The ALPBench Benchmark Suite for Complex Multimedia Applications *

Man-Lap Li Ruchira Sasanka 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

ABSTRACT

Multimedia applications are becoming increasingly important for a large class of general-purpose processors. Contemporary media applications are highly complex and demand high performance. A distinctive feature of these applications is that they have significant parallelism, including thread- , data-, and instruction-level parallelism, that is potentially well-aligned with the increasing parallelism supported by emerging multi-core architectures. Designing systems to meet the demands of these applications therefore requires a benchmark suite comprising these complex applications and that exposes the parallelism present in them.

This paper makes two contributions. First, it presents ALPBench, a publicly available benchmark suite that pulls together five complex media applications from various sources: speech recognition (CMU Sphinx 3), face recognition (CSU), ray tracing (Tachyon), MPEG-2 encode (MSSG), and MPEG-2 decode (MSSG). We have modified the original applications to expose thread-level and data-level parallelism using POSIX threads and sub-word SIMD (Intel's SSE2) instructions respectively. Second, the paper provides a performance characterization of the ALPBench benchmarks, with a focus on parallelism. Such a characterization is useful for architects and compiler writers for designing systems and compiler optimizations for these applications.

1. INTRODUCTION

Multimedia applications are becoming an important workload for general-purpose processors [5]. Emerging media applications are highly complex, incorporating increasingly intelligent algorithms that are more control intensive than in the past and incorporating increasing functionality. These applications demand high performance and energy efficiency. At the same time, they present new opportunities, especially in the form of various types of parallelism. The effective design of processors for these applications therefore requires a benchmark suite comprising contemporary complex media applications (versus individual kernels) that exposes the parallelism in these applications.

This work makes two contributions. First, it presents ALPBench, a suite of existing and emerging complex media applications, modified to expose thread-level and data-level parallelism (TLP and DLP respectively). ALPBench is publicly available from *http://www.cs.uiuc.edu/alp/alpbench/*. The current release includes five applications: speech recognition (derived from CMU Sphinx3.3 [28]), face recognition (derived from CSU face recognizer [3]), ray tracing (same as Tachyon [31]), MPEG-2 encode (derived from MSSG MPEG-2 encoder [25]), and MPEG-2 decode (derived from MSSG MPEG-2 decoder [25]). We modified the original applications to expose TLP by using POSIX threads (ray tracing was already parallelized) and to expose DLP by inserting sub-word SIMD (Intel SSE2) instructions into the most commonly used routines.

We believe that the applications in ALPBench will be routinely used on general-purpose processors to fulfill user requirements such as video conferencing, DVD/HDTV playback and recording, gaming and virtual reality, video editing, authentication, and personal search/organization/mining of media/digital information. The applications chosen represent a spectrum of media processing, covering video, speech, graphics, and image processing. The applications are all fairly complex, especially in contrast to kernels that are often used in multimedia studies. It is important to study these applications in their entirety because many effects are difficult to identify in a study that only evaluates kernels [9].

The second contribution of this work is a characterization of the parallelism and performance for the five ALPBench applications. We find that these applications contain multiple forms of parallelism – TLP, DLP, and ILP (instruction-level parallelism). For TLP, we find that all the applications have coarse-grain threads and most show very good thread scalability. Therefore, these applications are a good match for emerging processors with chip-multiprocessing (CMP) and simultaneous multi-threading (SMT). For DLP, we find that four out of five of these applications are amenable to sub-word SIMD instructions (SIMD for short). Many current general-purpose processors already support such SIMD media instruction sets (e.g., MMX/SSE [14]). We also report on the interaction between different forms of parallelism

*This work is supported in part by a gift from Intel Corp., an equipment donation from AMD Corp., and the National Science Foundation under Grant No. CCR-0096126, CCR-0209198, and EIA-0224453. Ruchira Sasanka was supported by an Intel graduate fellowship.

and the effects of the memory system on these applications (including their working sets, bandwidth requirements, and memory latency tolerance).

There are several prior studies that evaluate the individual applications in ALPBench [4, 10, 16, 17, 19, 20, 23, 24, 26, 30, 35]. There are also several popular benchmark suites that already target media applications and contain some of the applications in ALPBench; e.g., MediaBench [21], Berkeley multimedia workload [30], MiBench [8], and EEMBC [7]. This work differs from most of the above work because our focus is on the *parallelism* in these applications. Section 5 provides a detailed description of the related work.

2. APPLICATIONS

This section describes our applications and the enhancements we made to them. To extract parallelism, we threaded the applications and inserted SIMD instructions in the frequently used functions. For threading, we used POSIX threads (Pthreads). For most cases, straightforward parallelization was sufficient for the relatively small systems we consider (e.g., static scheduling of threads). For SIMD, we used Intel SSE2 and a more aggressive simulated version called ALP SIMD [29], which is modeled after SSE2.[1] SIMD hand-coding is prevalent practice for these applications and the maximum number of static assembly instructions inserted for any given application is about 400 (for MPEG-2 encoder). In some cases, we made a few algorithmic modifications to the original applications to improve performance.

The following sections describe algorithmic modifications (where applicable), major data structures, application phases, thread support, and SIMD support.

2.1 MPEG-2 Encoder (MPGenc)

We use the MSSG MPEG-2 encoder [25]. MPGenc converts video frames into a compressed bit-stream. A video encoder is an essential component in VCD/DVD/HDTV recording, video editing, and video conferencing applications. Many recent video encoders like MPEG-4/H.264 use similar algorithms.

A video sequence consists of a sequence of input images, which are in the YUV format; i.e., one luminance (Y) and two chrominance (U,V) components. Each encoded frame is characterized as an I, P, or B frame. I frames are temporal references for P and B frames and are only spatially compressed. On the other hand, P frames are predicted based on I frames, and B frames are predicted based on neighboring I and P frames.

Modifications: We made two algorithmic modifications to the original MSSG code: (1) we use an intelligent three-step motion search algorithm [18] instead of the original full-search algorithm and (2) we use a fast integer discrete cosine transform (DCT) butterfly algorithm based on the Chen-Wang algorithm [34] instead of the original floating point matrix-based DCT.

Data Structures: Each frame consists of 16x16 pixel macroblocks. Each macroblock consists of four 8x8 luminance blocks and two 8x8 chrominance blocks, one for U and one for V.

Phases: The phases in MPEG-2 include motion estimation, form prediction, quantization, discrete cosine transform (DCT), variable length coding (VLC), inverse quantization, and inverse DCT (IDCT).

The first frame is always encoded as an I-frame. For an I-frame, the compression starts with DCT, which transforms blocks from the spatial to the frequency domain. Following DCT is quantization which operates on a given 8x8 block, a quantization matrix, and a quantization value. The operations are performed on each pixel of the block independent of each other. After quantization, VLC is used to compress the bit stream. VLC uses both Huffman and run-length coding. This completes the compression.

For predictive (P and B) frames, the compression starts with motion estimation. In motion estimation, for each macroblock of the frame being currently encoded, we search for a "best-matching" macroblock within a search window in a previously encoded frame. The distance or "match" between two macroblocks is computed by calculating the sum of the differences between the pixels of the blocks. The original "full-search" algorithm performs this comparison for all macroblocks in the search window. Instead, we use a three-step search algorithm. This breaks a macroblock search into three steps, searching at the (i) center, (ii) around the edges, and (iii) around the center of the search window. A subsequent step is taken only if the previous step does not reveal a suitable match. Motion estimation is the longest (most compute intensive) phase for P and B frames. After motion estimation, the best-matched block is subtracted from the current block to get the error (form predictions). The rest of the compression for P and B frames is the same as that for an I-frame.

To process subsequent frames, it is necessary to decode the encoded frame. For this purpose, inverse quantization and IDCT are applied to the encoded frame. These inverse operations have the same properties as their forward counterparts.

We removed the rate control logic from this application. The original implementation performs rate control after each macroblock is encoded, which imposes a serial bottleneck. For the threaded version, rate control at the end of a frame encoding would be more efficient but we did not implement this.

Threads: We create a given number of threads at the start of a frame and join them at the end of that frame. Within a

[1]The publicly released version of ALPBench only contains SSE2 SIMD instructions.

frame, each thread encodes an independent set of contiguous macroblock rows in parallel. Each thread takes such a set through all the listed phases and writes the encoded stream to a private buffer. Thread 0 sequentially writes the private buffers to the output.

SIMD: Integer SIMD instructions are added to all the phases except VLC. 8b (char) sub-words are used in macroblocks; 16b (short) words are used to maintain running sums. The main SIMD computation in motion estimation is a calculation of sum of absolute difference (SAD) between two 128b packed words of two macroblocks. PSAD (packed SAD) instructions in SSE2 are used for this purpose. The result of the SAD is accumulated in a register. For half pixel motion estimation, it is necessary to find the average of two 128b records. This is achieved using PAVG (packed average) SSE2 instructions.

We obtained optimized SSE2 code for DCT and IDCT from [12] and [13], respectively. This code uses sub-word sizes of 16b (short) and multiply accumulate instructions for common multiply accumulate combinations. Quantization involves truncation operations. We use packed minimum and packed maximum for performing the truncations [14].

Before DCT and after IDCT, the encoder performs a block subtraction (prediction formation) and a block addition where a block of frequency deltas are added or subtracted from a block. We use packed saturated addition and subtraction for these operations.

2.2 MPEG-2 Decoder (MPGdec)

We use the MSSG MPEG-2 decoder [25]. MPGdec decompresses a compressed MPEG-2 bit-stream. Video decoders are used in VCD/DVD/HDTV playback, video editing, and video conferencing. Many recent video decoders, like MPEG-4/H.264, use similar algorithms.

Data Structures: Same as for MPGenc.

Phases: Major phases for MPGdec include variable length decoding (VLD), inverse quantization, IDCT, and motion compensation.

The decoder applies the inverse operations performed by the encoder. First, it performs variable-length Huffman decoding. Second, it inverse quantizes the resulting data. Third, the frequency-domain data is transformed with IDCT to obtain spatial-domain data. Finally, the resulting blocks are motion-compensated to produce the original pictures.

Threads: In our implementation, thread 0 identifies the slices (contiguous rows of blocks) in the input encoded bit-stream. When a given number of slices are identified, they are assigned to a new thread for decoding. This results in staggered thread creation and completion times, affecting scalability.

Each thread takes each block in a slice through all the phases listed above and then writes each decoded block into a non-overlapping region of the output image buffer.

SIMD: Integer SIMD instructions are added to IDCT and motion compensation. IDCT uses the SIMD code used in MPGenc. Motion compensation contains sub-functions like add-block (adding the reference block and error or frequency deltas) and saturate. These operations are performed using packed addition with saturate on 16b words.

2.3 Ray Tracing (RayTrace)

We use the Tachyon ray tracer [31]. A ray tracer renders a scene using a scene description. Ray tracers are used to render scenes in games, 3-D modeling/visualization, virtual reality applications, etc. Dubey includes ray tracing as one of the key challenging applications for future general-purpose processors [6].

The ray tracer takes in a scene description as input and outputs the corresponding scene. A scene description normally contains the location and viewing direction of the camera, the locations, shapes, and types of different objects in the scene, and the locations of the light sources.

Data Structures: The constructed scene is a grid of pixels. The pixels are colored based on the light sources and objects in the scene. The objects are maintained in a linked list. The color of each pixel is determined independently.

Phases: This application does not have distinct phases at a high level. At start, based on the camera location and the viewing direction specified, the viewing plane is created to represent the grid of pixels to be projected from the scene to the resulting picture. To project the correct color for each pixel, a ray is shot from the camera through the viewing plane into the scene. The ray is then checked against the list of objects to find out the first object that the ray intersects. After that, the light sources are checked to see if any of the light rays reach that intersection. If so, the color to be reflected is calculated based on the color of the object and the color of the light source. The resulting color is assigned to the pixel where the camera ray and the viewing plane intersect. Moreover, since objects can be reflective or transparent, the ray may not stop at the first object it intersects. Instead, the ray can be reflected or refracted to other directions until it intersects another object.

Threads: Each thread is given N independent rays to trace, where N is the total number of pixels in the viewing plane divided by the number of threads in the system.

SIMD: No DLP support is added since we could not identify DLP among either different rays or within a ray. Computations for different rays can be different since even neighboring rays can intersect different objects. Within a ray, significant control intensive computation made it hard to find

DLP.

2.4 Speech Recognition (SpeechRec)

We use the CMU SPHINX3.3 speech recognizer [28]. A speech recognizer converts speech into text. Speech recognizers are used with communication, authentication, and word processing software and are expected to become a primary component of the human-computer interface.

Data Structures: The major data structures used include:

(1) 39-element feature vectors extracted from an input speech sample.

(2) Multiple lexical search trees built from the language model provided. Each tree node is a 3-state hidden Markov model (HMM) and describes a phoneme (sound element).

(3) Each senone (a set of acoustically similar HMM states) is modeled by a Gaussian model. Each Gaussian model contains two arrays of 39-element vectors (mean and variance) and one array of coefficients.

(4) An array that stores candidates of recognized words. Each array element also contains a word ID and a back pointer to the previously recognized word. The word ID is used to lookup the actual word from the dictionary. A sequence of words linked by the back pointers forms a hypothesis.

(5) A dictionary (hash table) of known words.

Phases: The application has three major phases: feature extraction, Gaussian scoring, and searching the language model/dictionary. First, the feature extraction phase creates 39-element feature vectors from the speech sample. The Gaussian scoring phase then matches these feature vectors against the phonemes in a database. It evaluates each feature vector based on the Gaussian distribution in the acoustic model (Gaussian model) given by the user. In a regular workload, there are usually 6000+ Gaussian models. The goal of the evaluation is to find the best score among all the Gaussian models and to normalize other scores with the best one found. As this scoring is based on a probability distribution model, multiple candidates of phonemes are kept so that multiple words can be matched. The final phase is the search phase, which matches the candidate phonemes against the most probable sequence of words from the language model and the given dictionary. Similar to the scoring phase, multiple candidates of words (hypotheses) are kept so that the most probable sequence of words can be chosen.

The algorithm can be summarized as follows. At start, we make the root node of each lexical search tree active. The following steps are repeated until speech is identified:

(i) *Feature extraction:* This phase creates a feature vector from the speech sample.

(ii) *Gaussian scoring:* A feature vector is compared against Gaussian models of most likely senones and a similarity score is computed for each senone.

(iii) *Search phase - part 1* For each active node in each lexical search tree, the best HMM score for it is calculated. Then the overall best HMM score among all nodes is calculated (call this S_{ob}).

(iv) *Search phase - part 2* All nodes with HMM scores below $S_{ob} - threshold$, where $threshold$ is a given threshold, are deactivated and the children of the still active nodes are also activated. If the current active node is a leaf node with high enough score, the word is recognized and inserted into the hypothesis array and the dictionary is looked up to find the spelling.

For reporting results, the startup phase, where some data structures are initialized, is ignored since it is done only once for the entire session and it can be optimized by loading checkpointed data [24].

Threads: We parallelized both the Gaussian scoring and the search phases. We did not parallelize the feature extraction phase since it takes only about 2% of the execution time (with a single thread). A thread barrier is used for synchronization after each phase. In the Gaussian scoring phase, we divide the Gaussian models among threads to calculate senone scores. In the search phase, we divide the active nodes evenly among the threads. In the second part of the search phase, since the updates of the hypotheses involve writing to a shared data structure, we use fine grain locking to synchronize these updates. This locking makes this phase less scalable than the Gaussian scoring phase.

SIMD: We enhanced the Gaussian scoring phase with single precision floating point (32b subwords) SIMD instructions. The score computation consists of a short loop which performs a packed subtraction (SUBPS) and two packed multiplications (MULPSs). The scalar score is then obtained through reduction operations: packed addition (ADDPS) and a horizontal sum (SREDF).

2.5 Face Recognition (FaceRec)

We use the CSU face recognizer [3]. Face recognizers recognize images of faces by matching a given input image with images in a given database. Face recognition is used in applications designed for authentication, security, and screening. Similar algorithms can be used in other image recognition applications that perform image searches and data-mining.

This application uses a large database (called subspace) that consists of multiple images. The objective of phase recognition is to find the image in the subspace that best matches a given input image. A match is determined by taking the "distance" or difference between two images.

Modifications: The CSU software tries to find the pairwise distances among all images in the database since its objective is to find the effectiveness of the distance finding algorithm. We modified the application so that a separate input image is compared with each image in the subspace to emulate a typical face recognition scenario (e.g., a face of a

subject is searched in a database).

Data Structures: Each image is a single column vector with thousands of rows. The subspace is a huge matrix where each column is an image.

Phases: This application is first trained with a collection of images in order to distinguish faces of different persons. Moreover, there are multiple images that belong to the same person so that the recognizer is able to match face images against different expressions and lighting conditions. Then, the training data is written to a file so that it can be used in the recognition phase. Since training is done offline, we consider only the recognition phase for reporting results.

At the start of the recognition phase, the training data and the image database are loaded. The subspace matrix is created from the image database. The rest of the recognition phase has two sub-phases:

(i) Projection: When an input image is given, it is normalized and projected into the large subspace matrix that contains the other images. The normalization involves subtracting the subspace's mean from the input. Then that normalized image is "projected" on to the subspace by taking the cross product between the normalized image and the subspace.

(ii) Distance computation: Computes the difference between each image in the subspace and the given image by finding the similarity (distance).

Threads: In the projection sub-phase, each thread is given a set of columns from the subspace to multiply. In the distance-computation sub-phase, each thread is responsible for computing distances for a subset of images in the database.

SIMD: Both the sub-phases of the recognition phase contain short loops that perform multiplication and addition/subtraction. We use double precision FP (64b) packed subtraction (SUBPD), packed multiplication (MULPD), and packed addition (ADDPD) SIMD instructions in these sub-phases.

3. METHODOLOGY

For this study, we primarily obtain results from a chip multiprocessor (CMP) simulator called ALPSim. ALPSim allows us to study the parallelism and scalability of systems under different conditions. To augment these results, where practically feasible, we also present data obtained on a real Pentium 4 system.

ALPSim is an execution-driven cycle-level simulator derived from RSIM [11], and models wrong path instructions and contention at all resources. ALPSim simulates all code in C libraries but only emulates operating system calls.

The CMP system modeled by ALPSim contains out-of-order superscalar processor cores with private L1 data and instruction caches and a shared unified L2 cache. Each thread is run on a separate CMP processor (i.e., the number of CMP processors is equal to the maximum number of threads used in an experiment). The simulation parameters used are given in Table 1. Following the modern trend of general purpose processor architectures, almost all processor resources are partitioned (clustered) and caches are banked.

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
Integer ALUs (32b)	2	2
FP/SIMD Units (128b)	2	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, 4 Way, 64B line, 1 Port (for single-threaded cases)	4
	512K, 8 Way, 64B line, 1 Port (for multithreaded cases)	32

Bandwidth and Contentionless Latencies @ 4 GHz	
Parameter	**Value (cycles @ 4 GHz)**
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	2
L1 D-Cache Hit Latency	3
L2 Cache Hit Latency	16
L2 Miss Latency	256
Memory Bandwidth	16 GB/s

Table 1: Base architecture parameters for ALPSim. Note that several parameter values are *per partition or bank.*

The ALP SIMD programming model used with ALPSim roughly emulates Intel's MMX/SSE2 with multiple 8-, 16-, 32-, or 64-bit sub-words within a 128-bit word. Most common opcodes are supported; e.g., packed addition, subtraction, multiplication, absolute difference, average, horizontal reduction, logical, and pack/unpack operations. SIMD operations use the FP register file and FP units.

ALPSim uses SPARC binaries for non-SIMD code. Pthreads-based C code is translated into binary using the Sun cc 4.2 compiler with options -xO4 -xunroll=4 -

xarch=v8plusa. SIMD code resides in a separate assembly file, organized as blocks of instructions and simulated using hooks placed in the binary. When such a hook is reached while simulating, the simulator switches to the proper block of SIMD instructions. ALPSim emulates synchronization operations (locks and barriers), but properly charges for any waiting time due to contention for a synchronization variable.

To complement the results obtained using ALPSim, we obtained data using a 3.06 GHz Pentium 4 system with SSE2 running the Linux 2.4 kernel (henceforth referred to as **P4**). The processor front-side bus operates at 533 MHz (quad-pumped) and the system has 2GB of PC2100 DDR memory. The applications for P4 were compiled using the Intel icc compiler with maximum optimization level O3 and options -march=pentium4 -mcpu=pentium4. We aligned data arrays at 16B boundaries for best performance as suggested in [14]. We used the Intel VTune performance analyzer and the performance counter (sampling) mode to obtain results without any binary instrumentation. We only obtained single-thread data on P4.

We used the following inputs for the applications. For MPGenc and MPGdec, we used DVD resolution (704x480) input streams from ftp://ftp.tek.com/tv/test/streams/Element/index.html. For RayTrace, we used a 512x512 resolution picture (a scene of a room with 20 objects), available with the original application. For SpeechRec, we used a dictionary/vocabulary of 130 words with the input speech sample containing the words "Erase T M A Z X two thousand five hundred and fifty four" (obtained from http://www.speech.cs.cmu.edu/databases/an4/). For FaceRec, we used a database of 173 images (resolution 130x150) with an input image of the same resolution (available with the original application).

4. RESULTS

This section first presents our results characterizing the different types of parallelism in ALPBench, followed by results on the interaction between these types of parallelism. Since we observe that our parallelism results are sensitive to the memory system parameters, we next present memory system related data, including the working set sizes, the effect of increasing memory latencies (i.e., frequency scaling), and the effect of supporting more threads on the memory bandwidth. Finally, we give the application-level real-time performance of our applications on the P4 system.

Although the results we show are sensitive to the size of the inputs, the overall parallelism should improve or remain the same with larger inputs for all applications.

4.1 TLP

Figures 1(a) and (b) show the speedup achieved with multiple threads on ALPSim with an ideal 1 cycle memory system and a realistic memory system respectively. The threads do not use SIMD instructions. The ideal memory system results are obtained with perfect 1 cycle L1 caches to study the TLP scalability independent of the memory system parameters (since, as shown in Section 4.5, the scalability is sensitive to these parameters). The system with the realistic memory system is as described in Section 3.

As shown in Figure 1, MPGenc, FaceRec, and RayTrace scale very well up to 16 threads with both ideal and realistic memory parameters since the threads are independent and hence do not require extensive synchronization. MPGdec also scales well, but its scalability could be further improved by addressing two limitations: (i) staggered thread creation and (ii) load imbalance. The effects of both limitations diminish with larger inputs (e.g., HDTV input improved the speedup of 16 threads by 14%-15% for both ideal and realistic memory parameters). The second limitation can also be addressed by dynamic slice assignment [10].

The thread scalability of SpeechRec is somewhat limited, largely due to the fine-grain synchronization (locking) used in the search phase [20]. We found that larger dictionaries improve the scalability. We also note that the thread scalability of SpeechRec is slightly better with the realistic memory parameters. This is because the execution time of the single thread version with realistic memory is dominated by the time stalled for memory. The multithreaded versions reduce that stall time considerably due to the memory parallelism offered by multiple threads, countering the negative effect due to synchronization. The ideal memory system does not see this positive memory parallelism effect.

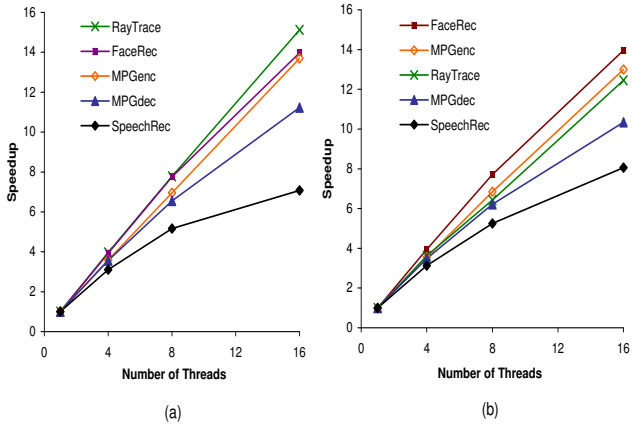

Figure 1: Scalability of TLP without SIMD instructions (a) with an ideal 1-cycle memory system, and (b) with realistic memory parameters (as in Table 1).

4.2 DLP

Figure 2 gives the speedups achieved with SSE2 (on P4) and ALP SIMD (on ALPSim) over the original non-SIMD single-threaded application. The results with SSE2 show the speedups achievable on existing general-purpose processors. The results with ALP SIMD indicate the speedups possible with a more general form of SIMD on a simulated 4 GHz processor. Overall, our applications (except RayTrace)

achieve significant speedups with ALP SIMD and modest to significant speedups with SSE2.

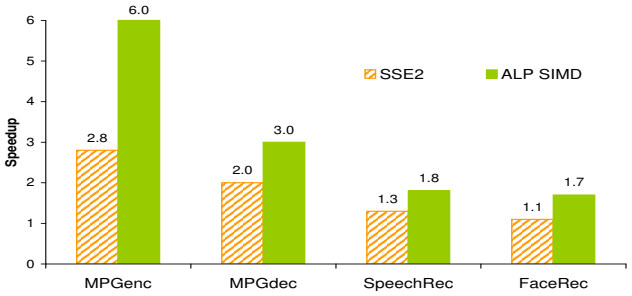

Figure 2: Speedup with SSE2 and ALP SIMD.

For all applications, the speedups with ALP SIMD are higher than the speedups with SSE2 due to several reasons. First, the latency of most SIMD instructions on ALPSim is 1 cycle whereas all SSE2 instructions have multi-cycle latencies. Further, the 128b SSE2 is implemented as two 64b operations on all Pentium processors that support SSE2, essentially halving the peak throughput when compared to ALPSim. Specifically, FaceRec fails to achieve any significant speedup with SSE2 because FaceRec uses double precision 64b operations. Second, the simulated processor has a different pipeline and hardware resources.Third, ALPSim supports SIMD opcodes that are more advanced than those in SSE2. For instance, horizontal sub-word reductions are available in ALPSim but not in SSE2 (although they are available with SSE3[2]).

4.2.1 SIMD Speedups of Individual Phases with SSE2

All parts of an application do not see the same level of performance improvement with SIMD support. To understand which parts are responsible for the overall SIMD speedup (or lack thereof), Table 2 shows the percentage of execution time and the SSE2 speedup of each phase in each application on P4. The total SSE2 speedup for each application is also given. The sampling error for small phases can be significant and their speedup cannot be measured reliably; therefore, we mark phases with non-SSE2 execution time less than 2% or aggregates of such small phases as N/A. Further, a phase cannot be completely separated from other phases; e.g., instructions from multiple adjacent phases can overlap in an out-of-order processor and branch histories and cache data due to one phase can affect another. Small slowdowns due to SSE2 seen in a few cases in Table 2 are artifacts of such anomalies.

MPGenc and MPGdec see good overall speedups with SSE2. All major phases of MPGenc and all but the VLD phase in MPGdec achieve speedups with SSE2. IDCT of MPGdec and DCT/IDCT phases of MPGenc achieve excellent speedups due to the use of optimized SSE2 code for these phases.

[2]We did not use SSE3 for our applications since it is fairly new and most existing systems do not support it.

	no-SSE2	with SSE2	
	% ExTime	% ExTime	Speedup
MPGenc			
Motion Estimation	64.3	66.3	2.69
DCT/IDCT	9.6	6.3	4.24
Form Predictions	5.6	11.9	1.3
Quant/IQuant	18.8	9.2	5.69
VLC	1.3	3.8	N/A
Other	0.4	2.5	N/A
Total	100.0	100.0	**2.78**
MPGdec			
IDCT	36.6	13.8	5.38
Motion Compensation	41.9	40.4	2.10
- Saturate	8.3	10.4	1.61
- Add Block	9.3	5.7	3.3
- Form Predictions	21.5	20.4	2.14
- Clear Block	2.8	3.9	1.44
VLD	20.3	43.3	0.95
Other	1.2	2.5	N/A
Total	100	100	**2.03**
SpeechRec			
Feature Extraction	1.6	2.1	0.97
Gaussian Scoring	89.0	85.3	1.34
- Vector Quantization	35.4	26.5	1.73
- Short-list Generation	10.5	13.7	0.99
- Gaussian Eval	35.7	34.3	1.34
- Others	7.4	10.8	N/A
Search	7.7	10.5	0.94
Other	1.7	2.1	N/A
Total	100.0	100.0	**1.29**
FaceRec			
Projection	88.0	88.8	1.11
Distance Computation	7.4	5.7	1.47
Other	5.3	6.9	N/A
Total	100.0	100.0	**1.12**

Table 2: Percentage execution time and SSE2 speedup for major phases of each application (except for RayTrace) on P4. Small phases (i.e., phases with non-SSE2 execution time less than 2% or aggregates of such small phases) where the speedup cannot be measured reliably are marked as N/A.

The motion estimation phase of MPGenc achieves very good speedups with SSE2 due to the use of byte operations and the elimination of data-dependent branches using PSAD (packed sum of absolute difference) instructions. Similarly, quantization achieves excellent speedups due to the elimination of branches by using PMAX and PMIN instructions to truncate.

In MPGdec, sub-phases of motion compensation phase like saturate, add block, and form prediction achieve good speedups with SSE2 since they contain straightforward loops with (saturated) additions and subtractions. However, VLD, which is a significant portion of the total application, does not see any speedup resulting in a lower overall speedup than MPGenc.

SpeechRec achieves reasonable speedup with SSE2 (due to its use of 32b single precision floats, the peak possible speedup is roughly 2X on Gaussian scoring). As expected, SIMD instructions lead to significant speedups in the two most dominant sub-phases of the Gaussian scoring phase. However, the overall speedup is limited by phases without DLP.

FaceRec fails to achieve significant speedups with SSE2

due to FaceRec's use of double precision 64b operations as described above. However, it records a small overall speedup due to the elimination of overhead instructions.

4.3 ILP

	ALPSim		P4 (Pentium 4)	
App	**Base**	**SIMD**	**Base**	**SIMD**
MPGenc	1.20	1.23 [4.24]	1.45 (1.87)	0.70 (1.03)
MPGdec	1.38	1.17 [3.31]	1.26 (1.73)	0.73 (1.14)
RayTrace	1.33	N/A	0.48 (0.73)	N/A
SpeechRec	0.35	0.39 [0.67]	0.38 (0.57)	0.34 (0.45)
FaceRec	0.32	0.30 [0.48]	0.51 (0.61)	0.43 (0.47)

Table 3: Instructions per cycle achieved on ALPSim and P4 for single-thread applications. For the ALP SIMD case, the number of sub-word operations retired per cycle is also given within square brackets. For P4, x86 micro-instructions per cycle is given in parenthesis.

Table 3 gives instructions-per-cycle (operations per cycle) achieved on ALPSim and x86 instructions per cycle (micro-instructions per cycle) achieved on P4. Note that the IPC values for ALPSim and P4 cannot be directly compared because they do not use the same instruction set, processor, or memory parameters.

FaceRec and SpeechRec fail to achieve large ILP due to their working sets not fitting in the cache (Section 4.5). Other applications show reasonable ILP on ALPSim. However, the SIMD versions of MPGenc and MPGdec and the base version of RayTrace achieve significantly lower IPC on P4 than on ALPSim. Specifically, for the SIMD versions of MPGenc and MPGdec, as described in Section 4.2, the longer SSE2 latencies and the lack of true 128-bit functional units lower the IPC on P4. For RayTrace, P4 sees lower IPC than ALPSim due to three main reasons: (i) longer latencies and repetition intervals (lower throughput) of the FP units of P4, (ii) the smaller 8K L1 cache of P4 achieves lower hit rates than the 32K L1 of ALPSim (Figure 3), and (iii) branch performance (P4 has a much deeper pipeline, and in RayTrace, 10% of all instructions are branches with a somewhat high misprediction rate of 4%).

P4, however, sees higher IPC for some applications due its lower frequency, L2 hardware prefetcher, and differences in the ISA (e.g., x86 ISA uses far more register spill instructions than SPARC ISA used with ALPSim).

Although SpeechRec and FaceRec have low ILP due to the high memory latencies, we observed that their IPC values become higher (1.5 and 1.3 respectively, with SIMD) when the memory latency is reduced to 42 cycles to simulate a 500 MHz processor or a 500 MHz frequency setting on a processor with frequency scaling. Further, we noticed that, reducing the fetch/retire width from 4 to 2 reduces the IPC of SIMD versions of MPGenc, MPGdec, and RayTrace by 35%, 33%, and 38% respectively. This again underscores the importance of ILP for these applications.

4.4 Interactions Between TLP, DLP, and ILP

App	1T	4T	8T	16T
MPGenc	1.00	0.99	1.01	1.09
MPGdec	1.00	1.12	1.27	1.55
SpeechRec	1.00	1.09	1.23	1.49
FaceRec	1.00	0.97	0.96	0.96

Table 4: Ratio of TLP speedup without SIMD to TLP speedup with SIMD for 1, 4, 8, and 16 threads. A ratio higher than 1.0 shows a reduction in TLP speedup due to SIMD. The data uses ALPSim with a 1 cycle ideal memory system.

4.4.1 Interaction Between TLP and DLP

Most SIMD sections in our applications occur within the parallel (vs. serial) portions of the code. Thus, comparing the SIMD and non-SIMD versions of a threaded application, the parallel portion runs faster in the SIMD version, making the serial portion more dominant in that version. In our applications, the use of SIMD therefore generally reduces TLP speedup.

Specifically, for each application, Table 4 shows the ratio of the TLP speedup of its non-SIMD version to the TLP speedup of its SIMD version (for 1, 4, 8, and 16 threads, obtained on ALPSim using a 1 cycle ideal memory system). A ratio higher than 1.0 shows a reduction of TLP speedup due to SIMD. We see larger ratios for MPGdec and SpeechRec because they have relatively large SIMD-free serial sections and significant SIMD use within the thread-parallel sections. Further, note that the above ratio increases with the number of threads for MPGdec and SpeechRec, limiting their TLP scalability in the presence of SIMD. This has significant implications for the TLP scalability of emerging multi-core/multi-threaded commercial processors that already support SIMD instructions. The above ratio stays close to 1 for MPGenc and FaceRec since they do not have large serial sections.

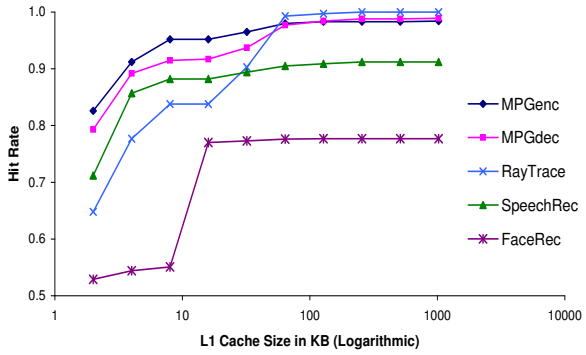

Figure 3: L1 cache hit rates. All applications except RayTrace use the SIMD version.

4.4.2 Interaction Between DLP and ILP

Exploiting SIMD instructions in a given piece of code should theoretically reduce the amount of ILP present in that section of code since one SIMD instruction can replace multiple independent non-SIMD instructions. Table 3 shows this

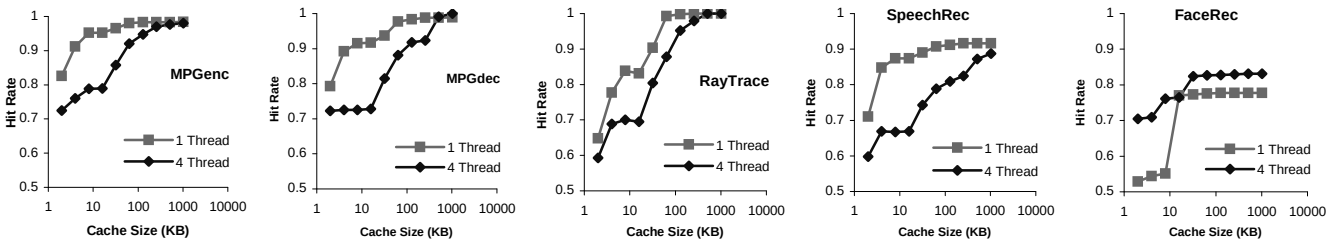

Figure 4: L2 cache hit rates (with L1 caches disabled). All applications (except RayTrace) use the SIMD version.

decrease for all applications except MPGenc and SpeechRec with ALPSim.

The exceptions occur for multiple reasons. First, SIMD code can provide increased resource usage relative to non-SIMD code. For example, in ALPSim, (and P4), SIMD instructions use the FP pipeline and FP functional units (thereby increasing resource utilization for integer code). Second, SIMD instructions can reduce contention to critical processor resources (e.g., load/store queue entries, cache ports) by combining several non-SIMD instructions into one instruction. Third, some SIMD instructions can reduce data dependent branches that are usually hard to predict and compromise ILP in non-SIMD code (e.g., packed sum of absolute difference or PSAD, packed maximum, and packed minimum). SIMD code also reduces the number of conditional branch instructions by reducing the number of loop iterations and the branches associated with them.

4.4.3 Interaction Between TLP and ILP

The interaction between TLP and ILP is well known. Specifically, with TLP, the presence of multiple threads could change cache behavior, affecting ILP. As discussed later, multiple threads can cause both positive and negative cache effects (Figure 4). For instance, in our CMP system, an application can see a reduction in ILP if there is no constructive sharing in the shared L2 cache (reducing the effective L2 size for each thread) or if there is false sharing in the L1 cache. Conversely, if there is constructive sharing in the L2 cache, then ILP is increased as each thread sees some prefetching effects from the accesses of other threads.

Table 5 gives the percentage reduction of per thread IPC in a 16-thread CMP with respect to the IPC of a single-thread processor. The IPC does not include the effect of synchronization instructions. We see that multiple threads cause a small to modest reduction in per-thread IPC due to the effects discussed above. This reduction is smallest in FaceRec due to some constructive data sharing among threads in the L2 cache (Section *4.5.1*).

4.5 Sensitivity to Memory Parameters

As expected, the parallelism of our applications is sensitive to the memory parameters. To understand this sensitivity, we next report the cache hit ratios/working sets of our applications, how our applications scale with increasing processor frequency (memory latencies), and the memory bandwidth requirement with increasing number of threads.

App.	MPGenc	MPGdec	RayTr.	Sp.Rec	FaceRec
IPC Reduc.	6.1%	8.8%	10.6%	7.8%	2.7%

Table 5: Percentage reduction of per thread IPC in the 16-thread CMP with respect to the IPC of a 1-thread processor (with realistic memory system). The IPC does not include synchronization instructions.

4.5.1 Working Sets

Figure 3 gives the L1 data cache hit ratios obtained using ALPSim with SIMD for different L1 cache sizes (2K to 1024K). Using the concepts described in [35], all applications, except FaceRec, have first-level working sets about 8KB since the first knee of all hit-rate curves occurs around 8KB. FaceRec has the first-level working set of 16KB. Both RayTrace and MPGdec can further benefit significantly from a cache size up to 64KB. MPGenc sees a slight benefit if the cache size is further increased to 64KB. FaceRec and SpeechRec, however, do not benefit much from increasing the cache size after 8KB and 16KB, respectively (up to 1MB).

Figure 4 shows the shared L2 cache hit rates for different cache sizes (2K - 1024K) for both single-thread and 4-thread versions of each application with SIMD. The L1 caches were disabled for this experiment to study the effect of data sharing between multiple threads in L2. If the threads share a significant portion of data and the single thread version achieves a given hit rate with x KB, the 4-thread version should be able to achieve the same or a better hit rate with $4x$ KB of cache. From Figure 4, we see that only threads in FaceRec share a significant portion of data since the 4-thread version achieves better hit rates than the single-thread version for a given cache size. This is because the threads in FaceRec share parts of the large subspace matrix (database). For the other applications, the data is mostly partitioned among the threads and then mostly exclusively accessed by one thread.

Even FaceRec, which exhibits some data sharing, does not share all the data in L2 since a significant portion of its memory accesses still have to go to memory.

To summarize, first we see that three of our applications have very good cache hit rates whereas two have low hit rates. Low cache hit rates reduce ILP when memory la-

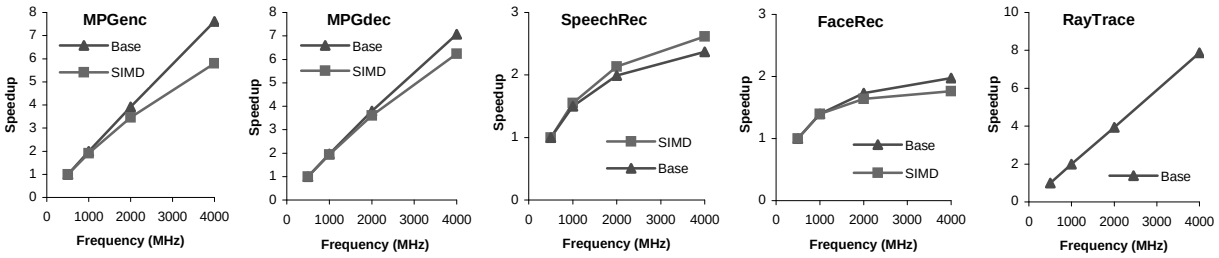

Figure 5: Frequency scalability for single thread applications. The SIMD data are with ALP SIMD. RayTrace does not have a SIMD version.

tencies are high. We also see that increasing TLP demands larger caches to accommodate the working sets of our applications since their threads do not share much data.

4.5.2 Sensitivity to Memory Latency or Processor Frequency

Figure 5 shows the speedup achieved by the base (non-SIMD) and SIMD versions of each single-thread application on ALPSim when the processor (die) frequency is scaled from 4 GHz to 500 MHz. Consequently, the time (in processor cycles) to access memory decreases linearly from 240 cycles (at 4 GHz) to 30 cycles (at 500 MHz) (i.e., the L2 miss latency is the memory/bus access time plus 16 cycles for all frequencies). The other parameters given in Table 1 are not changed. Specifically, the parameters used at 4 GHz are identical to those given in Table 1. Speedups reported for non-SIMD (SIMD) are with respect to the non-SIMD (SIMD) single thread version of the application running at the lowest frequency (500 MHz). Such frequency scaling data is important since many systems, especially mobile systems running these media applications, run at lower frequencies. Further, many such systems employ dynamic frequency scaling to run at lower frequencies than the maximum frequency supported by the processor to reduce power and energy consumption. The following describes how each application scales when the frequency is *increased* from the lowest (500 MHz) to the highest (4 GHz).

Figure 5 shows that the base cases of RayTrace, MPGenc and MPGdec scale well with increasing frequency since most of their working sets fit in the caches. The base cases of FaceRec and SpeechRec show poor scalability after 1 or 2 GHz. This is mainly due to their larger working sets not fitting in caches (Figure 3).

Two factors affect the relative scalability between SIMD and non-SIMD versions of the same application. On the one hand, the SIMD version has a lower computation to memory ratio than the base case and hence is more sensitive to longer memory latencies. This is because the SIMD case reduces loop overhead and address calculation overhead instructions, which are compute instructions. This effect causes the SIMD versions of MPGenc, MPGdec, and FaceRec to show lower scalability. The effect is more prominent in MPGenc due to its use of small sub-words; in that case, SIMD can reduce the loop iteration and overhead significantly. On the other hand, the SIMD version exposes more memory level parallelism to the out-of-order core – since the SIMD loops use fewer instructions per loop, the instruction window can contain a larger number of load instructions than possible in the non-SIMD case. This effect gives the SIMD version better scalability when the application has a significant L2 miss rate. Specifically, SpeechRec benefits from this effect. Although, the SIMD version of FaceRec should benefit from the same effect since it has high L2 miss rates, the increase in memory level parallelism for the SIMD version of FaceRec is low due to its use of double precision operations. That is, the number of additional loop iterations that can fit in the instruction window is not as large as that for SpeechRec.

To summarize, three out of five of our applications scale well with frequency. We see that higher memory latencies affect applications with and without SIMD differently. SIMD versions of all our applications except SpeechRec show somewhat poorer scalability with increasing frequency than their non-SIMD counterparts.

4.5.3 Memory Bandwidth

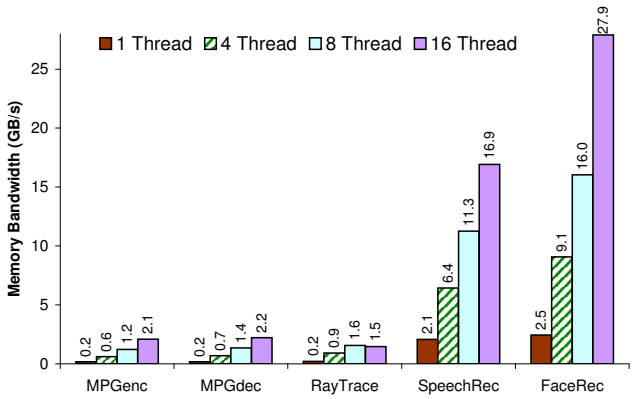

Figure 6: Memory bandwidth (in GB/s) at 4 GHz without SIMD.

Figure 6 shows how memory bandwidth demand increases for each application (non-SIMD) with the number of threads. The results were obtained on ALPSim without ALP SIMD using the parameters in Table 1 (with the processor at 4GHz). MPGenc, MPGdec, and RayTrace have relatively low bandwidth requirements since they have smaller working sets. However, FaceRec and SpeechRec demand much larger memory bandwidth since their working sets do not

Application	Performance
MPGenc	21.8 fps (704x480 DVD resolution)
MPGdec	166.3 fps (704x480 DVD resolution)
RayTrace	0.75 fps (512x512 resolution)
SpeechRec	9.0 words/sec.
FaceRec	152.1 130x150 images/sec.

Table 6: Application-level real-time performance obtained by single threaded versions of our applications on the P4 system.

fit in the L2 cache. The increase in bandwidth generally follows the TLP speedup of applications. For all applications that have DLP, SIMD versions will demand more bandwidth since they execute faster. These results show that the bandwidth of MPGenc, MPGdec, and RayTrace can be fulfilled by existing memory systems (assuming a maximum of 8.5 GB/s memory bandwidth on current personal computers with DDR2 memory). However, for SpeechRec and FaceRec, CMP systems that support 8 or more threads will have to support a higher memory bandwidth than supported today on many general-purpose systems [15].

4.6 Application-Level Real-time Performance

Table 6 shows the application-level real-time performance results for each application on P4 (Section 3). The results are for single-threaded applications with SSE2 (except for RayTrace). The approximate performance for systems with a higher number of threads and lower frequencies can be derived using the thread/frequency scaling results presented earlier. MPGdec already achieves the required real-time performance on current systems. The performance of RayTrace is far from real-time. Although MPGenc comes close to the required real-time performance of 30 frames per second, larger inputs (e.g., HDTV) will demand much higher performance. Similarly, although SpeechRec and FaceRec achieve reasonable performance with the given small input sets, much larger inputs anticipated in the future (e.g., much larger dictionaries for SpeechRec, higher resolution image/face recognition used with personal/database search, and much larger image databases) will demand much higher processing capabilities.

5. RELATED WORK

There have been many studies that characterize the individual applications used in ALPBench.

Several papers characterize MPEG-2. Chen et al. [4, 10] characterize various phases of MPGdec on a real system and discuss and evaluate slice assignment policies, and data vs. functional partitioning for parallelization. They also accelerate MPGdec with SSE instructions. However, they do not perform a thread-scaling or a frequency scaling study. We also modified MPEG encoder to use an intelligent motion search algorithm and to use an optimized algorithm for discrete cosine transform. Iwata et al. [16] propose a number of coarse-grained parallel implementations of MPEG-2 decoding and encoding. They evaluate the performance of these implementations on a multiprocessor, compare the performance against a single and wide issue superscalar processor, and report results with multi-threading for 4 and 8 processors. They also find that thread scalability of MPGenc is better than that for MPGdec. However, they report 8 thread results only with a single issue processor. We do the TLP scalability study up to 16 threads using the same processor configuration. They also report 50% speedup with MIPS based SIMD instructions for MPGdec. We report SSE2 speedups for both MPGenc and MPGdec; we are able to achieve much higher speedups (2X) with SSE2 for MPGdec. We also perform a frequency scalability study.

Several researchers have discussed the Eigenface face recognition algorithm used in this study [2, 23, 32]. Specifically, Mathew et al. [23] characterize architectural features such as cache hit rates and IPC on several embedded architectures. In addition to such characterizations, we analyze the thread and SIMD parallelism of this application. Vorbruggen [33] describes a similar face recognition algorithm used with SPEC CPU2000 but does not perform an evaluation.

Ravishankar [27] describes the algorithms, data structures, inputs/outputs of Sphinx3 used with this study, but does not provide any evaluation. Mathew et al. [24] provide a detailed analysis of Sphinx3. They identify the three distinct processing phases (Section 2.5) and quantify the architectural requirements for each phase. They also describe the large memory footprint and find the Gaussian and search phases to be the dominant ones. They also developed a parallel version of Sphinx3 that runs the three major phases (i.e., feature recognition, Gaussian scoring, and search) using three threads and report a speedup of 1.67. Instead of this type of functional partitioning, we parallelize Sphinx3 using data partitioning (i.e., each phase are divided into N symmetric threads). This method gives better speedups and is more scalable. Mathew et al. also develop a special-purpose accelerator for the dominant Gaussian scoring phase. The accelerator consists of specialized multipliers and adders to perform the specific multiply accumulate operation done in the inner loop of Gaussian scoring. To overcome the latency of FP multiply accumulate operations, they pipeline multiple independent iterations. Instead of using a separate co-processor, we use the SIMD units to exploit the DLP in the Gaussian scoring phase.

Baugh et al. characterize and parallelize Sphinx2 [1][3]. They divide Sphinx2 into multiple phases and use work queues in between phases. Then they use asymmetric threads to execute each phase. They also investigate using symmetric threads within each phase. In contrast, we use symmetric threads that span both Gaussian scoring and search phases and do not use work queues. They report speedups up to 2.7X using both asymmetric and symmetric threads (a total of 6 threads). They also show preliminary results where they achieve speedups up to 6.8X with

[3]Sphinx2 has somewhat different Gaussian models than those used in Sphinx3 [20].

10 threads. They do not investigate exploiting SIMD in this study.

Krishna et al. [19] analyze parameters affecting the performance of Sphinx2 with special emphasis on the memory system. They also find poor cache performance (Figure 3), poor memory reference predictability, and potential for using multiple threads albeit with higher demands on the memory system. Based on the insights from that work, they propose architectural SMT techniques to exploit the TLP in Sphinx3 [20]. They develop an architecture with multiple speech processing elements that are capable of generating their own threads and report good speedups (e.g., approximately 12X speedup with 16 speech processing elements and 4 thread contexts per processing element). They also perform partitioning of search tree nodes; for thread creation and synchronization, they use a fork/join model with a special barrier instruction (called EPOCH) and locks. We use a similar approach to exploit TLP and also investigate the use of sub-word SIMD instructions.

Woo et al. [35] present a characterization of a different version of RayTrace with other applications introduced with the SPLASH-2 benchmark. They also report working set characteristics similar to those reported here and also report good TLP scalability. However, they do not study the scalability of RayTrace with frequency and do not consider SIMD instructions. Nguyen et al. [26] also characterize RayTrace provided with SPLASH-2 as part of their work in evaluating multiprocessor scheduling policies. They also study the TLP scalability and identify the sources of speedup loss. However, they do not study working sets or frequency scalability of this application.

In addition to the above studies that focus on the individual applications, there have been several multimedia benchmark suites that target applications included in ALPBench and report their characteristics. These include MediaBench [21], Berkeley multimedia workload [30], MiBench [8] and EEMBC [7]. All these suites include MPEG encoder and decoder. Additionally, MiBench includes the Sphinx2 speech recognizer, and both MediaBench and Berkeley multimedia workload contain the RASTA speech recognizer. Berkeley multimedia workload also includes the POVray3 ray tracer as part of the suite. While all of the studies report single thread workload characteristics of the media applications, they do not characterize DLP and TLP.

Finally, in extended versions of this work, we also characterize the behavior of the ALPBench applications for other forms of DLP; i.e., conventional vectors [22] and a recently proposed specialized form of vector/stream support called SVectors and SStreams [29].

6. CONCLUSION

Complex media applications are becoming increasingly popular on general-purpose systems such as desktops, laptops, and handheld systems. This paper presents a publicly released suite of five such complex media applications and characterizes the parallelism and performance of them.

Our characterization shows that these applications have multiple levels of parallelism - TLP, DLP, and ILP. For TLP, we find that all our applications have coarse-grain TLP and most of them show good thread scalability. As for DLP, we find that these applications produce good speedups with sub-word SIMD. We also study the interaction between two forms of parallelism and find that exploitation of DLP could reduce effectiveness of TLP. Further, we also study the effects of the memory system on these applications, and report the different working sets, bandwidth requirements, and memory latency tolerance in these applications. Our characterization of parallelism can be used by processor/system architects and compiler writers to provide better support for complex media applications.

7. REFERENCES

[1] L. Baugh, J. Renau, J. Tuck, and J. Torrellas, "Sphinx Parallelization," Dept. of Computer Science, University of Illinois, Tech. Rep. UIUCDCS-R-2002-2606, 2002.

[2] R. Beveridge, D. Bolme, M. Teixeira, and B. Draper, "The CSU face identification evaluation system user's guide," http://www.cs.colostate.edu/evalfacerec/algorithms/version5/faceIdUsersGuide.pdf, 2003.

[3] R. Beveridge and B. Draper, "Evaluation of face recognition algorithms," http://www.cs.colostate.edu/evalfacerec/, 2003.

[4] Y.-K. Chen *et al.*, "Media Applications on Hyper-Threading Technology," *Intel Technology Journal, Vol.6, Issue 1*, 2002.

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

[6] P. Dubey, "Recognition, Mining and Synthesis Moves Computers to the Era of Tera," *Technology@Intel Magazine*, February 2005.

[7] EDN Embedded Microprocessor Benchmark Consortium, "The EEMBC benchmark suite," 1997.

[8] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE 4th Annual Workshop on Workload Characterization*, December 2001.

[9] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2002.

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

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

[12] *Intel Application Notes AP-922*, Intel Corporation, 1999.

[13] *Intel Application Notes AP-945*, Intel Corporation, 1999.

[14] *The IA-32 Intel Architecture Optimization Reference Manual*, Intel Corporation, 2004.

[15] Intel Corporation, "Intel 925XE express chipset," http://www.intel.com/products/chipsets/925xe/, 2005.

[16] E. Iwata and K. Olukotun, "Exploiting Coarse-Grain Parallelism in the MPEG-2 Algorithm," Computer Systems Lab, Stanford University, Tech. Rep. CSL-TR-98-771, 1998.

[17] H. Kalva, A. Vetro, and H. Sun, "Performance Optimization of an MPEG-2 to MPEG-4 Video Transcoder," in *Proc. of SPIE Conf. on Microtechnologies for the New Millennium, VLSI Circuits and Systems*, 2003.

[18] K. I. T. Koga, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-Compensated Interframe Coding for Video Conferencing," in *Proc. of the 1981 National Telesystems Conference*, 1981.

[19] R. Krishna, S. Mahlke, and T. Austin, "Insights Into the Memory Demands of Speech Recognition Algorithms," in *Proc. of the 2nd Annual Workshop on Memory Performance Issues*, 2002.

[20] ——, “Architectural Optimizations for Low-Power, Real-time Speech Recognition,” in *Proc. of the Intl. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, 2003.

[21] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, “MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems,” in *Proc. of the 29th MICRO*, 1997.

[22] M.-L. Li, “Data-Level and Thread-Level Parallelism in Emerging Multimedia Applications,” Master’s thesis, Univ. of Illinois, Urbana-Champaign, 2005.

[23] B. Mathew, A. Davis, and R. Evans, “A Characterization of Visual Feature Recognition,” Univ, of Utah, Tech. Rep. UUCS-03-014, 2003.

[24] B. Mathew, A. Davis, and Z. Fang, “A Low-Power Accelerator for the SPHINX 3 Speech Recognition System,” in *Proc. of the Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, 2003.

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

[26] T. D. Nguyen, R. Vaswani, and J. Zahorjan, “Parallel Application Characterization for Multiprocessor Scheduling Policy Design,” in *Job Scheduling Strategies for Parallel Processing, Volume 1162 of Lecture Notes in Computer Science, Springer-Verlag*, 1996.

[27] M. K. Ravishankar, “Sphinx-3 s3.X decoder,” http://cmusphinx.sourceforge.net/sphinx3/, 2004.

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

[29] R. Sasanka, M.-L. Li, S. V. Adve, Y.-K. Chen, and E. Debes, “ALP: Efficient Support for All Levels of Parallelism for Complex Media Applications (Submitted for publication),” Dept. of Computer Science, University of Illinois, Tech. Rep. UIUCDCS-R-2005-2605, July 2005.

[30] N. T. Slingerland and A. J. Smith, “Design and Characterization of the Berkeley Multimedia Workload.” *Multimedia Syst.*, vol. 8, no. 4, 2002.

[31] J. E. Stone, “Taychon raytracer,” http://jedi.ks.uiuc.edu/~johns/raytracer/, 2003.

[32] M. Turk and A. Pentland, “Face Recognition Using Eigenfaces,” in *Journal of Cognitive Neuroscience, Vol. 3*, 1991.

[33] J. C. Vorbruggen, “187.facerec: CFP2000 benchmark description,” http://www.spec.org/osg/cpu2000/CFP2000/, 2000.

[34] 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.

[35] S. C. Woo *et al.*, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” in *Proc. of the 22th Annual Intl. Symp. on Comp. Architecture*, 1995.