The Dissertation Committee for Anand Ramachandran
certifies that this is the approved version of the following dissertation:

Energy-Aware Embedded Media Processing: Customizable Memory Subsystems and Energy Management Policies

Committee:

Margarida F. Jacome, Supervisor

Jacob Abraham

Mauricio Breternitz Jr.

Doug Burger

Nur Touba

Gustavo De Veciana

Energy-Aware Embedded Media Processing: Customizable Memory Subsystems and Energy Management Policies

by

Anand Ramachandran, B. Tech., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2004

Dedicated to my parents.

Acknowledgments

I would first like to thank my parents whose blood, sweat and tears this PhD really represents. I cannot thank them enough for their constant support and encouragement. This work is dedicated to them.

There are innumerable other people who have helped shape my life during these college years – from the B.Tech days with Bana, Ramu, HariK, SSK and Pram, through the time Pali and Koru picked me up at the old Austin airport, to the fun-filled days with "Rio- and AID-Junta." I cannot possibly do justice to all those people in these couple of pages, but I do wish to thank them all.

I would like to thank my advisor, Margarida Jacome, first, for helping me move to the computer engineering department, second, for teaching me and guiding me to do high quality research, and finally, for moulding me to be a better person. While she is known the world over for being a good researcher, people who know her first-hand, first think of her as a nice person. I hope I have imbibed some of her qualities, and hope to follow closely in her footsteps.

I would like to thank my committee members for their encouragement and support. In particular, I would like to thank Doug Burger, and his students, Hrishi and Raj for their help with the implementation of cache-decay in Simplescalar. I would also like to thank Gustavo de Veciana for his patient

help with my first paper.

A PhD can be boring and frustrating at times, and it helps to have friendly colleagues to share your good and bad times with. Helvio and Meena made the office a fun place. Chitra and Kedar were always encouraging, while Heather made work seem like fun. While Viktor taught us how to live life, Jeff showed us how to live it. Satish was a patient listener and engaging (sometimes sarcastic) conversationalist whose company I cherished. Nandu was always there to help while Madhavi patiently and empathetically listened to all my rants.

I was fortunate to have my cousin, Anand, with me during some of the toughest moments of my PhD life. His mere presence was very calming, which often helped me finish my work on time. I would also like to thank my siblings Vasanthi (and Sreeram), Ramesh, Gayetri (and Bhaskar) for their patience and continued support during my long time at school.

Finally, I would like to thank my wife, Gayathri, who has been a pillar of strength and support during these years. Her boundless patience and love helped me ride through the difficult times with ease, making this PhD the most rewarding experience of my life.

Energy-Aware Embedded Media Processing: Customizable Memory Subsystems and Energy Management Policies

Publication No. _____

Anand Ramachandran, Ph.D.
The University of Texas at Austin, 2004

Supervisor: Margarida F. Jacome

The design of energy-efficient data memory architectures for embedded system platforms has received considerable attention in recent years. In this dissertation we propose a special-purpose data memory subsystem, called Xtream-Fit, targeted to streaming media applications executing on both generic uniprocessor embedded platforms and powerful SMT-based multi-threading platforms. We empirically demonstrate that Xtream-Fit achieves high energy-delay efficiency across a wide range of media devices, from systems running a single media application to systems concurrently executing multiple media applications under synchronization constraints. Xtream-Fit's energy efficiency is predicated on 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. 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 to find the best memory configuration for the target application(s). Extensive experimental results show that Xtream-Fit reduces energy-delay product substantially – by 32% to 69% – as compared to 'standard' general-purpose memory subsystems enhanced with state of the art cache decay and SDRAM power mode control policies.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Embedded systems are pervasive in modern life. State-of-the-art embedded technology drives the ongoing revolution in consumer and communication electronics, and is on the basis of substantial innovation in many other domains, including medical instrumentation, process control, etc. [40]. 32-bit embedded processors account for more than 90% of the marketshare [46]. Portable electronics and telecommunication equipment constitute a large segment of this embedded market. In fact, of the 500 million ARM processors sold in 2002 more than 80% were used in mobile consumer electronics equipment, such as cell-phones and personal digital assistants (PDAs) [46]. Marketwatchers [5] further predict that the media processing class of applications will soon dominate the consumer electronics market — this domain is very vast, including applications for image, audio and video encoding and decoding, speech and handwriting recognition, etc.

The ability to realize such complex applications in cost effective devices is fueled primarily by Moore's Law, which enables the fabrication of ever faster and denser integrated circuits (ICs). Unfortunately, as CMOS technology rapidly scales, the challenges that must be overcome to deliver each new

generation of electronic products multiply. In the last few years, power dissipation has emerged as a major concern. In fact, projections on power density increases due to CMOS scaling clearly indicate that this is one of the fundamental problems that will ultimately preclude further scaling [7, 29]. Although the power challenge is indeed considerable, much can be done to mitigate the deleterious effects of power dissipation, thus enabling performance and device density to be taken to truly unprecedented levels by the semiconductor industry throughout the next ten to fifteen years.

Power dissipation has a direct impact on packaging and cooling costs, and can also affect system reliability, due to electro-migration and hot-electron degradation effects. Thus, the ability to decrease power, while offering similar performance and functionality, critically enhances the competitiveness of a product. Furthermore, for battery operated portable systems, maximizing battery lifetime translates into maximizing duration of service, an objective of paramount importance for this class of products. Thus, power and energy have emerged as *primary figures of merit* in contemporaneous embedded system design.

Embedded systems come in many varieties and with many distinct design optimization goals and requirements. Even when two products provide the same basic functionality, say, video/audio playback, they may have fundamentally different characteristics, namely, different performance and quality-of-service requirements, one may be battery operated and the other not, etc. The implications of such product differentiation are of paramount importance

when power and energy are considered. Clearly, the higher the system's required performance/speed (defined by metrics such as throughput, latency, bandwidth, response time, etc.), the higher will be its power dissipation [21] — the key question is, how much higher does it need to be? To answer this question, i.e., to realize highly energy-efficient systems, one may need to consider static and dynamically adaptive energy conservation techniques, consider specialized architectures, and exploit the specific characteristics of the target embedded application on 'specialized' devices/subsystems. There is, thus, an increasing interest in the design of 'energy-aware' systems, where the objective is to *minimize* the energy spent to deliver the required level of performance and/or quality of service guarantees [22, 30].

In this research we focus on *energy-aware* design methodologies targeting the data memory subsystem of embedded media processing systems. We consider a generic embedded system architecture with a single processor core and associated data and program memory subsystems. Although there are certainly many alternative platforms, e.g., using multiple processors and/or programmable media streaming hardware accelerators, such as *Imagine* [34], the generic embedded system platform targeted in this research is very relevant in today's embedded systems market, particularly in the context of systems with tight energy budgets and non-stringent performance requirements [23]. For such a platform, on-chip caches are typically responsible for a significant percentage of a chip's overall power dissipation, e.g., more than 40% for the StrongARM-110 is reported in the study presented in [41]. Yet another data

point is presented in [55], where the energy consumption of a SmartBadge[1] decoding an MPEG video is analyzed. The study indicates that when a low power SRAM is used to implement on-chip memory, it consumes approximately 30% of the total system energy. Thus, due to its substantial impact on both energy consumption and performance, the memory subsystem is a prime candidate for customization to an application's requirements by embedded system designers [12, 43, 45, 50], thus motivating the work presented in this dissertation.

The main contributions of this work are:

1. A special-purpose data memory subsystem, called Xtream-Fit, targeted to a generic uniprocessor embedded platform or an SMT-based multithreading platform, executing streaming media applications. We empirically demonstrate that Xtream-Fit achieves high energy-delay efficiency across a wide range of media devices, from systems running a single media application to systems concurrently executing multiple applications under synchronization constraints.

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.

[1] A SmartBadge is an embedded system consisting of a StrongARM-1100 processor, FLASH and SRAM memories, sensors, and a modem/audio analog front end on a printed circuit board (PCB) powered by batteries through a DC-DC converter.

3. A tuning methodology for the memory subsystem that exposes a single customization parameter, thus enabling simple and yet effective design space exploration methodology to find the best — (i.e., most energy efficient) memory configuration for the target application(s).

The organization of this dissertation is as follows — We first present relevant background (Chapter 2) and survey previous research in the area of energy-efficient data memory subsystems (Chapter 3).

We then present our core research contributions — viz., a task-based processing model for media applications (Chapter 4) that works in conjunction with a novel data memory subsystem (Chapter 5) to achieve high energy-delay efficiency for media applications. We then present experimental results for embedded systems running a single media application (Chapter 6). We then discuss a generalized model of Xtream-Fit for embedded systems concurrently running multiple media applications (Chapter 7) and present experimental results for the same (Chapter 8). We conclude with a brief summary and present opportunities for future work (Chapter 9).

Chapter 2

Background

There are two sources of power dissipation in synchronous CMOS circuits: dynamic and static. In this chapter we start by briefly discussing these two components of power dissipation. We then provide some background on state-of-the-art system level power models for memory components used to support architecture-level design space exploration. We finally conclude with a brief discussion on the energy-efficiency metric adopted in this work.

2.1 Power Dissipation in CMOS Circuits

Digital CMOS circuits have two main types of power dissipation: *dynamic* and *static*. Dynamic power is dissipated when the circuit performs the function(s) it was designed for, e.g., logic and arithmetic operations (computation), data retrieval, storage, and transport, etc. Ultimately all of this activity translates into switching of the logic states held on circuit nodes. Dynamic power dissipation is thus primarily proportional to $C.{V_{DD}}^2.f.r$, where C denotes the total circuit capacitance, V_{DD} and f denote the circuit supply voltage and clock frequency, respectively, and r denotes the fraction of transistors ex-

pected to switch at each clock cycle [20, 47].[1] In other words, dynamic power dissipation is impacted to first order by circuit size/complexity, speed/rate, and switching activity.

In turn, static power dissipation is associated with preserving the logic state of circuit nodes in the absence of switching activity, and is caused by leakage currents in a transistor when it has been turned off. While barely negligible a few technology generations ago, static power dissipation in CMOS circuits is becoming substantial, and the mechanisms that cause such leakage are worsening as CMOS scaling progresses [8]. Indeed, leakage currents increase exponentially with increasing chip temperature and with decreasing threshold voltage. As CMOS technology scales, and the number of devices that can be integrated in the same area increases, chip power density also increases with corresponding deleterious effects on temperature. Moreover, with the scaling down of supply voltages that accompanies device scaling, threshold voltages need to be scaled down to deliver faster transistors making leakage currents and thus static power dissipation increasingly problematic.

2.2 Power Models for Memory

This section addresses high-level modeling and power estimation techniques for memory elements, aimed at assisting early system and architecture-

[1] Short circuit power dissipation, which occurs due to the presence of a direct current path from V_{DD} to GND during the process of actual switching is usually classified as dynamic power dissipation. However, this power dissipation is quite small when compared to other sources of power dissipation.

level design space exploration. First, one should note that it would be unrealistic to expect a high degree of accuracy on power estimates produced during such an early design phase, since accurate power modeling requires detailed physical level information that may not yet be available. Moreover, highly accurate estimation tools (working with detailed circuit/layout level information) would be too time consuming to allow for any reasonable degree of design space exploration [9, 40, 47]. Thus, practically speaking, power estimation during early design space exploration should aim at ensuring a high degree of fidelity rather than necessarily accuracy. Specifically, the primary objective is to assess the relative power efficiency of different candidate memory architectures. Estimates that correctly expose relative power trends across the design space provide the designer with the necessary information to guide the exploration process.

Dynamic Power. The high regularity of memory structures (e.g., caches) permits the use of simple, yet good fidelity power estimation techniques, relying on automatically synthesized "structural designs" for such components. CACTI (Cache Access and Cycle TIme) is a widely used memory model that implements this synthesis-driven power estimation paradigm. Specifically, given a specific cache hierarchy configuration (defined by parameters such as cache size, associativity, and line size), as well as information on the minimum feature size of the target technology, it internally generates a coarse structural design for such a cache configuration [63]. It then derives delay and power estimates for that particular design, using parameterized

built-in C models for the various constituent elements, namely, SRAM cells, row and column decoders, word and bit lines, pre-charge circuitry, etc. [31, 53]. Thus, during design space exploration, a designer may consider a number of alternative L1 and L2 cache configurations, and use CACTI to obtain "access-based" power dissipation estimates for each such configuration. Naturally, the memory access traces used by CACTI should be generated by a micro-architecture simulator (e.g., Simplescalar) working with a memory simulator (e.g., Dinero [16]), so that they reflect the bandwidth requirements of the embedded application of interest.

Static/Leakage Power. Similarly, a high-level model for static energy consumption in on-chip SRAM structures is obtained by considering the leakage energy in the memory arrays.[2] Static energy is typically estimated for a single SRAM cell using low-level SPICE simulations and is then extrapolated for the number of cells in memory. In this work, we use the leakage energy per bit presented in [64] for estimating leakage energy in SRAMs.

Off-Chip SDRAM Power. Detailed specifications on operating modes, operating currents and timing for off-chip memories, (e.g., SDRAM parts) are typically available from vendors. Using such parameters, simple off-chip memory models can be built to generate energy consumption estimates by simply analyzing an off-chip access trace. In simple terms, the *dynamic* energy consumed in an off-chip memory access is computed by summing up the energy

[2]Typically, the leakage power dissipated in the rest of the memory is negligible, and for simplicity, can be discarded [26].

needed to activate (open) an SDRAM bank row buffer, the energy used by the actual read/write operation, and the energy needed to precharge (close) the SDRAM bank row buffer.[3] (As indicated before, such off-chip access traces are typically obtained by tracking the reads and writes using cycle accurate simulations). *Static* energy in off-chip memory, in turn, is estimated by adding the self-refresh energy and the standby energy consumed in the memory during different modes of operation, for the duration of the program.

2.3 Assessing Energy Efficiency

A design is "energy-efficient" if it can deliver "maximum" performance for a given energy budget or, conversely, "minimize" energy for a given performance target. Accordingly, assessing the energy efficiency of a system requires considering, both, performance and energy, simultaneously. The seminal paper [21] suggests the use of Energy-Delay Product as a simple metric to assess the energy efficiency of a system. This metric is very well suited to compare alternative systems that deliver a similar performance level, and will thus be used throughout this dissertation to assess the relative energy-efficiency of memory subsystems under such conditions.

[3]The cost of opening/closing a row is amortized by performing several read/write operations from that row together in burst mode.

Chapter 3

Related Work

Xtream-Fit's high energy efficiency is predicated on a novel task-based execution model that exposes/enhances opportunities for exploiting *stream granularity prefetching* and advanced dynamic *energy conservation* techniques. In this chapter, we briefly review previous contributions to those areas.

3.1 Data Prefetching

There is a significant body of research on hardware and software techniques for data prefetching. *Hardware based* data prefetching techniques try to predict when a given piece of data will be needed, so as to load it into cache before it is actually referenced by the application (i.e., used by a demand access), see e.g. [13, 19, 48]. In contrast, *software based* prefetching techniques work by inserting prefetch instructions for selected data references at carefully chosen points in the program - such explicit prefetch instructions are executed by the processor, to move data into cache, see e.g., [11, 36, 42, 66]. Advanced, highly efficient software-based prefetching schemes, based on the independent execution of stream granularity load/store instructions, have also been proposed, mostly in the context of special purpose processors and hardware ac-

celerators [12, 34]. In the sequel we briefly review Imagine, a stream-oriented coprocessor/accelerator for media kernels that exploits such techniques [34].

3.2 Special Purpose Memory Subsystems for Programmable Accelerators

The Imagine accelerator chip is controlled by a host processor, which sends it stream instructions, namely, *load* and *store* instructions, to move streams between memory and a *128 KB* stream register file (SRF); *operate* instructions, to invoke kernels residing on the control store of Imagine's microcontroller, etc. During kernel execution, the same instruction executes on Imagine's eight arithmetic clusters in single-instruction, multiple-data mode [34]. In contrast, Xtream-Fit is a special purpose memory subsystem designed for less performance demanding embedded systems, where a single microprocessor core (say, a StrongARM SA-1110) executes the complete streaming media application. Similarly to our work, Imagine's stream memory transfers are executed independently from computations, that is, the memory subsystem can be prefetching streams to the SRF (for the next kernel execution) while the current kernel is executing [34]. In sharp contrast, though, the host processor is ultimately responsible for controlling the "dynamic" pace of such stream transfers and kernel executions, as it sends stream instructions to the Imagine chip. Moreover, to the best of our knowledge, no dynamic energy management policies have (so far) been implemented in Imagine. Although such issues/features may have a questionable relevance for a media kernel acceler-

ator such as Imagine, they are critical to the energy-efficiency of the class of embedded system architectures targeted in this dissertation. Similarly, in [15] a methodology to evaluate memory architecture design trade-offs for multiprocessor video signal processing is proposed, however, energy-efficiency is not addressed in the paper, nor are energy saving policies proposed for the targeted memory architectures.

3.3 Novel Memory Subsystem Designs

Seminal work on decoupled access/execute architectures, explicitly separating access and computation *instructions*, can be found in [49, 57]. The effectiveness of keeping a small partition of main memory (that is, a Scratch-Pad) on-chip has been extensively studied in the literature, see e.g. [12, 23, 32, 43, 45]. Most such contributions consider embedded system architectures with a standard cache hierarchy and a Scratch-Pad memory, and propose methods for deciding which data to assign to the Scratch-Pad. Several novel cache designs have also been proposed in recent years, see e.g. [6, 25, 52, 58]. 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 6.7, properly "configured" hardware-controlled caches typically provide sufficient bandwidth for media applications, yet they are not energy efficient – see also [56]. 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 in [58].

Indeed, when the variation on "optimal" burst sizes over time can be effectively "tracked" by the dynamic controller proposed in [58], 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 input/output 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., "Region-based Caching" [24] and "Cool Caches" [60]. Region based caching is an elegant horizontal partitioning strategy aimed at reducing power dissipation by exploiting the nature of memory allocation conventions [24]. Significant reduction in energy-delay product (in comparison to a conventional cache design) is reported for Mediabench applications, when two small (*2 KB*) horizontally partitioned region-based D-caches are added to the regular L1 D-cache, one for stack data and one for global data [24]. The Cool-Cache architecture, yet another novel memory subsystem design, significantly reduces energy consumption for multimedia applications (from 25% up to 77%, when compared to *64 KB direct mapped* and *4-way* caches) by eliminating cache tags – namely, scalars are directly mapped into a Scratch-Pad memory area, and a compile-time speculative approach using a small register area is used to eliminate tag-lookup for non-scalar memory accesses [60].

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 that exposes/enhances opportunities for efficient stream-granularity prefetching and aggressive dynamic energy conservation policies targeting, both, on-chip and off-chip memory components.

3.4 Techniques for Cache Leakage Power Reduction

Several techniques for reducing leakage power of caches have been proposed in recent years – the key idea behind such techniques is to turn-off cache lines holding data that is not likely to be reused [33, 65]. We found that, in general, such techniques work extremely well in the context of media benchmarks, due to the "well-behaved" stream-oriented patterns of memory access and low degree of data reuse characteristic of such applications. In fact, simple cache line turn-off policies (e.g., cache decay [33], which wait for a fixed number of cycles since the last access to a line before shutting it down), perform as well as more sophisticated/costly adaptive policies. In Section 6.8, we contrast leakage power reduction results achieved by cache decay versus our software/task driven shutdown policies.

3.5 Techniques for Exploiting SDRAM Low-Power Modes

Dynamic energy conservation techniques based on selectively controlling the operating mode of off-chip SDRAM modules have also been proposed in recent years, see e.g., [14, 17]. Most techniques relevant to the embedded system architecture targeted in this paper assume a standard cache hierarchy and target Rambus DRAM technology with multiple power saving modes [1]. The key challenge is to avoid spurious/costly mode transitions when the RDRAM idles for only a "small" number of cycles. As discussed in Section 5.2.1, Xtream-Fit greatly enhances the effectiveness of power mode control policies (for streaming media applications), by consolidating SDRAM activity/idleness.

3.6 Summary

In the chapters that follow, we will show 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 experimental results in Section 6.7), 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.

Chapter 4

A Processing Model for Media Applications

We propose a special-purpose data memory subsystem called *Xtream-Fit* (see Figure 4.1) and an associated processing model aimed at achieving high energy-delay efficiency for streaming media applications [50, 51]. In this chapter we discuss the processing model and detail the memory organization and associated energy saving policies in the next.

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-

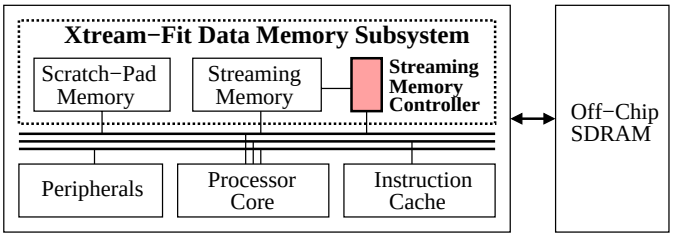


Figure 4.1: Embedded media processing platform with Xtream-Fit.

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 Section 4.1 we discuss the key characteristics of streaming media applications and the Xtream-Fit processing model. Section 4.2 illustrates the application of this task-based decomposition methodology for a media application. Section 4.3 defines the notion of *task granularity* and describes a scheduling policy for the synchronized execution of these tasks.

4.1 A Task-Based Model for Media Processing

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. For example, MPEG2 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

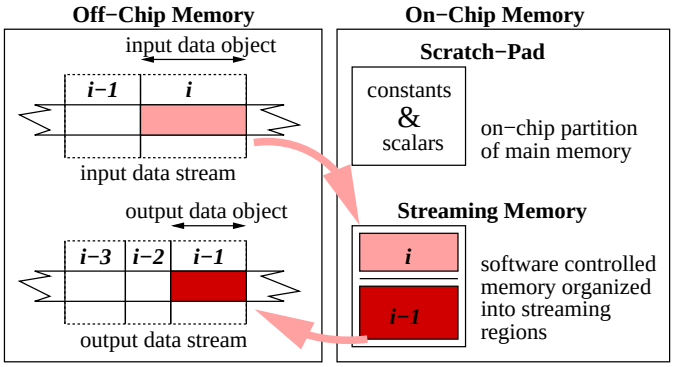


Figure 4.2: Efficient on-chip storage for streaming media applications.

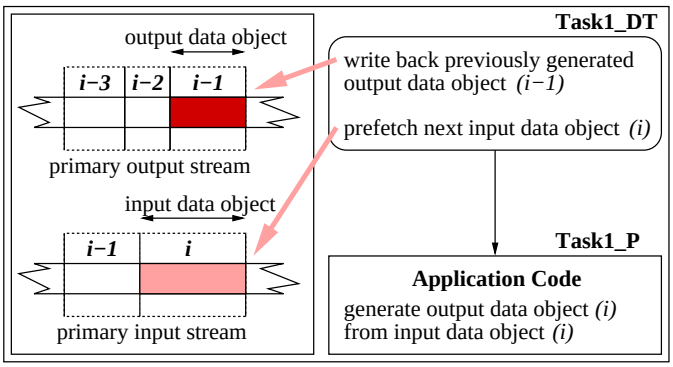
block basis (8x8 pixel block), etc. [38].

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 [18, 34, 54]. 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 4.2) provide energy-efficient on-chip storage for these two types of data: (1) constants and scalars; and (2) low reuse streaming data.

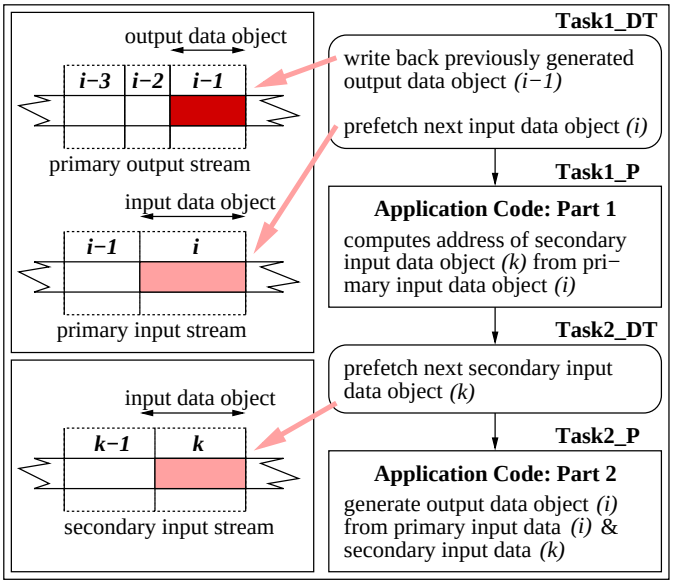
At the core of this work lies the idea of decomposing the media application into tasks, which encapsulate continuous processing loads for one of the two main architectural subsystems, namely, the data memory subsystem (Xtream-Fit) and the processing subsystem (processor core). Thus, once a task begins 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



(a) Media application with minimum number of tasks



(b) Media application with four tasks

Figure 4.3: Task decomposition methodology: key principles.

back data streams, while the second processes/generates those streams. Figure 4.3(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 4.3(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 4.3(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 4.3(b)) is defined for that purpose. Similarly, in order to 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 4.3(b).

4.2 Task Decomposition: MPEG2 Decoder Example

The above task decomposition methodology was successfully applied to several representative media applications from Mediabench [38], consistently resulting in the identification of only a few basic data transfer and processing tasks. A natural result of this task decomposition is a consolidation of idle times, i.e., avoiding scattered fine grain interleaving of stalls, which are

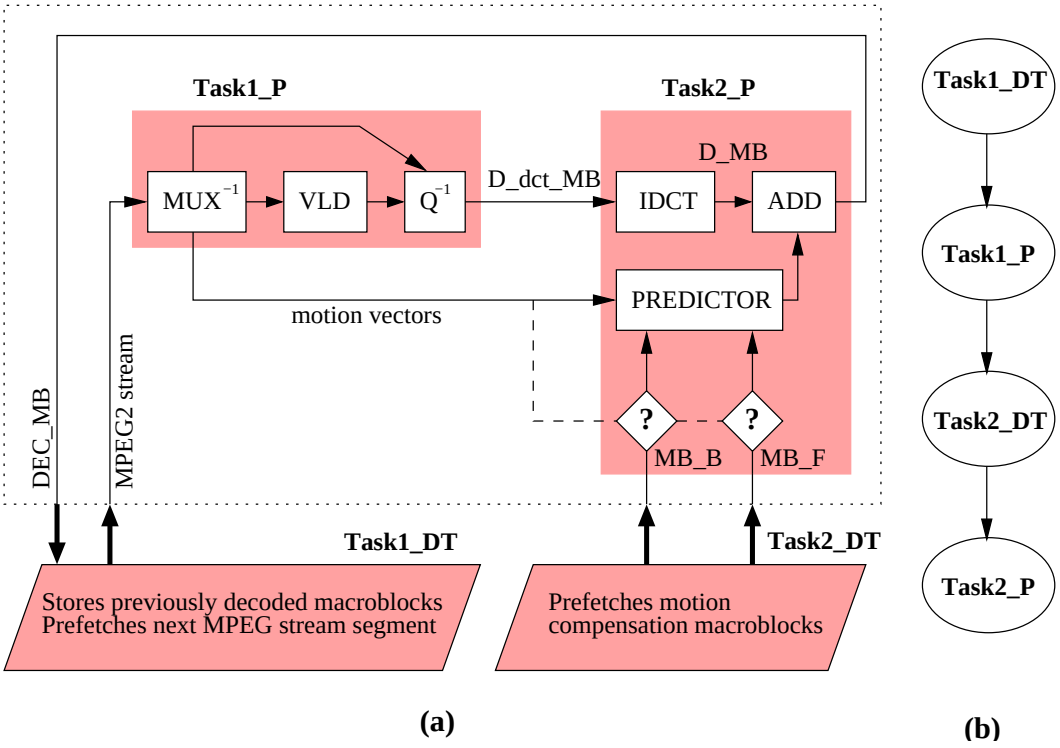


Figure 4.4: Block diagram of the MPEG2 decoder.

difficult to exploit for energy conservation purposes. We now illustrate our task decomposition methodology for a real media application. Consider the block diagram of an MPEG2 decoder shown in Figure 4.4 (a) – the input is the MPEG2 input stream (essentially, a sequence of compressed macroblocks), and the output is a corresponding stream of decoded/decompressed macroblocks (each denoted DEC_MB in Figure 4.4 (a)) [38].

Observe that, in the MPEG2 decoder application, the memory accesses required by the motion compensation part of the algorithm (represented by the PREDICTOR block in Figure 4.4 (a)) are conditioned on the motion vectors

extracted from the MPEG2 input stream for each macroblock (see inputs to PREDICTOR block in Figure 4.4 (a)). Based on this simple observation, one can define four main tasks for the MPEG2 decoder application, two executing on the processing subsystem, and two executing on the memory subsystem. These tasks have a level of granularity corresponding to a single macroblock, that is, process one macroblock at a time, and are as follows:

- $Task1\_DT$ (data transfer task): stores (writes back) a previously decoded macroblock into main memory (thus releasing space in the Streaming Memory for the macroblock about to be decoded), and then prefetches a fixed size MPEG2 stream segment[1] into a corresponding Streaming Memory region[2].

- $Task1\_P$ (processing task): extracts the differential DCTed macroblock (denoted D_dct_MB in Figure 4.4 (a)) and the motion vectors from the previously stored MPEG2 stream segment, and stores them in the corresponding Streaming Memory regions.

- $Task2\_DT$ (data transfer task): based on the motion vectors in storage, prefetches zero, one, or two motion compensation macroblocks (denoted MB_B and MB_F in Figure 4.4 (a)), from previously decoded frames.

[1]When dealing with variable size input stream data objects (e.g., compressed macroblocks in MPEG2), an upper bound on the maximum size of such data objects is used by the corresponding data transfer task.

[2]Streaming Memory organization into regions is discussed in Chapter 5

- $Task2\_P$ (processing task): performs an inverse discrete cosine transform (IDCT) on the previously extracted D_dct_MB macroblock, producing a differential macroblock, averages two motion compensation macroblocks (if needed), adds the result (or a single motion compensation macroblock) to the previous differential macroblock (if needed), and stores the decoded macroblock (denoted DEC_MB in Figure 4.4 (a)) in the corresponding Streaming Memory region.

The corresponding task graph capturing those basic data dependencies, i.e., defining the required interleaving between the four tasks, is shown in Figure 4.4(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 this case, the C program available in the Mediabench benchmark suite – into the two shaded components indicated in Figure 4.4(a). In contrast, data transfer tasks are specially written small code segments that are 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.

4.3 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 input stream data objects are *jointly* fetched and then jointly processed by the subsequent (granularity g) tasks. Figures 4.5(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 5.1 and 5.2.

We now discuss dynamic scheduling policies for data transfer and pro-

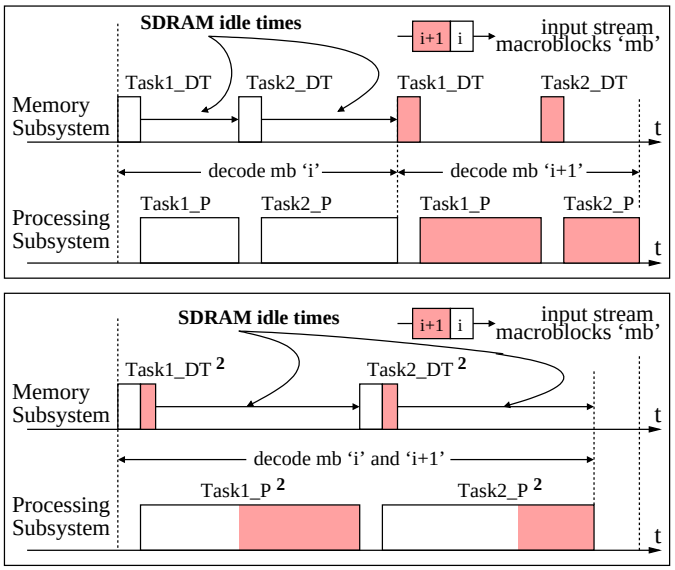


Figure 4.5: Scheduling snapshot for $g = 1$ and $g = 2$ (not to scale).

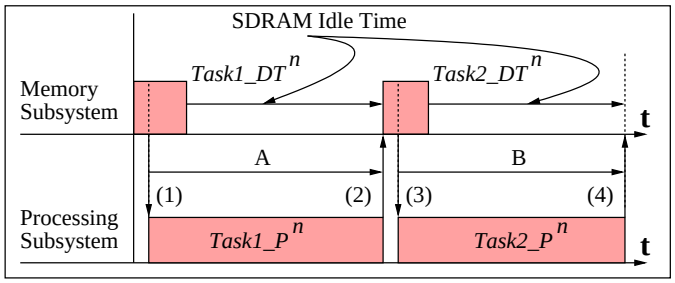


Figure 4.6: Scheduling snapshot for arbitrary g (not to scale).

cessing tasks. Although the delay of individual task types may vary significantly during the execution of a typical streaming media application, the computation to 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. [27, 28, 34, 54].

The delay dominance of one task type over the other has actually very positive implications. Specifically, it enables the implementation of remarkably simple dynamic synchronization policies between data transfer and processing tasks. In order to illustrate this last point, consider again the MPEG2 decoding application and assume, for generality, a task granularity of n macroblocks. The dynamic scheduling policy for the processing tasks would be as follows. $Task1\_P^n$ can commence processing the new batch of 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 $Task1\_DT^n$ – this is indicated by arrow (1) in Figure 4.6. Observe that, due to the relative delays between both tasks, no additional synchronization is needed between "producer" $Task1\_DT^n$ and "consumer" $Task1\_P^n$.

Similarly, as soon as the motion compensation data for the first macroblock is stored in the Streaming Memory by $Task2\_DT^n$, $Task2\_P^n$ can commence the processing of the corresponding macroblock – this is indicated by arrow (3) in Figure 4.6. 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, as indicated by arrows (2) and (4) in Figure 4.6. The delay and energy overhead incurred by those simple dynamic task scheduling/synchronization policies is essentially negligible.

These scheduling policies could be implemented very simply – e.g., on the completion of granularity g tasks, the processing system could signal the Streaming Memory Controller (either directly, or with the aid of the underlying operating system), and wait for the Streaming Memory Controller to complete the data transfers and signal it to start again. We only need to keep track of the number of processing tasks that have already been executed since the last data transfers.

Chapter 5

Memory Organization and Energy Management Policies

Xtream-Fit's data memory subsystem replaces a traditional on-chip data cache with a Streaming Memory and a Scratch-Pad as shown in Figure 4.1. The high energy efficiency of Xtream-Fit results from a novel, synergistic combination of these special-purpose memory subsystem components and the task-based processing model that enhances the *predictability* and *efficiency* of resource usage. Tuning Xtream-Fit to the requirements of a specific media application and target performance, i.e., optimizing energy-delay efficiency, is a surprisingly simple process, involving a *single* customization parameter - *task granularity* (g). Providing the ability to explore the complex energy trade-offs involved in such an optimization, by varying just this single parameter, is yet another key novel contribution of this work. Section 5.1 describes the main components of the data memory subsystem and Section 5.2 presents the energy conservation policies. Section 5.3 details the design space exploration process for Xtream-Fit.

5.1 Memory Organization for Media Processing

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. Data traffic in and out of the memory subsystem is controlled by the Streaming Memory Controller, which executes the data transfer tasks described in Section 4.1.

5.1.1 Software Controlled 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 the media application's basic input data objects. 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 5.1. Accordingly, a Streaming Memory with task granularity 2, has twice the size of a Streaming Memory with task granularity 1, and so forth.

5.1.2 Scratch-Pad memory

In our proposed memory subsystem, power hungry off-chip data accesses are "minimized" by mapping/assigning all constants (and/or infrequently changed data) required by the media application, as well as scalar

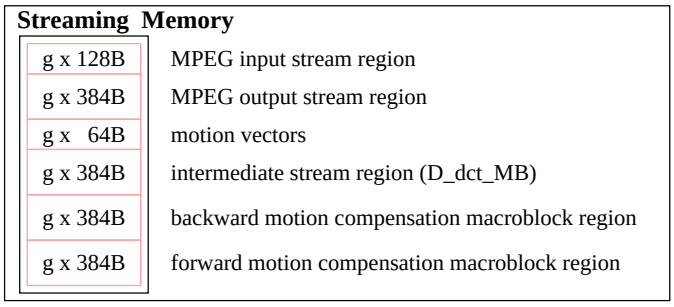


Figure 5.1: Organization of Streaming Memory regions for MPEG2 decoder.

variables, to Xtream-Fit's on-chip Scratch-Pad memory. [32, 44, 60, 61]. Accordingly, the Scratch-Pad SRAM is mapped into an address space disjoint from the off-chip main memory, yet, is connected to the same address and data buses. In the case of the MPEG2 decoder benchmark, roughly *2 KB* is needed to store arrays of quantization and IDCT coefficients, tables of header identifier codes, etc. As one would expect, this size is essentially invariant with respect to the selected task granularity. Furthermore, a memory size of *2 KB* is sufficient for most Mediabench programs [61].

5.1.3 Streaming Memory Controller

Data traffic in and out of the memory subsystem is controlled by the Streaming Memory Controller, which executes the data transfer tasks. The Streaming Memory Controller could be implemented by augmenting an existing *Direct Memory Access (DMA) Controller*, on the processor core. Alternatively, it could be implemented using a light-weight, low-power microcontroller, programmed to execute data-transfer tasks, turn on and turn off Streaming

Memory regions, etc.

5.2 Energy Conservation Policies

5.2.1 SDRAM: Burst/Page Mode 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/page mode (read and write) accesses to the row can be performed. By aggressively exploiting burst/page mode, one can substantially decrease average power consumption and delay of memory accesses to individual stream elements [45, 54].

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/page mode prefetching and delayed storage of stream segments of a predefined, optimized size (see Section 5.3 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/page mode. The task granularity parameter g, defined with respect to the media stream's basic data objects (e.g., macroblocks for MPEG2), establishes the degree to which one wishes to take advantage of burst/page mode read/write SDRAM accesses during the execution of data transfer tasks.

Furthermore, as tasks execute on both subsystems, one can actually *predict* when the next burst of SDRAM accesses will occur (namely, at the start of a new data transfer task), as well as when the next (consolidated) idling period will commence (corresponding to the end of a data transfer task). Such an ability to ensure well-defined, consolidated intervals of SDRAM idleness is critical to the effectiveness of energy conservation techniques exploiting SDRAM low-power modes of operation, particularly when exit latencies from such modes are non-negligible. In order to better access such benefits, our initial experiments considered two types off-chip memory, namely, Rambus DRAM [1] and low-power SDRAM modules [2, 3].

RDRAM's multiple power modes (namely, *active*, *standby*, *nap* and *power-down*) have corresponding resynchronization costs (i.e., exit latencies) varying from only a few cycles (standby $\rightarrow$ active) up to several thousand cycles (power-down $\rightarrow$ active). Note that, due to such resynchronization costs, RDRAM's idling intervals need to be *long enough*, so as to actually enable substantial energy savings with minimal impact on performance. It follows that Xtream-Fit's consolidation of off-chip main memory accesses into large, well defined time intervals, corresponding to the execution of data transfer tasks, is very effective from a power mode control standpoint. Indeed, consolidation of activity and idleness creates more opportunities for energy conservation, while decreasing the potential impact of resynchronization cycles in performance. In particular, note that by varying Xtream-Fit's task granularity parameter g, one can directly control the average duration of RDRAM's idling times and

corresponding frequency of transitions between idleness and activity (see e.g., Figures 4.5(a) and (b)).

However, for the generic class of embedded system architectures targeted in this dissertation, we found that low-power SDRAM modules such as the ones available in [2, 3], are able to provide sufficient memory bandwidth for the compute bound (not memory bound) streaming media applications of interest. In other words, when compared to systems using RDRAMs, system architectures using those low power memories delivered identical performance and throughput at a much lower energy cost. Thus, for the sake of considering system architectures that truly optimize energy consumption, in Chapter 6 we present experimental results only for the low-power SDRAM modules. Since such memories have a single low-power operation mode, with an exit latency of only a few cycles [2, 3], it follows that a simple policy that immediately switches the SDRAM to the low-power mode, as soon as a data transfer task concludes execution, and switches it back to active mode, right before a new data transfer task starts executing, performs optimally.

5.2.2 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* shut down policies, as tasks execute on both subsystems. In Figure 4.6 for example, as $Task1\_P^n$ concludes the processing of the n compressed macroblocks

sequentially stored in the MPEG2 *input stream region* (see time interval **A** in Figure 4.6), the corresponding sub-regions where such data is stored are selectively shut down (i.e., *Vdd-gated* [33]). At the end of the execution of $Task1\_P^n$, and during the time interval until $Task1\_DT^n$ starts the prefetching of the next batch of compressed macroblocks (see time interval **B** in Figure 4.6), 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 (i.e., backward and forward)* will be permanently shut down during the execution of $Task1\_P^n$. Experimental results in Section 6.8 show that such low cost task/software driven shut down policies can considerably reduce leakage/static power dissipation on the Streaming Memory.

5.3 Design Space Exploration and Tuning Methodology

Given a streaming media application, the selection of a specific processor core is driven by power/performance targets and other considerations which are beyond the scope of this work. Once the processor core is selected, Xtream-Fit's design space exploration process, aimed at minimizing *energy-delay product* [20] for the memory subsystem, is quite simple and systematic, and can be conducted by simply varying the task granularity parameter g for a number of representative input data sets. As with any non-trivial system, design space exploration is indeed needed, because there are conflicting cost/benefit trends in the memory subsystem architecture. 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 dynamic and leakage power dissipation in on-chip memory, increase as well.

Figure 5.2 shows a sample of the design space exploration results for the MPEG2 decoder application when running the *mobile.mpg* input data set on a MIPS R10000 processing element. Figure 5.2(a) shows the impact of task granularity g on the energy consumed by the on-chip memory (Streaming Memory and Scratch-Pad) and the SDRAM during the decoding of the MPEG2 stream. We plot energy consumed in mJ on the y-axis, and different Xtream-Fit configurations, that is, values for parameter g, on the x-axis. As expected, with increasing granularity, the energy consumed in on-chip memory increases but that on the SDRAM decreases.

For the same input data set, (i.e., *mobile.mpg*), Figure 5.2(b) shows the total energy consumption on the Streaming Memory, Scratch-Pad and SDRAM in mJ versus total decoding delay (in simulation cycles), assuming several task granularities. Figure 5.2(c) shows the resulting energy-delay product for the various task granularities considered during the design space exploration. As illustrated in Figures 5.2(a), (b) and (c), by simply varying parameter g, one can systematically move across the conflicting energy consumption curves, i.e., explore the design space, so as to easily find the point of maximum energy efficiency (i.e., minimum energy-delay product), for a particular target processor/performance – see Figure 5.2(c).

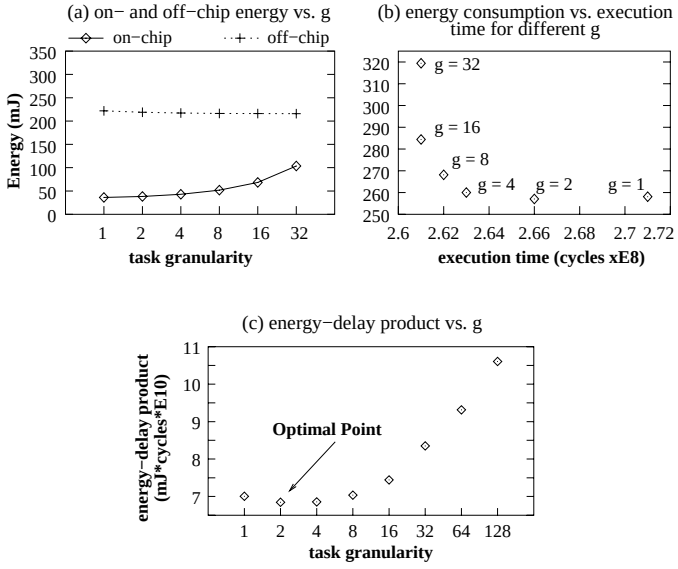


Figure 5.2: Xtream-Fit: Task granularity driven design space exploration for MPEG2 considering an MIPS R10000 core and input data set *mobile.mpg*.

Hence, although there is a significant degree of "specialization" in Xtream-Fit, the process of customizing it to an application is actually quite simple and systematic. Given a target application, it starts with the definition of an Xtream-Fit configuration for task granularity $g = 1$. In order to do so, the application code needs to be first carefully analyzed, in order to identify the set of basic (i.e., $g = 1$) processing and data transfer tasks, and generate the corresponding task graph (see Section 4.1 and Section 4.2). Using the task graph (see e.g., Fig.4(b)), and relying on the simple policies discussed in Section 4.1, the dynamic scheduling of the various tasks can be then established (see Section 4.3). Then, the Streaming Memory regions for the various input, output and intermediate data objects produced/consumed by those tasks are defined and sized (see Section 5.1 and Figure 5.1). Subsequently, the task-

based shut down policies for Streaming Memory regions holding dead data are defined and implemented in Xtream-Fit's Streaming Memory Controller (see Section 5.2.2). The default SDRAM power mode control policy is also trivially implemented in the Streaming Memory Controller (see Section 5.2.1). Finally, the Scratch-Pad memory (aimed at holding scalars and constants) is sized, as discussed in Section 5.1.

Once an Xtream-Fit configuration for task granularity $g = 1$ is defined, configurations for alternative task granularities can be easily derived from it. Namely, for a configuration with granularity $g = n$, each processing task, when invoked, iterates n times; each data transfer task, when invoked, prefetches and/or writes back $k \times n$ basic data objects, exploiting page mode access whenever possible[1]; the SDRAM is switched to its low power mode only after each data transfer task executes n times; etc.

Design space exploration consists of systematically varying the task granularity parameter (g = 1, 2, 4, 8, 16, etc.[2]), and determining the corresponding energy-delay product for each such alternative configuration (for a representative set of input data sets). The design space exploration process ends when an Xtream-Fit configuration with minimum energy-delay product is found. Section 6.1 describes an experimental set up that can be used to de-

[1]Note that, for certain tasks, the set of data objects to be accessed may not be stored contiguously. For example, the $2 \times n$ motion compensation macroblocks prefetched by MPEG2's $Task2\_P^n$ are stored at "arbitrary" locations.

[2]Depending on the application, a maximum granularity may need to be established, based on specifications pertaining jitter and other considerations.

termine the resulting energy-delay product for each alternative configuration and input data set.

A few important comments on the complexity of the design space defined by the Xtream-Fit memory architecture. Since energy consumption in on-chip/off-chip memory increases/decreases with g (see e.g., Figure 5.2(a)), while performance increases with g yet at a much slower rate[3], one should expect to always find a *single* minimum energy-delay product point for any input data set. That is, the complexity of the design space defined by Xtream-Fit is in fact very reduced. Perhaps most importantly, for all media benchmarks and representative input data sets considered in our experiments (see Table 6.1), we found that: (1) for a given application, substantially different input data sets consistently give very similar "optimal" task granularity points; and (2) energy-delay product tends to vary very little in the vicinity of the "optimal" granularity point for a given (*application*, *input data set*) pair.

Consider, for example, Figure 5.3(a), which shows the percentage increase in energy-delay product, for a range of g-values, with respect to the minimum energy-delay product found (through design space exploration) for each of five MPEG2 input data sets running on a MIPS R10000 processor core. As it can be seen, for four of the five input data sets, the "optimal" granularity is $g = 4$, while for the remaining data set the "optimal" granularity is 2. Note further that the percentage increase in energy-delay product result-

[3]This is so due to the 'computation bound' nature of media applications.

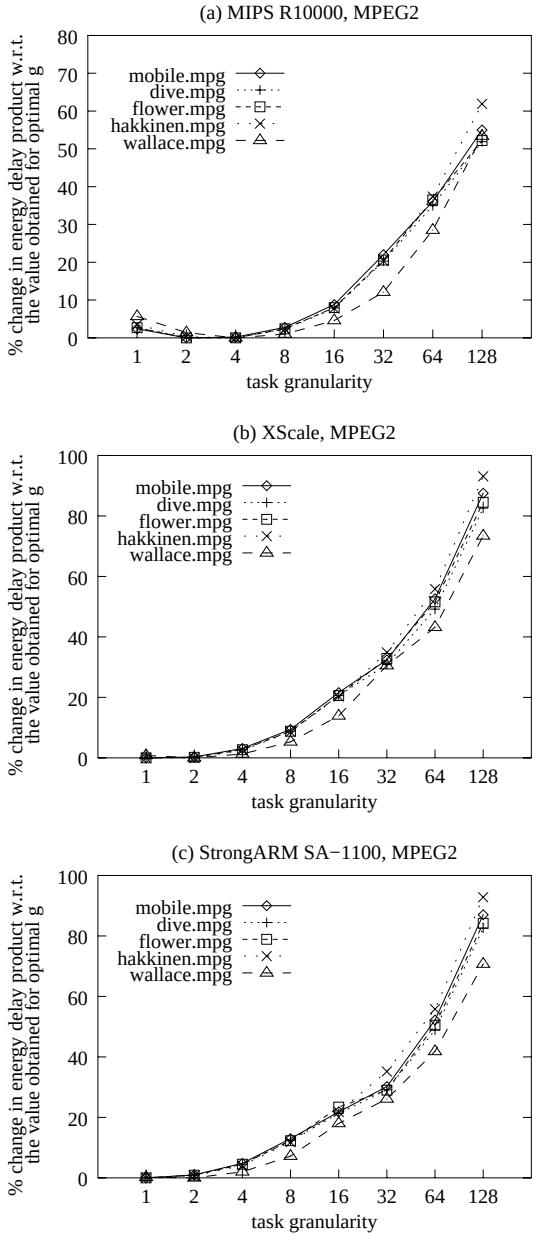


Figure 5.3: Xtream-Fit: Task granularity driven design space exploration for MPEG2 considering multiple input data sets.

ing from using granularity $g = 2$ for the four input data sets with "optimal" granularity 4 is under 2%, while the increase in energy-delay product resulting from using granularity $g = 4$ for the input data set with "optimal" granularity 2 is under 1%. Figures 5.3(b) and (c) show similar trends for configurations using an ARM XScale processor core and a StrongARM SA-1110 processor core, for the same input data sets. Similar results were obtained for the other representative media benchmarks considered in our experiments – see also Section 6.9. Such results are very positive, since they indicate that, even though one may not be able to select a task granularity that is "optimal/ideal" for all possible application workloads, the resulting energy-delay product for an arbitrary input data set will in general be very close to the "minimum" possible to achieve. Additional experimental data is provided in Section 6.9, empirically supporting this claim.

Chapter 6

Experimental Results: Single-Application Media Systems

In this chapter, we present the experimental methodology used to evaluate the effectiveness of the Xtream-Fit data memory subsystem and discuss the results thereof. We first compare the performance and energy consumption of Xtream-Fit to a set of reference systems using "standard" cache configurations. Specifically, we consider a select set of memory configurations that could be easily offered ("off-the-shelf") for each of the processor cores targeted in our experiments, and assess the energy-efficiency of Xtream-Fit relative to those "standard" memory configurations. This first set of experiments thus aims at comparing Xtream-Fit to systems developed with a reduced customization effort. We then performed a much more exhaustive design space exploration on the reference cache configurations (for each processor core and benchmark application) so as to determine the best possible reference system to compare against Xtream-Fit. This second set of experiments was designed to assess the effectiveness of Xtream-Fit relative to aggressively customized cache hierarchies. An important additional observation is that *all* reference cache configurations (for both sets of experiments) were augmented/enhanced with state-of-the-art techniques to reduce power consumption. Namely, cache decay

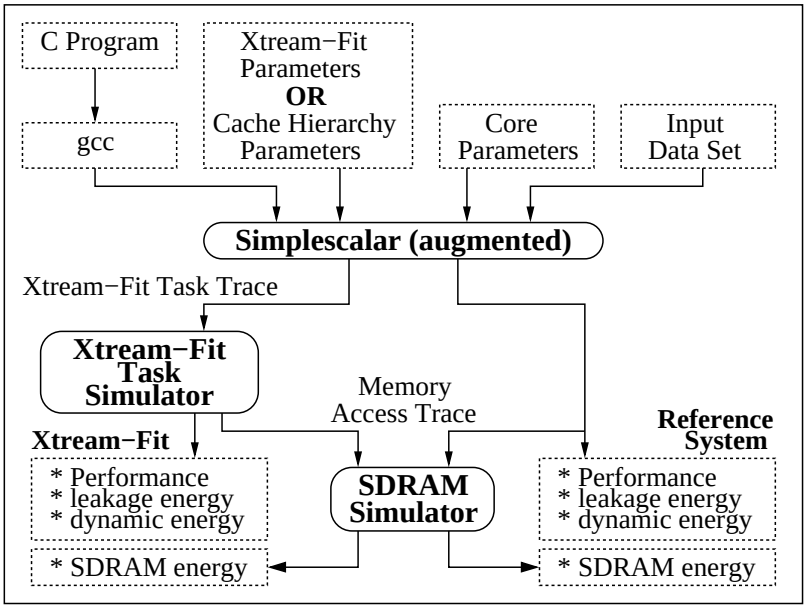


Figure 6.1: Experimental methodology to evaluate Xtream-Fit.

policies were implemented to minimize leakage power in the cache hierarchy of the reference systems and the optimal policy exploiting SDRAM low power modes, discussed in Section 5.2.1, was integrated in both Xtream-Fit and the reference systems to minimize energy consumption in off-chip memory. Although some of these techniques might not be widely available or supported in current processors, we did include them in our reference systems in order to ensure that our experimental results truly reflect the *very best* (in terms of energy efficiency) achievable with "traditional" cache hierarchies.

6.1 Experimental Setup

The experimental framework used to evaluate Xtream-Fit and the reference systems is shown in Figure 6.1. The flowchart in the figure shows the inputs specified for each experiment and the various simulators used to generate the outputs of interest. Specifically, each experiment requires the following inputs:

1. The media benchmark – a C program from the Mediabench suite (see Table 6.1);
2. The input data set for the selected benchmark (see Table 6.1);
3. The set of core parameters – a description of the processor core (see Table 6.2);
4. The memory hierarchy specification – SDRAM parameters, and either Xtream-Fit or cache hierarchy parameters (see details below).

The *Simplescalar simulator* [10] was augmented with cache decay policies and modified to provide the following additional information:

1. Leakage and dynamic energy consumption for on-chip memory blocks;
2. A task trace for the Xtream-Fit system.

For systems with a traditional memory hierarchy, performance was calculated by measuring the total number of simulation cycles taken to process an entire

data set (i.e., from start to completion of the program). To counter the effects due to startup of the program, we used sufficiently large input data sets, e.g., a (512×512) pixel picture for the JPEG encoder.

For the Xtream-Fit system, Simplescalar automatically generates a task trace for the specified task granularity parameter g. The individual tasks are then scheduled by the *Task Simulator* using the policies described in Section 4.3. As argued before, the synchronization time between different data transfer tasks and between data transfer and processing tasks is essentially negligible. However, to stress the Xtream-Fit system to the fullest, we used a very conservative synchronization time of 100 clock cycles.

6.2 Energy Estimation

The leakage energy in on-chip memories for both systems was calculated based on the low-V_t data given in Table 2 of [64]. The data was extrapolated to the sizes of different memories used in the experiments. The dynamic energy for both systems was calculated using CACTI [53] as described in Chapter 2.

To measure SDRAM energy, a detailed model of a *mobile SDRAM*, from *Micron Technologies* [2], specifically, part number MT48V2M32LFC-8 @ 2.5V, was built and incorporated in the SDRAM simulator shown in Figure 6.1. We chose Micron's mobile SDRAM as it represents one of industry's best low-power SDRAMs and is widely used in media devices. It has a number of power saving features, including low operating voltage, low operating currents and temperature controlled self-refresh.

Table 6.1: Benchmarks and input data sets used in our experiments.

Benchmark	Description	Input Data Set	Description
MPEG2	Video compression decoder	P:mobile.mpg	60 frames video
		dive.mpg	85 frame video
		flower.mpg	150 frame video
		hakkinen.mpg	200 frame video
		wallace.mpg	130 frame video
JPEG	Lossy image compression encoder	P:lena.ppm	512x512 image
		bike.ppm	768x512 image
		circles.ppm	256x256 image
		slope.ppm	256x256 image
		text.ppm	512x512 image
G721	Voice compression encoder	P:clinton.s.pcm	32KB raw data
		clap.pcm	268KB raw data
		keillor.pcm	36KB raw data
		piano.pcm	184KB raw data
		sitar.pcm	156KB raw data

6.3 Benchmarks and Input Data Sets

Since we are targeting streaming media applications, we used benchmarks from the Mediabench Suite [38]. We selected one representative application from the video, image, and audio compression domains. The benchmarks were compiled with *gcc* with optimizations turned on ($-O2$). A brief description of the applications and their input data sets is given in Table 6.1. In order to evaluate Xtream-Fit's energy-delay efficiency under different workload conditions, for each benchmark we selected five input data sets exhibiting significantly different characteristics. For the MPEG2 benchmark, we chose video of CIF and QCIF resolutions, with fast moving sports scenes (*hakkinen.mpg*, *dive.mpg*), and relatively slow moving scenes with low (*flower.mpg*)

Table 6.2: Simplescalar parameters for core configurations.

Parameters	StrongARM SA-1110	XScale	MIPS R10000
Fetch/Retire Rate	2	2	4
RUU Size	8	16	64
LSQ Size	8	8	32
Out-Of-Order	No	Yes	Yes
Memory	4B/cycle	8B/cycle	64B/cycle
Memory Latency	32, 1	32, 1	64, 2
Functional Units	2 ALU, 1 Mult	2 ALU, 1 Mult	4 ALU, 2 Mult

and high (*mobile.mpg*) change in background scenery. For JPEG, we selected images with different sizes and different content including a classical image from the image processing community (*lena.ppm*), an image containing text only (*text.ppm*), an image with substantial amount of color and background detail (*bike.ppm*) and black and white images with and without sharp boundaries (*circles.ppm* and *slope.ppm*, respectively). Finally, for G721, our input data set includes the voice of a man speaking (*clinton.s.pcm* and *keillor.pcm*), sound of an audience applause (*clap.pcm*) and instrumental music (*piano.pcm* and *sitar.pcm*).

6.4 Machine Models

We modeled three processor cores, specifically, two processors from the ARM family, *Intel's StrongARM* and *Intel's XScale*, and a *MIPS R10000*. The ARM cores were used to evaluate Xtream-Fit in the context of low-power, less performance demanding embedded systems. The MIPS core was used to assess its performance in the context of more performance demanding systems

Table 6.3: "Optimal" Xtream-Fit task granularity for primary input data sets.

Benchmark	Family of Configurations	"Optimal" Task Granularity	Xtream-Fit Configuration
MPEG2	SA-1110 (X1-g)	$g = 1$	$X1 - 1$
	XScale (X2-g)	$g = 1$	$X2 - 1$
	MIPS R10000 (X3-g)	$g = 2$	$X3 - 2$
JPEG	SA-1110 (X1-g)	$g = 2$	$X1 - 2$
	XScale (X2-g)	$g = 4$	$X2 - 4$
	MIPS R10000 (X3-g)	$g = 4$	$X3 - 4$
G721	SA-1110 (X1-g)	$g = 64$	$X1 - 64$
	XScale (X2-g)	$g = 64$	$X2 - 64$
	MIPS R10000 (X3-g)	$g = 128$	$X3 - 128$

requiring faster, more powerful processors. Details on the processor configurations used in our experiments are given in Table 6.2. Some of the parameters of those machines, including issue_width, and ruu_size have been aggressively dimensioned in order to stress the bandwidth requirements of the memory subsystem. We would like to have also considered DSPs, e.g., Starcore, TMS320 families, etc., yet our experiments require the availability of an *open-source* cycle-by-cycle simulator, such as Simplescalar [10] and, unfortunately, such simulators are not available for commercial DSPs. Still, the range of machines considered in our experiments provide a solid basis for assessing basic trends on the relative energy-efficiency of Xtream-Fit across a large range of sustainable IPC (instructions-per-cycle) levels, and thus memory bandwidth requirements. In fact, similar evaluation methodologies have been frequently used in literature, to overcome the lack of *open-source* simulators alluded to above, see e.g., [35, 37].

Table 6.4: Sizes of Streaming Memory and Scratch-Pad in Xtream-Fit.

	MPEG2	**JPEG**	**G721**
Scratch-Pad	2048 B	2048 B	64 B
Streaming Memory	2048 $B \times g$	256 $B \times g$	$2B \times g$

6.5 Design Space Exploration: Xtream-Fit

As discussed in Section 5.3, given a processor core and a target media application, Xtream-Fit's design space exploration process is quite simple and systematic – it can be conducted by simply varying *task granularity parameter* g until the point of minimum energy-delay product is found. Such a point (i.e., value of g), defines the "optimal" Xtream-Fit configuration for the particular processor and application.

Accordingly, embedded systems implemented with "optimal" Xtream-Fit configurations are represented by the corresponding processor core (StrongARM → X1, XScale → X2, and MIPS R10000 → X3) and the "optimal" task granularity is given by the appended integer, representing parameter g. Table 6.3 gives the "optimal" task granularity experimentally determined for the set of media benchmarks and processors considered in our study (assuming a select set of input data sets, see discussion below). For example, the "optimal" task granularity for the MPEG2 benchmark, for input data set *mobile.mpg*, running on the MIPS R10000 core (X3 family) was found to be $g = 2$, and therefore the corresponding "optimal" Xtream-Fit configuration is denoted $X3 - 2$.

The sizes of Xtream-Fit's Streaming Memory and Scratch-Pad Memory

for the applications under study are given in Table 6.4. Naturally, when one varies g, the size of Xtream-Fit's Streaming Memory varies accordingly – as indicated in the Table. The size of the Scratch-Pad memory, however, is fixed for each application and was determined by a static analysis of the program.

As discussed in Section 5.3, substantially different input data sets consistently gave very similar "optimal" configuration points for the three representative media benchmarks considered in our experiments. Moreover, energy-delay product tended to vary very little in the vicinity of the "optimal" granularity point for a given (*application, input data set*) pair. Thus, in order to be able to present actual performance, energy and energy-delay product *numbers*, and analyze concrete trends on such numbers (see Section 6.7), the "optimal" task granularities presented in Table 6.3, and the detailed results presented in Figures 6.2, 6.3, 6.4 and 6.5, 6.6, 6.7 were determined considering a single input data set for each application, namely, *mobile.mpg* for MPEG2, *lena.ppm* for JPEG and *clinton.s.pcm* for G721. We denote these input data sets as the *primary* input data sets (indicated with a **P** in Table 6.1). For completeness, in Section 6.9, we present the overall percentage increase in energy-delay product for each media benchmark resulting from running all input data sets considered for the particular benchmark with the "optimal" granularity determined for the corresponding primary input data set, and discuss simple techniques to incorporate multiple input data sets in the design space exploration process, if deemed necessary.

6.6 Design Space Exploration: Reference Systems

Our experiments are aimed at determining Xtream-Fit's relative improvement in energy-delay product with respect to properly configured reference data memory subsystems, as well as analyzing percentage savings achieved by Xtream-Fit on static and dynamic energy consumption with respect to such baselines. Thus, similarly to Xtream-Fit, the design space exploration performed for the reference memory subsystems was directed towards minimizing energy-delay product for the primary input data sets. Accordingly, the detailed results presented in Figures 6.2, 6.3, 6.4 and 6.5, 6.6, 6.7 were determined considering those input data sets.

The overall design space for the reference data memory systems is defined by the following eight parameters: L1 D-Cache *size*, L1 D-cache *block size*, L1 D-Cache *associativity*, L1 D-Cache *kill-window size*, L2 D-Cache *size*, L2 D-cache *block size*, L2 D-Cache *associativity* and L2 D-Cache *kill-window size*.

Case 1: "Standard" Cache Hierarchies. In this set of experiments we consider a limited set of cache configurations approximating those offered for the actual processor cores (see Tables 6.5 and 6.6). Thus, in this set of experiments the eight parameters listed above are only "partially" explored, yet kill-window sizes were carefully tuned for each individual configuration (see tuning ranges in Table 6.6) so as to minimize energy-delay product.

Case 2: "Best" Cache Hierarchies. The objective of this second

set of experiments was to determine the "best/optimal" cache hierarchy configuration for each processor core and media benchmark through an exhaustive search over the entire design space, that is, to determine the set of values for the *eight* cache hierarchy parameters listed above leading to a minimum energy-delay product. The parameter ranges considered in our exploration are given in Tables 6.7 and 6.8 respectively.

Unfortunately, due to the "conflicting" nature of most such design parameters [23, 62], the actual design space for each processor-benchmark pair

Table 6.5: Experimental validation of Xtream-Fit with respect to "standard" memory configurations (**Case 1**).

Core/Processor	**Reference Systems**			**Xtream-Fit**
	Label	L1-Cache	L2-Cache	Label
StrongARM SA-1110	$R1a$	4 KB	None	$X1-g$
	$R1b$	16 KB	None	
	$R1c$	32 KB	None	
XScale	$R2a$	4 KB	None	$X2-g$
	$R2b$	16 KB	None	
	$R2c$	32 KB	None	
XScale	$R2a^*$	4 KB	16 KB	
	$R2b^*$	16 KB	64 KB	
	$R2c^*$	32 KB	128 KB	
MIPS R10000	$R3a$	4 KB	16 KB	$X3-g$
	$R3b$	16 KB	64 KB	
	$R3c$	32 KB	128 KB	

Table 6.6: Cache configuration for "standard" reference systems (**Case 1**).

Cache	**Block Size**	**Associativity**	**Kill Windows**
L1-Cache	32B	4-way	4000 - 1024000
L2-Cache	64B	4-way	64000 - 1024000

was found to be consistently very complex. In order to ensure that we were indeed identifying the minimum energy-delay product configuration for each case, we had to exhaustively search the entire design space, i.e. perform up to 6183 detailed simulations per *processor-benchmark* pair. Table 6.9 lists out the results of our exhaustive design space exploration, i.e., the best performing cache hierarchy parameters for each processor-benchmark pair – notice the significant variation in parameter values across such pairs.

6.7 Overview of Results

Case 1: Xtream-Fit Compared to "Standard" Cache Hierarchies.

Figures 6.2, 6.3 and 6.4 summarize the results of our first set of ex-

Table 6.7: Experimental validation of Xtream-Fit with respect to "best" memory configurations (**Case 2**).

Core/Processor	**Reference Systems**			**Xtream-Fit**
	Label	L1-Cache	L2-Cache	Label
SA-1110	$R1 - opt$	4/16/32 KB	None	$X1 - g$
XScale	$R2 - opt$	4/16/32 KB	None	$X2 - g$
XScale	$R2 * - opt$	4/16/32 KB	4 times L1	$X2 - g$
MIPS R10000	$R3 - opt$	4/16/32 KB	4 times L1	$X3 - g$

Table 6.8: Design space exploration parameters for reference systems with "best" cache configuration (**Case 2**).

Cache	**Block Size**	**Associativity**	**Kill Windows**
L1-Cache	32B	1-way/2-way/4-way	1000 - 1024000
L2-Cache	64B/128B/256B	1-way/2-way/4-way	4000 - 1024000

Table 6.9: "Best" cache configurations for Case 2.

Core	L1 Cache				L2 Cache				energy*delay
	size	line	asso	k-win	size	line	asso	k-win	
MPEG2									
R1	32 KB	32	2	256000	-	-	-	-	7.21e+11
R2	32 KB	32	2	64000	-	-	-	-	4.21e+11
R2*	16 KB	32	2	4000	64 KB	64	4	256000	4.79e+11
R3	16 KB	32	1	4000	64 KB	64	4	256000	1.49e+11
JPEG									
R1	32 KB	32	2	64000	-	-	-	-	2.35e+09
R2	32 KB	32	2	1024000	-	-	-	-	1.27e+09
R2*	16 KB	32	1	4000	64 KB	64	2	1024000	1.23e+09
R3	32 KB	32	1	4000	128 KB	128	4	1024000	6.63e+08
G721									
R1	16 KB	32	1	64000	-	-	-	-	1.20e+09
R2	16 KB	32	1	16000	-	-	-	-	6.10e+08
R2*	4 KB	32	1	16000	16 KB	64	2	256000	5.34e+08
R3	16 KB	32	1	16000	64 KB	64	4	64000	1.52e+08

periments, contrasting Xtream-Fit's energy efficiency to that of a select set of "standard" cache hierarchies. Consider first the performance results for the the MPEG2 decoder application, shown in Figure 6.2(a). The four graphs in the figure plot decoding delays in processor cycles for primary input stream *mobile.mpg* grouped by processor core. Note that the number of cycles required to process the input data set varies by about one order of magnitude when one moves from the slowest core (StrongARM SA-1100) to the fastest core (MIPS R10000) – performing experiments exhibiting such wide performance variation is very critical, since it allows us to evaluate Xtream-Fit energy-delay efficiency under very different working conditions and bandwidth requirements. For each

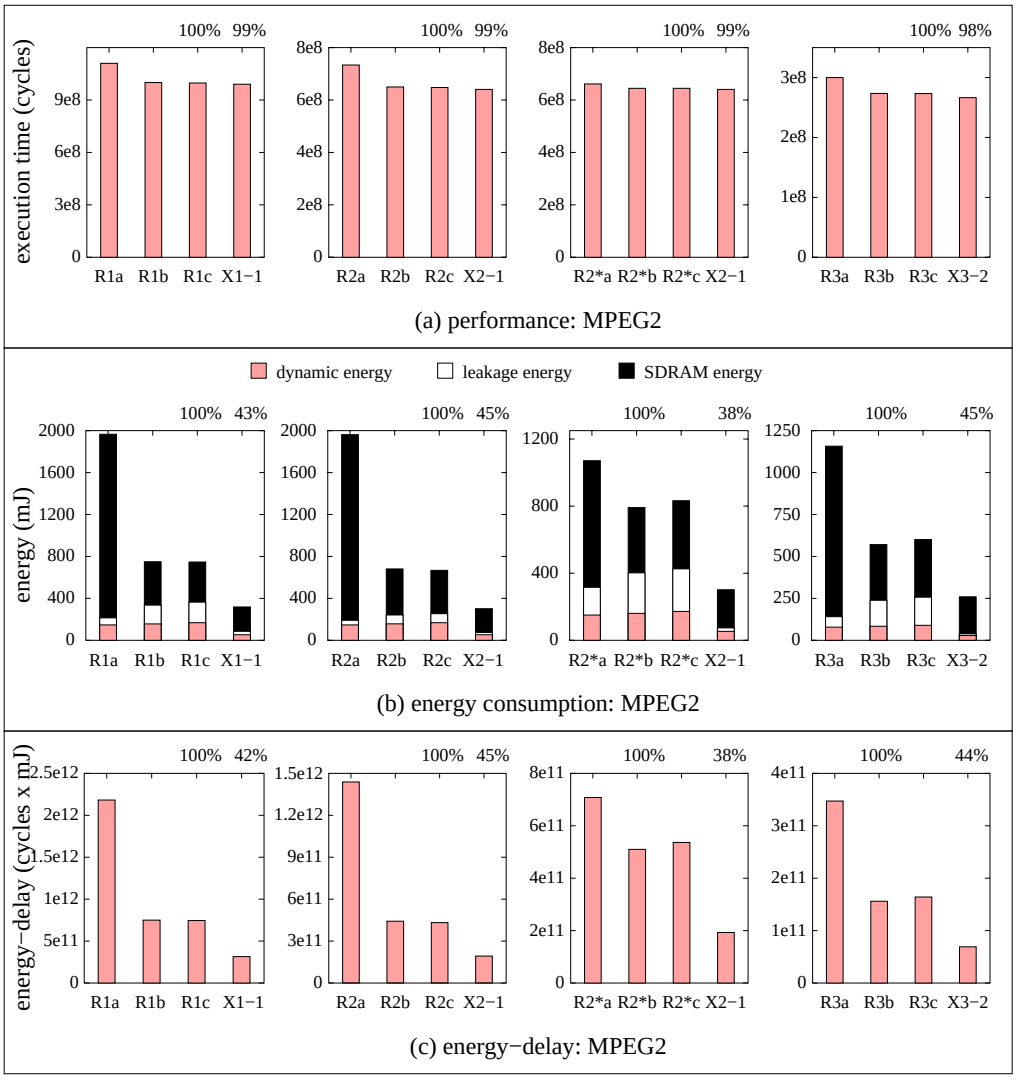


Figure 6.2: Comparative evaluation: "standard" cache hierarchy: MPEG2.

core, we normalize the performance of Xtream-Fit to the best reference configuration. For example, in the left-most bar-chart in Figure 6.2(a) that contrasts reference configurations R1a, R1b, and R1c to Xtream-Fit's X1-g (i.e., considers the StrongARM SA-1110 core), the best performing granularity was $g = 1$

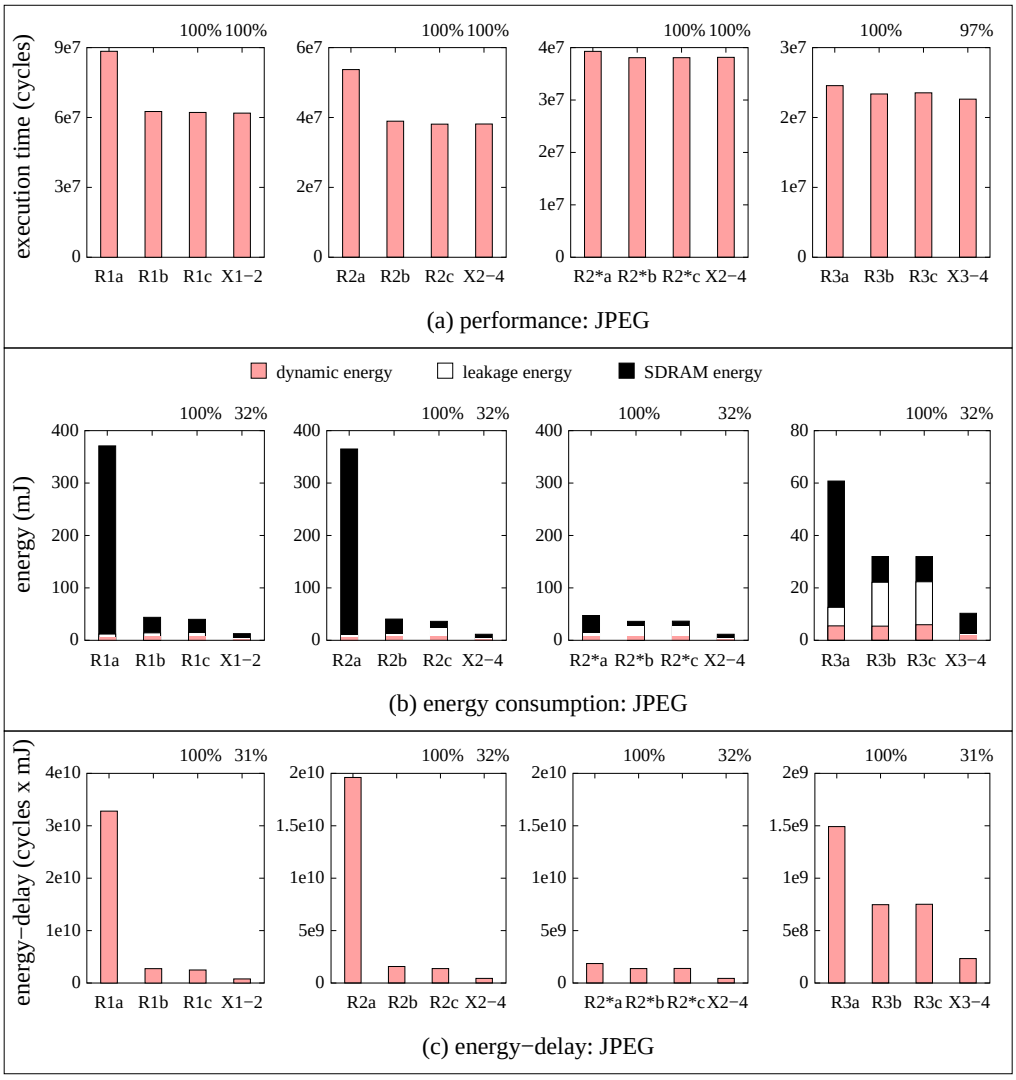


Figure 6.3: Comparative evaluation: "standard" cache hierarchy: JPEG.

(see Table 6.3 and Figure 5.2(c)), and the best performing reference was R1c (see Table 6.5). 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.

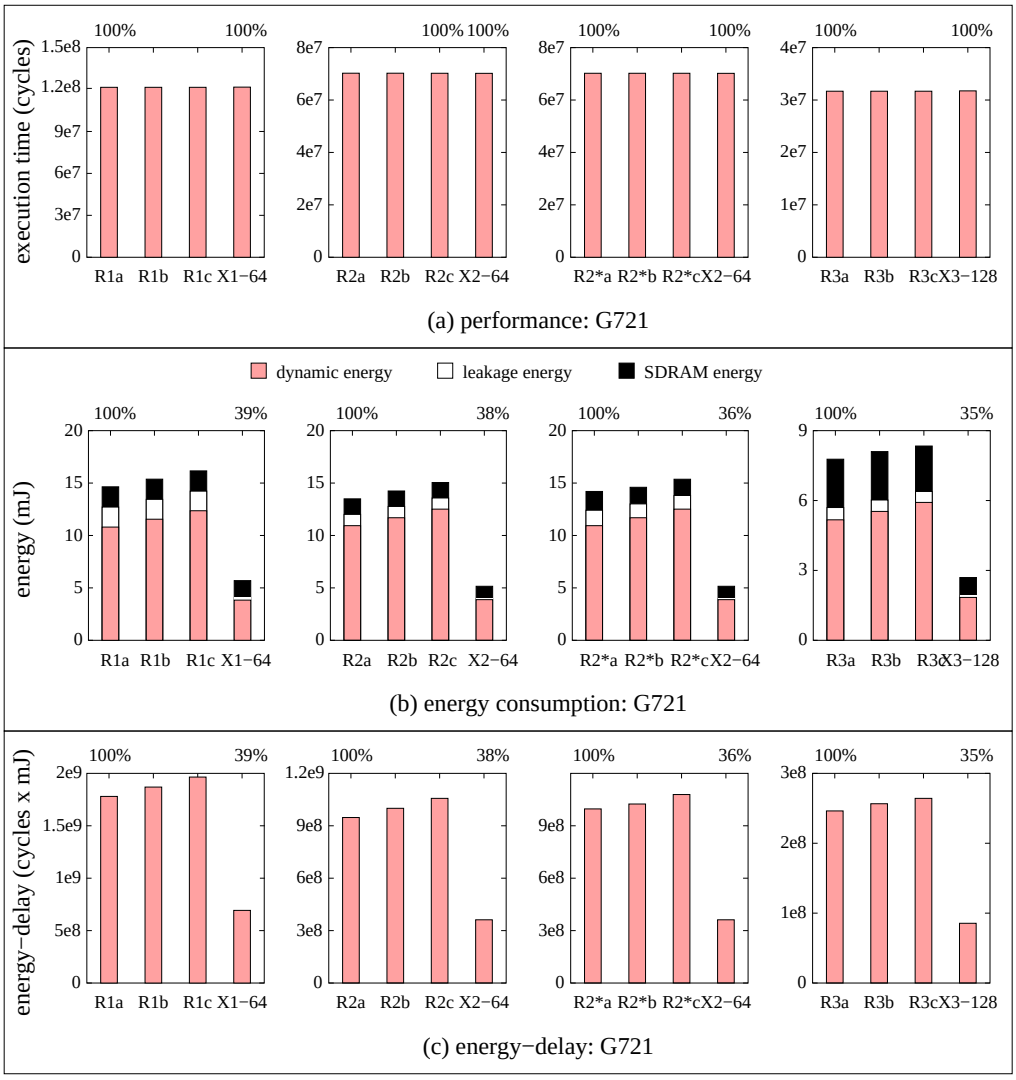


Figure 6.4: Comparative evaluation: "standard" cache hierarchy: G721.

For the same set of experiments on the MPEG2 benchmark, Figure 6.2(b) plots total energy consumption in mJ 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,

for MPEG2, Xtream-Fit consistently outperforms the best reference configuration, with decreases in total energy consumption ranging from 55% to 62%.

Finally, Figure 6.2(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 55% and 62%, when compared to the best performing reference configuration for any particular core.

Similarly, Figures 6.3(a)-(c) and 6.4(a)-(c) plot performance, energy consumption and energy-delay product for JPEG and G721 respectively, showing energy-delay product improvements ranging from 61% to 69%.

A brief note on the relevance of the energy savings reported above. Using the PowerAnalyzer tool available for the ARM family of processors [4], we verified that, for example, for the XScale default configuration (with a 32 KB, 4-way L1 on-chip cache), the energy consumption in the on-chip data memory subsystem was consistently 26% to 30% of the total energy consumed by the processor (excluding I/O pad drivers) when running the set of media benchmarks considered in our experiments, i.e., MPEG2 decoder, JPEG encoder and G721 encoder. Even though such percentages are already quite substantial, we should further note that the PowerAnalyzer tool does not consider off-chip memories. Thus, the actual percentage of energy consumed by the data memory subsystem (including also off-chip memories) should be substantially higher than the values reported above, attesting to the significance of the energy savings achieved by Xtream-Fit.

Case 2: Xtream-Fit Compared to "Best" Cache Hierarchies.

Figures 6.5, 6.6 and 6.7 summarize the results of our second set of experiments, contrasting Xtream-Fit's energy efficiency with that of "best" cache configurations. The results are organized in a manner similar to that used for the "standard" cache configurations, except that the reference data point presented for each processor-benchmark pair is the best (i.e., minimum energy-delay product) cache hierarchy configuration found during the corresponding design space exploration (i.e., out of the 6183 explored/simulated configurations for the particular processor-benchmark pair). The corresponding Xtream-Fit design point for each processor-benchmark pair is obviously identical to that used in Figures 6.2, 6.3 and 6.4.

Overall Assessment of Results. A quick comparison between Xtream-Fit's improvement with respect to "standard" reference systems versus Xtream-Fit's improvement with respect to "best" reference systems, summarized in Figures 6.5, 6.6 and 6.7 and Figures 6.2, 6.3 and 6.4, reveals the following:

1. In spite of our exhaustive design space exploration, the performance of the "best" reference systems was found to be very similar to that of the standard reference systems. The performance of the Xtream-Fit based systems was found to be marginally better than that of the reference systems.

2. When compared to the standard reference systems, energy consumption in the "best" reference systems decreased significantly for all processor

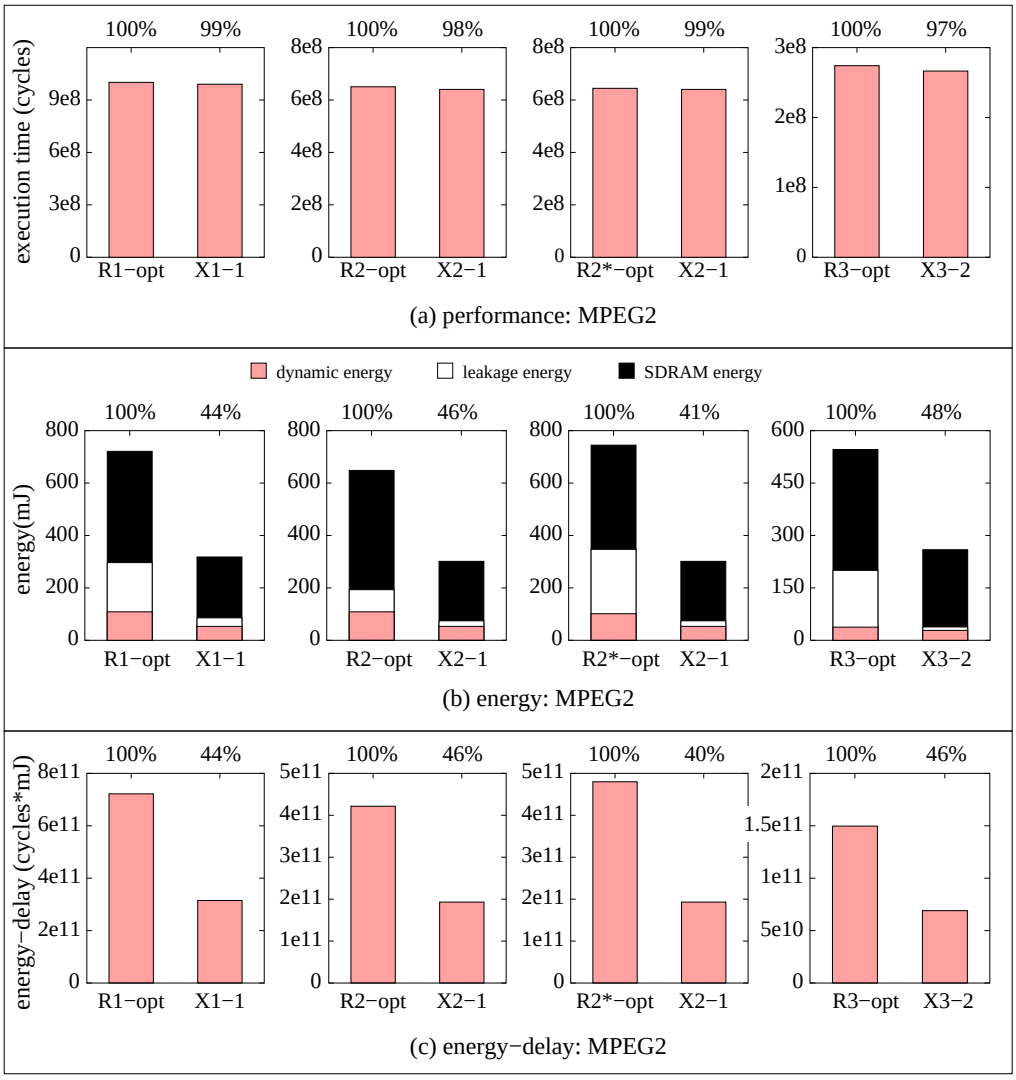


Figure 6.5: Comparative evaluation: "Best" cache hierarchy: MPEG2.

cores for the G721 benchmark, while it was only marginally better for MPEG2 and JPEG.

3. When compared to the "best" reference systems, Xtream-Fit still delivers a significant improvement in energy-delay product, ranging from 32% to

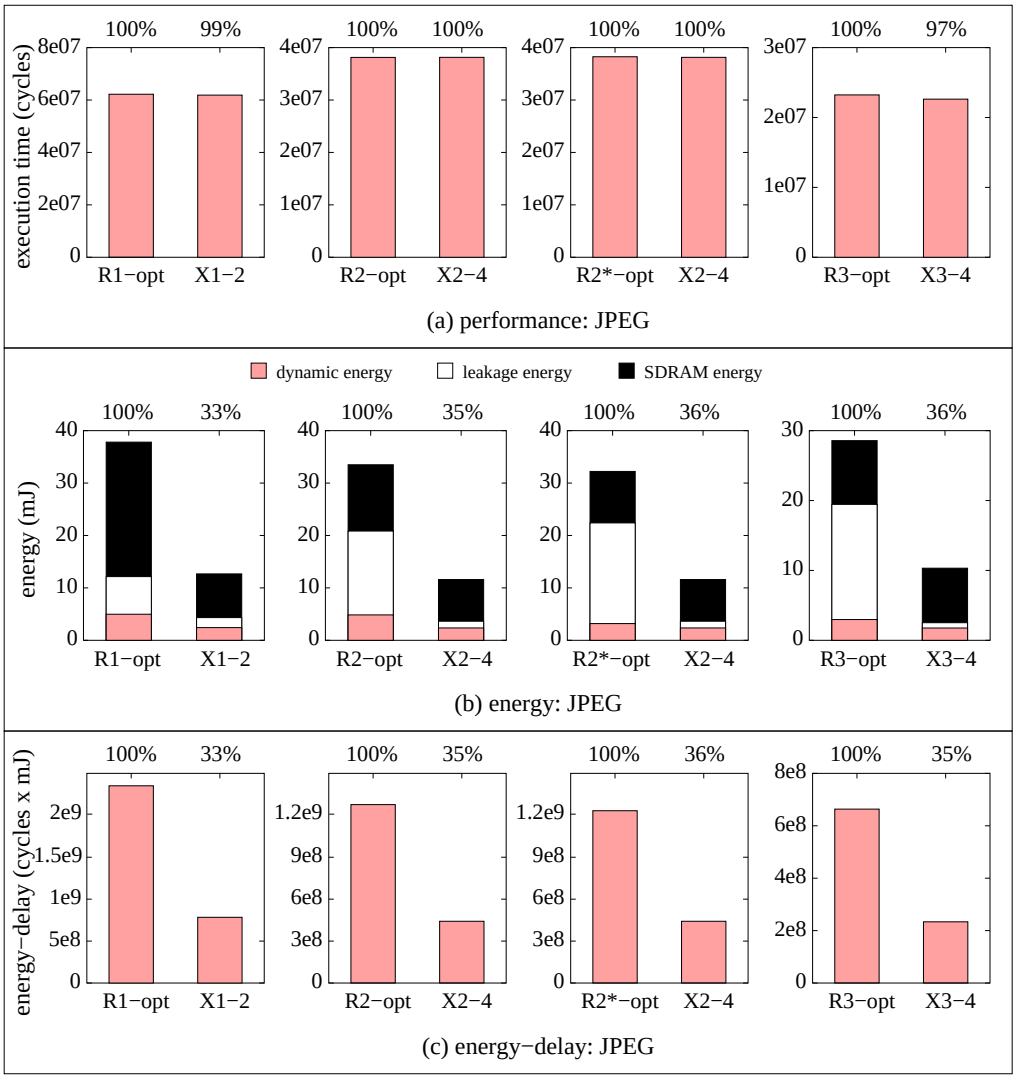


Figure 6.6: Comparative evaluation: "Best" cache hierarchy: JPEG.

69%.

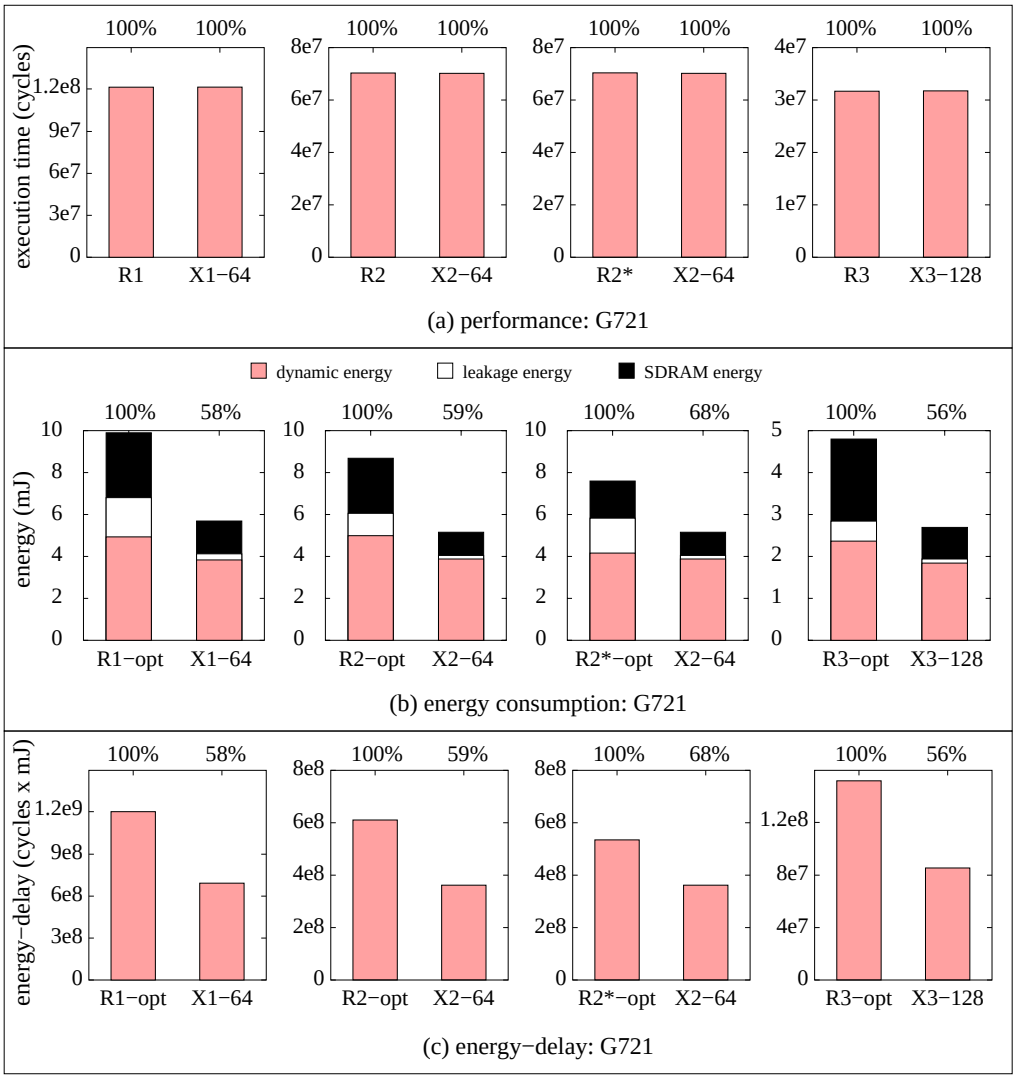


Figure 6.7: Comparative evaluation: "Best" cache hierarchy: G721.

6.8 Detailed Analysis of Results

Consistent with studies on cache performance for multimedia applications [56], we found that traditional cache hierarchies perform quite well. In fact, with sufficiently large cache sizes and an adequate degree of associativity,

the performance of the reference systems came close to that of a hypothetical perfect memory system. In spite of that, we consistently found that the execution time of media systems with Xtream-Fit was marginally better than that of the reference systems. Note that, Xtream-Fit's scheduling policy requires processing tasks to wait until data required by the first processing task has been prefetched into memory by the first data transfer task. Yet, the delay dominance of processing tasks over data transfer tasks, and Xtream-Fit's ability to take advantage of burst/page mode access, more than offsets the "on-demand access" features of modern out-of-order processors.

However, the key improvement achieved by Xtream-Fit is on energy-delay efficiency, i.e., on aggressively decreasing the energy cost of supporting the required memory bandwidth. As shown in Figures 6.2(b), 6.3(b), 6.4(b) and Figures 6.5(b), 6.6(b), 6.7(b) Xtream-Fit configurations consume only a fraction (32% to 68%) of the energy consumed by the best performing reference configurations. A comparative analysis of Xtream-Fit and the reference systems for the three different energy components is provided below.

Off-chip SDRAM Energy. Xtream-Fit's savings in SDRAM energy consumption are very significant across both sets of experiments (14% to 68% savings). 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.

On-chip Leakage Energy. Even though the cache decay scheme used in the reference systems works extremely well in "low reuse" media applications, throughout our experiments we found that Xtream-Fit's on-chip memories consistently consumed significantly less leakage energy (52% to 95% improvement). This is due to two factors. First, at any given point in time, Xtream-Fit's Streaming memory stores just enough data required by the "next" granularity g tasks. Secondly, the organization of the Streaming Memory into data-object based regions allows us to efficiently/selectively shut down these regions as the data they hold becomes "dead."

On-chip Dynamic Energy. The percentage of dynamic energy savings enabled by Xtream-Fit with respect to the reference configurations varies across various processor-benchmark pairs, from marginal (7%) to very significant (67%). It should be noted that such savings 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. Indeed, most of Xtream-Fit's dynamic energy savings are 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).

So, when such power hungry features are "truly" needed, that is, when they lead to reference systems with *minimum energy-delay product* (see associativity parameters for "best" reference cache configurations listed in Table 6.9), then Xtream-Fit's relative dynamic energy savings will be correspondingly more substantial. This last point illustrates once again the complex energy-delay trade-offs involved in the selection of the eight ("best") cache hierarchy parameters for the reference systems, in contrast to the simplicity of the design space exploration required by Xtream-Fit, and the substantial improvements in energy-delay product *consistently* delivered by the resulting Xtream-Fit based media systems.

6.9 Analysis over Multiple Input Data Sets

As indicated previously, the experimental data presented in Sections 6.7 and 6.8 was derived considering a single input data set for each media benchmark. Such "primary" input data sets were arbitrarily selected among the five data sets considered for each benchmark, each exhibiting very different characteristics (see Section 6.3). In this section, we assess the impact of an application's workload on the "optimal" task granularity parameter g. We do so by empirically evaluating the 'sensitivity' of Xtream-Fit's energy-delay efficiency with respect to 'small' variations on parameter g around the "optimal" value for a given workload.

Table 6.10 provides experimental results for the five MPEG2 input data sets. The table is organized into three columns, each showing the data for a

particular processor core. In each column, the first sub-column, titled g_{opt}, lists out the "optimal" granularity obtained for that input data set. The adjoining sub-column labeled $\%(1 - EDP\_Pr/EDP\_Op)$, gives the percentage increase in energy-delay product resulting from running a given input data set using the "optimal" granularity derived for the primary input data set (which is highlighted and labeled **P**) rather than the "optimal" granularity derived for that particular data set. Obviously, for the primary input data set (first line), the energy-delay product values will be identical, since both correspond to running the primary input set with its "optimal" g value – i.e., this entry is always 0%. For the subtable corresponding to the StrongARM SA-1100, for example, only one input data set (*wallace.mpg*) has a corresponding "optimal" granularity different from that derived for the primary data set. Note also that energy-delay product is insignificantly impacted (increase of 0.21%) for that input data set, indicating that the granularity derived for the primary input data set would essentially deliver "minimal" energy-delay product also for *wallace.mpg*.[1] The results reported on the remaining two subtables of Table 6.10 (pertaining MPEG2), as well as on Tables 6.11 (JPEG) and 6.12 (G721) exhibit very similar trends, providing strong empirical evidence that the 'sensitivity' of Xtream-Fit's energy-delay efficiency with respect to 'small' variations on the g parameter around the "optimal" value for a given workload is indeed likely to be very small for our three media benchmarks.

[1] In fact, the 0.21% percent increase is so small that it falls within the margin of error of our estimation models, and thus has no actual significance.

Table 6.10: Assessing the impact of workload variations on Xtream-Fit's energy-delay product improvement – MPEG2.

	StrongARM SA-1100		**XScale**		**MIPS R10000**	
	g_{opt}	$1-\frac{EDP\_Pr}{EDP\_Op}$ %	g_{opt}	$1-\frac{EDP\_Pr}{EDP\_Op}$ %	g_{opt}	$1-\frac{EDP\_Pr}{EDP\_Op}$ %
P:mobile	1	0%	1	0%	2	0%
dive	1	0%	1	0%	4	0.07%
flower	1	0%	1	0%	4	0.04%
hakkinen	1	0%	1	0%	4	0.30%
wallace	2	0.21%	2	0.80%	4	1.34%

Table 6.11: Assessing the impact of workload variations on Xtream-Fit's energy-delay product improvement – JPEG.

	StrongARM SA-1100		**XScale**		**MIPS R10000**	
	g_{opt}	$1-\frac{EDP\_Pr}{EDP\_Op}$ %	g_{opt}	$1-\frac{EDP\_Pr}{EDP\_Op}$ %	g_{opt}	$1-\frac{EDP\_Pr}{EDP\_Op}$ %
P:lena	2	0%	4	0%	4	0%
bike	2	0%	4	0%	4	0%
circles	2	0%	4	0%	4	0%
slope	2	0%	4	0%	4	0%
test	2	0%	4	0%	4	0%

Table 6.12: Assessing the impact of workload variations on Xtream-Fit's energy-delay product improvement – G721.

	StrongARM SA-1100		**XScale**		**MIPS R10000**	
	g_{opt}	$1-\frac{EDP\_Pr}{EDP\_Op}$ %	g_{opt}	$1-\frac{EDP\_Pr}{EDP\_Op}$ %	g_{opt}	$1-\frac{EDP\_Pr}{EDP\_Op}$ %
P:clinton.s	64	0%	64	0%	128	0%
clap	32	0.14%	64	0%	128	0%
keillor	32	0.14%	64	0%	128	0%
piano	32	0.16%	64	0%	128	0%
sitar	32	0.14%	64	0%	128	0%

Thus, the results reported above strongly suggest that, at least for the three representative media applications considered in this work, one could simply select a single input data set among the ones representative for the application of interest, and determine the "optimal" g for such a data set, using the proposed design space exploration methodology. Yet, if a designer would want to be sure that the selected granularity is indeed the one most likely

to maximize the overall energy-delay efficiency achieved by Xtream-Fit over varying workloads, a design space exploration considering several input data sets can obviously be performed. Indeed, we conducted such an exploration, considering five input data sets per benchmark, and obtaining for each application and processor, *at most* two alternative granularities for Xtream-Fit (see Tables 6.10, 6.11 and 6.12). It would thus be relatively simple to consider such a small set of competing g values, together with some application dependent measure of likelihood for the various representative input data sets, in order to determine the actual granularity most likely to maximize Xtream-Fit's energy-delay efficiency for such representative working conditions. Although this effort would be somewhat futile for the case studies considered in this work since the decrease in energy-delay efficiency for the alternative granularity is essentially negligible, it is conceivable that for some media applications, such an exploration across multiple data sets may indeed be required.

Chapter 7

Supporting Multi-Application Media Systems

In this chapter we consider the general case of media devices executing multiple applications under *synchronization* and possibly *throughput* constraints. We extend our task-based model in order to represent two types of concurrency relevant to our target media applications – interleaving and parallel execution of applications. Accordingly, we consider two corresponding target platforms, one with Simultaneous Multithreading machines (SMT), for the parallel execution case (see Figure 7.1(a)), and another with conventional superscalar machines (see Figure 7.1(b)), for the interleaving case. We incorporate the notion of synchronization constraints in our task-based execution model and show that such constraints can be easily supported in the proposed memory subsystem.

7.1 Memory Organization

For this more general, multiple applications case, the proposed data memory subsystem architecture remains essentially the same, except that, (1) the Streaming Memory is now partitioned among the various concurrent applications, and each partition is organized into regions, as in the single application

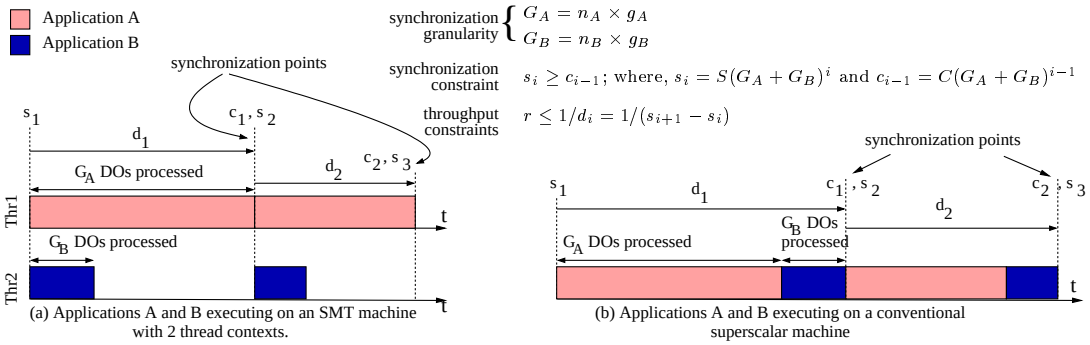


Figure 7.1: Executing two media applications under synchronization constraints: $(G_A + G_B)$ basic data objects (DOs), from applications **A** and **B**, respectively, must be jointly processed.

case, and, (2) the Scratch-Pad is also partitioned across applications. The size of each Streaming Memory and Scratch-Pad partition is derived similarly to the single application case, since each can be seen as a logically independent Streaming Memory or Scratch-Pad. Note that, for the conventional superscalar processor based multi-application platform, one could consider having a single Streaming Memory and Scratch-Pad shared across all applications. In our experiments, we observed that the relatively small sizes of the Scratch-Pad and Streaming Memory required by our characteristic media applications did not justify such an overhead of repeatedly storing and retrieving data for every context switch.

7.2 Task-based Processing Model with Synchronization Constraints

We now define synchronization constraints in the context of Xtream-Fit's task-based execution model. Assume that two streaming media applications, say, applications **A** and **B** in Figure 7.1, are to be executed simultaneously on a media device. We start by introducing the notion of *synchronization granularity* which, in simple terms, defines the number of basic data objects from each application that must be processed "concurrently." Assume, for example, that the specified synchronization granularity for applications **A** and **B** is G_A and G_B, respectively. Using these values, one may now define a corresponding synchronization constraint (see Figure 7.1),

$$s_i \geq c_{i-1}$$

where:
$s_i = S(G_A + G_B)^i$, is the time at which block *"i"* of basic data objects from applications **A** and **B** starts being processed, and
$c_{i-1} = C(G_A + G_B)^{i-1}$, is the completion time of block *"i-1"* of basic data objects from applications **A** and **B**.

Throughput constraints may also be defined, e.g., by placing an upper bound on the time elapsed between the starting times of two consecutive synchronization blocks – (see Figure 7.1),

$$(1/r) \geq S(G_A + G_B)^i - S(G_A + G_B)^{i-1}$$

where:

$1/r$, is the rate at which synchronization blocks are processed, and

$S(G_A + G_B)^{i-1}$, and $S(G_A + G_B)^i$ are the start times of consecutive synchronization blocks $i-1$ and i, respectively.

In the context of streaming media devices, the latter is typically defined as a *soft constraint*, that is, a constraint that may be occasionally violated, and larger windows of time may be considered, in which case the soft throughput constraint would be expressed as

$$(1/r_x) \geq S(G_A + G_B)^i - S(G_A + G_B)^{i-x},\ \textit{where}: x \geq 1$$

Enforcing synchronization constraints is very simple in our approach, involving only a minor extension to the previous framework. Namely, we only need to keep track of the number of granularity g tasks of each individual application that still need to be executed in order to complete the current synchronization block. For example, consider Figure 7.1, which shows a snapshot of two applications executing concurrently: (a) on an SMT machine with two thread contexts (Thr1 and Thr2); and (b) on a conventional superscalar machine. Note that, when the SMT core is used, each application runs concurrently in its own thread context, while when the conventional superscalar machine is used, the applications switch contexts periodically. For simplicity, the figure represents only the time it took to process two consecutive synchronization blocks, $(G_A + G_B)^1$ and $(G_A + G_B)^2$, with no explicit indication of the processing and data transfer tasks executed during such period. In-

deed, assuming that applications **A** and **B** are using task granularities g_A and g_B, respectively, each task from **A** and **B** will be correspondingly executed $n_A = G_A/g_A$ and $n_B = G_B/g_B$ times during the processing of a single synchronization block. During such a processing of the synchronization block, the scheduling policies used for each application's processing and data transfer tasks are identical to those used for a single application media system (see Section 4.3). The only difference is that, once the i^{th} set of n_A tasks completes, the next set of n_A tasks can only start executing if the corresponding i^{th} set of n_B tasks has also completed, and vice-versa – this ensures that the synchronization constraint specified in Figure 7.1 is met. As before, implementing these policies in this more general case, is also quite simple. The only difference is that in addition to the previous case, we also need to keep track of the number of sets of granularity g tasks ($n = G/g$) that have been completed for each application.

7.3 Design Space Exploration and Tuning Methodology

Note that, while G_A and G_B, the synchronization granularities used in the example in Figure 7.1, are *'specified'* for a particular device and execution scenario – e.g., they may represent synchronization constraints between audio and video, task granularities g_A and g_B are not. In fact, just as in the previous case (single application), g_A and g_B are the key tuning parameters used to maximize the energy-delay efficiency of Xtream-Fit for a particular set of target media applications. The only extra restriction posed to g_A and g_B is that ($n_A =$

G_A/g_A) and ($n_B = G_B/g_B$) should be integral. If this is not possible, then, the corresponding synchronization block (say, of application **A**), is executed $n_A = \lfloor G_A/g_A \rfloor$ times in 'standard' mode and the remaining $g_A{}' = G_A - n_A g_A$ tasks are executed in a shorter burst.

As will be seen in Chapter 8, the "smooth" design space induced by Xtream-Fit enables, once again, the use of a very simple tuning methodology to find "optimal" task granularities for the multi-application execution scenario. Specifically, we found experimentally that for most cases the "optimal" task granularity can be determined *independently for each application*, by executing it *alone*:

1. on the same target machine, for the superscalar case; or

2. on a target machine with as many issue slots as the average IPC sustained by the particular application/thread, when executing *concurrently* with the remaining applications, for the SMT case.[1]

As before, for each such application, the task granularity was varied (i.e., increased) during design space exploration, until the point of maximum energy-delay efficiency was found.

Our experimental results consistently show that, even when the task granularities derived using this simple method are not actually "optimal" for

[1] In the experiment used to assess the individual IPC of each thread, we assume a perfect memory subsystem and no synchronization constraints.

the concurrent scenario, the decrease in energy efficiency resulting from using such values is typically insignificant (within 2%, see e.g., Figure 8.5).

In the following chapter we describe the experiments and results for multiple media applications running on a single embedded system.

Chapter 8

Experimental Results: Multi-Application Media Systems

In this chapter, we evaluate the effectiveness of the Xtream-Fit data memory architecture for the general case, viz., an embedded system running multiple media applications concurrently. As mentioned in the previous chapter, we consider platforms with Simultaneous Multithreading cores[1] as well as conventional superscalar cores to evaluate Xtream-Fit. As before, the cache based data memory subsystem of the reference systems was augmented with state-of-the-art cache decay and SDRAM power mode control policies. Our experimental results show that, Xtream-Fit continues to be substantially more energy-efficient than the reference systems, even for this general case.

8.1 Experimental Setup

The experimental framework used to evaluate Xtream-Fit against the reference systems is similar to that used for the single application case – see Figure 6.1. The only difference is that, in this new framework, we can now

[1] A Simultaneous Multithreading machine can issue multiple instructions from multiple thread contexts each cycle and can thus execute several programs (threads) simultaneously [59]

evaluate the performance and energy consumption of systems running multiple media applications concurrently. Accordingly, the Simplescalar simulator in Figure 6.1 was replaced with a simultaneous multithreading processor simulator [39]. In all our experiments the number of 'hardware contexts' was set to the number of different applications executing on the SMT machine. The SMT processor simulator is however, highly configurable, and can also emulate a conventional superscalar processor running multiple applications simultaneously, i.e., execute one application at a time and switch contexts periodically. We were thus able to use the same simulator framework to compare the energy-efficiency of Xtream-Fit against reference systems under the varying bandwidth requirements of these two kinds of processors.

The inputs to the simulator now also include a *set* of one or more media programs, with corresponding input data sets, and synchronization constraints when applicable. The SMT processor simulator used for these experiments was also augmented to provide (1) leakage and dynamic energy consumption numbers (2) an off-chip memory access trace, and (3) a task trace for Xtream-Fit – performance and energy numbers were obtained as in the single application study, see Section 6.1.

8.2 Benchmarks

We consider two experimental scenarios representative of workloads for advanced multimedia devices, namely, video-phones and camera-phones (see column 2 in Table 8.1). Specifically, *Scenario 1* considers a device that enables

Table 8.1: Benchmarks

Single Application Scenario		Multi-Application Scenario	
MPEG2	Video compression decoder	MPEG2 G.721d G.721e	**SCENARIO 1** Video-phone Application
JPEG	Image compression encoder		
G.721d	Voice compression decoder	JPEG G.721d G.721e	**SCENARIO 2** Camera-phone Application
G.721e	Voice compression encoder		

Table 8.2: Simulator parameters for core configurations.

	1-wide core	2-wide core	4-wide core	8-wide core
Fetch/Retire Rate	1	2	4	8
Functional/Load-Store Units	1/1	2/2	4/2	8/4
RUU Size	8	16	32	128
LSQ Size	8	8	16	32
Out-Of-Order	no	yes	yes	yes

viewing your interlocutor during a phone conversation – the mix of applications considered in this case is an MPEG2 decoder, a G.721 decoder and a G.721 encoder. *Scenario 2* considers a device that enables the capture and encoding of pictures while talking on the phone – the mix of media applications in this case is a JPEG encoder, a G.721 decoder and a G.721 encoder. The component applications comprising Scenarios 1 and 2 are listed in column 1 of Table 8.1. The input data sets used to run these applications were the primary input data sets used in the experiments in the single application case.

Table 8.3: Reference Subsystems: Cache hierarchy configurations.

Size Range	Associativity	Line Size	Kill-Win Range
L1 Cache Parameters			
4KB-32KB	1/2Way	32B	4K-64K
L2 Cache Parameters			
16KB-256KB	1/2/4Way	64B-256B	64K-1024K

8.3 Machine Models

As mentioned previously, we considered platforms with both SMT and conventional superscalar cores. We modeled 4 processor cores with issue widths ranging from 1 to 8 for each machine type. The cores with smaller issue widths were used to assess the energy-efficiency of Xtream-Fit in the context of low-power systems, while the high issue width cores were used to assess it's efficiency in the context of higher performance systems. The key core configuration parameters for the 4 systems are shown in Table 8.2.

The data cache hierarchy parameters for the reference systems were aggressively varied, as shown in Table 8.3. When considering experiments for the reference systems with the SMT cores, we had the choice of using either unified L1 and L2 data caches or separate L1 and L2 data caches, with one cache per thread context. Note that, after an extensive design space exploration for these systems, we found that the best energy-delay product for the reference subsystems was consistently obtained for configurations with one L1 data cache per thread context, and a unified L2 data cache (shared by all contexts) – yet the 'best' performing cache sizes, associativity, kill window sizes, etc.,

varied quite substantially across the various media benchmarks. Naturally, for the superscalar machines, a standard hierarchy with a *single* L1 D-Cache and a single L2 D-Cache was used. Finally, for the Xtream-Fit configurations, we have one 2KB Scratch-Pad per application, and a Streaming Memory whose size depends on the corresponding task granularities (see Section 5.1).

8.4 Results: Single Application Case

We start by discussing the effectiveness of Xtream-Fit for single application media devices. These experiments are conceptually similar to the results presented in Chapter 6, yet we regenerated them with the SMT machines with a single hardware context, to use them as baselines for the multi-application case.

A summary of the experimental results obtained for the MPEG2 decoder (executing with input stream mobile.mpg), is presented in Figure 8.1. Specifically, Figure 8.1(a) plots decoding delays for platforms using Xtream-Fit vs. the corresponding best reference configuration, for 1, 2, 4 and 8-wide machines. **Xy-z** denotes a platform with an y-wide processor core and Xtream-Fit, where z is the "optimal" task granularity for the particular case. **Ry** denotes a platform with an y-wide processor core and the best reference memory subsystem, i.e., the one leading to minimum energy-delay product, for the particular case.

Not surprisingly, for most cases Xtream-Fit delivers marginal performance improvements (always within a single digit – see relative percentages

indicated at the top of Figure 8.1(a)), since properly tuned cache-based data memory subsystems perform very well for media workloads. For the same set of experiments, Figure 8.1(b) plots total energy consumption on memory components, i.e., in the Streaming Memory, Scratch-Pad Memory and SDRAM for Xtream-Fit, and in the cache hierarchy and SDRAM for the reference subsystems. As indicated in the figure, Xtream-Fit consistently outperforms the best reference configurations, with percentage improvements in total energy consumption varying between 60% and 65%. Finally, Figure 8.1(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 by 62%-66%, when compared to the best performing reference configuration, for any particular core. A comparative analysis between Xtream-Fit and the reference memory subsystems for the three different energy components is provided below.

8.4.1 On-chip Dynamic Energy

The dynamic energy savings enabled by Xtream-Fit are quite significant, being mostly 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).

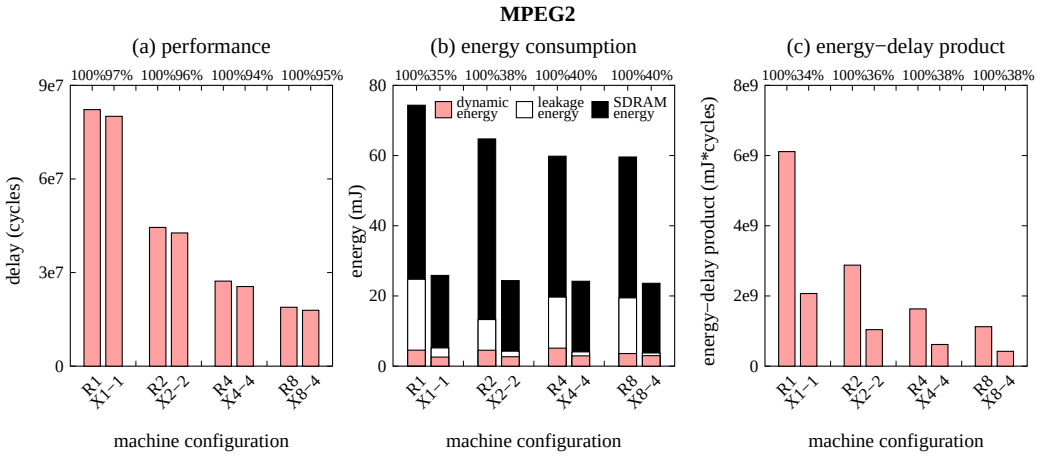


Figure 8.1: Comparative evaluation of performance, energy-consumption and energy-delay product, Xtream-Fit vs. Reference Subsystems for the MPEG2 decoder running on a 1-wide, 2-wide, 4-wide and 8-wide MIPS R10000 like machine.

8.4.2 On-chip Leakage Energy

Even though the cache decay scheme used in the reference subsystems works very well for "low reuse" media applications [33, 65], we found that Xtream-Fit's on-chip memories consistently consumed significantly less leakage energy (with 82% to 95% savings for the MPEG2 decoder). This is mostly due to Xtream-Fit's ability to efficiently/selectively shut down Steaming Memory regions, as the data objects they hold become "dead."

8.4.3 Off-chip SDRAM Energy

Xtream-Fit's savings in SDRAM energy consumption are also significant across all sets of experiments (50% to 61% savings for the MPEG2 experiments). These improvements occur despite the fact that we use the same

SDRAM shutdown policy for, both, Xtream-Fit and the reference subsystems. 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, and the virtual elimination of any possible data conflicts in on-chip memory (requiring live data eviction), are the two key advantages of Xtream-Fit over the reference configurations, leading to fewer and far more efficient accesses to external memory.

8.4.4 Trends with Processor Performance

We now briefly discuss the main trends observed for this set of experiments, as processor issue width increases, and as more bandwidth is required from the data memory subsystem. We consider Xtream-Fit first. As one moves from a 1-wide processor to a 8-wide processor, "optimal" task granularity increases by a factor of 4 (specifically, g^* is 1 for the 1-wide machine and 4 for the 8-wide machine), performance *improves* by a factor of 4.5 (see X1-1 and X8-4 in Figure 8.1(a)), and the total energy consumed by Xtream-fit *decreases* by about 9% (see X1-1 and X8-4 in Figure 8.1(b)). Thus, Xtream-Fit's energy-delay efficiency is clearly higher for the faster processors, since, both, delay and energy consumption decrease, as issue width increases.

Let us now analyze the specific trends for each of the three energy components, noting that for the 1-wide machine (X1-1), on-chip leakage energy and on-chip dynamic energy account each for roughly 10% of the total

energy consumed by Xtream-fit, while off-chip SDRAM energy accounts for about 80% of such energy. As we move from X1-1 to X8-4, total on-chip leakage energy decreases by 70%, on-chip dynamic energy increases by about 16%, and off-chip SDRAM energy decreases by about 4%. The observed decrease in on-chip leakage energy is due to two factors. First, the *2 KB* Scratch-Pad leaks for much less time for the 8-wide machine (or, stating it differently, the average amount of leakage energy wasted per 'unit of work', or task, is much smaller). Second, the average time a given data object is alive in the Streaming Memory is also smaller for the 8-wide machine – indeed, performance improved by a factor of 4.5 while task granularity increased by only a factor of 4.[2] The increase in on-chip dynamic energy results from two factors. The first of such factors is the increase in the size of the Streaming Memory for the 8-wide machine, due to the corresponding increase in task granularity. However, for our relatively small software-controlled memories, the corresponding increase in energy consumption is almost negligible. Indeed, the bulk of the observed increase in on-chip dynamic energy consumption is actually caused by a substantial increase in the number of memory accesses, due to the higher rate of misspeculations/branch mispredictions incurred by the wider out-of-order processor core. Finally, the small decrease in off-chip SDRAM energy consumption results, again, from the increase in task granularity, and thus on the size of data blocks accessed from main memory. Note finally that the small decrease observed in this case clearly indicates that the individual cost

[2]The size of the Streaming Memory for the MPEG2 decoder is $(g \times 2KB)$.

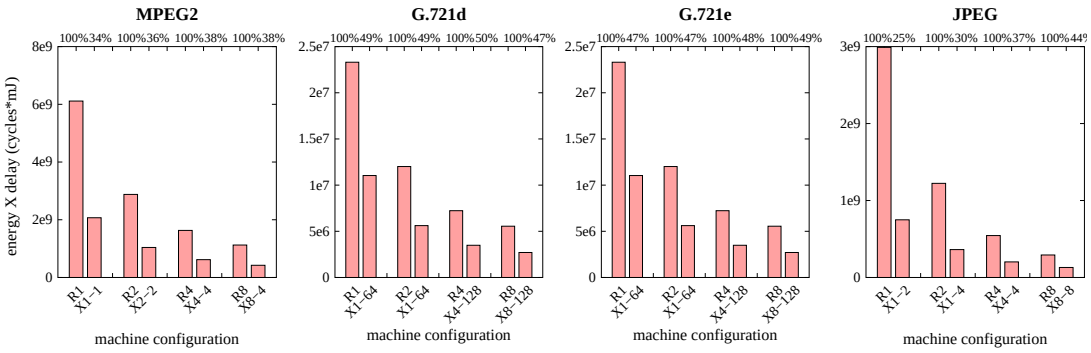


Figure 8.2: Comparative evaluation of energy-delay product for Xtream-Fit as compared to the "best" reference subsystems for all benchmarks running on a 1-wide, 2-wide, 4-wide and a 8-wide MIPS R10000 like machine.

of read/write accesses to the SDRAM is already quite amortized for a task granularity of 1.

Summarizing, if one can perform the same amount of processing work in less time, leakage and power down costs per 'unit of work' (incurred by on-chip and off-chip memories, respectively) decrease. In addition, the memory subsystem may be able to more aggressively prefetch data, with the associated benefits in terms of SDRAM access cost amortization. Obviously, faster processors are also more power hungry. Still, the point being made here is that, for the set of platforms/cores considered in our experiments, Xtream-Fit's energy-efficiency increases with higher bandwidth requirements, which is clearly a desirable feature.

Let us now analyze the results for the reference memory subsystems. While the tuning of Xtream-Fit was very simple, the tuning of the reference subsystems (in order to find the point of "maximum" energy delay effi-

ciency) was extremely expensive, requiring thousands of simulations (aggressively varying L1 and L2 cache sizes, and corresponding line sizes, degree of associativity, and kill window sizes – see Table 8.3). For the best reference subsystems, as one moves from a 1-wide processor to a 8-wide processor, performance improves by a factor of 4.3, and total energy consumption decreases by about 20%. Specifically, on-chip leakage energy consumption decreases by about 22% (yet some "fluctuations" occur for intermediate machines[3]), on-chip dynamic decreases by 21% and off-chip SDRAM decreases by about 19%. (Note that, for the 1-wide machine, on-chip leakage energy, on-chip dynamic energy, and off-chip SDRAM energy accounted for roughly 27%, 6% and 67% of the total energy.) Thus, although the trends seen here are somewhat similar to the ones observed for Xtream-Fit, the degree to which each energy component varies is quite distinct. A detailed discussion on the rationale for such variations would require a lengthy discussion on the many (conflicting) parameters defining a cache hierarchy [12, 23, 62], and is therefore not presented.

Note finally that Xtream-Fit's improvement in energy-delay efficiency with respect to the best reference subsystem slightly decreases as we move from the 1-wide to the 8-wide machine (66% for a 1-wide machine, 64% for a 2-wide machine, 62% for a 4-wide machine, and 62% for a 8-wide machine). As the processing subsystem becomes faster, the delay dominance of the processing task over the data transfer task decreases and as a result, Xtream-Fit's performance improvement slightly diminishes. It is important to note however

[3]In other words, leakage sometimes increases as we move to a wider machine.

that, this decrease in energy-efficiency is only marginal, and that Xtream-Fit continues to be energy-efficient even for faster/wider processors.

8.4.5 Summary of Experimental Data

Figure 8.2 shows the percentage improvements in energy-delay product achieved by Xtream-fit with respect to the best reference subsystems, for our four media benchmarks. As it can be seen, Xtream-Fit consistently delivers substantial improvements in energy-delay product, ranging from 50% to 75%.

The experimental results presented above were derived using a single input data stream per application. The results are however representative, since we have determined, through extensive experimental validation, that using the exact same task granularity g^* for very distinct input data sets consistently gives very similar energy-delay efficiency improvements with respect to the baseline reference subsystems (for each of the individual benchmarks), with variations within 2% – see e.g., Section 5.3.

8.5 Results: Multiple Applications Case

We now discuss the effectiveness of Xtream-Fit for media devices running multiple applications under synchronization constraints.

8.5.1 Analysis of Results for Scenario 1

A summary of the experimental results obtained for *Scenario 1*, i.e., an MPEG2 decoder running simultaneously with a G.721 encoder and a G.721

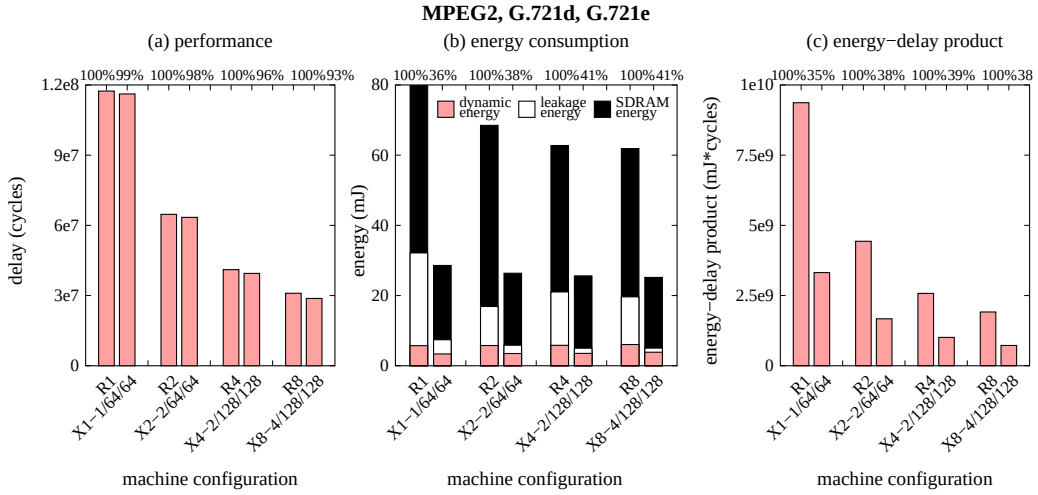


Figure 8.3: Comparative evaluation of performance, energy-consumption and energy-delay product for Xtream-Fit vs. Reference Subsystems, for the MPEG2:G.721d:G.721e mix (Scenario 1) running on a conventional superscalar machine (MIPS R10000 like).

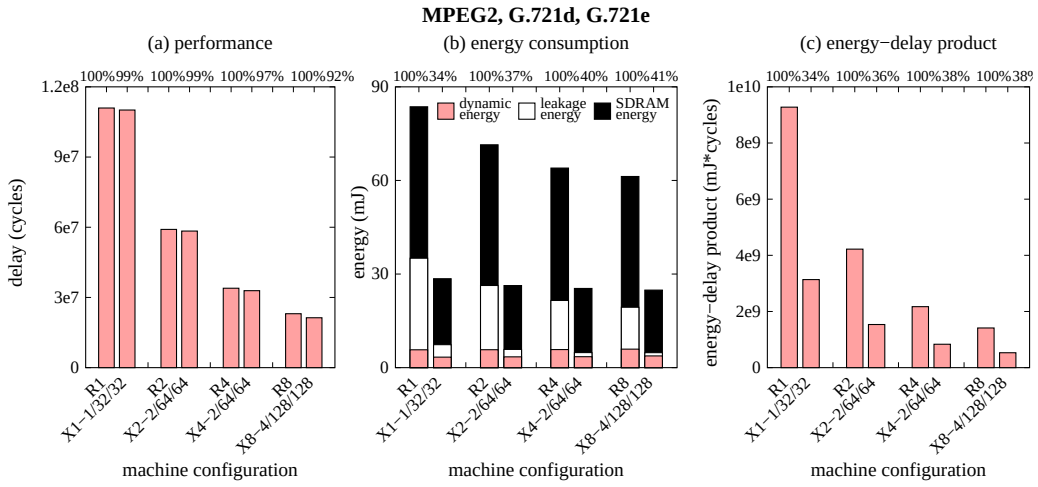


Figure 8.4: Comparative evaluation of performance, energy-consumption and energy-delay product for Xtream-Fit vs. Reference Subsystems, for the MPEG2:G.721d:G.721e mix (Scenario 1) running on an SMT machine.

decoder, is presented in Figures 8.3 and 8.4. As in the previous case, our experiments consider 1, 2, 4, and 8-wide traditional superscalar cores (Figure 8.3) and SMT cores (Figure 8.4). Since three applications run concurrently in this experiment, the labels for the Xtream-Fit configurations indicate the "optimal" task granularity for each such application. Specifically, **Xy-z/w/v** denotes a platform with Xtream-Fit and a *y-wide* processor, where the "optimal" task granularity for the MPEG2 decoder, the G.721 encoder and the G.721 decoder is z, w, and v, respectively. For this experiment, we specified that the decoding of each video frame (comprising 396 macroblocks) should be synchronized with the decoding and encoding of 1092 sound samples, which corresponds to a *synchronization granularity* of ($G_{MPEG2} = 396$, $G_{G.721d} = 1092$, $G_{G.721e} = 1092$).

Figures 8.3(a) and 8.4(a) plot decoding delays for platforms with superscalar and SMT cores, respectively. Similarly to the single application case, Xtream-Fit delivers marginal performance improvements with respect to the best reference subsystems. In terms of absolute performance, the SMT and superscalar based platforms with 1-wide processors achieve a rate of 5 video frames (CIF resolution) and 5+5 audio frames per second (32.2Kbps) (MPEG2, G.721d and G.721e); platforms with 2-wide processors achieve a rate of 10 video frames and 10+10 audio frames per second; and platforms with 4 wide processors achieve a rate greater than 30 video and 30+30 audio frames per second. The latter is a very reasonable performance for the video-phone application under consideration.

Figures 8.3(b) and 8.4(b) plot total energy consumption on memory

components. As it can be seen, Xtream-Fit again consistently outperforms the best reference configurations, delivering improvement in total energy consumption in the range of 59% to 66%. Finally, Figures 8.3(c) and 8.4(c) plot the corresponding energy-delay product metric. The results show very substantial improvements for Xtream-Fit, with decreases in energy-delay product of 61% to 66%.

For each individual scenario, the "optimal" task granularity (g^*) was determined on a per application basis, using the simple method discussed in Section 7.3. Figure 8.5 shows the percentage variation in energy-delay product resulting from using task granularities g^*, $g^*/2$ (represented at distance of -1 from g^* in the graph) and $2g^*$ (represented at distance of 1 from g^*), for each of the applications in *Scenario 1* (i.e., MPEG2 decoder, G.721 encoder and G.721 decoder). As it can be seen, in Figure 8.5 the variation in energy-delay product (when g^*, $2g^*$ and $g^*/2$ task granularities are used) is insignificant, suggesting that our simple design space exploration method should work very well for most cases.

Let us now analyze the energy consumed on the various memory components – see Figures 8.3(b) and 8.4(b). As we move towards wider machines, Xtream-Fit exhibits a very similar behavior to that observed for the single application case – on chip leakage energy decreases substantially, on-chip dynamic energy increases somewhat (roughly 12% from the 1-wide to 8-wide machine, due to a steady increase in memory accesses, caused by a higher rate of branch mispredictions), and off-chip energy decreases marginally (within a

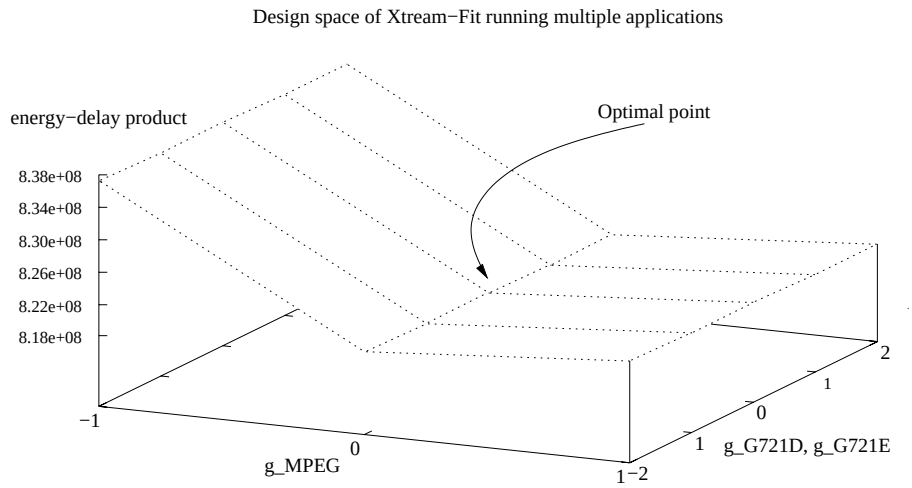


Figure 8.5: Xtream-Fit: Variations in energy-delay product for task granularities close to the "optimal" (Scenario 1).

single digit for all cases). The reference subsystems, in turn, exhibit a much less 'predictable' behavior, with no clear trend for on-chip leakage energy, as well as some fluctuations in on-chip dynamic energy and, of course, always perform much less energy efficiently than Xtream-Fit.

Note that SMT machines achieve better performance than their superscalar counterparts, since they are better able to take advantage of the available machine issue slots, by simultaneously exploiting the ILP available in the three threads. For example, for the set of Xtream-Fit experiments pertaining *Scenario 1*, the relative performance improvements achieved by the SMT cores (with respect to the their superscalar counterparts) vary from 5% (1-wide machines) to 26% (8-wide machines). The experimental data shows that Xtream-Fit performs similarly well for both families of processors, with fluctuations on relative energy-delay efficiency with respect to the reference

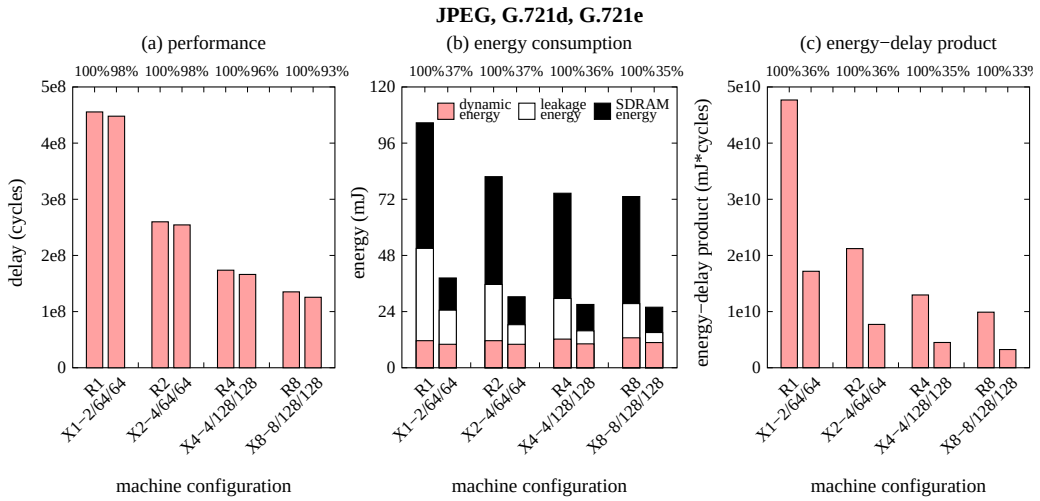


Figure 8.6: Comparative evaluation of performance, energy-consumption and energy-delay product for Xtream-Fit vs. Reference Subsystems, for the JPEG:G.721d:G.721e mix (Scenario 2) running on a conventional superscalar machine (MIPS R10000 like).

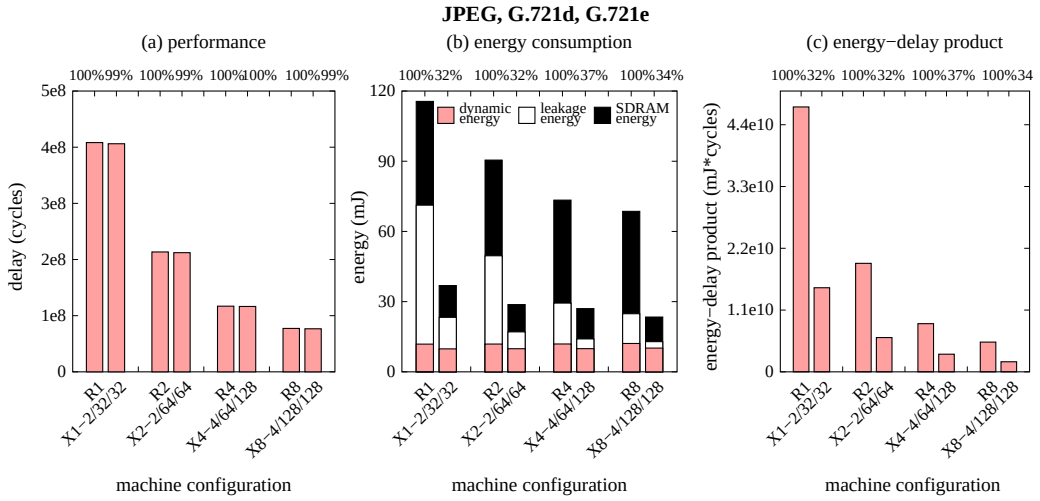


Figure 8.7: Comparative evaluation of performance, energy-consumption and energy-delay product for Xtream-Fit vs. Reference Subsystems, for the JPEG:G.721d:G.721e mix (Scenario 2) running on an SMT machine.

subsystems being mostly caused by corresponding fluctuations on the performance possible to achieve with these baselines.

8.5.2 Summary of Results for Scenario 2

Scenario 2 considers a JPEG decoder running simultaneously with a G.721 encoder and a G.721 decoder. For this experiment, we specified a somewhat artificial synchronization constraint, namely, that the encoding of 32 8×8 pixel blocks of a 512×512 image should be synchronized with the decoding and encoding of 1092 sound samples, corresponding to a synchronization granularity of ($G_{JPEG} = 32$, $G_{G.721d} = 1092$, $G_{G.721e} = 1092$).

A summary of the experimental results obtained for *Scenario 2* is presented in Figures 8.6 and 8.7. As before, Xtream-Fit's performance is marginally better than the best reference subsystems, it delivers very substantial improvements in total energy consumption (between 63% and 68%) and thus substantial improvements in energy-delay product (between 63% and 68%).

Chapter 9

Conclusions and Future Work

In this dissertation we propose a special-purpose data memory subsystem, called Xtream-Fit, targeted to streaming media applications executing on, both, generic uniprocessor embedded platforms and SMT-based multi-threading platforms. We empirically demonstrate that Xtream-Fit achieves high energy-delay efficiency across a wide range of media devices, from systems running a single media application to systems concurrently executing multiple media applications under synchronization constraints. Xtream-Fit's energy efficiency is predicated on 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. 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 to find the best memory configuration for the target application(s). Extensive experimental results show that Xtream-Fit reduces energy-delay product substantially – by 32% to 69% – as compared to 'standard' general-purpose memory subsystems enhanced with state of the art cache decay and SDRAM power mode control policies. Based on these results, we argue 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 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, while greatly simplifying the overall customization effort (hardware and software), when contrasted to the second group of solutions.

Our results indicate that Xtream-Fit scales seamlessly when used with a wide range of processors including both conventional superscalar as well as SMT machines, i.e, efficiency gains delivered by Xtream-Fit with respect to the reference subsystems are not substantially impacted by the actual ILP of the machine. This strongly suggests that, for media applications, Xtream-Fit will consistently be much more energy-efficient than a corresponding cache-based system dimensioned to deliver similar performance. For very high performance systems, e.g., using programmable accelerators such as Imagine, faster, more complex special purpose data memory architectures may be required.

An interesting future work direction would be to assess our approach's ability to support variable ("data driven") synchronization constraints. Some streaming media applications (or mixes of such applications) may require distinct synchronization granularities for different input data sets. Consider, for example, a media device that must simultaneously execute an MPEG2 decoder and a G.721 decoder. Different execution scenarios may be conceived for such a device. For example, one may need to decode video streams at CIF or

QCIF resolution (say, at a rate of 11880 macroblocks/s at 30 frames/s or 1485 macroblocks/s at 15 frames/s respectively) and with correspondingly different audio frame rates. These two scenarios would require different synchronization granularities, i.e., $(G_{MPEG2d}, G_{G.721d})$ values. At a first glance, it appears as though such diversity can be handled well in our approach. Specifically, since the number of macroblocks per frame is identified in the video stream, the required n_{MPEG2d} and $n_{G.721d}$ values for the particular input set characteristics (i.e., the number of times each task from each of the applications needs to be executed before a synchronization point is reached), could, for example, be quickly retrieved from a lookup table, and appropriately used.[1]

[1]Naturally, for each mix of applications, one may need to store several such tuples, indexed by relevant stream characteristics, and possibly with a default set of values.

Bibliography

[1] 128/144-MBit Direct RDRAM Data Sheet. Rambus Incorporated, http://www.rdram.com/.

[2] 128MBit Micron Mobile SDRAM Data Sheet. Micron Technology Incorporated, http://www.micron.com/.

[3] 128MBit Samsung Mobile SDRAM Data Sheet. Samsung Electronics, http://www.samsung.com/.

[4] http://www.eecs.umich.edu/ jringenb/power/. url.

[5] http://www.microcontroller.com/microcontrollers/marketplace.

[6] D. H. Albonesi. Selective Cache Ways: On-demand Cache Resource Allocation. In *Proceedings of the International Symposium on Microarchitecture*, 1999.

[7] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, 19(4), 1999.

[8] S. Borkar. Getting Gigascale Chips: Challenges and Opportunities in Continuing Moore's Law. *ACM Queue*, 1(7), 2003.

[9] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. *International Journal of Computer Simulation, special issue on Simulation Software Development*, 4, 1994.

[10] D. Burger, T. M. Austin, and S. Bennett. Evaluating Future Microprocessors: the SimpleScalar Tool Set. Technical report, Computer Sciences Department, The University of Wisconsin-Madison, 1996.

[11] D. Callahan, K. Kennedy, and A. Porterfield. Software Prefetching. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 1991.

[12] F. Catthoor, S. Wuytack, E. DeGreef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design.* Kluwer Academic Publishers, 1998.

[13] T. F. Chen and J. L. Baer. Effective Hardware-Based Data Prefetching for High Performance Processors. *IEEE Transactions on Computers*, 44(5), 1995.

[14] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M.J. Irwin. Scheduler-based DRAM Energy Management. In *Proceedings of the Design Automation Conference*, 2002.

[15] S. Dutta, W. Wolf, and A. Wolfe. A Methodology to Evaluate Memory Architecture Design Tradeoffs for Video Signal Processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(1), 1998.

[16] J. Edler and M. D. Hill. Dinero IV Trace-Driven Uniprocessor Cache Simulator, 1998. http://www.cs.wisc.edu/∼markhill/DineroIV/.

[17] X. Fan, C. S. Ellis, and A. R. Lebeck. Memory Controller Policies for DRAM Power Management. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2001.

[18] J. Fritts, W. Wolf, and B. Liu. Understanding multimedia application characteristics for designing programmable media processors. In *SPIE Photonics West, Media Processors*, 1999.

[19] J. W. C. Fu, J. H. Patel, and B. L. Janssens. Stride Directed Prefetching in Scalar Processor. In *Proceedings of the International Symposium on Microarchitecture*, 1992.

[20] R. Gonzalez, B. Gordon, and M. Horowitz. Supply and Threshold Voltage Scaling for Low Power CMOS. *IEEE Journal of Solid-State Circuits*, 32(8), 1997.

[21] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9), 1996.

[22] R. Graybill and R. Melhem, editors. *Power Aware Computing*. Kluwer Academic Publishers, 2002.

[23] P. Grun, N. Dutt, and A. Nicolau. *Memory Architecture Exploration for Programmable Embedded Systems*. Kluwer Academic Publishers, 2003.

[24] H. h. Lee and G. Tyson. Region-Based Caching: An Energy-Delay Efficient Memory Architecture for Embedded Processors. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2000.

[25] E. G. Hallnor and S. K. Reinhardt. A Fully-Associative Software-Managed Cache Design. In *Proceedings of the International Symposium on Computer Architecture*, 2000.

[26] H. L. Hanson. Static Power in Microprocessor Caches. Technical report, Department of Electrical and Computer Engineering, University of Texas at Austin, 2001.

[27] C. J. Hughes, P. Kaul, S. V. Adve, R. Jain, C. Park, and J. Srinivasan. Variability in the Execution of Multimedia Applications and Implications for Architecture. In *Proceedings of the International Symposium on Computer Architecture*, 2001.

[28] C. J. Hughes, J. Srinivasan, and S. V. Adve. Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications. In *Proceedings of the International Symposium on Microarchitecture*, 2001.

[29] http://public.itrs.net/.

[30] M. F. Jacome and A. Ramachandran. *Information Technology Handbook,* Editor: R. Zurawski, chapter Power Aware Embedded Computing. CRC Press, 2004.

[31] M. Kamble and K. Ghose. Analytical Energy Dissipation Models For Low Power Caches. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 1997.

[32] M. Kandemir, J. Ramanujam, M. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh. Dynamic Management of Scratch-Pad Memory Space. In *Proceedings of the Design Automation Conference*, 2001.

[33] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proceedings of the International Symposium on Computer Architecture*, 2001.

[34] B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, and A. Chang. Imagine: Media Processing with Streams. *IEEE Micro*, 21(2), 2001.

[35] J. Kin, C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Power Efficient Mediaprocessors: Design Space Exploration. In *Proceedings of the Design Automation Conference*, 1999.

[36] A. C. Klaiber and H. M. Levy. An Architecture for Software Controlled Data Prefetching. In *Proceedings of the International Symposium on Computer Architecture*, 1991.

[37] C. Lee, J. Kin, M. Potkonjak, and W. H. Mangione-Smith. Media Architecture: General Purpose vs. Multiple Application-Specific Programmable Processor. In *Proceedings of the Design Automation Conference*, 1998.

[38] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *Proceedings of the International Symposium on Microarchitecture*, 1997.

[39] D. Madon, E. Sanchez, and S. Monnier. A Study of a Simultaneous Multithreaded Processor Implementation. In *Proceedings of European Conference on Parallel Processing*, 1999.

[40] G. Micheli, R.Ernst, and W.Wolf, editors. *Readings in Hardware/Software Co-Design*. Morgan Kaufman Publishers, 2002.

[41] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, E. M. Cooper, D. W. Dobberpuhl, P. M. Donahue, J. Eno, A. Farell, G. W. Hoeppner, D. Kruckemyer, T. H. Lee, P. Lin, L. Madden, D. Murray, M. Pearce, S. Santhanam, K. J. Snyder, R. Stephany, and S. C. Thierauf. A 160 MHz 32b 0.5 W CMOS RISC Microprocessor. *Proceedings of the International Solid-State Circuits Conference, Digest of Technical Papers*, 1996.

[42] T. C. Mowry, M. S. Lam, and A. Gupta. Design and Evaluation of a Compiler Algorithm for Prefetching. In *Proceedings of the Interna-*

tional Conference on Architectural Support for Programming Languages and Operating Systems, 1992.

[43] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg. Data and Memory Optimization Techniques for Embedded Systems. *ACM Transactions on the Design Automation of Electronic Systems*, 6(2), 2001.

[44] P. R. Panda, N. D. Dutt, and A.Nicolau. Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications. In *Proceedings of the European Design and Test Conference*, 1997.

[45] P. R. Panda, N. D. Dutt, and A. Nicolau. *Memory Issues in Embedded Systems-On-Chip: Optimizations and Exploration.* Kluwer Academic Publishers, 1999.

[46] D. A. Patterson and J. H. Hennessy. *Computer Organization Design, A Hardware/Software Interface.* Morgan Kaufman Publishers, 2004.

[47] M. Pedram and J. M. Rabaey. *Power Aware Design Methodologies.* Kluwer Academic Publishers, 2002.

[48] S. S. Pinter and A. Yoaz. A Hardware-based Data Prefetching Technique for Superscalar Processors. In *Proceedings of the International Symposium on Computer Architecture*, 1996.

[49] A. R. Pleszkun and E. S. Davidson. Structured Memory Access Architecture. In *Proceedings of the International Conference on Parallel Processing*, 1983.

[50] A. Ramachandran and M. F. Jacome. Xtream-Fit: An Energy-Delay Efficient Data Memory Subsystem for Embedded Media Processing. In *Proceedings of the Design Automation Conference*, 2003.

[51] A. Ramachandran and M. F. Jacome. Xtream-Fit: An Energy-Delay Efficient Data Memory Subsystem for Embedded Media Processing. *To appear in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2004.

[52] P. Ranganathan, S. V.Adve, and N. P. Jouppi. Reconfigurable Caches and their Application to Media Processing. In *Proceedings of the International Symposium on Microarchitecture*, 2000.

[53] G. Reinman and N. M. Jouppi. CACTI 2.0: An Integrated Cache Timing and Power Model. Technical report, Compaq Computer Corporation, Western Research Lab, 2001.

[54] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens. Memory Access Scheduling. In *Proceedings of the International Symposium on Computer Architecture*, 2000.

[55] T. Simunic, L.Benini, and G. De Micheli. Energy-Efficient Design of

Battery-Powered Systems. *IEEE Transactions on Very Large Scale Integration Systems*, 9(1), 2001.

[56] N. T. Slingerland and A. J. Smith. Cache Performance for Multimedia Applications. In *Proceedings of the International Conference on Supercomputing*, 2001.

[57] J. E. Smith. Decoupled Access/Execute Computer Architectures. *ACM Transactions on Computer Systems*, 2(4), 1984.

[58] W. Tang, R. Gupta, A. Nicolau, and A. Veidenbaum. Fetch Size Adaptation vs. Stream Buffer for Media Benchmarks. In *Workshop on Media and Streaming Processors, International Symposium on Microarchitecture*, 2001.

[59] D. M. Tullsen, J. Eggers, and H. M. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In *Proceedings of the International Symposium on Computer Architecture*, 1995.

[60] O. S. Unsal, R. Ashok, I. Koren, C. M. Krishna, and C. A. Moritz. CoolCache for Hot Multimedia. In *Proceedings of the International Symposium on Microarchitecure*, 2001.

[61] O.S. Unsal, Z. Wang, I. Koren, C.M. Krishna, and C.A. Moritz. On Memory Behavior of Scalars in Embedded Multimedia Systems. In *Proceedings of the Workshop on Memory Performance Issues*, 2001.

[62] W.-T.Shiue and C. Chakrabarti. Memory Exploration For Low Power Embedded Systems. In *Proceedings of the Design Automation Conference*, 1999.

[63] S. J. E. Wilton and N. M. Jouppi. CACTI: An Enhanced Cache Access and Cycle Time Model. Technical report, Digital Equipment Corporation, Western Research Lab, 1996.

[64] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. Vijaykumar. An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches. In *High Performance Computer Architecture*, 2001.

[65] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte. Adaptive Mode Control: A Static-Power-Efficient Cache Design. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2001.

[66] D. F. Zucker, R. B. Lee, and M. J. Flynn. Hardware and Software Cache Prefetching Techniques For MPEG Benchmarks. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(5), 2000.

Vita

Anand Ramachandran was born in the city of Madras (Chennai), in Tamil Nadu, India, on 25 October 1972, the son of Sundaresan Ramachandran and Radha Ramachandran. He received the Bachelor of Technology degree in Chemical Engineering from the Indian Institute of Technology, Madras in 1995. He received the Master of Science in Engineering degree in Electrical and Computer Engineering in 1997. His research interests include embedded system design, and energy- and reliability-aware processor design and computing. Anand is currently with Intel Corp., Austin, TX.

Permanent address: 47, Saiva Muthaiya St.,
George Town,
Chennai, 600001,
India.

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.