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

Ruchira Sasanka Man-Lap Li Sarita V. Adve
University of Illinois at Urbana-Champaign
Department of Computer Science
{manlapli, sasanka, sadve}@cs.uiuc.edu

Yen-Kuang Chen Eric Debes
Architecture Research Labs
Intel Corporation
{yen-kuang.chen, eric.debes}@intel.com

UIUC CS Technical Report UIUCDCS-R-2005-2605, July 2005
(Submitted for Publication)

ABSTRACT

The real-time execution of contemporary complex media applications requires energy-efficient processing capabilities beyond those of current superscalars. 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 sub-word 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 sub-word SIMD. This technique lies between sub-word 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 1.1X to 3.4X and energy-delay product improvements of 1.1X to 5.1X for applications with DLP.
Keywords: SIMD, multimedia, data-level parallelism

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, due to a wide range of multimedia standards and proprietary solutions [10]. However, real-time execution of such complex media applications needs a considerable amount of processing power that often surpasses the capabilities of current superscalars. Further, high performance processors are often constrained by power/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.

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 [1], SCALE [25], VIRAM [23], and CODE [24]. 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.

This paper differs from the above works in one or both of the following important ways. First, we use more complex applications from our recently released ALPBench benchmark suite [27]. 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, due to our focus on GPPs, we impose the following constraints/assumptions on our work: (i) GPPs already exploit some DLP through sub-word SIMD instructions such as MMX/SSE (subsequently referred to as SIMD), (ii) 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 (iii) radical changes in the hardware and programming model are not acceptable for well established GPPs. Motivated by the properties our applications and the above constraints/assumptions, we propose a complete architecture called ALP.

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

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 [27], 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) due to 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, and vectors/streams of SIMD.

The above observations motivate supporting many forms of parallelism including ILP, TLP, and various forms of DLP. For DLP, they imply that conventional solutions such as dedicated multi-lane vector units may be over-kill. For example, the Tarantula vector unit has the same area as its scalar core [12], but will likely be under-utilized for our applications due to the significant non-DLP parts, short vector lengths, and frequent reductions. We therefore take an alternative approach in this paper that aims to improve upon SIMD, but without the addition of a dedicated vector unit.

To effectively support *a*ll *l*evels of *p*arallelism 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 auto-increments the current element of that register. Thus, a loop containing a SIMD instruction accessing SVector register V0 marches through V0, much like a vector 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 superscalar 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 [2, 30]). 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 5X to 56X, energy reduction from 1.7X to 17.2X, and energy-delay product (EDP) reduction of 8.4X to 970.4X. These results include benefits from a 4-way CMP, 2-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 1.1X to 3.4X, energy savings of 1.1X to 1.5X, and an EDP improvement of 1.1X 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; and the evaluations consider complete applications.

More broadly, our results show that conventional architectures augmented with evolutionary mechanisms can provide high performance and energy savings for complex media applications without resorting to radically different architectures and programming paradigms (e.g., Imagine, SCALE).

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 su-

perscalar core as in current GPPs. The SIMD programming model roughly emulates Intel's MMX/SSE2 with multiple 8, 16, 32, or 64 bit sub-words within a 128 bit word and with eight SIMD logical registers. Most common opcodes are supported; e.g., packed addition, subtraction, 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)

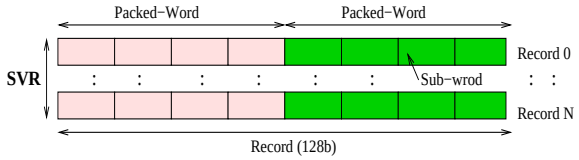

Figure 1: An SVR consists of records, a record consists of packed-words, and a packed-word consists of sub-words.

SVectors are built on three key enhancements to SIMD support:

1. SIMD Vector Registers (SVRs) hold a sequence of *records*, where each record itself is a sequence of (possibly strided) *packed-words* and each packed-word may contain multiple (contiguous) sub-words (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. SVector allocate (VALLOC) and SVector load (VLD) instructions. VALLOC and VLD allocate an SVR. VLD additionally loads a (possibly strided) sequence of packed-words into the SVR from memory. A slight variation of VALLOC, 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.

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 to V7, with a record size of 128 bits and sub-word 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

```
(1) VLD addr:stride:length ==> V0
(2) VLD addr:stride:length ==> V1
(3) VALLOCst addr:stride:length ==> V2
    do for all records in SVector
(4)   simd_add V0, V1 ==> simd_reg0
(5)   simd_mul simd_reg0, simd_reg1 ==> simd_reg2
(6)   simd_sub simd_reg2, #16 ==> V2
```

Figure 2: SVector code for $V2 = k * (V0 + V1) - 16$

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 number of packed-words per 128b record. All of this information is stored in the associated SVR descriptor.

As an example, Figures 2 gives SVector code for the computation $V2 = k*(V0+V1)-16$, where V0, V1, and V2 are SVRs, and k is a constant, stored in simd_reg1. The first two instructions load the two source SVectors from memory. The next instruction, VALLOCst, allocates a new SVR for writing V2. All of these instructions implicitly reset the CRP of V0, V1, and V2. 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 $\Rightarrow$ 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 (512B) 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 due to 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 (i) exceeds the maximum length of an SVR, and (ii) 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 vs. 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 [3, 7, 15]. Section 7 elaborates on the differences between SVectors and conventional vectors.

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 due to increased register locality. The SVRs increase the register space available that can be directly accessed by compute instructions, reducing loads/stores due to 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 (i) it would need a larger instruction width to encode the larger register space and (ii) a single large register file is energy inefficient and this price would be paid for *all* SIMD instructions. SVectors mitigate problem (i) 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 (ii) 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 16x16 or 8x8 blocks seen in MPEG, there may not be enough time for a hardware prefetcher to effectively learn the pattern [16]. Section 6 discusses experimental results that show that ALP's benefits exceed well beyond those for prefetching.

4. L1 cache space and bandwidth savings due to 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 MMX/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.

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 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 [36] 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.

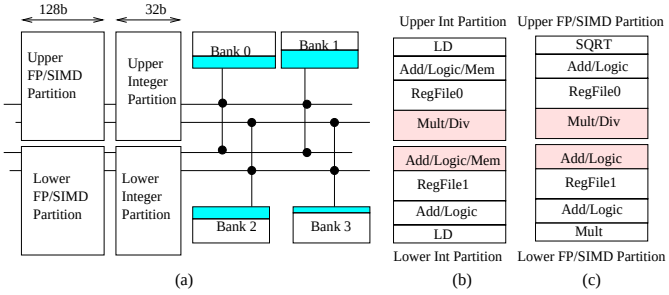

Figure 3: Integer, FP/SIMD, and L1 partitions/banks. (a) *Overview.* The integer and FP/SIMD execution units and register files consist of two partitions (upper and lower int or FP/SIMD execution partitions). The L1 cache consists of four banks. Each execution partition connects to all the four L1 banks – two 16B busses 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 (32b 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 (128b wide).* Similar to int, only the shaded units can access both register file partitions.

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 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 sub-banks) 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.

Table 1 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 500MHz (in 90nm technology) since ALP is targeted towards 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 results to hold with a wide range of parameter choices representative of modern superscalar GPPs. Section 6.1 reports

Parameter	Value PER PARTITION	# of Partitions
Phy Int Reg File (32b)	64 regs, 5R/4W	2
Phy FP/SIMD Reg File (128b)	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 (32b)	see Fig. 3	2
FP SIMD Units (128b)	see Fig. 3	2
Int SIMD Units (128b)	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 thread)	

Parameter	Value PER BANK	# Banks
L1 I-Cache	8K, 4 Way, 32B line, 1 Port	2
L1 D-Cache (Writethrough)	8K, 2 Way, 32B line, 1 Port	4
L2 Cache (Writeback, unified)	256K, 16 Way, 64B line, 1 Port	4

Bandwidth and Contentionless Latencies @ 500MHz	
Parameter	**Value (cycles @ 500MHz)**
ALU/Int SIMD Latency	8 (Div-32b), 2 (Mult-32b), 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

Table 1: Base architecture parameters. Note that several parameter values are *per partition or bank.* Section 6.1 reports some sensitivity results.

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 (Figure 3 and Figure 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 [2, 30]. 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. 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 busses 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 due to 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 un-aligned loads). This capability is generally found in caches to access individual words for writing. The home bank assembles two 16B records into a 32B 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 32B cache line (two 16B 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.

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

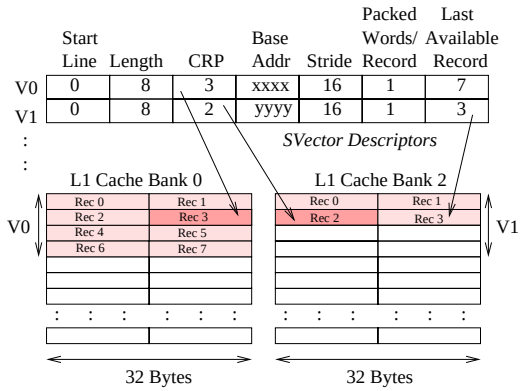

Figure 4: SVRs and SVector descriptor registers. Shaded cache lines contain SVRs whereas unshaded ones comprise normal cache data. Start Line, CRP, and Last Available Record are relative to the start of the cache.

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: (i) Start Line specifies the cache index address of the first record of the SVR, and (ii) 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 StartLine 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.

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.

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. Additionally, 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 happens at retirement.

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).

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 [24]. 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 due to 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 [24] 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). 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 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 [27]: MSSG MPEG-2 encoder and decoder [29] (MPGenc and MPGdec), Tachyon ray tracer [38] (RayTrace), Sphinx-3 speech recognizer [32] (SpeechRec), and CSU face recognizer [4] (FaceRec). 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. We additionally 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. A detailed description and characterization of the applications appear in [27].

	MPGenc	MPGdec	SpeechRec	FaceRec
Sub-word size	1B,2B	1B,2B	4B (float)	8B (double)
% SIMD instr.	24	24	17	66
Computation/Memory	1.70	1.84	1.43	1.00
DLP Granularity (in 128b words)	1,8,16	1,4,8	10	9750 (stream)
% Reductions	36	10	28	50

Table 2: DLP characteristics of applications.

Table 2 summarizes the DLP characteristics of our applications, except for RayTrace, which has only TLP and ILP.

The first row summarizes the sub-word sizes used by each application. MPGenc and MPGdec use smaller integer sub-words whereas SpeechRec and FaceRec use larger FP sub-words.

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 2 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 128b 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 [40].)

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., multi-laned 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 1).
- **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 (4 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 2-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

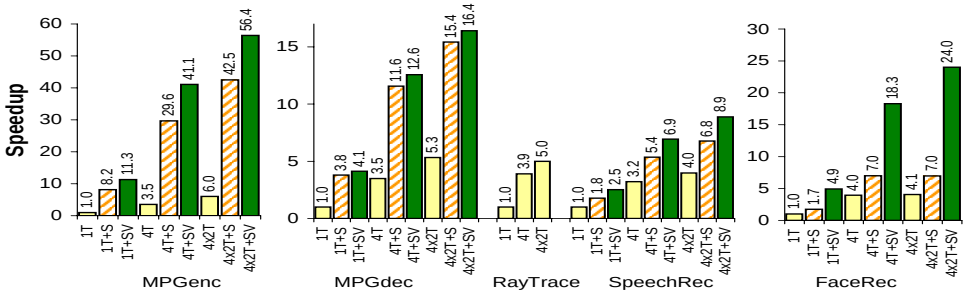

Figure 5: **Speedup of enhanced systems over the base 1T system for each complete application (ExecutionTimeBase/ExecutionTimeEnhanced).**

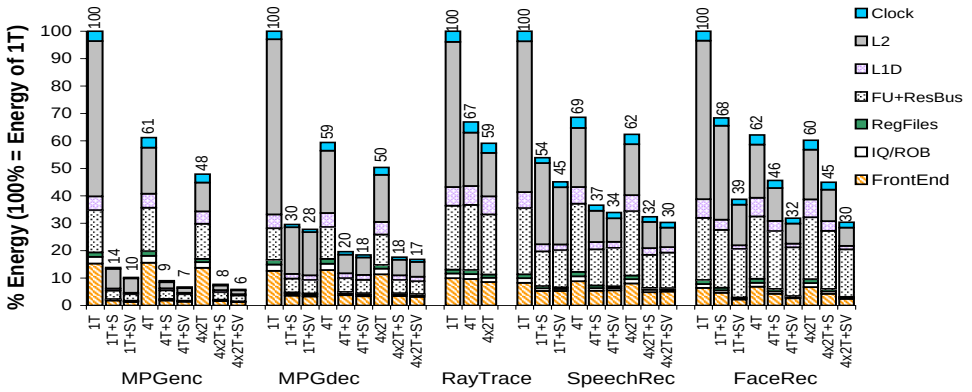

Figure 6: **Energy consumption and distribution of enhanced systems as a percentage of the energy consumption of the base system.**

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 [17], and model wrong path instructions and contention at all resources. We only emulate 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 [5] for dynamic power and HotLeakage [43] with temperature models from [37] for static power. We assume aggressive clock gating. Most components have 10% ungated circuitry [5]. 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 rename-map tables/descriptors, extra bits in caches).

6. RESULTS

6.1 Overall Results

Figures 5 and 6 and Table 3 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 5 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 (i.e., all except RayTrace); for the others, the means are computed across all applications. For reference, Table 4 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 5X to 56X (harmonic mean 9.2X), energy improvement of 1.7X to 17.2X (harmonic mean 2.8X), and EDP improvement of 8.4X 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 5 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 4-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 2-wide fetch/decode/retirement (but same issue width and functional units). Compared to this system, the 4-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 (due to 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 512x512 frames (a scene of a room with 20 objects), recognizing 30 words using a 130 word vocabulary/dictionary (SpeechRec), and recognizing 1,451 130x150 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).

App	1T +S	1T +SV	4T	4T +S	4T +SV	4x2T	4x2T +S	4x2T +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

Table 3: Energy Delay Product (EDP) improvement over the base (1T) system (EDPbase/EDPenhanced). Higher values are better.

To aid our analysis, Table 6 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-of-order processor, it is generally difficult to attribute execution time to different components. Following prior work [33], we follow a retirement-centric approach. 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 6 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 due to the low SIMD computation to memory ratio (Table 2). 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 pre-computation

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)

App	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 4: Instructions-per-cycle (operations-per-cycle) achieved by all systems.

Benefits of	SIMD			SVectors/SStreams		
Systems compared	1T+S/ 1T	4T+S/ 4T	4x2T+S/ 4x2T	1T+SV/ 1T+S	4T+SV/ 4T+S	4x2T+SV/ 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 compared	4T/ 1T	4T+S/ 1T+S	4T+SV/ 1T+SV	4x2T/ 4T	4x2T+S/ 4T+S	4x2T+SV/ 4T+SV	4x2T+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

Table 5: Mean speedup, energy improvement, and EDP improvement. 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).

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 [27]). 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 pre-computation phase results in more SIMD computation.

Reduction in SimdMem stalls is given by the difference between SimdMem 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 due to 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 4 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), and so more parallelism can be exploited. SpeechRec sees a slight decrease in SimdALU stall time due to the same reasons.

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 1 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.

[1]The original version of SpeechRec has a pre-computation phase to reduce the amount of work done in later phases. This pre-computation is omitted for +SV due to lack of any benefit.

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) due to 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 [27].

Reduction in other stalls results directly from the reduction in overhead instructions described above (most significantly in MPGenc and SpeechRec).

Energy benefits due to 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.

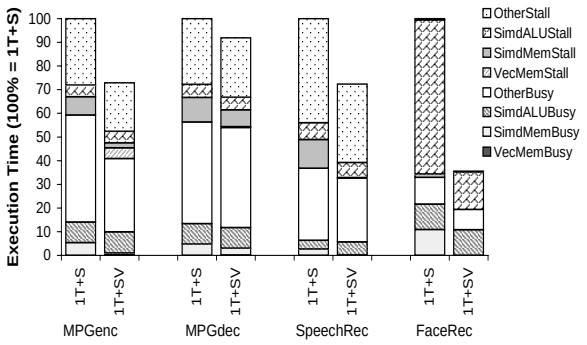

Figure 7: Execution time distribution for 1T+S and 1T+SV.

	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)

Table 6: # of instructions (operations) retired for 1T+S and 1T+SV systems as a percentage of instructions (operations) retired by 1T. The numbers for other +S (+SV) systems are the same as for 1T+S (1T+SV).

7. RELATED WORK

There is a vast amount of literature on conventional vector architectures and their recent uniprocessor and multiprocessor variations; e.g., VIRAM [23], CODE [24], Tarantula [12], T0 [3], out-of-order vectors [14], MOM [7], SMT Vectors [13], NEC SX [22], Cray X1 [8], and Hitachi SR [41]. 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 4-thread scalar core [12]). In return, they provide excellent performance and energy efficiency for medium-grain regular DLP; e.g., through multi-laned implementations.

However, our applications mostly exhibit small-grain DLP interspersed with control and reductions, and have parts that do not exhibit any DLP at all (Section 1 and 4). We therefore 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 other 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 2-16 operations in each instruction). Further, it may (or may not) be possible that significant algorithmic changes to our applications can expose larger grain DLP for which conventional vectors would be well suited. We are currently performing a detailed quantitative comparison between ALP and a representative vector machine (Tarantula) – reporting the results of such a study is outside the scope of this paper.

The Imagine architecture [1] (and its multiprocessor version, Merrimac [9]) 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, (i) 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; (ii) Imagine is designed as a co-processor that depends on a scalar host for irregular scalar computation while ALP's DLP support is tightly integrated into the superscalar core; and (iii) 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. At the same frequency, we found ALP's performance on MPEG2 encod-

ing comparable to that reported for Imagine (138 360x288 frames per second at 200 MHz but without B frames, half-pixel motion estimation, and Huffman VLC [1]). The details are omitted for lack of space.

A few architectures like SCALE [25], Pseudo Vector Machine (PVM) [26], conditional streams [21] of Imagine, and Titan [20] 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. So far, SCALE has been evaluated primarily for kernels; a comparison with ALP on complex applications is therefore difficult.

PVM provides support for vector/stream-like 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 [18], MOM [7, 34], and CSI [6]. 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 seperate pipeline for streams.

Several architectures like Smart Memories [28], TRIPS [35], and RAW [39] 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.

Several mechanisms enhance the memory system to support DLP. Impulse [42] augments the memory controller to map non-contiguous memory to contiguous locations. Processor in memory architectures like DIVA [11] increase memory bandwidth and decrease memory latency. Some DSP processors, as well as TRIPS, support software managed caches or scratchpad 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 auto-incrementing, loop repetition, and/or rotating registers. ALP achieves similar benefits with the unified mechanism of SVectors/SStreams.

Itanium [19], Cydra 5 [31], and Hitachi SR-8000 [41] 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.

Our previous work characterizes the parallelism and performance of the applications used in this paper [27]. 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 1.1X to 3.4X and EDP improvements of 1.1X 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.

In future work, we will report on a quantitative comparison with Tarantula.

9. REFERENCES

[1] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das. Evaluating the Imagine Stream Architecture. In *Proc. of the 31th Annual Intl. Symp. on Comp. Architecture*, 2004.

[2] D. H. Albonesi. Selective Cache Ways: On-Demand Cache Resource Allocation. In *Proc. of the 32nd Annual Intl. Symp. on Microarchitecture*, 1999.

[3] K. Asanovic. *Vector Microprocessors*. PhD thesis, Univ. of California at Berkeley, 1998.

[4] R. Beveridge and B. Draper. Evaluation of Face Recognition Algorithms. *http://www.cs.colostate.edu/evalfacerec/*, 2003.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proc. of the 27th Annual Intl. Symp. on Comp. Architecture*, 2000.

[6] D. Cheresiz, B. H. H. Juurlink, S. Vassiliadis, and H. A. G. Wijshoff. The CSI Multimedia Architecture. *IEEE Trans. VLSI Syst.*, 13(1), 2005.

[7] J. Corbal, R. Espasa, and M. Valero. MOM: A Matrix SIMD Instruction Set Architecture for Multimedia Applications. In *Proc. of the 14th Intl. Conf. on Supercomputing*, 1999.

[8] Cray Inc. Cray X1 System Overview. *www.cray.com*, 2005.

[9] W. J. Dally, P. Hanrahan, M. Erez, et al. Merrimac: Supercomputing with Streams. In *Proc. of 2003 ACM/IEEE conference on Supercomputing*, 2003.

[10] K. Diefendorff and P. K. Dubey. How Multimedia Workloads Will Change Processor Design. *IEEE Computer*, Sep. 1997.

[11] J. Draper, J. Chame, M. Hall, et al. The architecture of the diva processing-in-memory chip. In *Proc. of the 17th Intl. Conf. on Supercomputing*, 2002.

[12] R. Espasa, F. Ardanaz, J. Emer, et al. Tarantula: A Vector Extension to the Alpha Architecture. In *Proc. of the 29th Annual Intl. Symp. on Comp. Architecture*, 2002.

[13] R. Espasa and M. Valero. Simultaneous multithreaded vector architecture. In *Proc. of the 3rd Intl. Symp. on High-Perf. Comp. Architecture*, 1997.

[14] R. Espasa, M. Valero, and J. E. Smith. Out-of-order vector architectures. In *Proc. of the 25th Annual Intl. Symp. on Comp. Architecture*, 1997.

[15] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[16] M. Holliman and Y.-K. Chen. MPEG Decoding Workload Characterization. In *Proc. of Workshop on Computer Architecture Evaluation using Commercial Workloads*, 2003.

[17] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve. RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors. *IEEE Computer*, February 2002.

[18] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, programmability*. McGraw-Hill Inc., 1993.

[19] Intel Corporation. *Intel Itanium Architecture Software Developer's Manual*, 2001.

[20] N. P. Jouppi. A Unified Vector/Scalar Floating-Point Architecture. In *Proc. of the 8th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 1989.

[21] U. J. Kapsai, W. J. Dally, S. Rixner, et al. Efficient Conditional Operations for Data-parallel Architectures. In *Proc. of the 36rd Annual Intl. Symp. on Microarchitecture*, 2003.

[22] K. Kitagawa, S. Tagaya, Y. Hagihara, and Y. Kanoh. 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*, 2002.

[23] C. Kozyrakis. *Scalable Vector Media Processors for Embedded Systems*. PhD thesis, Univ. of California at Berkeley, 2002.

[24] C. Kozyrakis and D. Patterson. Overcoming the Limitations of Conventional Vector Processors. In *Proc. of the 30th Annual Intl. Symp. on Comp. Architecture*, 2003.

[25] R. Krashinsky, C. Batten, M. Hampton, et al. The Vector-Thread Architecture. In *Proc. of the 31th Annual Intl. Symp. on Comp. Architecture*, 2004.

[26] L. H. Lee. *Pseudo-Vector Machine for Embedded Applications*. PhD thesis, University of Michigan, 2000.

[27] M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The ALPBench Benchmark Suite for Multimedia Applications (Submitted for publication). Technical Report UIUCDCS-R-2005-2603, Dept. of Computer Science, University of Illinois, July 2005.

[28] K. Mai, T. Paaske, N. Jayasena, R. Ho, et al. Smart Memories: A Modular Reconfigurable Architecture. In *Proc. of the 27th Annual Intl. Symp. on Comp. Architecture*, 2000.

[29] MPEG Software Simulation Group. MSSG MPEG2 encoder and decoder. *http://www.mpeg.org/MPEG/MSSG/*, 1994.

[30] P. Ranganathan, S. Adve, and N. P. Jouppi. Reconfigurable Caches and their Application to Media Processing. In *Proc. of the 27th Annual Intl. Symp. on Comp. Architecture*, 2000.

[31] B. R. Rau, D. W. L. Yen, W. Yen, and R. A. Towie. The Cydra 5 Departmental Supercomputer: Design Philosophies, Decisions, and Trade-Offs. In *IEEE Computer*, 1989.

[32] R. Reddy et al. CMU SPHINX. *http://www.speech.cs.cmu.edu/sphinx/*, 2001.

[33] M. Rosenblum, E. Bugnion, and S. A. Herrod. Vector Vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks. In *Proc. of 20th ACM Symp. on Operanting Systems Principles*, 1995.

[34] F. Sanchez, M. Alvarez, E. Salam, A. Ramirez, and M. Valero. On the Scalability of 1- and 2-Dimensional SIMD Extensions for Multimedia Applications. In *Proc. of IEEE Intl. Symp. on Performance Analysis of Systems and Software*, 2005.

[35] K. Sankaralingam, R. Nagarajan, H. Liu, et al. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. In *Proc. of the 30th Annual Intl. Symp. on Comp. Architecture*, 2003.

[36] R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The Energy Efficiency of CMP vs. SMT for Multimedia Workloads. In *Proc. of the 20th Intl. Conf. on Supercomputing*, 2004.

[37] K. Skadron, T. Abdelzaher, and M. R. Stan. 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*, 2002.

[38] J. E. Stone. Taychon Raytracer. *http://jedi.ks.uiuc.edu/ johns/raytracer/*, 2003.

[39] M. Taylor, W. Lee, J. Miller, D. Wentzlaff, et al. Evaluation of the RAW Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *Proc. of the 31th Annual Intl. Symp. on Comp. Architecture*, 2004.

[40] Z. Wang. 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, 1984.

[41] Y.Tamaki, N. Sukegawa, M. Ito, et al. Node Architecture and Performance Evaluation of the Hitachi Super Technical Server SR8000. In *Proc. of the 11th Intl. Conf. on Parallel and Distributed Systems*, 1999.

[42] L. Zhang, Z. Fang, M. Parker, B. K. Mathew, et al. The Impulse Memory Controller. In *IEEE Transcations on Computers*, 2001.

[43] Y. Zhang, D. Parikh, K. Sankaranarayanan, et al. HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. Technical Report CS-2003-05, Univ. of Virginia, 2003.