Simultaneous Multithreaded DSPs: Scaling from High Performance to Low Power

Stefanos Kaxiras, Alan D. Berenbaum, Girija Narlikar
Bell Laboratories, Lucent Technologies
{kaxiras,adb,girija}@research.bell-labs.com

Abstract*—In the DSP world, many media workloads have to perform a specific amount of work in a specific period of time. This observation led us to examine how we can exploit Simultaneous Multithreading for VLIW DSP architectures to: 1) increase throughput in situations where performance is the most important attribute (e.g., base station workloads) and 2) decrease power consumption in situations where performance is bounded by the nature of the workload (e.g., wireless handset workloads). In this paper we discuss how we can multithread a commercial DSP architecture. We study its performance and power characteristics using simulation, compiled code, and realistic workloads that respect real-time constraints. Our results show that a multithreaded DSP (with no additional function units over the base architecture) can easily support a small number of compiled threads without seriously degrading their individual performance. Furthermore, in situations where the required performance is bounded, we show that a multithreaded DSP can perform as well as a non-multithreaded DSP but operating at a reduced clock frequency and low supply voltage (V_{dd}) with substantial power savings.*

1 Introduction

Communications are fast becoming one of the highest-volume applications of microprocessors—more specifically Digital Signal Processors (DSPs). According to market estimates, the DSP sector is projected to experience significant growth in the near future [1]. At the same time DSP architectures undergo significant changes. New developments and new standards in communications call for a significant increase in DSP performance while at the same time the mobile device market demands ever lower power consumption. Developers are also moving from hand-written code to compiled code. We are investigating simultaneous multithreaded VLIW DSPs as a promising direction to satisfy all these diverse demands.

In the near future we will see a significant increase in DSP processing requirements because of the transition to the third generation (3G) wireless technology pursued by the International Telecommunications Union (ITU). 3G wireless systems are based on Wideband Code Division Multiple Access (WCDMA) technology [2]. These systems will enable advanced mobile communications services in all five continents. They will simultaneously provide voice (phone calls), data (access to the Web, email), and video services. For wireless 3G multimedia applications, the IMT2000 committee of the ITU has proposed a roadmap for WCDMA that will enable data rates

of up to 384 kilobits per second (Kbps) in mobile applications and up to 2 megabits per second (Mbps) in stationary applications [1]. For mobile clients this means that a significant increase in performance must be provided at ever lower power consumption levels. Before such mobile devices appear, a third-generation wireless infrastructure must be deployed to accommodate the underlying WCDMA technology. Fundamental to the infrastructure are the countless wireless base stations that handle all communication among the mobile clients and provide the connection to optical, satellite, or microwave networks. The new generation of base stations will need to handle greater capacity, process higher data rates, and support multimedia data and video. However, at the same time we want base stations to be smaller (for easier installation), cheaper (for rapid deployment), and less power hungry (for simpler packaging and increased reliability).

Another change in the DSP world is the transition from hand-written code to compiled code. Communication standards are becoming more complex as more features are added and sophisticated wireless technology is used [1]. Voice and video standards are now specified by international committees in C [12]. The complexity of the standards code and the requirement for bit-exact output make hand-written code a difficult proposition. Compiled code becomes appealing when the benefits (reduced development time and cost) start to become more valuable than the drawback (reduced performance). Using multithreading to improve the performance of a *workload* rather than the performance of a single thread ameliorates the drawback of compiled code.

Multithreading was originally proposed as a way to increase throughput for a workload by hiding long latencies [5]. More recently Tullsen, Eggers and Emer proposed simultaneous multithreading (SMT) to increase utilization of out-of-order superscalar processors [3][4]. What makes SMT appealing in that context is that the same hardware mechanisms that support out-of-order execution can be used to handle multiple simultaneous threads [4].

In the DSP arena, VLIW [6] rather that out-of-order superscalar architectures have prevailed for simplicity and chip area reasons. Leading DSP architectures such as the TI 320C6x [7] or the Star*Core SC140 [8] leverage VLIW technology to provide multiple operations per cycle. In this paper we propose a SMT VLIW architecture using the Star*Core SC140 DSP as starting point. We provide replicated thread state (e.g., multiple register files) but we share a single set of function units among all threads. In each cycle we select multiple (variable length) instruction packets from ready threads—as many as we can accommodate—and assign them to the function units.

We simulate multithreaded DSP architectures running workloads consisting a mix of speech

encoders/decoders (GSM EFR), channel encoders/decoders (Trellis modulated channel encoding) and video encoders/decoders (MPEG-2). Our workloads approximate real-world processing in base stations and cell phones by respecting real time constraints. Our results show:

- A multithreaded DSP (with just the function units of the base DSP) can easily run a small number of compiled threads without significantly degrading their performance. By adding more load/store units—which are the bottleneck in our workloads—performance improves further. We found a small advantage in cost/performance over a chip-multiprocessor (CMP) DSP.
- Despite the increased complexity and utilization of a multithreaded DSP, in some cases we can use it to reduce power consumption. We show how we can exploit the high IPC of the multithreaded architecture to reduce clock frequency and voltage and thus *reduce* power (and conserve energy), in situations (such as in wireless handsets) where the required performance is bounded. Power consumption for the same workload can be reduced by a factor of 4 over a single-threaded DSP or by a factor of 1.47 over a CMP DSP also running at low frequency and low voltage.

Structure of this paper —In Section 2 we describe the base architecture and the multithreaded DSP architecture. In Section 3 we discuss our evaluation methodology and in particular our benchmarks, the simulator, and the compiler we use. Section 4 contains the results of our study for base station workloads (with an emphasis on throughput/cost) and Section 5 for cell phone workloads (with an emphasis on power). Finally we conclude with a summary in Section 6.

2 Multithreaded VLIW DSPs

Traditional DSPs have attempted to exploit the data parallelism inherent in signal processing algorithms through the use of compact instructions that combine data operations (such as multiply-accumulate—MAC) with iteration or other control flow operations. This approach assumes that a compiler cannot easily uncover the inherent parallelism in the algorithms, and this class of signal processors is typically hand-coded in assembly language for maximum efficiency. With sufficient coding effort, the result can be very compact code that helps make these processors energy efficient.

In the 1980’s compilers had become more effective in determining the parallelism in a program, and so a reasonable hardware-software trade-off was to construct hardware with many par-

allel data units but very simple instruction issue logic. A very long instruction word (VLIW) directs data flow to all the parallel data units simultaneously as instructed by the compiler, in lieu of more complex issue logic attempting to uncover parallelism at runtime [5]. As the implementation of microprocessors has become increasingly complex, the idea of moving some of that complexity onto the compiler has become more popular, and so a number of recent general purpose microprocessor architectures feature some variation on VLIW architecture [24][20].

The same trends have affected the DSP world. The simple issue logic required for VLIW designs has inspired several DSP architectures that claim very high peak operations-per-second [21][7][8]. The peak throughput is unlikely to be realized in real-world applications, since the compiler will not always be able to find independent operations that can use all available function units every cycle. In addition, there are several disadvantages that result from the long instruction word architecture. Because the long instructions specify many simple operations, instead of a few complex or compound functions, VLIW DSPs make less efficient use of code memory. The increased memory traffic due to larger code footprint, plus the wide internal busses and function units running in parallel, mean that VLIW DSPs consume more power to perform the same work as more traditional DSPs.

The base architecture we have chosen for this study is similar to the Star*Core developed by Motorola and Lucent Technologies [8]. The architecture contains four data ALUs (DALUs), which can perform single-cycle multiply-accumulates, two address AGUs (Address Generation Units) and a single bit-manipulation unit (BFU). There are 32 registers (16 data and 16 address registers). A single variable-length instruction packet can issue up to six operations to these function units in one cycle.

For application workloads that consist of several independent threads, the throughput of a DSP can be increased by replicating the core DSP engine on a single die [9] yielding a chip-multiprocessor (CMP) structure [10]. This same technique has been used in a number of network processors to obtain the throughput necessary to process broadband trunks [22]. However, the utilization of function units is likely to be low, since not all threads will be able to issue maximum-width operations simultaneously and so on any cycle at least some of the function units will be idle.

An alternative to the CMP structure is Simultaneous Multithreading (SMT) [3][4][25]. Independent instruction issue units, including the register arrays that comprise a thread’s state, simul-

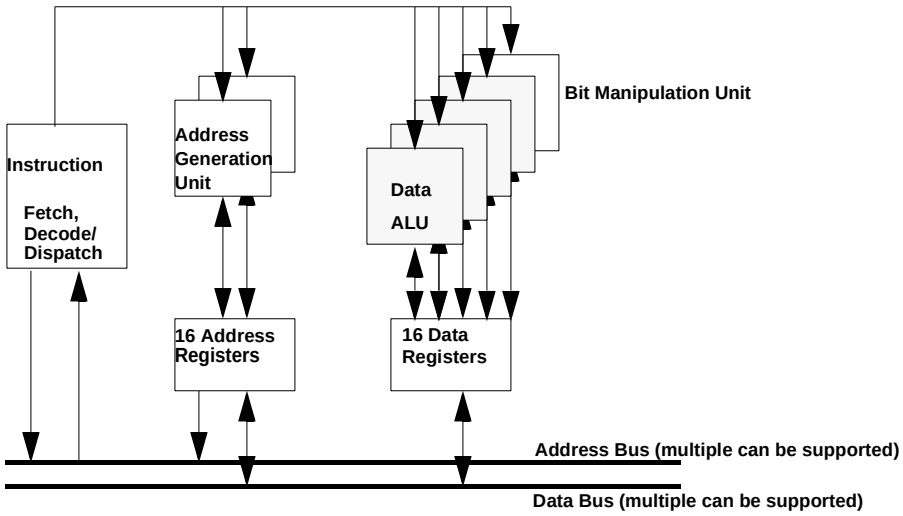

FIGURE 1. Block diagram of our base architecture (similar to Star*Core SC140)

taneously issue commands to a single shared set of function units. For the same number of register sets SMT architectures require fewer execution units than the aggregate number used in CMP designs, and can potentially make more efficient use of resources that are available. For example, recent work studied the application of SMT to network processors [23]. Because an SMT design makes more efficient use of a large number function units the compiler can in principle issue wider instructions during the relatively rare times a very high degree of parallelism is detectable, without the hardware cost of supplying function units that remain idle almost all the time. We did not address this potential advantage of SMT's in this study; for both CMP and SMT alternatives we used the same compiler, which generates a maximum of six operations in parallel. Therefore, a single thread cannot take advantage of a machine wider than our base VLIW.

Our model SMT architecture resembles the base architecture in that it retains the same function units and instruction issue logic. However, we can increase the number of data ALU's, address ALU's and bit manipulation units arbitrarily. The largest additional cost in implementing the SMT version of the base architecture is routing data to and from the register arrays to the function units and—in accordance to the SMT work [4]—we allowed an additional pipeline stage over the base model to account for wire and setup delays.

In the additional pipeline stage, we decide which threads will issue instructions to the ALUs. If threads have different priorities, higher priority threads get to issue first; low priority threads scavenge the leftover slots. Within a priority level we implement round-robin scheduling. In our

experiments a high priority thread will not be affected by any number of lower priority threads which nevertheless do make forward progress. In Section 5 we use priorities to meet real-time deadlines for speech encoding. Multiple threads can issue their VLIW packet as long as there are no resource conflicts. As in the base architecture, a thread holds on to its resources until its longest running instruction finishes. An optimization here would be to release resources as soon as possible for use by *other* threads. This improves performance with a small increase in issue logic complexity and bookkeeping state. A more aggressive optimization is to split instruction packets and issue individual instructions in cases we have resources available but not enough for the entire VLIW packet. This, in general, violates VLIW semantics[1] and requires either compiler support or run-time checking. In this work we examine the simplest SMT DSP architecture without the benefit of these optimizations.

We have also examined caches which are not prevalent in the DSP world. Even with caches the behavior of the multithreaded DSP in relation to the base DSP remained qualitatively the same. As in the SMT work [4] we did not observe thrashing with the DSP workloads we used. Because of the streaming nature of the data the effects of multithreading are minimal in the caching behavior of these programs. For the rest of this work we will assume no caches but only a memory system consisting solely of on-chip memory. Memory latency is 1 cycle and we model contention in the memory system through the two address ALUs.

3 Methodology

In this section we describe the DSP benchmarks used in both our handset and base station experiments. We then present the experimental setup used to compile and run the benchmarks.

3.1 Benchmarks

Our choice of benchmarks is intended to model next generation wireless handsets that supports multimedia (voice and video) applications, as well as wireless base stations. Real-time constraints play an important role in this domain, especially for voice-based applications. It becomes critical to ensure that their performance is not affected by other, non-real-time applications that are running on the same DSP. Therefore, instead of evaluating the performance of each multimedia benchmark in isolation, we have designed a set of benchmarks that run simultaneously on the

[1] For example a VLIW packet which writes some of its input registers cannot be split arbitrarily.

DSPs while obeying real-time constraints. For both the handset and the base station, we measure the power and performance for runs of one second duration.

For the handset, we assume the speaker is communicating over a video-enabled phone. The following benchmarks will run simultaneously in such a scenario:

- **Speech encoder and decoder.** Speech coding is used in cellular telephony for reducing the bit rate of the transmitted voice signal. A speech encoder transforms the original digital speech signal into a low-rate bitstream using lossy compression. The decoder restores this bitstream to an approximation of the original signal. The encoded bitstream is represented in fixed-length packets (frames); each frame must be generated by the encoder every few milliseconds, depending on the bit-rate. The decoder has similar real-time deadlines for decoding each input frame.

  We have used bit-exact C code for a GSM standard speech coder, namely, the Enhanced Full Rate (EFR) coder [11][12]. The standard requires a bit rate of 12.2Kbits/sec, with 244-bit frames transmitted every 20ms (such a bit rate is appropriate for wireless handsets). We therefore run the encoder and decoder on one frame of data every 20 ms, ensuring that it completes within the 20ms deadline.
- **Channel encoder and decoder.** The output of the speech encoder is fed into a channel encoder (see Figure 2). The channel encoder adds redundancy in a controlled manner, to protect against noise and interference during transmission. The output of the channel encoder is fed to a channel modulator, which interfaces to the communication channel. The channel encoder we used also includes a channel modulator, and performs trellis-coded modulation [15]. At the receiver's end, the digital demodulator takes in the waveform received from the channel, and approximates it with a digital sequence. This sequence is decoded by the channel decoder, and then fed to the speech decoder. The channel decoder that we run uses Viterbi's algorithm [16] and also performs digital demodulation.

  To match the real-time requirements of the speech coder, we run both the channel encoder and decoder once every 20 milliseconds, for one frame's worth of input. We start the speech and channel coders at the start of every 20 ms slot.
- **Video encoder and decoder.** A video phone can transmit and receive a video signal between speakers, and requires a coding scheme to compress the signal. We use an implementation of the MPEG-2 encoder and decoder provided by the MPEG Software Simulation Group [17].

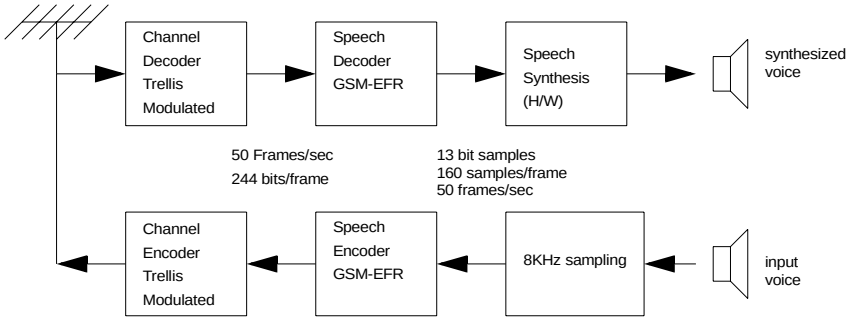

FIGURE 2. Speech processing in a cellular phone.

The MPEG encoder is the most computationally intensive of all our applications. Therefore, we picked image sizes and frame rates based on the capability of the DSP processor when running in single-threaded mode. The image sizes are 32 x 32 for the slower handset technology, and 64 x 64 for the faster technology (see Section 5 for details). The MPEG-2 coders were designed for high bit rates (and hence high frame rates). We had to reduce the frame rate to be more suitable for a handheld video phone; the code we run encodes only 2 frames per second. In the future, we plan to switch to a video coder designed specifically for lower bit rates. (Preliminary experiments with H.263, a low bit-rate coder, indicate that it results in very similar IPC performance as the MPEG-2 code used in this paper.) The decoder decodes a 64 x 64 image at 3 frames per second.

The MPEG encoder consumes significantly more cycles than any of the other benchmarks, and also suffers from low IPC. Therefore, to effectively reduce total running time by making better use of multiple execution units, the encoder needs to be explicitly parallelized at a coarse level. We do not currently support intra-process parallelism in our simulator. Instead, we approximate a parallel encoder with 4 independent MPEG encoder threads; each thread processes one quarter of the original image size.

All the above six codes are adapted from sample implementations of industry standards, and were not modified to aid optimization by the compiler. For the single-threaded experiments, we run 6 threads: an MPEG encoder and decoder, a speech encoder and decoder, and a channel encoder and decoder. In the experiments with the multithreaded core, we run 9 threads (three additional MPEG encoders); each of the 4 MPEG encoder threads processes a quarter of the original image. In both sets of experiments, the threads are run for a one second duration. The speech

and channel encoders and decoders repeat for one frame of data every 20ms. The MPEG threads run from start to completion. A base station does not need to perform any encoding or decoding of the video signal; therefore, we run only the 4 speech and channel coding threads.

3.2 Compiler

We use the Enterprise C/C++ compiler designed for the commercial SC100 family of DSP cores [14]. The compiler performs various high-level optimizations, as well as low-level, machine-dependent optimizations. The machine-dependent phases convert linear assembly code into parallel (VLIW) code, that can make good utilization of the SC140 architecture described in Section 2. The Enterprise compiler achieves close to 45% of the performance of hand-coded assembly, and nearly 90% with a very small amount of assembly code added to C for a typical DSP code[19]. All the binaries used in our experiments were generated from pure C. As shown in Section 4.1, the serial versions of the applications have only low to moderate amounts of instruction-level parallelism that can be extracted by the compiler.

The compiler compiles floating point arithmetic down to fixed-point instructions using emulation libraries. The MPEG codes, which make extensive use of floating point arithmetic, hence result in binaries containing a large number of fixed-point instructions with serial dependencies. Therefore the MPEG codes suffer from particularly low IPC.

3.3 Simulation environment

The experiments were carried out using a cycle-accurate, instruction-level simulator [18]. The simulator emulates the SC140 DSP core, including exception processing activity. The SC140 core for which the simulator was originally designed has a fixed number of AGUs (2) and MACs (4); the C/C++ compiler assumes this hardware configuration.

We have extended the original simulator by adding support for multiple threads, each with a separate address space and register file. The simulator models the architecture described in Section 2; the threads share execution units and can be prioritized.

4 Results: Base station workloads

Using the benchmarks described above we constructed two workloads to simulate real-world situations. The first workload conforms to base station requirements. Base stations are responsible for all communication among the mobile clients in a specific cell and also interface to other networks. Base stations support many wireless handsets (each on a separate channel) so throughput/

cost is the important metric in this case.

4.1 Throughput

Base stations are designed to support a maximum channel capacity and they typically contain enough DSPs to easily accommodate all channels. A “channel” in our workload consists of four programs run consecutively: a GSM-EFR encoder, a GSM-EFR decoder, a Trellis encoder, and a Trellis decoder. Each channel has to process one frame within 20ms (50 frames/sec). A workload comprises multiple independent channels. In this section, we establish the number of channels that can be supported (i.e., without breaking real-time constraints) by each architecture at various clock frequencies. We estimate the cost for the multithreaded DSP in terms of additional chip area and subsequently we compare cost/performance for an SMT system with both a CMP and a single-threaded DSP.

The base station workload is dominated by the GSM EFR encoder as seen in Table 1. The decoder runs for 0.4 million cycles. The total for all four programs is 0.5 million cycles which implies a minimum clock frequency of approximately 25MHz (so that enough cycles will be available in a time interval of 20ms).

	IPC	Cycles for 1 Frame
GSM EFR encoder	1.42	0.400 M
GSM EFR decoder	1.50	0.034 M
Trellis decoder	1.17	0.040 M
Trellis encoder	0.74	0.026 M
	Average: 1.39	Total: 0.500 M

TABLE 1: IPC and cycles for 4 threads (1 frame)

Figure 3 shows results of executing up to eight channels in the base architecture (serially) and in the multithreaded architecture (concurrently). The first graph in Figure 3 plots the elapsed cycles for the eight workloads and the two architectures. Elapsed cycles in the multithreaded architecture increase only slowly. The multithreaded DSP requires less that 2 million cycles to complete all eight channels. This translates to an operating frequency of just under 100MHz. In contrast the base DSP requires a clock frequency of 200MHz for the eight channels.

The second graph in Figure 3 shows the IPC for the two architectures. The IPC for the base case is constant no matter how many channels we run. The IPC of the multithreaded DSP shows a smooth increase as we add more channels but it flattens (topping at 3.16) as we add more than five channels. At this point the two AGUs that handle all loads/stores are saturated. In Figure 4 we plot

the utilization of the AGUs and DALUs. For each of the eight workloads we show the percentage of time no unit was used, one unit was used, two units, etc. With more than five channels we use both AGUs more than 90% of the time while the utilization of the four DALUs remains small, using just one DALU a little more that 30% of the time.

Since AGUs appear to be a major bottleneck we increase their number from two to four. No individual thread can use four AGUs simultaneously since all threads have been compiled for only two AGUs available. However, the benefits in the multithreaded architecture are considerable. Figure 5 shows elapsed cycles, IPC, AGU and DALU utilization with two additional AGUs in the multithreaded architecture. The number of cycles increases even more slowly with the number of channels and the maximum IPC climbs to 5.4. The AGUs are nowhere near saturation and the utilization of the DALUs increases (4 DALUs used simultaneously 30% of the time). Despite the lower utilization of the function units IPC still flattens after five channels because we approach the IPC ceiling of the architecture.

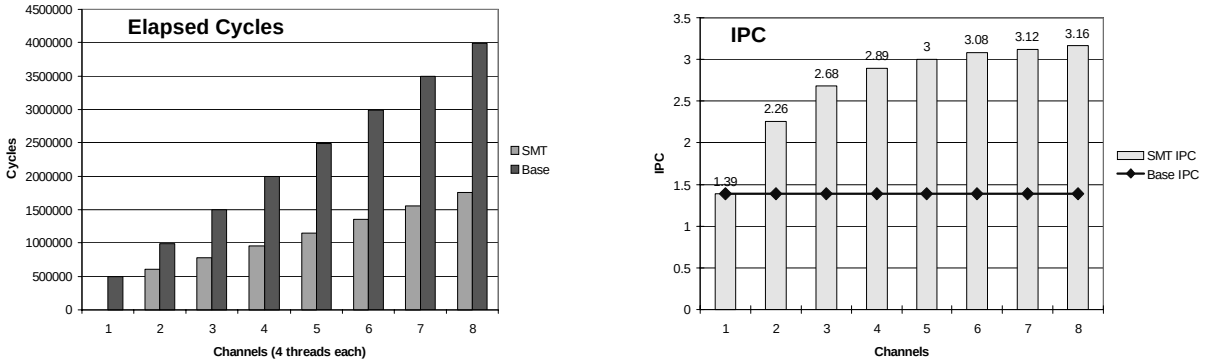

FIGURE 3. Execution time and IPC for multiple channels simulated on the base and multithreaded DSP. For the base DSP, elapsed cycles increase linearly with number of channels while IPC remains constant.

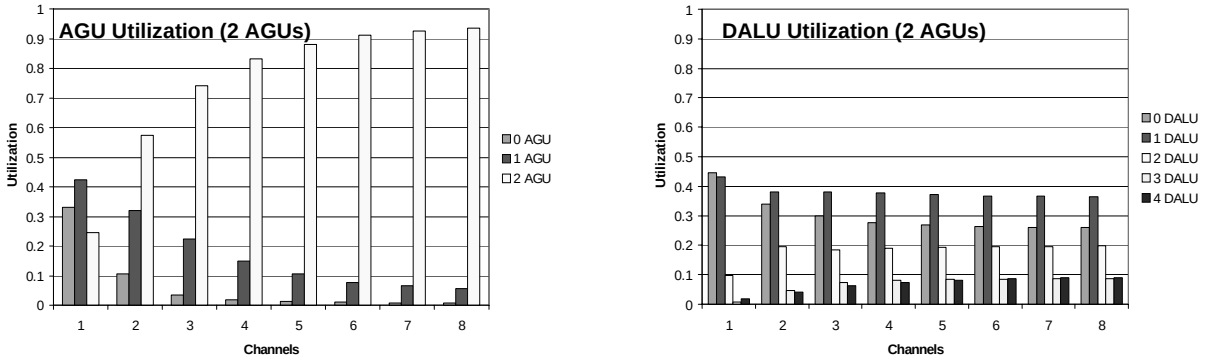

FIGURE 4. AGU and DALU utilization for base station workloads. The sets of three bars on the AGU graph denote the percentage of time 0, 1, or 2 AGUs were active. Similarly the sets five bars in the DALU graph denote the percentage of time 0, 1, 2, 3, or 4 DALUs were active.

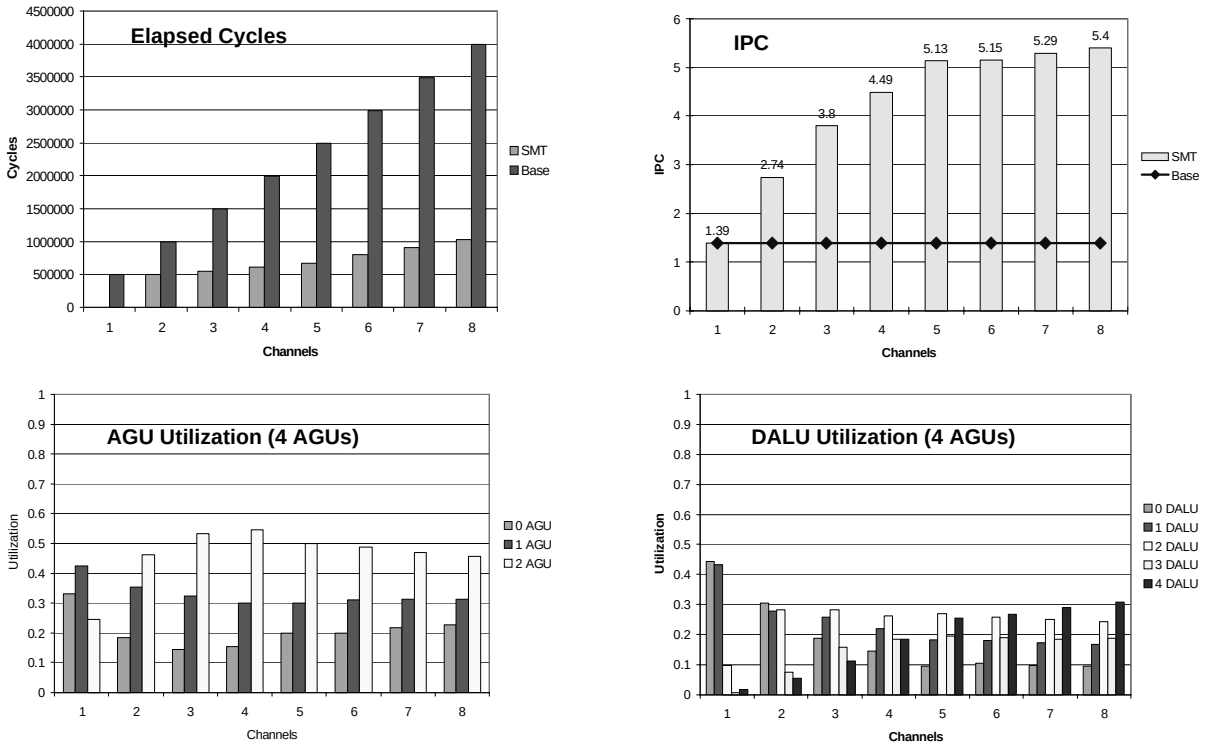

FIGURE 5. Increasing the AGUs from 2 to 4 for the base station workload.

4.2 Cost

The above results show that a multithreaded DSP can execute a number of compiled threads with a small slowdown over the execution of a single thread. Chip multiprocessor (CMP) DSPs such as Lucent's StarPro [9] are also designed for throughput. In the CMP case, however, cost increases linearly with the number of supported threads. For the multithreaded DSP cost increases slowly as we add more state (register files) and multithreading support. We measure the cost of a multithreaded architecture according to the additional chip area required. As a starting point we use subblock areas of a synthesized SC140 core[2]. Subblock area percentages are listed in Table 2. Approximately 31% of the chip area is devoted to the 4 DALUs and data register file, another 23% to the 2 AGUs and address register file, and the rest 47% is support circuitry and other function units (e.g., BFU) that are not replicated in the multithreaded architecture.

In the multithreaded architecture the main increase in area comes from additional register files. As in the SMT work [4], we increase the size of the existing register file to the appropriate number of registers and we use a thread ID to access the appropriate registers. The number of

2 Numbers can be different for custom cores.

	Area % of total
DALU MACs (4)	10%
Data registers	21%
BFU (Bit manipulation)	8%
Logic	24%
AGUs (2) + address registers	23%
Fixed HW	13%
Total	100%

TABLE 2: Subblock areas of a synthesized SC140 core

read/write ports in the register files depends on the number of function units. The larger register file is also slower but we have taken this into account in the extra pipeline stage we introduced (see Section 2). We estimate that support for one additional hardware context costs 33% of the chip area for additional registers (21% for data registers and 11% for address registers) and related routing. Issue logic doubles in size from 24% to 48% to accommodate thread scheduling logic.

Table 3 shows estimated areas for the multithreaded DSPs with up to five threads. The first five rows show the increase in area from additional data and address registers. The second five rows show area increase when we also double the number of AGUs. In this case, the size of the address register file increases both with the number of threads (number of registers) and with the number of AGUs because of the additional read/write ports needed (size of registers). We conservatively allow a factor of two in area increase for the additional read/write ports needed to support the additional AGUs.

	4 DALUs	**2 AGUs**	**Data register files**	**Address register files**	**Logic**	**Other**	**Total**	**CMP Area**
Base architecture	10%	11%	21%	12%	24%	22%	100%	100%
2 Threads	10%	11%	42%	24%	48%	22%	157%	200%
3 Threads	10%	11%	63%	36%	48%	22%	190%	300%
4 Threads	10%	11%	84%	48%	48%	22%	223%	400%
5 Threads	10%	11%	105%	60%	48%	22%	256%	500%
		4 AGUs						
Base architecture	10%	11%	21%	12%	24%	22%	100%	100%
2 Threads	10%	22%	42%	48%	48%	22%	192%	200%
3 Threads	10%	22%	63%	72%	48%	22%	237%	300%
4 Threads	10%	22%	84%	96%	48%	22%	282%	400%
5 Threads	10%	22%	105%	120%	48%	22%	327%	500%

TABLE 3: Chip area estimates for the multithreaded architecture (all estimates conservative). All percentages correspond to chip area of the base architecture.

Finally, in Figure 6, we show performance improvement over area increase for the multithreaded DSP and CMP implementations. We compute the performance improvement as the ratio

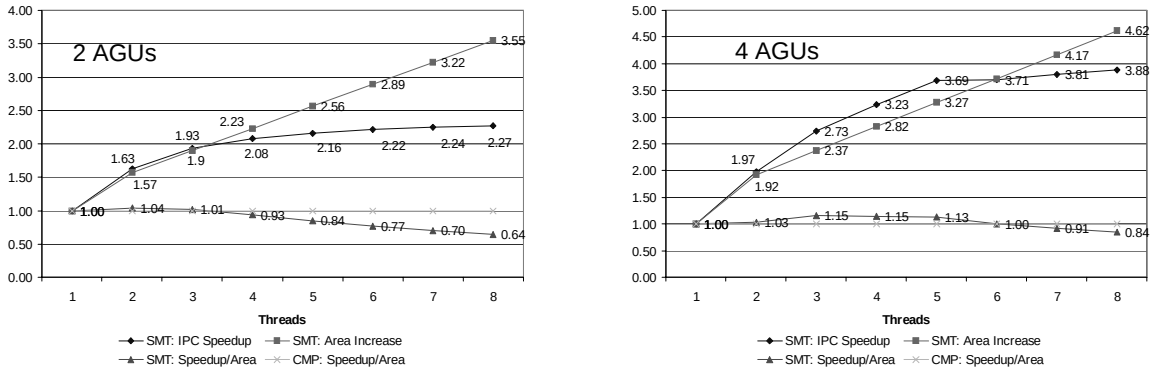

FIGURE 6. Performance/Cost for SMT and CMP DSPs: IPC speedup (over the base single-threaded architecture) divided by chip area increase (over base DSP area). The SMT DSP shows a slight advantage for a small number of threads, especially for the 4-AGU case.

of the IPC of the multithreaded DSP (or the CMP) over the base, single-core, single-threaded architecture. Area increases are derived from Table 3. The multithreaded DSP compares favorably to the CMP. The CMP delivers performance improvement linear to area increase and therefore its performance/area ratio will always be one. The multithreaded DSP, however, can actually achieve ratios greater than 1 for a small number of threads. For two AGUs its ratio barely exceeds 1 with two threads. For four AGUs the ratio comfortably exceeds 1—by more than 10%—with two, three, and four threads. Figure 6 also shows that the performance/area ratio for the multithreaded DSP also drops below 1 for more than three threads (2 AGUs) or seven threads (4 AGUs) meaning that we get diminishing returns beyond these points.

5 Results: cell-phone workloads

To satisfy the increased processing requirements of 3G wireless handsets, DSPs have to turn to ever higher clock frequencies. However, power consumption is still the overriding concern in mobile applications. In this section we show that an SMT DSP processor can be used to reduce power consumption in situations where the required performance is bounded by a fixed workload. By multithreading the workload we increase parallelism (IPC), and can therefore decrease clock frequency and still do the same amount of work in the same time. Decreasing frequency also allows us to decrease the supply voltage (voltage scaling). Both lower frequency and lower voltage contribute to a significant reduction in power consumption. A similar strategy is used by Transmeta in their LongRun technology [20]. Transmeta's strategy is to monitor—in software—the activity of the processor and reduce frequency and voltage to minimize idle time.

To study power consumption we examine two different IC manufacturing technologies: .16µ

(e.g., SC140) and .25µ used to manufacture Lucent's DSP1628 [13]. The range of clock frequencies and the corresponding scaling of minimum V_{dd} for both technologies is shown in Figure 7. For our study we use the low voltage .25µ technology. We study a different workload for each technology intended to stress the base architecture to its limits in the available frequency range.

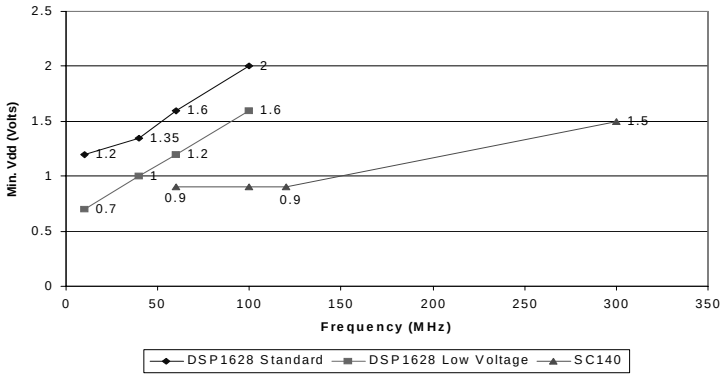

FIGURE 7. Minimum V_{dd} vs. frequency for .25µ DSP1628 (data adapted from [13]) and .16µ SC140 processes. V_{dd} does not decrease below 0.9Volts in the .16µ process in frequencies below 120MHz.

We simulate a full second of processing for the base architecture and the multithreaded architecture with five hardware contexts. We use GSM-EFR speech codecs[3], Trellis channel codecs, and MPEG-2 codecs. The speech and channel codecs run every 20ms (50 times in total) while the MPEG codecs run until completion without restarting within the simulated second. In a real implementation the base DSP would context switch among all threads every 20 ms but we do not penalize the serial execution with context switch costs.

The speech and channel threads involve a fixed amount of computation and their spacing every 20ms dilutes the IPC as a function of clock frequency: the higher the frequency, the lower the IPC. The MPEG-2 encoders are by far the longest threads and dominate IPC; the MPEG-2 decoder is an order of magnitude smaller. Table 4 lists the characteristics of the MPEG codes; the remaining applications are the same as in Table 1. The image size for the MPEG encoder was chosen such that the entire workload would complete just in time (1 second) on the single-threaded DSP running at the highest possible frequency. When executed as part of the entire workload, encoding 32 x 32 pixel-frames completes on the base architecture in 1 second operating at 93MHz, while a 64 x 64 pixel frame can be encoded in one second at just over 300MHz. Therefore, assuming each of the two different manufacturing technologies (Figure 7), we fix these

[3] "codec" is shorthand for encoder/decoder

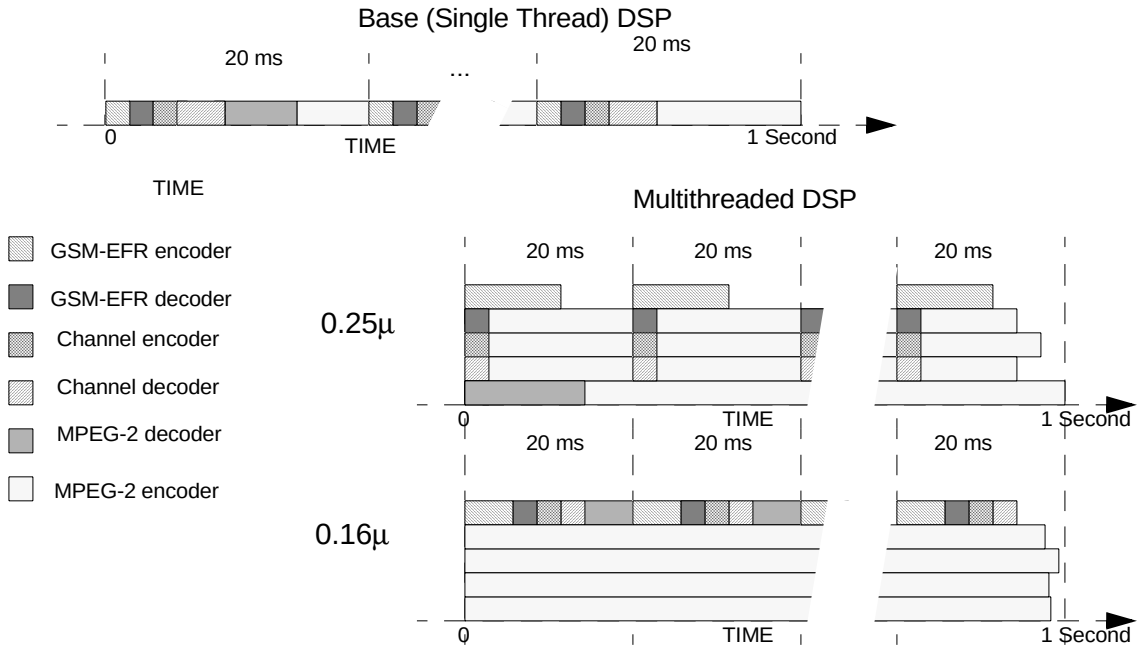

FIGURE 8. Cell-phone workload. The GSM and Channel codecs run once every 20ms and have to finish within this time. The MPEG-2 codecs consume the rest of the cycles within the one second we simulate. In the base architecture (top) threads context-switch with zero overhead. In the multithreaded architecture threads run in five hardware contexts.

image sizes for the MPEG encoder respectively. Four encoders, each encoding one-fourth the image size, are used for the multithreaded experiments.

We assign threads to the five hardware contexts of the multithreaded DSP as follows:

- .25µ technology: one of the five hardware contexts executes the speech encoder, while the other four hardware contexts each execute an MPEG encoder thread along with one of the other four remaining applications (speech decoder, channel encoder, channel decoder, MPEG decoder).
- .16µ technology: the four MPEG encoders run on separate hardware contexts, while the remaining five applications (speech codecs, channel codecs, and MPEG decoder) all execute on the fifth hardware context.

Figure 9 shows the IPC as a function of clock frequency for the two technologies and for two and four AGUs. As frequency increases the multithreaded DSP is able to finish early the large MPEG encoder threads, leaving the speech and channel threads running every 20ms. The idle time involved reduces the average IPC significantly over the period of one second we simulate.

5.1 Power Computation

Dynamic power consumption is the chief source of power consumption in CMOS processors.

Program	IPC	Cycles
MPEG-2 encoder 16x16 frame (2 frames/sec) 4 copies used in multithreaded DSP @ 28-93MHz to approximate 32x32 frame	0.58	16921130
MPEG-2 encoder 32x32 frame (2 frames/sec) 4 copies used in multithreaded DSP @ 72-300MHz to approximate 64x64 frame 1 copy used in base architecture @ 93 MHz	0.59	67013987
MPEG-2 encoder 64x64 frame (2 frames/sec) 1 copy used in base architecture @ 300MHz	0.61	268016411
MPEG-2 decoder 64x64 frame (3 frames/sec) used in all experiments	1.02	645008

TABLE 4: IPC and cycles for MPEG threads on the base architecture.

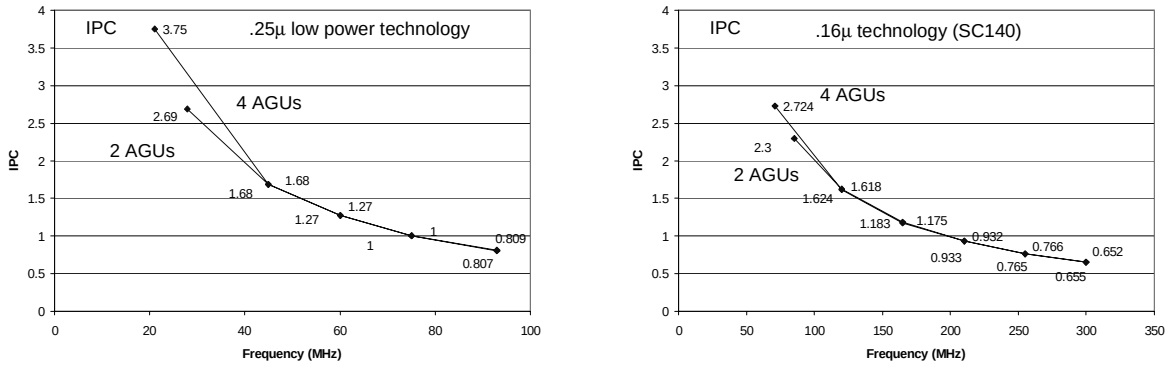

FIGURE 9. Average IPC of the fixed cell phone workload running for one second as a function of clock frequency for a multithreaded DSP with 5 hardware contexts.

It is defined by the formula:

$$P = a \times C \times Vdd^2 \times F$$

where F is the operating frequency, V_{dd} is the supply voltage and C is load capacitance of the circuit. The term a is an *activity factor* as used by Brooks, Tiwari, and Martonosi [26] to represent the average switching activity of transistors per clock cycle; it takes a value between zero (no switching) and one (switching every cycle). We use our cycle-accurate simulator to determine the activity factors for the different subblocks on the DSP core.

We derive the average power consumption over the one second period for an architecture (single- or multi-threaded) operating at frequency F using the following steps:

1. Given F, V_{dd} is computed from Figure 7 (we conduct experiments for both 0.25µ and 0.16µ IC technologies).
2. Load capacitance for each subblock is derived from the area estimates in Table 3. For the multithreaded case, we use the load capacitances given 6 threads (hardware contexts). We assume conservatively that the increase in the load capacitance of a subblock will be analogous to the increase in area [27]. The chip area increase was discussed in Section 4 (see

Table 3).

3. The activity factor for each unit is derived from the AGU and DALU utilizations, which are determined by the simulator running the entire workload for the 1-second time period. For example, if over the 1-second run, the simulator finds that on average 1 DALU (out of 4) is busy per cycle, the DALU utilization a_{dalu} is computed to be 1/4. The AGU utilization a_{agu} is derived in a similar manner, and also depends on the number of AGUs in the architecture being simulated. The AGU and DALU utilizations as a function of clock frequency are given in Figure 10.
4. The average power consumption is then computed by the formula:

$$P = [\ a_{dalu} \times (C_{dalu} + C_{data\_reg}) + a_{agu} \times (C_{agu} + C_{addr\_reg}) + 1 \times C_{rest}\ ] \times V_{dd}^2 \times F$$

where C_{dalu} , C_{data_reg} ,C_{agu} , and C_{addr_reg} are the load capacitances of the DALU, data registers, AGU, and address registers respectively. C_{rest} is the load capacitance of the remaining chip (logic and other subblocks), which is assumed to have a constant activity factor of 1. Because every access to the AGU involves accesses to the address registers, we assume the activity factor of the address registers is the same as that of the AGUs, namely, a_{agu} as computed in step (3). Similarly, the activity factor of the data registers is assumed to be equal to a_{dalu}, the activity factor of the DALUs.

For our study, we do not need to examine directly *energy* or *energy-delay* metrics. This is because we run for a fixed amount of time (1 second) and we execute the same workload. Average power consumption, therefore, can be translated directly to energy by multiplying by one second (time).

5.2 Comparison of power consumption

Recall that the inputs to the MPEG encoder(s) were chosen such that the entire workload would complete at a certain frequency (close to the maximum allowable operating frequency) on the base architecture. Therefore, for the base architecture, we compute power consumption at only this frequency, using the method described in Section 5.1.

In the multithreaded case, for the .25µ technology, the lowest feasible operating frequency was 28 MHz (using 2 AGUs). Lowering the frequency further did not allow the workload (in par-

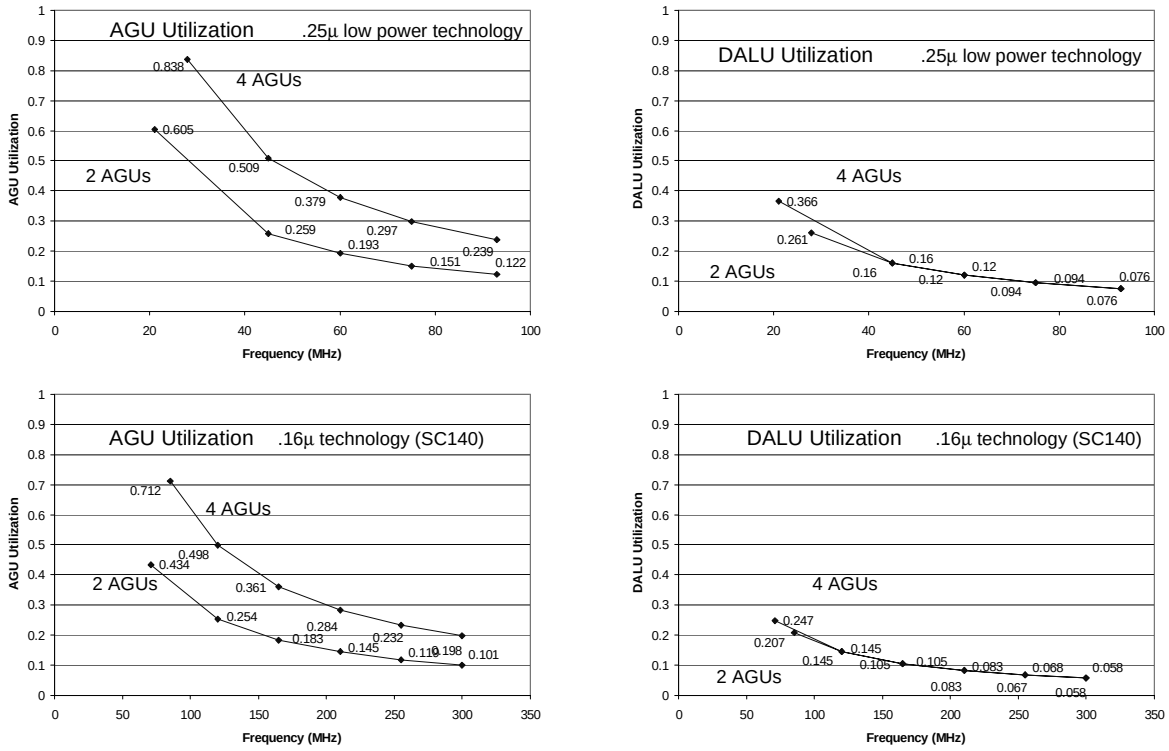

FIGURE 10. AGU and DALU utilization as a function of frequency for the multithreaded DSP running a fixed workload for one second.

ticular, the speech encoder) to meet the 20ms real-time deadline. Increasing the number of AGUs to 4 increases performance, and therefore allows the frequency to be further lowered to 21MHz. At these low frequencies the speech encoder needs to be run in high priority to ensure it meets its deadlines. Similarly, for the .16µ technology, the lowest feasible operating frequency was determined to be 85 MHz using 2 AGUs and 71 MHz using 4 AGUs. In this case, the MPEG encoder (given the larger image size) was the limiting factor in further reducing the frequency. For the .16µ technology voltage scaling stops at 120MHz and we cannot go below 0.9Volts V_{dd} for lower frequencies (Figure 7).

We ran the multithreaded experiments at different frequencies ranging from the lowest to the highest feasible operating frequencies. For each frequency, we computed the ratio of power consumptions of the single-threaded and multithreaded DSPs; the results are shown in Figure 11. Substantial savings in power (up to a factor of 4) are possible at low frequencies using the SMT version of the DSP. As clock frequency is increased, however, it begins consuming more power because the effect of increased frequency and V_{dd} outweighs the effect of lower utilization (activity factor). Further, the load capacitances of the subblocks are higher for the multithreaded DSP,

and therefore as frequency is increased, it eventually begins to consume more power than the single-threaded DSP. The break-even points are at 67MHz and 220MHz for the .25µ and .16µ technologies respectively.

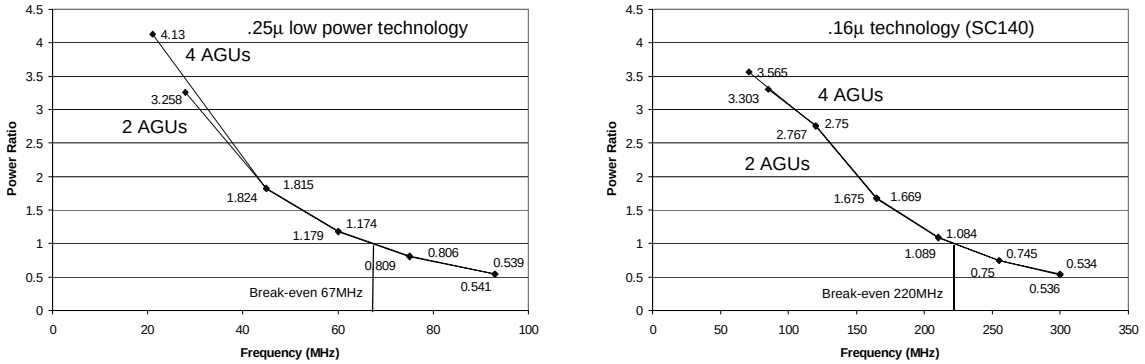

FIGURE 11. Power ratio (Base/Multithreaded) executing the same workload in one second. Ratios greater that one indicate that the multithreaded DSP is better. We vary the frequency of the multithreaded DSP to the lowest possible frequency that can safely accommodate the workload. Base frequency cannot be decreased without breaking real-time constraints.

Increasing the number of AGUs allows a lower operating frequency, and this benefit outweighs the increased load capacitance of the AGUs and address registers. As seen in Figure 9, this benefit is cancelled out at higher frequencies and therefore more AGUs do not help reduce power consumption.

We also computed the power consumption of a CMP system, with all the processors running at the same frequency. For the .25µ technology, the CMP system is assumed to have five processors; one processor executes the speech encoder, four processors execute MPEG encoders, and one of the other 4 benchmarks (exactly as the SMT workload for .25µ depicted in Figure 8). For the .16µ technology, the CMP system again needs only 5 processors: 4 running the MPEG encoders and one running the remaining benchmarks (as in Figure 8). At the higher operating frequencies for this manufacturing technology, the speech encoder can be executed with other benchmarks without violating real-time deadlines. The resulting power consumption for the CMP system is also shown in Figure 12. For both technologies, the lowest feasible operating frequency for the CMP system is lower than that of the SMT system; this is because the most computationally intensive benchmarks run on separate processors and do not contend for resources. However, as seen from Figure 12, the benefit of a lower frequency is outweighed by the increased load capacitance of the CMP system. At the lowest feasible frequency, the SMT system still consumes 21% (47%) less power than the CMP system for the .25µ (.16µ) technology. Furthermore, as the

frequency is increased, the CMP consumes significantly more power than the SMT system.

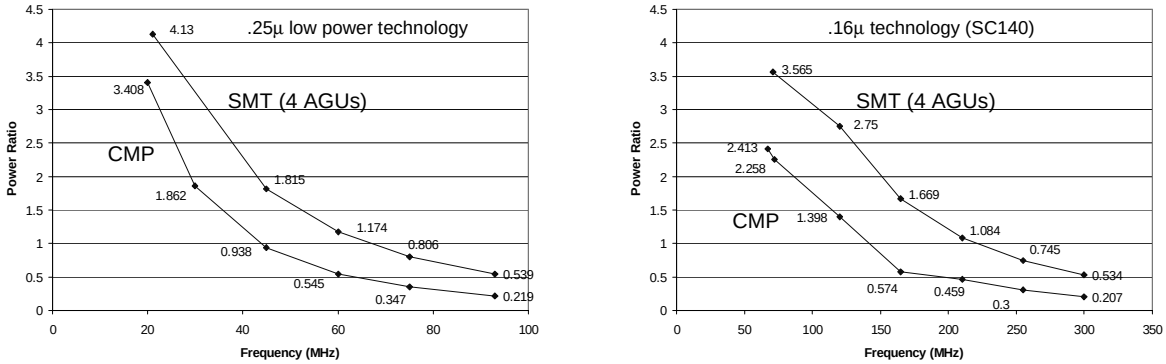

FIGURE 12. CMP vs. SMT on power consumption. Power ratio is given as CMP/base and SMT/base (higher is better). The CMP architecture can go to lower frequencies but it does not surpass the SMT architecture in terms of power efficiency.

6 Conclusions

Signal processing and real-time applications often require a different figure of merit than minimizing the number of cycles required to complete a task. Minimization of power may be more important, and real-time may mean that minimizing the time to complete a task is unimportant. We ran two series of experiments that represent workloads typical of real world multimedia applications, both of which are very sensitive to power consumption. One model, the mobile telephone base station, requires maximizing the amount of work that can be done in a fixed amount of time while minimizing the number or size of processors. The second model, the wireless handset, requires minimizing power consumed for a fixed amount of work.

The experiments show that using SMT it is possible to save area and/or power in comparison with a CMP implementation or a single processor implementation that runs at a higher clock rate:

- In the base station example, the SMT design was capable of exceeding the increase in cost with its increase in performance—something that the CMP system cannot do.
- The wireless handset demonstrates the power advantage of running tasks in parallel at a lower clock rate and supply voltage. The SMT can reduce power consumption by a factor of 4 over a single DSP running at high frequency and high voltage to complete the same real-time workload at the same period of time. Compared to a CMP that also can run at low frequency and low voltage for the same workload, the SMT retains a significant advantage in power consumption being more power efficient by a factor of 1.47.

  We used a commercial DSP architecture as the base of our study, and did not modify the com-

piler or other software tools. The results are therefore conservative in that it is possible to optimize the programs to exploit the SMT configuration and extend the efficiency advantages of SMT over CMP organizations. On the other hand, our study is constrained by the compiler we used and the workloads we chose. Our compiled codes do not exhibit high IPC so a multithreaded architecture can easily accommodate multiple of them. However, we believe that compiled code is becoming increasingly important in the development cycle for DSP applications and an architecture that ameliorates reduced compiled performance is likely to find acceptance. In many applications, the power and cost benefits of the SMT approach could make it a more attractive alternative to a simpler CMP design.

7 Acknowledgments

We would like to thank Nevin Heintze, Rae Mcllelan, Tor Jeremiassen, Cliff Young, and Brian Kernighan for their comments on drafts of this paper. We would also like to thank Paul D'Arcy and Isik Kizilyalli who provided data for this parer.

8 References

[1] M. Baron, "Breaking the $4 Billion Barrier -- 1999 DSP Vendor Market Shares," In *Stat Group Research Report No. ML00-01MS*. February, 2000.

[2] Ojanpera and R. Prasad, Wideband CDMA for Third Generation Mobile Communications, Artech House, Oct. 1998.

[3] Dean Tullsen, Susan Eggers, and Henry Levy "Simultaneous Multithreading: Maximizing On-Chip Parallelism," In *Proceedings of the 22rd Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June 1995, pages 392-403.

[4] Dean Tullsen, Susan Eggers, Joel Emer, Henry Levy "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, Philadelphia, May 1996.

[5] B, J. Smith, "Architecture and Applications of the HEP multiprocessor computer System," *In SPIE Reat Time Signal Processing IV*, pp 241-248, 1981.

[6] J. Fisher, "VLIW architectures: an inevitable standard for the future?" *Journal of Supercomputer, Vol. 7, No 2*, pp 29-36. Mar 1990.

[7] Texas Instruments. TMS320C6211 Fixed Point Digital Signal Processor—Product Review, August 1998. SPRS073.

[8] "Star*Core Launches First Architecture," *Microprocessor Report 12:14*, 10/26/98.

[9] "Lucent rolls out its first Star*Core-based DSP, promises to double Internet chip capacity", *Semiconductor Business News*, 6/12/00.

[10] Basem A. Nayfeh and Kunle Olukotum, "Exploring the Design Space for a Shared-Cache Multiprocessor," In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, Chicago, April 1994.

[11] European Telecommunications Standards Institute. “Digital cellular telecommunications system: Enhanced Full Rate (EFR) speech transcoding (GSM 06.60),” March 1997.

[12] European Telecommunications Standards Institute. “Digital cellular telecommunications system: ANSI-C code for the GSM Enhanced Full Rate (EFR) speech codec,” March 1997.

[13] I.C Kizilyalli et al. “A WSi/WSiN/Poly:Si Gate CMOS Technology for 1.0V DSPs,” In *Proceedings of the First International Symposium on ULSI Process integration, The Electrochemical Society Proceedings Vol. 99-18*. pp 347-352.

[14] Star*Core, “SC100 C/C++ Compiler User’s Manual”, available at http://www.starcore-dsp.com/.

[15] E. Biglieri, D. Divsalar, P. McLane, and M. K. Simon, “Introduction to Trellis-Coded Modulation with Applications”, Macmillan, New York, 1991.

[16] A. J. Viterbi, “Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm”, *IEEE Trans. Information Theory, vol. IT-13*, 260-269, April 1967.

[17] MPEG Software Simulation Group (MSSG), http://www.mpeg.org/MPEG/MSSG/

[18] Tor E. Jeremiassen, “A DSP with Caches - A Study of the GSM-EFR Codec on the TI C6211,” *IEEE International Conference on Computer Design*, pages 138-145, October 1999.

[19] MicroDesign Resources, “Embedded Processor Watch, #78”, December 14, 1999. http://www.mpronline.com

[20] Keith Diefendorff, “Transmeta Unveils Crusoe: Supersecret Startup Attacks Mobile Market with VLIW, Code Morphing,” *Microprocessor Report*, 1/24/00.

[21] “Philips Hopes to Displace DSPs with VLIW,” *Microprocessor Report 8:16*, 12/5/94.

[22] L. Gwennap, “Network processors race toward 10-gigabit goal,” *Electrical Engineering Times, Issue 1118*, 6/19/00.

[23] P. Crowley, M. Fiuczynski, J.-L. Baer and B. Bershad, “Characterizing Processor Architectures for Programmable Network Interfaces.” *Proceedings of the 2000 International Conference on Supercomputing*, Santa Fe, NM, May 2000.

[24] “IA-64 and Merced--What and Why,” *Microprocessor Report*, 12/30/96

[25] Michael Slater, Linley Gwennap, Keith Diefendorff, Peter Glaskowsky “Alpha 21464 Targets 1.7 GHz in 2003,” *Microprocessor Watch Issue #25*, MicroDesign Resources, November 18, 1999.

[26] David Brooks, Vivek Tiwari, Margaret Martonosi, “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations,” In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, Vancouver Canada, June 2000.

[27] Tohru Ishihara and Hiroto Yasuura, “Experimental Analysis of Power Estimation Models of CMOS VLSI Circuits,” *IEICE Trans. Fundamentals, Vol.E80-A No.3, pp.480-486*, March 1997.