practice for these applications and the maximum number of static assembly instructions inserted (for MPGenc) is about 400. All applications exploit TLP and ILP. All applications except for RayTrace exploit SIMD and SVectors; only FaceRec exploits SStreams. Detailed descriptions and characterizations of the applications appear in Li et al. [2005].

Table II summarizes the DLP characteristics of our applications, except for Ray Trace, which has only TLP and ILP.

The first row summarizes the subword sizes used by each application. MP-Genc and MPGdec use smaller integer subwords, whereas SpeechRec and FaceRec use larger FP subwords.

The % *SIMD instr.* row gives the percentage of total dynamic instructions that are SIMD in the version of the code with ALP SIMD. The relatively small percentage shows the importance of supporting ILP and TLP for these complex applications.

The *computation/memory* row of Table II gives the ratio of SIMD computation to memory operations (and instructions) in the version of the code with ALP SIMD. All our applications show low computation to memory ratios. This indicates that efficient support for memory operations could lead to significant performance and energy improvements for these applications.

The *DLP granularity* row shows the DLP granularity in 128-b SIMD words (i.e., the iteration count of SIMD loops). All our applications except FaceRec exhibit small-grain DLP (i.e., small vector lengths), while FaceRec exhibits very large-grain DLP (i.e., streams). It may (or may not) be possible to increase the vector lengths with much effort using substantially different algorithms. However, such changes were not obvious to us and, we believe, are beyond the scope of this work. (Some obvious changes substantially increase the amount of work to be done and significantly reduce performance, e.g., using a full-search instead of an intelligent search in MPGenc, using matrix multiplication-based DCT/IDCT in MPGenc/MPGdec instead of an optimized Chen–Wang butterfly algorithm [Wang 1984].)

The % *reductions* row shows the dynamic number of DLP operations that are part of a reduction, as a percentage of the total dynamic DLP compute

operations in the version of the code with ALP SIMD. This relatively high percentage combined with the small DLP granularity underscores the importance of supporting efficient reductions for these applications. Some implementations are not efficient in supporting reductions when the DLP granularity is small (e.g., multilaned vector units). This property indicates that such implementations may be a poor match for these applications.

5. SIMULATION METHODOLOGY

We simulate the following systems, based on Section 3:

- 1T: The base system with only one thread and no SIMD or SVectors/SStreams (Figure 3 and Table I).
- 1T+S: 1T system with SIMD instructions.
- 1T+SV: 1T+S system with SVectors/SStreams.
- 4T, 4T+S, 4T+SV:. Analogous to the above three systems, respectively, but with four cores in each system (four way CMP) and each core running one thread.
- 4x2T, 4x2T+S, 4x2T+SV. Analogous to 4T, 4T+S, 4T+SV respectively, but each core is a two-thread SMT.

We refer to the 1T system as the base and to the others as enhanced. ALP is 4x2T+SV. Note that we keep the total L2 cache size the same in 1T and 4T systems to ensure that the CMP benefits do not come simply from a larger cache size.

To model the above systems, we use an execution-driven cycle-level simulator derived from RSIM [Hughes et al. 2002]. The simulator models the processor and the L1 and L2 cache pipelines in detail, including wrong-path instructions, contention at all resources, and contention and communication overhead of multiple cache banks. The simulator only emulates operating system calls.

Pthread-based C code is translated into binary using the Sun cc 4.2 compiler with options -xO4 -xunroll=4 -xarch=v8plusa. DLP code resides in a separate assembly file, organized as blocks of instructions and simulated using hooks placed in the binary. When the simulator reaches such a hook, it switches to the proper block of DLP instructions in the assembly file.

We integrate Wattch [Brooks et al. 2000] for dynamic power and HotLeakage [Zhang et al. 2003] with temperature models from Skadron et al. [2002] for static power. We assume aggressive clock gating. Most components have 10% ungated circuitry [Brooks et al. 2000]. To favor the base system, we model only 2% ungated power for caches and functional units. Since the base system takes longer to execute a given application, having more ungated circuitry would make it consume more power. Since the exact amount of clock gating is highly implementation-dependent, we made the choices to favor the base system. We model energy for supporting SVectors/SStreams (e.g., for SVRs, vector renamemap tables/descriptors, extra bits in caches).



Fig. 5. Speedup of enhanced systems over the base 1T system for each complete application (ExecutionTimeBase/ExecutionTimeEnhanced).



Fig. 6. Energy consumption and distribution of enhanced systems as a percentage of the energy consumption of the base system.

6. RESULTS

6.1 Overall Results

Figures 5 and 6 and Table III present our high-level results. For each application, they, respectively, provide the execution time speedup achieved over the base system (single-thread superscalar), the energy consumed normalized to the base, and the *improvement* in energy-delay product (EDP) over the base for each system. Each energy bar also shows the distribution of energy among different components. Table V (see later) summarizes the above data by reporting the *harmonic* means of the speedup, energy *improvement*, and EDP improvement for key pairs of systems. For the DLP enhancements (+S and +SV), the means are computed across the applications that use those enhancements

18

Table III. Energy Delay Product (EDP) Improvement Over the Base (1T) System (EDPbase/EDPenhanced)^a

	1T	1T		4T	4T		4x2T	4x2T
Арр	+S	+SV	4T	+S	+SV	4x2T	+S	+SV
MPGenc	58.8	110.4	5.8	328.8	612.1	12.6	550.5	970.4
MPGdec	12.8	14.9	5.9	59.1	68.1	10.6	87.3	97.8
RayTrace	N/A	N/A	5.5	N/A	N/A	8.6	N/A	N/A
SpeechRec	3.3	5.6	4.7	14.7	20.5	6.4	20.9	29.4
FaceRec	2.6	12.7	6.4	15.3	57.5	6.7	15.5	79.2

^aHigher values are better.

Table IV. Instructions-per-Cycle (Operations-per-Cycle) Achieved by All Systems

App	1T	1T+S	1T+SV	4T	4T+S	4T+SV
MPGenc	1.8 (1.8)	2.5(8.5)	2.3 (10.3)	6.3 (6.3)	9 (30.9)	8.3 (37.5)
MPGdec	2.1 (2.1)	2.3 (6.4)	2.4 (6.6)	7.4 (7.4)	6.9 (19.5)	7.2 (20.1)
RayTrace	1.9 (1.9)	N/A	N/A	7.1 (7.1)	N/A	N/A
SpeechRec	1.6 (1.6)	1.5(2.2)	1.8 (2.9)	5.3 (5.3)	4.5 (6.8)	5.1 (8)
FaceRec	1.3 (1.3)	1.3(2.2)	2.2(3.4)	5.2(5.2)	5.3(8.7)	8.1 (12.8)
-			_			
	4x2T	4x2T-	-S = 4x2T	+SV		

MPGenc	10.7 (10.7)	12.8 (44.4)	11.4 (51.5)
MPGdec	11.3 (11.3)	9.2 (25.9)	9.4 (26.2)
RayTrace	9.8 (9.8)	N/A	N/A
SpeechRec	6.6 (6.6)	5.7 (8.6)	6.6 (10.3)
FaceRec	5.3 (5.3)	5.3 (8.7)	10.7 (16.9)

Table V. Mean Speedup, Energy Improvement, and EDP Improvement^a

Benefits of	SIMD			SVectors/SStreams		
Systems	1T+S/	4T+S/	4x2T+S/	1T+SV/	4T+SV/	4x2T+SV/
compared	1T	4T	4x2T	1T+S	4T+S	4x2T+S
Speedup	2.63	2.52	2.41	1.48	1.43	1.46
Energy	2.41	2.29	2.25	1.3	1.21	1.21
EDP	5.06	4.69	4.52	1.83	1.67	1.69

Benefits of	CMP			SMT			ALP
Systems	4T/	4T+S/	4T+SV/	4x2T/	4x2T+S/	4x2T+SV/	4x2T+SV/
compared	1T	1T+S	1T+SV	4T	4T+S	4T+SV	1T
Speedup	2.88	3.38	3.24	1.05	1.23	1.32	9.24
Energy	1.26	1.5	1.38	0.91	1.1	1.11	2.81
EDP	4.52	5.08	4.48	1.18	1.35	1.45	22.69

 a All means are *harmonic* means of the ratio of the less enhanced to the more enhanced system. The means for the DLP enhancements (SIMD and SVectors/SStreams) are over the DLP applications (i.e., all except RayTrace).

(i.e., all except RayTrace); for the others, the means are computed across all applications. For reference, Table IV gives the instructions and operations per cycle (IPC and OPC) for each application and system.

Our data validates our claims that complex media applications demand support for a wide spectrum of parallelism and ALP effectively provides such support. Specifically, all the techniques in ALP are important and effective.

Across all our applications, compared with the base 1T system, ALP, with all the enhancements (i.e., 4x2T+SV), shows a speedup of 5 to 56X (harmonic mean 9.2X), energy improvement of 1.7 to 17.2X (harmonic mean 2.8X), and EDP

improvement of 8.4 to 970X (harmonic mean 22.7X). All comparisons are made at the same voltage/frequency for all systems—ALP's energy savings could be further improved by using dynamic voltage scaling at the cost of some reduction in its performance speedups.

Table V clearly shows each technique in ALP contributes significantly to the above benefits, especially when considering the relative complexity/area overhead of the technique. Specifically, comparing the effect of an enhancement over a system with all the other enhancements, we see that the mean improvement in EDP from adding SIMD instructions to 4x2T is 4.5X, from adding SVector/SStreams to 4x2T+S is 1.7X, from adding four-way CMP to 1T+SV is 4.5X, and from adding SMT to 4T+SV is 1.4X. The means for the DLP enhancements (SIMD and SVector/SStream) are over the DLP applications (i.e., except for RayTrace).

We also performed experiments with the 4x2T+SV system restricted to twowide fetch/decode/retirement (but same issue width and functional units). Compared to this system, the four-wide system reduced execution time from 5 to 22% (mean 12%), validating the need for the ILP support in ALP.

We also increased the L1/SVR hit time from 1 to 4 cycles. The speedup of SVectors/SStreams over SIMD remained within 6%, except that FaceRec saw a 15–20% increase (because of the use of SStreams that successfully hide the higher latency). Similarly, we also experimented with higher processor frequency (i.e., longer memory latencies) and generally found increased benefits for SVectors/SStreams over SIMD (since SVRs reduce the impact of the higher latency).

Since it is possible to emulate SStreams using SVectors by strip mining long loops, we also performed experiments using SVectors, instead of SStreams, for FaceRec, the only application that uses SStreams. However, SVectors are not as effective as SStreams for hiding memory latency. This is because SVector loads have to be explicitly scheduled for maximal latency hiding whereas hardware automatically schedules record loads well in advance for SStreams. As a result, we found that there is a 17% performance degradation with SVectors with respect to SStreams for FaceRec. This benefit from SStreams may appear modest, but comes at a small additional hardware cost.

Finally, as an indication of application-level real-time performance, for each second, ALP supports MPEG2 encoding of 73 DVD resolution frames, MPEG2 decoding of 374 DVD frames, ray tracing of 5 512 \times 512 frames (a scene of a room with 20 objects), recognizing 30 words using a 130-word vocabulary/dictionary (SpeechRec), and recognizing 1451 130 \times 150 images in a 173-image database (FaceRec). Although the above application performance may seem more than currently required in some cases, it is expected that these applications will be run with larger inputs in the future, requiring higher performance (except perhaps for MPEG decode, which shows ample performance even for future larger inputs).

6.2 Analysis of SIMD Vectors/Streams

Section 2.3 qualitatively described the benefits of SVectors/SStreams over SIMD. We next relate our quantitative data to those benefits. We only consider the DLP applications here (i.e., all except RayTrace).

21

MPGenc MPGdec RayTrace SpeechRec FaceRec 1T+S 17 (59) 28 (80) N/A 51(77)58 (95) 1T+SV 11 (52) 27(75)N/A 45 (70) 34(53)

^aThe numbers for other +S (+SV) systems are the same as for 1T+S (1T+SV).



Table VI. Number of Instructions (operations) Retired for 1T+S and 1T+SV Systems as a Percentage of Instructions (Operations) Retired by 1T^a

Fig. 7. Execution time distribution for 1T+S and 1T+SV.

To aid our analysis, Table VI gives the total instructions and operations retired for the 1T+S and 1T+SV systems as a percentage of the base 1T system. (The numbers for the other +S (+SV) systems are the same as for the 1T+S (1T+SV) systems.) Further, Figure 7 shows the different components of execution time in these systems, normalized to the total time of 1T+S. For an out-oforder processor, it is generally difficult to attribute execution time to different components. Following prior work, we use a retirement-centric approach [Pai et al. 1996; Rosenblum et al. 1995]. Let r be the maximum number of instructions that can be retired in a cycle. For a cycle that retires a instructions, we attribute a/r fraction of that cycle as busy, attributing 1/r cycle of busy time to each retiring instruction. We charge the remaining 1 - a/r cycle as stalled, and charge this time to the instruction at the top of the reorder buffer (i.e., the first instruction that could not retire). This technique may appear simplistic, but it provides insight into the reasons for the benefits seen.

We categorize instructions as: Vector memory (VecMem) (only for 1T+SV), SIMD memory (SimdMem), SIMD ALU (SimdALU), and all others. In Figure 7, the lower part of the bars shows the busy time divided into the above categories, while the upper part shows the stall components. The busy time for a category is directly proportional to the number of instructions retired in that category. We also note that the "other" category includes overhead instructions for address generation for SimdMem instructions and SIMD loop branches; therefore, the time spent in the DLP part of the application exceeds that shown by the Simd category of instructions. The figure shows that the benefits of

SVectors/SStreams arise from the following:

• **Reduction in busy time** occurs due to the reduction in instruction count from SimdMem and related overhead (Other) instructions (Table VI and benefit 1 in Section 2.3).

Eliminating SIMD loads should eliminate a significant fraction of the total SIMD and associated overhead instructions for our applications because of the low SIMD computation to memory ratio (Table II). This effect can be clearly seen in MPGenc and FaceRec.

In SpeechRec, as a fraction of total instructions, the SIMD instructions are small. Nevertheless, the benefit of reducing SimdMem/overhead instructions (and associated stalls) is large enough that it allows skipping of a precomputation phase.¹ This results in a further reduction of the "other" instructions.

For MPGdec, SVectors could not remove many of the SIMD loads, because it uses an optimized IDCT algorithm with random memory access patterns [Li et al. 2005]). Consequently, SVectors see a limited benefit from the reduction of instruction count. This, in turn, lowers the execution time benefit for SVectors.

In general, we do not expect the SimdALU instruction count to change, since +SV performs the same computations. However, there is a slight increase in SpeechRec because skipping the precomputation phase results in more SIMD computations.

- **Reduction in SimdMem stalls** is given by the difference between Simd-Mem stalls in +S and VecMem stalls (plus SimdMem stalls, if any) in +SV. The benefit occurs, because of the reduction in SimdMem instructions and increased load latency tolerance (benefits 1 and 3 in Section 2.3). However, the magnitude of this benefit is quite small for all applications. This is because (1) most memory accesses either hit in the L1 cache or the L2 cache and the out-of-order processor can tolerate these latencies, and (2) the L2 misses that do occur see a relatively low miss penalty, since we model a low-frequency processor.
- **Reduction in SimdALU stalls** is significant specially in FaceRec because (1) a larger number of independent SimdALU instructions fit in the instruction window because of the elimination of intervening SimdMem and overhead instructions (benefit 2 of Section 2.3) and (2) better load latency tolerance results in the ALU instructions obtaining their operands sooner (benefit 3 in Section 2.3). FaceRec, in particular, has two dependent four-cycle FP SimdALU instructions within a SIMD loop iteration, which feed into an FP reduction running through the loop iterations. This incurs a relatively large stall time in 1T+S. In 1T+SV, more of the (mostly independent) iterations fit in the reorder buffer (since about 50% of the instructions per iteration are eliminated), thus, more parallelism can be exploited. SpeechRec sees a slight decrease in SimdALU stall time as a result of the same reasons.

 $^{^{1}}$ The original version of SpeechRec has a precomputation phase to reduce the amount of work done in later phases. This precomputation is omitted for +SV as a result of lack of any benefit.

ACM Transactions on Architecture and Code Optimization, Vol. 4, No. 1, Article 3, Publication date: March 2007.

MPGdec and MPGenc do not have much of a SimdALU stall time to start with because they use integer instructions and also have independent instructions within iterations.

To confirm that not all of the benefits in SimdALU stall time came from load latency tolerance in FaceRec and SpeechRec, we also ran both 1T+S and 1T+SV versions of all applications with a perfect cache where all memory accesses take one cycle. We continued to see benefits in SimdALU stalls (for FaceRec and SpeechRec) and total execution time from +SV (e.g., for FaceRec, 1T+SV showed a speedup of 1.8X over 1T+S with a perfect cache). These experiments also show that techniques, such as prefetching, cannot capture all the benefits of SVectors/SStreams.

It is possible to obtain more exposed parallelism for SIMD (+S) systems using larger resources. Although we already simulate an aggressive processor, we also conducted experiments where we doubled the sizes of the physical SIMD register file, FP/SIMD issue queue, and the reorder buffer. Of all our applications, FaceRec showed the largest speedup with increased resources— 1.67X for the SIMD version. However, SVectors/SStreams (+SV) continued to show significant benefits over SIMD, even with larger resources (1.63X over SIMD for FaceRec) resulting from other sources, such as the reduction of SIMD load instructions and overhead. Thus, SVectors/SStreams can be viewed as a way to achieve the benefits of much larger resources and more, without the higher power consumption and slower processor clock speeds associated with larger resources.

It may be possible to further improve SIMD performance by providing more SIMD logical registers. However, all the loop bodies in our applications, except the large tables, can comfortably fit in the logical SIMD registers provided. Fitting the larger tables would require a much larger register file (e.g., 32 additional SIMD registers for DCT/IDCT coefficient tables). We also note that our out-of-order core already performs dynamic unrolling effectively to use the much larger physical register file, and ALP SIMD already achieves much better performance compared with SSE2 [Li et al. 2005].

- **Reduction in other stalls** results directly from the reduction in overhead instructions described above (most significantly in MPGenc and SpeechRec).
- **Energy benefits** because of SVectors/SStreams come largely from the reduction of instruction count. Comparing corresponding +S and +SV systems in Figure 6, we can see energy reduction in almost every component.

7. RELATED WORK

7.1 Vector Architectures

There is a vast amount of literature on conventional vector architectures and their recent uniprocessor and multiprocessor variations, e.g., VIRAM [Kozyrakis 2002], CODE [Kozyrakis and Patterson 2003], MOM [Corbal et al. 1999], SMT Vectors [Espasa and Valero 1997], NEC SX [Kitagawa et al. 2002], Cray X1 [Cray Inc. 2005], and Hitachi SR [Tamaki et al. 1999]. Some of these systems also focus on integrating a vector unit with a general-purpose

processor, e.g., vector unit with a superscalar core by Quintana et al. [1999], simple vector microprocessors by Lee and Stoodley [1999], out-oforder vectors [Espasa et al. 1997], T0 [Asanovic 1998], and Tarantula [Espasa et al. 2002]. Such systems require investment in a relatively large special-purpose dedicated vector unit (e.g., the Tarantula vector unit is the same size as the four-thread scalar core [Espasa et al. 2002]). In return, they provide excellent performance and energy efficiency for medium- to large-grain regular DLP, e.g., through multilaned implementations.

However, our applications often exhibit small-grain DLP interspersed with control and reductions, and have parts that do not exhibit any DLP at all (Sections 1 and 4). Therefore we chose to explore a lighter weight DLP mechanism, SVectors/SStreams, that could be tightly integrated into expected GPP designs that already have superscalar cores, CMP, SMT, and SIMD. Our results show that the resulting architecture, ALP, is effective in exploiting the different levels of parallelism in complex media applications, and SVectors/SStreams, in particular, show significant benefits over ALP's enhancements. Further, ALP does so primarily using existing data paths and storage with only modest modifications to a conventional superscalar core. Thus, we believe that this paper has demonstrated a valuable design point between pure SIMD and conventional vectors.

Nevertheless, conventional vectors would have the advantage of reduced dynamic compute instructions (ALP uses SIMD compute instructions, which encode only 2 to 16 operations each). To quantify the performance loss resulting from not implementing a separate full-scale vector unit, we compared SVectors with an architecture similar to Tarantula [Espasa et al. 2002] (the most recent proposal for integrating a vector unit with a general-purpose superscalar processor). The detailed comparison is given in Li [2005]. For a fair comparison, we enhanced our Tarantulalike implementation to process multiple packed words in each lane. We call this out-of-order vector architecture VoS (Vector of SIMD). Like Tarantula, VoS is a separate multilaned vector unit and accesses vectors from the L2 cache. We used the same number of computation units and memory bandwidth for both VoS and SVectors. We performed more aggressive unrolling with VoS (than with SVectors) to prefetch vectors before they are used. We ran these experiments at 4 GHz for both SVectors and VoS.

We found SVectors and VoS to be comparable for MPGdec and SpeechRec (SVectors was slightly better for SpeechRec). For MPGenc and FaceRec, VoS provided 1.2 and 3.5X speedups over SVectors, respectively. For MPGenc, most of the benefit comes directly from the reduction in instruction count (SIMD compute and associated overhead instructions). FaceRec, in addition, also sees increased memory level parallelism from VoS vectors. For the SStream version of FaceRec, we used only two source streams. At 4 GHz, these were unable to hide the entire L2 miss latency of 256 cycles. (Recall that once a stream fills an SVR, a request for the next record is sent out only after the head of the SVR is consumed. Thus, the number of outstanding stream loads is limited by the instruction window size of the processor.) VoS, on the other hand, sees more memory level parallelism—because of fewer instructions, increased unrolling, and dynamic vector register renaming, it is able to issue multiple vector load

instructions. Although it is possible to use multiple SStreams to increase memory level parallelism, it would involve nontrivial code transformations, since SStreams still depend on the limited number of SIMD registers for computation. For reference, the performance benefit obtained by SVectors/SStreams over SIMD for these experiments was 1.28X for MPGenc, 1.05X for MPGdec, 1.55X for SpeechRec, and 4.27X for FaceRec.

7.2 Other Architectures

The Imagine architecture [Ahn et al. 2004] and its multiprocessor version, Merrimac [Dally et al. 2003], are also motivated by support for large amounts of DLP, specifically streams. ALP's focus on small-grain DLP and the constraint of integration within a GPP results in significant design differences. Specifically, (1) for computation, Imagine provides ALU clusters that work in lockstep while ALP uses independent SIMD units to exploit ILP and TLP along with fine-grain DLP; (2) Imagine is designed as a coprocessor that depends on a scalar host for irregular scalar computation, while ALP's DLP support is tightly integrated into the superscalar core; and (3) unlike ALP, Imagine needs a substantially new programming model to manipulate streams. ALP and Imagine share similarities in the handling of data—ALP's combination of the SIMD register file and SVRs is analogous to Imagine's storage hierarchy with a local register file for intermediate computation and a stream register file for stream data. For MPEG2 encoding, Imagine reports 138360×288 fps at 200 MHz, but without B frames, half-pixel motion estimation, and Huffman VLC [Ahn et al. 2004]. At the same frequency, ALP achieves comparable performance with B frames, half-pixel motion estimation, and Huffman VLC.

A few architectures like SCALE [Krashinsky et al. 2004], Pseudo Vector Machine (PVM) [Lee 2000], conditional streams [Kapasi et al. 2003] of Imagine, and Titan [Jouppi 1989] cater to fine-grain DLP. SCALE combines TLP and vectors in a concept called vector-thread architectures, which uses a control processor along with a vector of virtual processors. It can exploit DLP interspersed with control; however, it uses a new programming model while ALP extends the established GPP programming model. Thus far, SCALE has been evaluated primarily for kernels; a comparison with ALP on complex applications is, therefore, difficult.

PVM provides support for vector/streamlike processing of loops that are difficult to vectorize. Two source vectors are associated with two registers. A compute instruction accessing such a register implicitly accesses the next element of the associated vector. The PVM implementation does not support a cache hierarchy and all vector data accessed by compute instructions is transferred from memory space. This shares similarity with SVectors/SStreams, but has some key differences. Our SVectors use vector load instructions to bring data into the SVR in a pipelined way and enable preloading of data. Any data that is spilled from the SVRs is held in the L2 cache for some time. In contrast, PVM supports a fast scratchpad memory space, somewhat analogous to our SVR. However, there are no vector load instructions to bring data into this space; data can be moved to scratchpad only through functional units using explicit instructions.

Conditional streams provide limited fine-grain DLP support for Imagine they allow different operations on different records on a stream. However, conditional streams change the order of the resulting records.

Titan uses a different approach to cater to DLP interspersed with control. It uses successive *scalar* FP registers to store a vector allowing individual vector elements to be accessed. All compute instructions are vector instructions and scalar operations have a length of 1. It is difficult to map such a design on to current renamed/out-of-order cores.

At a high level, SVectors exploit two dimensional DLP, as done by traditional SIMD array processors [Hwang 1993], MOM [Corbal et al. 1999; Sanchez et al. 2005], and CSI [Cheresiz et al. 2005]. This is because SVectors are, in turn, composed of small vectors (SIMD). However, unlike ALP, MOM uses vector/matrix instructions for computation and uses a large matrix register file. Similarly, unlike ALP, CSI uses a memory-to-memory stream architecture with a separate pipeline for streams.

Several architectures like Smart Memories [Mai et al. 2000], TRIPS [Sankaralingam et al. 2003], and RAW [Taylor et al. 2004] support all forms of parallelism. Instead of supporting a DLP-based programming model, like vectors/streams in the ISA, these architectures support efficient mapping/scheduling of multiple instructions that work on independent data and schedule communication among them. For example, TRIPS' main support for DLP consists of rescheduling loop iterations for computation without requiring prefetching and other repeated front-end overhead (called revitalization). RAW allows direct accessing of operands from the network, eliminating some explicit loads. SmartMemories can morph memories into many structures; ALP uses a more restricted type of morphing cache for SVRs. Unlike ALP, both Smart Memories and TRIPS require switching to a different mode to support DLP (resulting in mode changes between different parts of an application). Unlike ALP, both RAW and Smart Memories expose underlying hardware details and communication to the programming model.

Also related to our work are various mechanisms to enhance memory system performance. The most closely related is the stream memory controller (SMC) [McKee et al. 1996], which focuses on compile-time detection of streams and execution-time reordering of resulting memory accesses to maximize memory bandwidth. On the processor side, SMC provides separate hardware FIFO stream buffers controlled with memory-mapped registers. The processor references the next element of a stream buffer using such a memory-mapped register. While the main focus of this work (reordering memory accesses) is orthogonal to ours, the ability to access stream data has similarity with SStreams (and SVectors). The main difference is that our design integrates vectors and streams much more tightly within a modern out-of-order processor pipeline. Thus, we do not have special hardware stream buffers accessed through memory-mapped registers, which incur additional loads and stores. Instead, our SVRs are implemented within the (reconfigurable) cache, allowing compute (SIMD) instructions to directly access them using existing data paths, without additional loads and stores.

Other related work on memory system enhancements includes Impulse [Zhang et al. 2001] and the related parallel vector access unit [Mathew et al. 2000], which augment the memory controller to map noncontiguous memory to contiguous locations. Processor in memory architectures, like DIVA [Draper et al. 2002], increase memory bandwidth and decrease memory latency. Some DSP processors, as well as TRIPS, support software managed caches or scratch-pad memories, which usually need explicit loads/stores to be accessed. To reduce loads/stores, they support memory-to-memory addressing modes and DMA. SVRs achieve similar benefits without loads/stores.

To support regular computation, DSP processors include indexed addressing modes with autoincrementing, loop repetition, and/or rotating registers. NonDSP processors, such as Motorola 68K, provided autoincrementing modes. ALP achieves similar benefits with the unified mechanism of SVectors/ SStreams. For instance, SVectors provide nonspeculative block-prefetch capability, which is more efficient than loading individual array elements using autoincrementing.

Itanium [Intel Corporation 2001], Cydra 5 [Rau et al. 1989], and Hitachi SR-8000 [Tamaki et al. 1999] use rotating registers to hold data elements that are accessed sequentially. Rotating registers are used to provide different registers for different instances (in different loop iterations) of the same variable. In out-of-order processors, renaming provides the same functionality, albeit at a higher hardware cost. Rotating registers, which are usually a part of the general-purpose register file, are loaded with scalar load instructions. In contrast, SVectors use vector loads to bring a sequence of data records into the data arrays of reconfigured L1 cache. Further, rotating registers can hold variables that are accessed only within a given iteration. Therefore, unlike SVRs, such registers cannot store more permanent state (e.g., a table that is used many times or a variable used across iterations). SVectors do not have such limitations, i.e., SVectors can be loaded in advance and used repeatedly.

There is a large amount of work on special-purpose processors for media applications, including DSP processors, media processors, and ASICs. In contrast to these, ALP aims to use relatively evolutionary changes to GPPs to enable continued high performance for conventional applications. A comprehensive description of all special-purpose processors is outside the scope of this paper.

Finally, our previous work characterizes the parallelism and performance of the applications used in this paper [Li et al. 2005]. However, that work evaluates only SIMD support for exploiting the DLP of our applications. This paper presents our novel DLP support, SVectors and SStreams, in the context of a complete architecture that targets multiple levels of parallelism for our target applications.

8. CONCLUSIONS

We seek to provide energy-efficient performance for contemporary media applications in a GPP. We observe that these applications require efficient support for different types of parallelism, including ILP, TLP, and multiple forms of DLP. Given existing support for SIMD instructions in GPPs, the additional

DLP in these applications is either fine-grained or stream-based, and exhibits a relatively high ratio of memory to compute operations. Based on these observations and current GPP trends, we propose a complete architecture called ALP. ALP uses a CMP with superscalar cores with SIMD and SMT, enhanced with a novel mechanism of SIMD vectors and streams (SVectors/SStreams). SVectors/SStreams exploit many advantages of conventional vectors, without the cost of a dedicated vector unit.

Using several complex media applications, we show that all the techniques used in ALP are, indeed, important and effective and no single type of parallelism alone suffices. Specifically, SVectors/SStreams give speedups of 1.1 to 3.4X and EDP improvements of 1.1 to 5.1X for the applications that have DLP, over and above all of the other enhancements in ALP. The results of this paper are applicable to the applications with properties described in Section 1 and can be extended to other applications with similar properties.

REFERENCES

- AHN, J. H., DALLY, W. J., KHAILANY, B., KAPASI, U. J., AND DAS, A. 2004. Evaluating the imagine stream architecture. In Proc. of the 31st Annual Intl. Symp. on Comp. Architecture.
- ALBONESI, D. H. 1999. Selective cache ways: On-demand cache resource allocation. In Proc. of the 32nd Annual Intl. Symp. on Microarchitecture.
- ASANOVIC, K. 1998. Vector Microprocessors. Ph.D. thesis, Univ. of California at Berkeley.
- BEVERIDGE, R. AND DRAPER, B. 2003. Evaluation of face recognition algorithms. http://www.cs. colostate.edu/evalfacerec/.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. of the 27th Annual Intl. Symp. on Comp. Architecture*.

- CHERESIZ, D., JUURLINK, B. H. H., VASSILIADIS, S., AND WIJSHOFF, H. A. G. 2005. The CSI multimedia architecture. *IEEE Trans. VLSI Syst. 13,* 1.
- CORBAL, J., ESPASA, R., AND VALERO, M. 1999. MOM: A matrix SIMD instruction set architecture for multimedia applications. In Proc. of the 14th Intl. Conf. on Supercomputing.

CRAY INC. 2005. Cray X1 System Overview. http://www.cray.com/products/x1e/.

- DALLY, W. J., HANRAHAN, P., EREZ, M., ET AL. 2003. Merrimac: Supercomputing with streams. In *Proc. of 2003 ACM/IEEE conference on Supercomputing*.
- DIEFENDORFF, K. AND DUBEY, P. K. 1997. How multimedia workloads will change processor design. *IEEE Computer*.
- DRAPER, J., CHAME, J., HALL, M., ET AL. 2002. The architecture of the diva processing-in-memory chip. In *Proc. of the 17th Intl. Conf. on Supercomputing*.
- ESPASA, R. AND VALERO, M. 1997. Simultaneous multithreaded vector architecture. In Proc. of the 3rd Intl. Symp. on High-Perf. Comp. Architecture.
- ESPASA, R., VALERO, M., AND SMITH, J. E. 1997. Out-of-order vector architectures. In Proc. of the 25th Annual Intl. Symp. on Comp. Architecture.
- ESPASA, R., ARDANAZ, F., EMER, J., ET AL. 2002. Tarantula: A vector extension to the alpha architecture. In Proc. of the 29th Annual Intl. Symp. on Comp. Architecture.
- HENNESSY, J. L. AND PATTERSON, D. A. 2002. Computer Architecture: A Quantitative Approach. Morgan Kaufmann, San Mateo, CA.
- HOLLIMAN, M. AND CHEN, Y.-K. 2003. MPEG decoding workload characterization. In Proc. of Workshop on Computer Architecture Evaluation using Commercial Workloads.
- HUGHES, C. J., PAI, V. S., RANGANATHAN, P., AND ADVE, S. V. 2002. RSIM: Simulating shared-memory multiprocessors with ILP processors. *IEEE Computer*.
- HWANG, K. 1993. Advanced Computer Architecture: Parallelism, Scalability, Programmability. McGraw-Hill, New York.
- Intel Corporation 2001. Intel Itanium Architecture Software Developer's Manual. Intel Corporation, Santa Clara, CA.

- JOUPPI, N. P. 1989. A unified vector/scalar floating-point architecture. In Proc. of the 8th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems.
- KAPASI, U. J., DALLY, W. J., RIXNER, S., ET AL. 2003. Efficient conditional operations for data-parallel architectures. In Proc. of the 36th Annual Intl. Symp. on Microarchitecture.
- KITAGAWA, K., TAGAYA, S., HAGIHARA, Y., AND KANOH, Y. 2002. A hardware overview of SX-6 and SX-7 supercomputer. http://www.nec.co.jp/techrep/en/r_and_d/r03/r03-no1/rd02.pdf.
- KOZYRAKIS, C. 2002. Scalable vector media processors for embedded systems. Ph.D. thesis, Univ. of California at Berkeley.
- KOZYRAKIS, C. AND PATTERSON, D. 2003. Overcoming the limitations of conventional vector processors. In *Proc. of the 30th Annual Intl. Symp. on Comp. Architecture*.
- KRASHINSKY, R., BATTEN, C., HAMPTON, M., ET AL. 2004. The vector-thread architecture. In Proc. of the 31st Annual Intl. Symp. on Comp. Architecture.
- LEE, C. G. AND STOODLEY, M. G. 1999. Simple vector microprocessors for multimedia applications. In Proc. of the 31st Annual Intl. Symp. on Microarchitecture.
- LEE, L. H. 2000. Pseudo-vector machine for embedded applications. Ph.D. thesis, University of Michigan.
- LI, M.-L. 2005. Data-level and thread-level parallelism in emerging multimedia applications. M.S. thesis, Univ. of Illinois, Urbana-Champaign.
- LI, M.-L., SASANKA, R., ADVE, S. V., CHEN, Y.-K., AND DEBES, E. 2005. The ALPBench benchmark suite for multimedia applications. In *IEEE Intl. Symp. on Workload Characterization*.
- MAI, K., PAASKE, T., JAYASENA, N., HO, R., ET AL. 2000. Smart memories: A modular reconfigurable architecture. In Proc. of the 27th Annual Intl. Symp. on Comp. Architecture.
- MATHEW, B. K., MCKEE, S. A., CARTER, J. B., AND DAVIS, A. 2000. Design of a parallel vector access unit for sdram memories. In Proc. of the 6th Intl. Symp. on High-Perf. Comp. Architecture. 39–48.
- McKEE, S. A., ALUWIHARE, A., CLARK, B. H., KLENKE, R. H., LANDON, T. C., OLIVER, C. W., SALINAS, M. H., SZYMKOWIAK, A. E., WRIGHT, K. L., WULF, W. A., AND AYLOR, J. H. 1996. Design and evaluation of dynamic access ordering hardware. In *Proceedings of the 10th International Conference on* Supercomputing. ACM Press, New York. 125–132.
- MPEG SOFTWARE SIMULATION GROUP. 1994. MSSG MPEG2 encoder and decoder. http://www.mpeg.org/MPEG/MSSG/.
- PAI, V. S., RANGANATHAN, P., ADVE, S. V., AND HARTON, T. 1996. An evaluation of memory consistency models for shared-memory systems with ILP processors. In Proc. of the 7th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems. 12–23.
- QUINTANA, F., CORBAL, J., ESPASA, R., AND VALERO, M. 1999. Adding a vector unit to a superscalar processor. In Proc. of the 14th Intl. Conf. on Supercomputing.
- RANGANATHAN, P., ADVE, S., AND JOUPPI, N. P. 2000. Reconfigurable caches and their application to media processing. In *Proc. of the 27th Annual Intl. Symp. on Comp. Architecture.*
- RAU, B. R., YEN, D. W. L., YEN, W., AND TOWIE, R. A. 1989. The Cydra 5 departmental supercomputer: Design philosophies, decisions, and trade-offs. In *IEEE Computer*.
- REDDY, R. ET AL. 2001. CMU SPHINX. http://www.speech.cs.cmu.edu/sphinx/.
- ROSENBLUM, M., BUGNION, E., AND HERROD, S. A. 1995. The impact of architectural trends on operating system performance. In Proc. of 20th ACM Symp. on Operanting Systems Principles.
- SANCHEZ, F., ALVAREZ, M., SALAM, E., RAMIREZ, A., AND VALERO, M. 2005. On the scalability of 1and 2-dimensional SIMD extensions for multimedia applications. In *Proc. of IEEE Intl. Symp.* on *Performance Analysis of Systems and Software*.
- SANKARALINGAM, K., NAGARAJAN, R., LIU, H., ET AL. 2003. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In Proc. of the 30th Annual Intl. Symp. on Comp. Architecture.
- SASANKA, R., ADVE, S. V., CHEN, Y.-K., AND DEBES, E. 2004. The energy efficiency of CMP vs. SMT for multimedia workloads. In Proc. of the 20th Intl. Conf. on Supercomputing.
- SKADRON, K., ABDELZAHER, T., AND STAN, M. R. 2002. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In Proc. of the 8th Intl. Symp. on High-Perf. Comp. Architecture.
- STONE, J. E. 2003. Taychon raytracer. http://jedi.ks.uiuc.edu/~johns/raytracer/.
- TAMAKI, Y. SUKEGAWA, N., ITO, M., ET AL. 1999. Node architecture and performance evaluation of the Hitachi super technical server SR8000. In *Proc. of the 11th Intl. Conf. on Parallel and Distributed Systems*.

TAYLOR, M., LEE, W., MILLER, J., WENTZLAFF, D., ET AL. 2004. Evaluation of the RAW microprocessor: An exposed-wire-delay architecture for ILP and streams. In *Proc. of the 31st Annual Intl. Symp.* on Comp. Architecture.

WANG, Z. 1984. Fast algorithms for the discrete cosine transform and for the discrete fourier transform. In *IEEE Transactions in Acoustics, Speech, and Signal Processing. Vol. ASSP-32.*

ZHANG, L., FANG, Z., PARKER, M., MATHEW, B. K., ET AL. 2001. The impulse memory controller. In *IEEE Transcations on Computers*.

ZHANG, Y., PARIKH, D., SANKARANARAYANAN, K., ET AL. 2003. HotLeakage: A temperature-aware model of subthreshold and gate leakage for architects. Tech. Rep. CS-2003-05, Univ. of Virginia.

Received July 2005; revised August 2006; accepted August 2006