Xtream-Fit: An Energy-Delay Efficient Data Memory Subsystem for Embedded Media Processing*

Anand Ramachandran and Margarida F. Jacome
Department of Electrical and Computer Engineering
The University of Texas at Austin
randy|jacome@ece.utexas.edu

ABSTRACT

In this paper we propose a novel special-purpose data memory subsystem, called Xtream-Fit, aimed at achieving high energy-delay efficiency for streaming media applications. A key novelty of Xtream-Fit is that it exposes a single customization parameter, thus enabling a very simple and yet effective design space exploration methodology. A second key contribution of this work is the ability to achieve very high energy-delay efficiency through a synergistic combination of: (1) special purpose memory subsystem components, namely, a Streaming Memory and Scratch-Pad Memory; and (2) a novel task-based execution model that exposes/enhances opportunities for efficient prefetching, and aggressive dynamic energy conservation techniques targeting on-chip and off-chip memory components. Extensive experimental results show that Xtream-Fit reduces energy-delay product by 46% to 83%, as compared to general-purpose memory subsystems enhanced with state of the art Cache Decay and SDRAM power mode control policies.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems—*media processing, energy efficiency*; B.3 [**Memory Structures**]: Cache memories—*software controlled cache, scratch-pad, streaming memory*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

streaming memory, scratch-pad, energy delay product, low power, media processing, design space exploration, configurability

1. INTRODUCTION

Microprocessor cores, as opposed to packaged, off-the shelf processors, are being increasingly used in high volume embedded systems [1, 2, 3]. Achieving high energy-delay efficiency, via chip customization, is the main objective driving this trend. It is well known that the memory subsystem is responsible for a significant percentage of the overall power dissipation of most off-the-shelf processors, e.g., more than 40% for the StrongARM-110 [4]. The memory subsystem is thus a prime candidate for customization to an application's requirements by embedded system designers [1, 2, 3].

In this paper, we propose a special-purpose data memory subsystem architecture, Xtream-Fit (see Figure 1), aimed at achieving high energy-delay efficiency for streaming media applications. Xtream-Fit's high energy efficiency is achieved through the use of an on-chip software-controlled Streaming Memory, a Scratch-Pad Memory, and an aggressive exploitation of burst/page mode access and low power modes available in modern SDRAMs.

*This work is supported in part by NSF ITR Grant ACI-0081791, and NSF Grants CCR-9901255 and ECS-0225448.

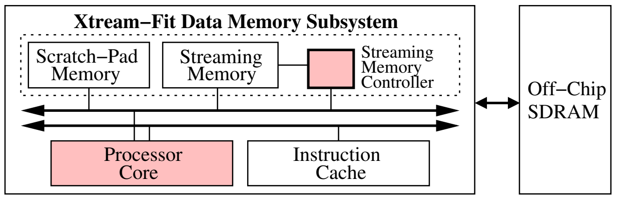

Figure 1: Embedded media processing w/ Xtream-Fit

In Xtream-Fit, data transfers (to/from the off-chip SDRAM) are independent stream-granularity operations executed by a dedicated Streaming Memory Controller. The relative "pacing" of such data transfers with respect to actual computations is established via a dynamic synchronization of data transfer tasks, executing on the Streaming Memory Controller, and processing tasks, executing on the embedded processor core.

We will show that, when such tasks are properly defined, their synchronized execution maximizes opportunities for energy savings, and exposes critical *energy-delay trade-offs*, in two important ways. First, it enables a tight control over the patterns of off-chip memory accesses/references generated by the memory subsystem, namely, over their inherent locality and periodicity, which in turn allows for a better exploitation of modern SDRAMs' energy-delay efficient access modes. Second, it leads to a consolidation of system components' idle/dead times (e.g., off-chip SDRAM and/or specific regions of the on-chip Streaming Memory), into larger and more predictable time intervals. As will be seen, this is critical for optimizing the profitability of energy saving techniques based on selective exploitation of low power modes.

In contrast to previous approaches, which require the *tuning* of several (possibly conflicting) customization parameters, the proposed Xtream-Fit data memory subsystem exposes a single customization parameter, thus enabling a simple yet highly effective design space exploration methodology. Our extensive experimental results for a representative set of applications from the Mediabench suite [5], considering two families of processor cores (namely, ARM and MIPS), empirically demonstrate that Xtream-Fit can reduce total energy consumption of on-chip and off-chip memories by 43% to 83% with no negative impact in performance, thus improving energy-delay product by as much as 83%, as compared to general-purpose memory subsystems enhanced with state-of-the-art Cache Decay and SDRAM power mode control policies, working with a similar processor core [6, 7, 8, 9].

The paper is organized as follows. Section 2 discusses key characteristics of streaming media applications and introduces the main concepts/ideas exploited in Xtream-Fit. Sections 3 and 4 describe the main components of the proposed data memory subsystem and associated energy conservation policies. Section 5 contrasts Xtream-Fit with previous related work. Section 6 describes the methodology and experimental set up used to assess the energy-delay efficiency of Xtream-Fit and presents detailed simulation results and analysis. Conclusions are given in Section 7.

2. XTREAM-FIT PROCESSING MODEL

Input and output streams of media applications are typically specified as sequences of relatively small *basic data objects*, which can be processed/generated (quasi-) independently.

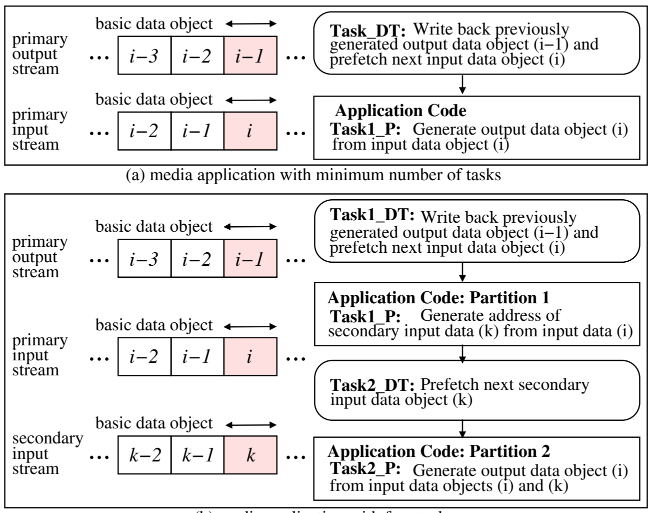
basic data object
primary output stream
i−3
i−2
i−1
Task_DT: Write back previously generated output data object (i−1) and prefetch next input data object (i)
basic data object
primary input stream
i−2
i−1
i
Application Code
Task1_P: Generate output data object (i) from input data object (i)
(a) media application with minimum number of tasks
basic data object
primary output stream
Task1_DT: Write back previously generated output data object (i−1) and prefetch next input data object (i)
Application Code: Partition 1
Task1_P: Generate address of secondary input data (k) from input data (i)
primary input stream
Task2_DT: Prefetch next secondary input data object (k)
secondary input stream
k−2
k−1
k
Application Code: Partition 2
Task2_P: Generate output data object (i) from input data objects (i) and (k)
(b) media application with four tasks

Figure 2: Task decomposition: key principles

For example, MPEG compression and decompression of frames/ pictures in a video stream is performed on a *macroblock* basis (16x16 pixel block), JPEG encoders/decoders work on a block basis (8x8 pixel block), etc. [5].

Media applications usually perform sequences of complex operations and transformations on streams of such basic data objects, often exhibiting high locality of reference yet low data reuse [10, 11, 12, 13]. In addition, a myriad of constants (and/or infrequently changed data) are periodically reused during the processing of those basic data objects – examples include arrays of quantization, filtering, DCT, IDCT, FFT and other transform coefficients. Xtream-Fit's Scratch-Pad Memory and Streaming Memory (see Figure 1) provide energy-efficient on-chip storage for these two types of data: (1) constants and scalars; and (2) low reuse streaming data.

2.1 Processing and Data Transfer Tasks

At the core of this work lies the idea of decomposing the media application into tasks, which encapsulate continuous processing loads for one of the two main architectural subsystems, namely, the data memory subsystem (Xtream-Fit) and the processing subsystem (processor core). Thus, for example, once a task begins execution on the processing subsystem, it will not be stalled waiting for data transfers. Similarly, a task being executed on the memory subsystem will make full use of the memory bandwidth until completion.

Accordingly, streaming media applications are decomposed into at least one data transfer task and one processing task – the first prefetches and writes back data streams, while the second processes/generates those streams. Figure 2(a) shows an application that breaks down into two such tasks. Specifically, at iteration i, the data transfer task $Task\_DT$ starts by storing (writing back) the previously generated *output data object* (i.e., data object $(i-1)$ of the output stream), and then prefetches the next *input data object* (i.e., data object i of the input stream). When the data transfer task ends, a processing task, encompassing the entire application code, starts executing – during such execution, the input data object just prefetched is processed so as to generate the corresponding output data object. Note that output data object i will be written back only at the next iteration $i+1$, and so forth. Many media applications, e.g., JPEG and G721 encoders and decoders, can be decomposed into only two such basic tasks.

Some applications may however require more than two tasks. The key rule driving further task decomposition is as follows:

> *Memory accesses dependent on input or intermediate data, when extant, should be organized or clustered into independent data transfer tasks, such that continuous workloads on the two subsystems can be ensured.*

For example, consider a media application that requires one additional input data object from some secondary input stream, as it processes each data object from the primary input stream. Assume also that the address of this additional input data object is encoded in the primary input data object itself. Data dependent accesses such as this define the boundaries for further task decomposition. Indeed, one can only load the secondary data object after its address has been determined/computed by a processing task, using the primary input data (namely, $Task1\_P$ in Figure 2(b)). Thus, if one wishes to define data transfer tasks that ensure continuous workloads on the memory subsystem, the loading of this secondary data object cannot be performed by the same data transfer task that loads the primary input data object (namely, $Task1\_DT$ in Figure 2(b)) – or, else, workload continuity would not be guaranteed, since such a task would stall, waiting for the address of the secondary data object. Accordingly, an additional data transfer task (denoted $Task2\_DT$ in Figure 2(b)) is defined for that purpose. Similarly, in order ensure continuous workloads on the processing subsystem, the computations/decoding that determines the address of the secondary data object (using the primary input data) cannot be executed by the same processing task that eventually generates the primary output data object. This application would thus be decomposed into four tasks, interleaved as indicated in Figure 2(b).

The above task decomposition methodology was successfully applied to several representative media applications from Mediabench [5], consistently resulting in the identification of only a few data transfer and processing tasks. In Figure 3(a), we illustrate the task decomposition for the MPEG2 decoder application. Note that, the memory accesses required by the motion compensation part of the algorithm (represented by the Predictor block in Figure 3(a)) are conditioned on the motion vectors extracted from the MPEG2 input stream for each macroblock. Thus, four tasks are needed for this application. The corresponding task graph capturing those basic data dependencies, i.e., defining the required interleaving between the tasks, is shown in Figure 3(b).

Once the set of basic tasks is identified, the application code is partitioned accordingly. For example, the two processing tasks defined for the MPEG2 decoder, namely, $Task1\_P$ and $Task2\_P$, can be obtained by partitioning the actual application code – in our case, the C program available in the Mediabench benchmark suite – into the two shaded components indicated in Figure 3(a). In contrast, data transfer tasks are specially written small code segments executed by Xtream-Fit's Streaming Memory Controller. For example, $Task1\_DT$ is essentially a single Xtream-Fit's *store stream* instruction, which writes back the output data object just generated, followed by a single *load stream* instruction, which prefetches a fixed size MPEG2 stream segment corresponding to the next macroblock[1].

2.2 Task Granularity and Scheduling

We define *task granularity* g to be the number of primary input data objects processed during a single execution of an application's task graph. Note that once a set of minimum granularity ($g = 1$) tasks is defined for a streaming media application (using our task decomposition methodology), parameter g can be easily increased, so that g primary stream data objects are *jointly* fetched and then jointly processed by the subsequent (granularity g) tasks. Figures 4(a) and (b) show an execution snapshot of MPEG2 tasks with granularities $g = 1$ (i.e., one macroblock decoding at a time) and $g = 2$ respectively. The importance of being able to explicitly vary task granularity will become evident in Sections 3 and 4.

We now discuss dynamic scheduling policies for data transfer and processing tasks. Although the delay of individual task types may vary significantly during the execution of a typical streaming media application, the computation to

[1]When dealing with variable size input stream data objects (e.g., compressed macroblocks in MPEG), an upper bound on the maximum size of such data objects is used by the corresponding data transfer task.

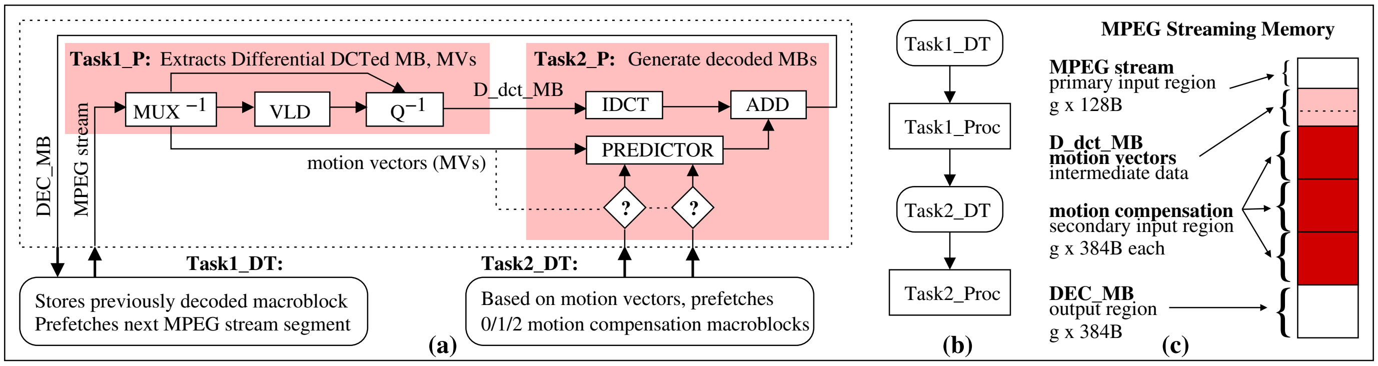

Figure 3: MPEG2 Decoder block diagram: (a) task decomposition (b) task graph (c) Streaming Memory

memory access rate remains consistently high throughout any such execution trace. In our experiments we found this ratio to be between one and two orders of magnitude, for minimum granularity tasks ($g = 1$). As task granularity increases, the dominance becomes even more significant. Those numbers essentially confirm the well known fact that media applications are computation bound, see e.g. [10, 11, 12, 13].

The delay dominance of one task type over the other has actually very positive implications. Specifically, it enables the implementation of very simple dynamic synchronization policies between processing and data transfer tasks. In order to illustrate this last point, consider again the MPEG2 decoding application and assume, a task granularity of *2* macroblocks. The dynamic scheduling policy for the processing tasks would be as follows. $Task1\_P^{g=2}$ can commence processing the new batch of *two* macroblocks as soon as the first input stream segment (corresponding to the first macroblock in the new batch) is stored in the Streaming Memory by $Task1\_DT^2$ (see Figure 4). Observe that, due to the relative delays between both tasks, no additional synchronization is needed between "producer" $Task1\_DT^2$ and "consumer" $Task1\_P^2$.

Similarly, as soon as the motion compensation data for the first macroblock is stored in the Streaming Memory by $Task2\_DT^2$, $Task2\_P^2$ can commence the processing of the corresponding macroblock. As in the previous case, no additional synchronization is needed between the two tasks. The dynamic scheduling policy for the data transfer tasks is even simpler – they are started on completion of the logically preceding processing task. The delay overhead incurred by those simple dynamic task scheduling/synchronization policies is essentially negligible – see results in Section 6.

3. XTREAM-FIT ON-CHIP DATA MEMORY

Streaming Memory. Control over (i.e., predictability of) on-chip memory accesses is enhanced by organizing Xtream-Fit's Streaming Memory into a set of regions, each capable of individually holding one of the input, output or intermediate data streams used or generated during the processing of a media application's basic input data object. The Streaming Memory for the MPEG2 decoder application, for example, is organized into six regions, with corresponding *sizes* parameterized by task granularity g – see Figure 3(c). Accordingly, a Streaming Memory with task granularity 2, has roughly twice the size of a Streaming Memory with task granularity 1, and so forth.

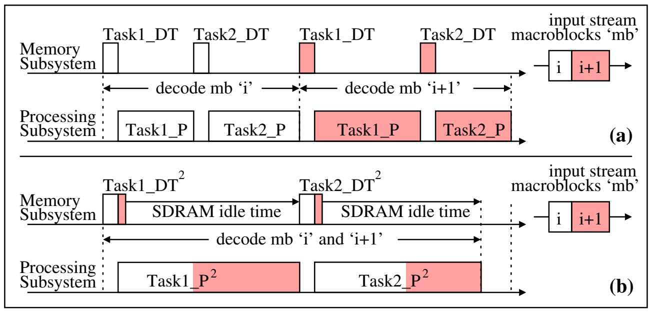

Figure 4: MPEG2 decoder: Scheduling snapshot

Scratch-Pad Memory. Power hungry off-chip data accesses are minimized in our proposed memory subsystem by mapping all constants and scalar variables, to an on-chip partition of main memory, i.e., a Scratch-Pad memory [14, 15, 16, 17]. Our analysis and experiments confirm the results of the study in [17], showing that a memory size of *2 KB* is sufficient to hold all such data for most Mediabench programs.

4. ENERGY CONSERVATION POLICIES

4.1 SDRAM: Burst Access, Low-Power Modes

Modern SDRAMs cache the most recently accessed row of each bank on a row buffer. While a row is active in the buffer, an arbitrary number of single-cycle burst mode (read and write) accesses to the row can be performed [6, 7]. By aggressively exploiting burst mode, one can substantially decrease the average power consumption and delay of memory accesses to individual stream elements [3, 13].

Xtream-Fit's data transfer tasks provide the proper scope for such an exploitation, namely, they allow embedded system designers to actually *program* (that is, statically ensure) burst mode prefetching and delayed storage of stream segments of a predefined, optimized size (see Section 4.3 on Design Space Exploration). For example, consider $Task1\_DT$ of the MPEG2 decoder application, which starts by writing back a previously decoded macroblock and then prefetches a fixed size input stream segment. A proper (sequential) layout of the corresponding MPEG2 input and output streams in the SDRAM enables both of these accesses to be performed in burst mode. The task granularity parameter g, defined with respect to the media streams' basic data objects (e.g., macroblocks for MPEG2), establishes the degree to which one wishes to take advantage of burst mode read/write SDRAM accesses during the execution of data transfer tasks.

Yet another advantage of our proposed task based processing model is that it ensures well defined, consolidated intervals of off-chip memory idleness, see e.g., Figure 4(b). This is critical for effectively exploiting the low power modes of operation available in such memories, particularly when exit latencies are non-negligible. In our initial experiments, we considered two types of off-chip memory, namely, Rambus DRAM modules [18] (high bandwidth, several low power modes, some with considerable exit latencies) and low power SDRAM modules (lower bandwidth, single low power mode with a small exit latency) [19, 20]. Since the low power SDRAM memory always provided sufficient memory bandwidth at a much lower energy cost than the RDRAM memory, solutions based on the latter were abandoned. We adopted a simple policy, which switches the SDRAM to its low power mode as soon as a data transfer task concludes execution, since it consistently led to minimum energy consumption, with no noticeable impact in performance.

4.2 Software Controlled Streaming Memory: Reducing Leakage Power

The *region-based* organization of Xtream-Fit's Streaming Memory allows for the implementation of simple and yet highly effective selective memory shut down policies, as tasks execute on both subsystems. In Figure 4(b) for example, as $Task1\_P^{g=2}$ concludes the processing of the 2 compressed macroblocks sequentially stored in the MPEG2 input stream region, the corresponding sub-regions where such data is stored are selectively shut down (that is, clock gated [8]). At the end of the execution of $Task1\_P^2$, and during the time interval until $Task1\_DT^2$ starts the prefetching of the next batch of compressed macroblocks, the entire MPEG2 input stream region will remain shut down. Note also that, as a result of similar shut down policies implemented by previous tasks, the MPEG2 output stream and the two motion compensation macroblock regions will be permanently shut down during the execution of $Task1\_P^2$. Experimental results in Section 6 show that such low cost task/software driven shut down policies can considerably reduce leakage/static power dissipation on the Streaming Memory.

4.3 Design Space Exploration

Given a streaming media application, the selection of a specific processor core is driven by specific power/performance targets and other considerations which are beyond the scope of this paper. Once one or more of such cores are selected, Xtream-Fit's design space exploration process, aimed at minimizing energy-delay product in the memory subsystem, is quite simple and systematic, and can be conducted by simply varying the task granularity parameter g. Specifically, as task granularity g increases, both power dissipation and average delay of *off-chip data transfers* decreases accordingly. However, the corresponding size of the Streaming Memory, and thus *power dissipation in on-chip memory*, increases. As illustrated in Figure 5 for the MPEG2 decoder application, by simply varying parameter g, one can systematically move across those two conflicting energy consumption curves, i.e., explore the design space, so as to easily find the point of maximum energy efficiency, for a particular target processor/performance.

5. RELATED WORK

Recent work geared towards augmenting general purpose memory hierarchies with power aware features include dynamic policies aimed at minimizing cache leakage, e.g., Cache Decay [8], and SDRAM dynamic power mode control policies [6, 7].

Cache/Scratch-Pad partitioning and reconfiguration strategies include [21, 22, 23, 24, 15, 25, 26]. The above techniques focus primarily on performance optimization, as opposed to energy-delay product optimization, and this distinction is critical. Indeed, as evidenced by our experimental results in Section 6, properly "configured" hardware-controlled caches typically provide sufficient bandwidth for media applications, but are not energy efficient – see also [10, 11, 12]. Still, a number of such performance oriented schemes may also lead to energy savings, a notable example being the general-purpose Adaptive Line Size Caches [26]. Indeed, when the variation on "optimal" burst sizes over time can be effectively "tracked" by the dynamic controller, SDRAM energy savings are likely to result. However, if the pattern of "optimal" burst sizes oscillates "quickly" between very distinct values, e.g., due to the interleaving of read/write accesses to *compressed* vs. *decompressed* input/output streams, the controller may end up "stabilizing" on some "average" value over all such distinct sizes. In contrast, burst sizes in Xtream-Fit are software controlled and can be thus "optimally" programmed for each such individual stream-access, with no "tracking" delays and associated transition related energy overheads.

Energy-efficiency of on-chip caches has been directly addressed in recent research, e.g., "Cool Caches" [16] and "Region-based Caching" [28]. Region-based caching aims at reducing energy by dividing the cache along OS memory regions (viz., stack, global, heap). Cool-Caches reduce energy consumption by eliminating cache tags. Namely, scalars are mapped to a Scratch-Pad and a compile time speculative approach (using a small register area) is used to eliminate tag-lookups for non-scalar accesses. Note that, Xtream-Fit also exploits "non-conventional" memory subsystem components, namely, a Streaming Memory and Scratch-Pad Memory. In contrast, though, one of its key unique strengths is a novel task-based execution model and associated single customization parameter that exposes opportunities for efficient stream-granularity prefetching and aggressive dynamic energy conservation policies targeting, in an *integrated* way, the energy-delay efficiency of both, *on-chip* and *off-chip* memory accesses.

There is also considerable previous work on methodologies for designing high-performance energy-efficient *custom* data memory subsystems for programmable hardware accelerators, e.g. [1], as well as work on memory subsystems for high-performance coprocessors for media kernels, e.g. [12]. Most such work focuses on sheer performance maximization via an aggressive customization of processing and memory subsystems, thus targeting a segment of the embedded system's market very different from that targeted in this paper.

In conclusion, we claim that Xtream-Fit provides a unique alternative to: (1) "standard/general-purpose" data memory hierarchies enhanced with state-of-the-art power-aware features found in contemporaneous off-the-shelf processors; and (2) complex, highly specialized hierarchical data memory subsystems found on many programmable hardware accelerators. Specifically, Xtream-Fit enables an aggressive reduction of energy-delay product when compared to the first group of solutions (see Section 6), while greatly simplifying the overall customization effort (hardware and software), when contrasted to the second group of solutions. Such "middle point" is likely to be attractive for many segments of the embedded systems market.

6. EXPERIMENTS AND RESULTS

We evaluated the effectiveness of Xtream-Fit across a wide range of processor core configurations, data memory configurations and benchmarks. Specifically, we considered two ARM cores (Intel's StrongARM SA-1110 and XScale) and a MIPS R10000 core, in order to assess Xtream-Fit's relative efficiency across a variety of contexts, ranging from highly power constrained, less performance demanding embedded systems, to more performance demanding ones (see processor configurations in Table 1). The last column of Table 1 shows the corresponding Xtream-Fit based system configurations, parametrized by task granularity g. Each such configuration has a 2 KB Scratch-Pad and a Streaming Memory whose size depends on g (see Section 3). Columns 2-4 of Table 1 summarize the twelve "general-purpose" *reference* embedded system configurations used to empirically evaluate Xtream-Fit's energy-delay efficiency. The various cache configurations specified in the table correspond to actual standard/default configurations for the corresponding cores, plus a few "promising" variants created by us (e.g., considering a configuration with a 4 KB L1 D-cache for all the cores ensures that we always have a reference design point whose on-chip memory size is very close to the smallest size possibly used in Xtream-Fit, etc.). In addition, we augmented each such reference system with state-of-the-art features available (or

Table 1: System Configurations

Core	**Reference System**			**Xtream Fit**
	Name	**L1 Cache** (32B, 4-way)	**L2 Cache** (64B, 4-way)	
StrongARM SA-1110 width = 2, #ALUs = 2, in-order, mem lat = 32,1	R1a	4KB	None	X1-g
	R1b	16KB		
	R1c	32KB		
XScale, width = 2 #ALUs = 2, out-of-order, mem lat = 32,1	R2a	4KB	None	X2-g
	R2b	16KB		
	R2c	32KB		
	R2*a	4KB	64KB	
	R2*b	16KB		
	R2*c	32KB		
MIPS R10000, width = 4, #ALUs = 4, out-of-order mem lat = 64,2	R3a	4KB	64KB	X3-g
	R3b	16KB		
	R3c	32KB		

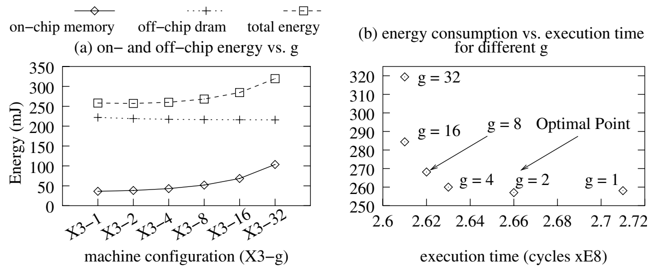

Figure 5: Design space exploration, MPEG2

easily implementable) in contemporaneous off-the-shelf processors. Note that, Cache Decay [8] policies do not work well with Adaptive Line Size Cache schemes [26]. Since Cache Decay is known to deliver significant energy savings in L1 and L2 D-caches for media applications, and the set of media compression/decompression benchmarks considered in our experiments is particularly challenging for an Adaptive Line Size Cache scheme (in that it exhibits significant variations on "optimal" burst sizes across different streams), Cache Decay was clearly the most promising technique in terms of energy savings. Accordingly, Cache Decay [8] and SDRAM power mode control policies [6, 7] were incorporated in all reference systems, and properly tuned on a per application basis.

Xtream-Fit's performance and energy efficiency was contrasted to that of the twelve reference systems for three representative benchmarks from the Mediabench suite, one from the audio, one from the video and one from the image compression/decompression domain. The benchmarks, detailed in Table 2, were compiled with optimizations turned on ($-O2$) and simulated using the Simplescalar tool-set for the ARM and the PISA instruction sets [29], and performance estimates were obtained for the corresponding input sets.

The dynamic energy consumption for the on-chip caches, Streaming Memory and Scratch-Pad components, was modeled using CACTI [30]. Similar to [8], we evaluated the leakage energy in on-chip memory structures using the low-V_t data given in Table 2 of [31]. To measure SDRAM energy, a detailed model of the mobile SDRAM from *Micron Technologies* [20], specifically, part number MT48V2M32LFC-8 @ 2.5V, was integrated into Simplescalar.

6.1 Results: Design Space Exploration

Figure 5 shows a sample of design space exploration results for the MPEG2 decoder application, considering a MIPS based embedded system, i.e., the X3-g family of configurations (see Table 1). Figure 5(a) shows the impact of the customization parameter, i.e., task granularity g, on the total energy consumed by the on-chip and off-chip memories. As expected, the energy consumption in on-chip memories increases while that in off-chip memories decreases with increasing g. For the same input set, Figure 5(b) shows the total energy consumed in on- and off-chip memories and the total decoding delay (execution cycles) for different task granularities. As seen in Figure 6(c), the minimum energy-delay product for this family of configurations is achieved for $g = 2$.

6.2 Comparative Evaluation

We now evaluate Xtream-Fit's effectiveness by comparing it to the set of alternative configurations specified in Table 1. We start with the MPEG2 decoder application – Figure 6(a) plots decoding delays for input stream mobile.mpg, with experimental results grouped/organized by processor core. For each core, we normalize the performance of Xtream-Fit to the best reference configuration. For example, in the left-most bar-chart in Figure 6(a), that contrasts reference configurations R1a, R1b, and R1c to Xtream-Fit's X1-g (i.e., considers the StrongARM-SA1110 core), the best performing granularity was $g = 1$, and the best performing reference was R1c. Accordingly, we annotate X1-1 with the fractional delay relative to R1c. As it can be seen, Xtream-Fit never performs worse than the corresponding best reference system.

Table 2: Representative Mediabench benchmarks

Program	Description	Input	Description
MPEG2	Video compression decoder	mobile.mpg	60 frames video
JPEG	Image compression encoder	lena.jpg	512x512 image
G721	Voice compression encoder based on CCITT G.711, G.721 and G.723 standards	clinton.short.pcm	32KB raw data

For the same set of experiments, Figure 6(b) plots total energy consumption on memory components, i.e., the Streaming Memory, Scratch-Pad Memory and SDRAM for Xtream-Fit, and in the cache hierarchy and SDRAM for the reference systems. As it can be seen, Xtream-Fit consistently outperforms the best reference configurations, with decreases in total energy consumption ranging from 43% to 76%.

Finally, Figure 6(c) plots the corresponding energy-delay product metric, for the same set of experiments. The results show very substantial improvements for Xtream-Fit – specifically, it decreases energy-delay product between 46% and 77%, when compared to the best performing reference configuration for any particular core. Similarly, Figures 6(d-i) plot performance, energy consumption and energy-delay product for JPEG and G721 respectively, showing energy-delay product improvements ranging from 64% to 83%.

6.3 Analysis of Results

Xtream-Fit's performance is marginally better than that of the reference systems in spite of the fact that the scheduling policy requires all subsequent processing tasks to wait until the *required data* has been prefetched into memory by the data transfer task. Indeed, Xtream-Fit's ability to take advantage of burst mode access more than offsets the "on-demand access" of modern out-of-order processors.

As shown in Figures 6(b)(e)(h), Xtream-Fit configurations consume only a fraction (17% to 57%) of the energy consumed by the best performing reference configuration. A comparative analysis between Xtream-Fit and the reference systems for the three different energy components is provided below.

On-chip Dynamic energy. The dynamic energy savings enabled by Xtream-Fit are quite significant (64% to 71% improvement). It should be noted that they are not merely a consequence of different on-chip memory sizes used by both approaches. Note, for example, that the $g = 1$ Xtream-Fit configuration for MPEG2 has a 4 KB on-chip memory (i.e., a 2 KB Streaming Memory and 2 KB Scratch-Pad) which is the same on-chip memory size used by the reference configuration with a 4 KB L1 Cache; yet there is a significant difference in dynamic energy consumption between the two systems. This difference is due to the hardware simplicity of the (software controlled) Streaming Memory and Scratch-Pad, in contrast to the more "power hungry" features of traditional caches, needed to deliver good performance (e.g., associativity).

On-chip Leakage energy. For the reference configurations, Cache Decay schemes were implemented (for both L1 and L2 caches) and the "kill-windows" were carefully tuned on a per application and configuration basis, so as to truly minimize leakage energy, without compromising performance. Cache Decay schemes work extremely well in "low reuse" media applications [8]. However, throughout our experiments, we found that Xtream-Fit's on-chip memories consistently consume significantly less leakage energy (44% to 87% improvement). This is due to two factors. First, Xtream-Fit's Streaming memory stores just enough data corresponding to the granularity g tasks. Secondly, the organization of the Streaming Memory into regions allows us to gradually shut down these regions as the data they hold becomes "dead."

Off-chip SDRAM energy. Xtream-Fit's savings in SDRAM energy consumption are also very significant (up to 98% improvement). Recall that, each individual read or write access to the SDRAM requires two separate activate/precharge actions, which consume a significant amount of energy. The ability to *individually* control burst sizes for *each* distinct data stream, via data transfer tasks, is one of the key advantages of Xtream-Fit over the reference configurations since it leads to far fewer accesses to external memory.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a special-purpose data memory subsystem for embedded media processing. Our results show that Xtream-Fit delivers up to an order of magnitude improvement in energy-delay product, as compared to a general purpose memory subsystem enhanced with state-of-the-art Cache Decay and SDRAM dynamic power mode control policies. Xtream-Fit's performance is predicated on a novel task-based execution model that exposes/enhances opportunities for efficient stream-granularity prefetching and aggressive software-based energy conservation techniques.

Xtream-Fit exposes a single customization parameter, thus enabling a very simple and yet effective design space exploration methodology for energy-delay product optimization on a per application basis. We are currently extending the approach to handle multiple applications through an appropriate interleaving of their corresponding tasks.

8. REFERENCES

[1] F. Catthoor et al. *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design.* KAP, 1998.

[2] P. R. Panda et al. Data and Memory Optimization Techniques for Embedded Systems. *ACM TODAES*, 6(2), 2001.

[3] P.R. Panda et al. *Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration.* KAP, 1998.

[4] J. Montanaro et al. A 160MHz 32b 0.5W CMOS RISC Microprocessor. *In ISSCC Digest of Technical Papers*, 1996.

[5] C. Lee et al. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. *In MICRO*, 1997.

[6] V. Delaluz et al. Scheduler-Based DRAM Energy Management. *In DAC*, 2002.

[7] X. Fan et al. Memory Controller Policies for DRAM Power Management. *In ISLPED*, 2001.

[8] S. Kaxiras et al. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. *In ISCA*, 2001.

[9] H. Zhou et al. Adaptive Mode Control: A Static-Power-Efficient Cache Design. *In PACT*, 2001.

[10] C. Hughes et al. Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications. *In MICRO*, 2001.

[11] C. Hughes et al. Variability in the Execution of Multimedia Applications and Implications for Architecture. *In ISCA*, 2001.

[12] B. Khailany et al. Imagine: Media Processing with Streams. *In IEEE Micro*, 2001.

[13] S. Rixner et al. Memory Access Scheduling. *In ISCA*, 2000.

[14] M. Kandemir et al. Dynamic Management of Scratch-Pad Memory Space. *In DAC*, 2001.

[15] P. R. Panda et al. Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications. *In ETDC*, 1997.

[16] O. Unsal et al. Cool-Cache for Hot Multimedia. *In MICRO*, 2001.

[17] O. Unsal et al. On Memory Behavior of Scalars in Embedded Multimedia Systems. *In WMPI, ISCA*, 2001.

[18] http://www.rambus.com/.

[19] http://www.micron.com/.

[20] http://www.samsung.com/.

[21] D. H. Albonesi. Selective Cache Ways: On-demand Cache Resource Allocation. *In MICRO*, 1999.

[22] L. Benini et al. A Recursive Algorithm for Low-Power Memory Partitioning. *In ISLPED*, 2000.

[23] D. Chiou et al. Application-Specific Memory Management in Embedded Systems Using Software-Controlled Caches. *In DAC*, 2000.

[24] V. Milutinovic et al. The Split Temporal/Spatial Cache: Initial Performance Analysis. *In SCIzzL*, 1996.

[25] P. Ranganathan et al. Reconfigurable Caches and their Application to Media Processing. *In ISCA*, 2000.

[26] W. Tang et al. Fetch Size Adaptation vs. Stream Buffer for Media Benchmarks. *In WMSP, MICRO*, 2001.

[27] S. VanderWiel et al. When Caches Are Not Enough: Data Prefetching Techniques. *IEEE Computer*, 30(7), 1997.

[28] H. Lee et al. Region-Based Caching: An Energy-Delay Efficient Memory Architecture for Embedded Processors. *In CASES*, 2000.

[29] D. Burger et al. Evaluating Future Microprocessors: The SimpleScalar Tool Set *Technical Report, University of Wisconsin, Madison*, 1996.

[30] S. Wilton et al. An Enhanced Access and Cycle Time Model for On-chip Caches. *Technical Report, DEC WRL*, 1994.

[31] S.-H. Yang et al. An IC/Arch Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches. *In HPCA*, 2001.

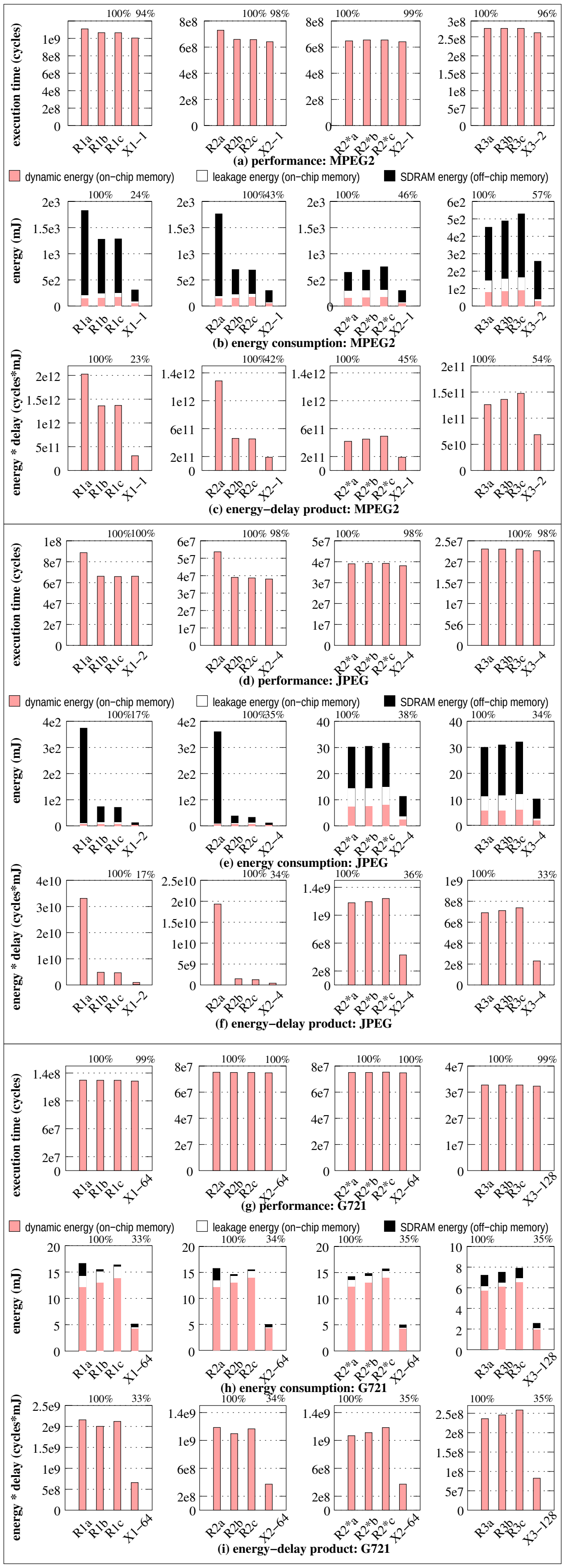

Figure 6: Comparative evaluation