Memory System Design Space Exploration for Low-Power, Real-Time Speech Recognition

Rajeev Krishna, Scott Mahlke, and Todd Austin
Advanced Computer Architecture Lab
University of Michigan (Ann Arbor)
{rkrishna, mahlke, austin}@eecs.umich.edu

ABSTRACT

The recent proliferation of computing technology has generated new interest natural I/O interface technologies such as speech recognition. Unfortunately, the computational and memory demands of such applications currently prohibit their use on low–power portable devices in anything more than their simplest forms. Previous work has demonstrated that the thread level concurrency inherent in this application domain can be used to dramatically improve performance with minimal impact on overall system energy consumption, but that such benefits are severely constrained by memory system bandwidth. This work presents a design space exploration of potential memory system architectures. A range of low–power memory organizations are considered, from conventional caching to more advanced system–on–chip implementations. We find that, given architectures able to exploit concurrency in this domain, large L2 based cache hierarchies and high bandwidth memory systems employing data stream partitioning and on–chip embedded DRAM and ROM technologies can provide much of the performance of idealized memory systems without violating the power constraints of the low–power domain.

Categories and Subject Descriptors: B.3.2 [Memory Structures]: Design Styles; C.3 [Special Purpose and Application Based Systems]

General Terms: Performance Design

1. INTRODUCTION

The tight power and cost constraints of most portable embedded systems is primarily responsible for the exclusion of sufficient computational resources to manage natural I/O applications. This combined with the continued development and evolution of the computing technologies in question (mostly precluding ASIC–style solutions), and the peculiar computational demands of these applications, make this a very challenging design space. Recent work has demonstrated, however, that fairly straightforward architectural mechanisms that exploit thread level concurrency and other domain specific characteristics can dramatically improve performance and moderate or reduce overall energy consumption [10, 11]. The key remaining bottleneck to realizing these performance goals, however, is memory system performance.

The primary challenge to memory system design for speech recognition (and similar natural I/O) systems is access to knowledge base data. In speech recognition, this knowledge base can represent tens to hundreds of megabytes of data, and the input driven nature of this application leads to poor locality and predictability in the memory reference stream, causing difficulties for simple caching or DMA techniques. The throughput requirements of such applications can greatly exceed the capabilities of low–power memory systems generally employed in portable computing devices [11].

This paper explores memory system designs to support real-time speech recognition on low power, portable, parallel platforms. As automated design exploration flows are not viable due to the complexities of the application domain, we perform a systematic, intuition driven manual exploration. Given the target domain, we do not extend our analysis to high performance memory systems found on higher–end platforms, focusing instead on technologies that could realistically be implemented in a low–cost consumer device. The results of this exploration are presented here. We find that moderate sized hybrid Chip Multiprocessor / Multi-Threading (CMP/MT) architectures (4 simple pipelines, 2–4 hardware contexts each), combined with multilevel caching that targets program metadata (dynamic programming lists and the like) provide good tradeoffs in maximizing performance while reducing overall energy consumption.

We begin this analysis in Section 2 with a general overview of our speech recognition infrastructure, introduction of our baseline low–power parallel architecture, and a brief look at the memory stream characteristics of this application space. Section 3 will present our evaluation methodology, describing our architectural simulator and discussing our approach to power and performance estimation. In Section 4, we begin our memory space exploration by considering methods of reducing demand on the memory system. Section 5 expands on this analysis by considering system modifications that affect available bandwidth and off–chip power dissipation. We consider a general overview of the design space relative to a cost metric of on–chip area in Section 6 and conclude with a brief summary and look at future directions for this work in Section 7. An expanded version of this work, detailing individual tradeoffs between hardware, software, and various

CODES+ISSS'04, September 8–10, 2004, Stockholm, Sweden

architectural components in far greater detail, is available as a technical report [10].

2. BACKGROUND

We begin our analysis by considering the problem of speech recognition and presenting a hybrid CMP/MT low-power architecture designed to expose the inherent concurrency in this application domain. This will be a very brief overview in order to provide an understanding of the remaining memory system results. Further details on speech recognition [13], the CMU–Sphinx speech recognizer [9], performance characterization [1, 10], and a similar architectural model [10] are available in cited works.

Speech Recognition

The basic goal of speech recognition is to translate an acoustic language input into its corresponding textual representation. This task is greatly complicated by natural variations in spoken language that are difficult for discrete systems to deal with. Examples include variations in intonation and accent (acoustic level), and variations in meaning such as the difference between "there" and "their" (linguistic level).

Accommodating this flexibility in a discrete computing system requires the use of probabilistic modeling techniques. For example, the CMU-Sphinx speech recognition system employed in this study utilizes probabilistic Gaussian models of pre-trained phonetic sequences to score each discrete unit of input speech against its knowledge base data. It then applies these scores to a phonetic / linguistic model of English language constructs (represented as a Hidden Markov Model or HMM), producing a set of potential recognition hypotheses. The recognition hypothesis with the highest overall probability is presented as the final result.

The nature of the memory reference stream produced by these applications is tied to their internal stochastic search methods. As a result, applications in this class generally place high demands on the memory system and demonstrate poor locality and predictability of access. While these characteristics present challenges for memory system design, the relative independence of computation across data elements presents the opportunity for improved performance through concurrent execution. This potential is recognized by a number of previous studies [1, 10, 3]. Attempts to take advantage of this inherent concurrency, however, further exacerbate the memory problem. Thus, while substantial opportunity for performance improvement exists, achievable performance strongly tied to memory system bandwidth [10].

Parallel Architecture and Programming Model

In order to explore opportunities to improve system performance through concurrent execution, we develop a hybrid CMP/MT architecture designed for low power devices. We present a brief overview of this architecture and of salient performance points here, but a full discussion is available in cited work [10].

Figure 1 shows a high-level overview of the processor architectural model used in this study. The system is based on a standard Intel XScale 400 MHz embedded processor with FPA floating point extensions to accommodate the floating point requirements of speech recognition, and a series of ISA extensions to control the speech processing unit.

The speech processing unit is made up of a number of simple, in–order integer pipelines, each with multiple hardware thread contexts. Each pipeline utilizes a small (2k, 4way) data cache to capture data actively under evaluation. Fine grain mutual exclusion is achieved through a combination of static workload partitioning and key based exclusion (similar to previous techniques [7]). When global, non–data related mutual exclusion is required, a slower global lock management infrastructure can be used. Dynamic load balancing is utilized on top of static workload distribution to handle per–search–iteration imbalances. The processing elements share a small control bus with each other and with the main processor. The aforementioned static workload distribution techniques keep utilization of this resource very low, allowing a single, 8–bit wide bus to accommodate application needs without introducing significant latencies.

Finally, the entire system shares a small communication bus, and a wide bus which leads to a memory controller and possibly a flash or otherwise alternate memory configuration. As this work considers memory system issues, we will begin with the depicted 100MHz, 64bit wide memory bus architecture, and develop our experimental variants from there.

Performance analysis under idealized memory constraints (fixed 50 cycle memory latency and unlimited bandwidth) demonstrates that this architectural model is quite effective at exploiting the concurrency inherent in speech recognition. On a series of test inputs to a hand-parallelized version of the Sphinx–2 library, near ideal speedups were observed, with fall off from ideal directly attributable to intolerable system latencies and often mitigated by the addition of extra thread contexts [10]. The introduction of a realistic memory system model (utilizing a highly detailed SDRAM model [15]), however, nearly eliminates the speedups observed under idealized conditions. Evaluations on similar architectures strongly suggest that the bottleneck introduced by the memory system is not due to internal resource conflicts, but rather pure bandwidth and request rate [10]. It is from this point that we begin our current evaluation.

3. EVALUATION METHODOLOGY

The base simulation infrastructure utilized in this study is a modified version of the SimpleScalar/ARM toolset [2] running a hand parallelized version of the CMU-Sphinx speech recognition engine (v2.0.4) compiled with GNU-GCC for the ARM platform. A search tree traversal profile was generated using a training input stream and partitioned using the hMetis graph partitioning toolset. Due to the nature of the search tree, it was possible to generate efficient partitions with no intra–word edge cuts. Our parallelization efforts have moved approximately 92% of the search phase code over to parallel execution on the speech coprocessor. About half of the eight percent remaining on the main processor is executed concurrently with coprocessor code. All experiments were performed using a 11400 word vocabulary and corresponding language model generated by use of the Cambridge Statistical Modeling Toolkit [5] and based on transcripts of famous speeches and selections of text available online through the Gutenberg Project.

Power Estimation

In considering the power / energy implications of our designs, it is important to consider the overall energy consumed by the entire system, not just the aspect of the system that

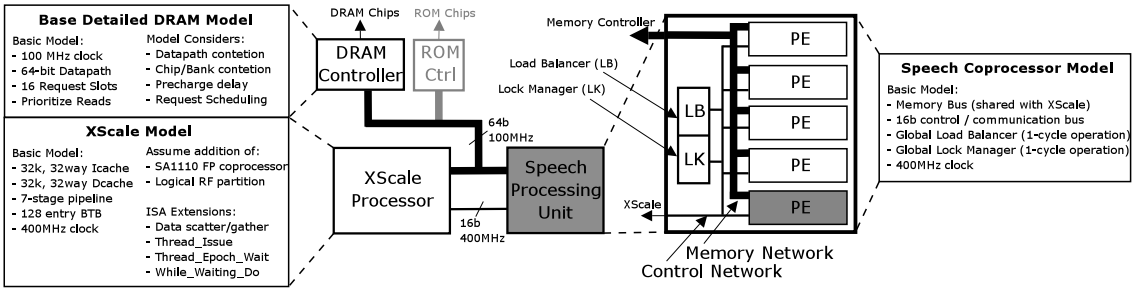

Figure 1: Overview of proposed architecture.

we are manipulating. As an actual hardware implementation of our architecture is impractical at this stage of development, we construct full system power estimates by combining data from a number of sources, relying on the general accuracy of relative (as opposed to absolute) energy estimates to provide some insights into the energy performance of our system.

Our energy estimates are developed from activity based power estimates. Each component is assumed to dissipate either active, idle, or standby power depending on its operating status over the course of simulation. Power estimates for the datapath component of the main XScale processor are generated from Intel PXA250 processor design documents [4], and transistor count based Spice power estimates. As the execution cores of the speech co-processor processing elements are based on the same technology as the XScale, we utilize conservative area scaling estimates of the relevant XScale die area to arrive at power consumption for these components. These estimates (along with a substantial error margin) lead to a value of 200mW for the XScale datapath, and 70mW for processing element datapaths (active power). Idle power dissipation is assumed to be 25% of active power, tracking XScale idle dissipation and data found in other works [6]. We assume a lower power idle NOP for the XScale (as opposed to the standard 'MOV' approach), as it performs relatively little work in our design.

Storage resource power estimates (register files and caches) are based on register bit equivalent techniques [12] and data from the Cacti 3 [14] toolset. Each added hardware thread context (basically a register file and some routing) is accounted for by 10mW of active power dissipation. Active cache dissipation is based on single bank and routing power (keeping with power conservation techniques used on the XScale) and idle power dissipation is based on total cache size.

Power dissipation along two data buses are accounted for by total energy required for all single bit transitions occurring along each bus. On–chip control bus energy is estimated based on a 0.2fF/mm capacitance estimate taken from the Berkeley Predictive Technology Modeland estimating 1.5 mm/PE. Memory bus energy utilizes 60pF off–chip capacitances per line, and 10pF for "off–chip, on–package" estimates (based on data from cited works [8]).

Memory system power estimation techniques vary depending on the specific components under consideration. SDRAM and DDR estimates are generated using the Micron Technologies System Power Calculator, which accounts for active, precharge, read/write, and background dissipation based on activity rate. We utilize device characteristics for Micron 1.7V low power SDRAM and 2.4V DDR chips. This is also used for embedded DRAM, providing a conservatively high estimate. Flash components are based on energy data from Micron Technology Q-Flash datasheets and assuming a 4–chip page mode flash module, resulting in a per–active–access energy dissipation of ≈12nJ. ROM energy estimates account for pre–decode, word–line, and bitline dissipation for a hypothetical 64MB ROM array using techniques presented in cited work [16], resulting in a per–active–access energy dissipation of ≈4.5nJ.

4. CACHING EFFECTS

We begin our analysis by considering methods of reducing the memory demand of speech recognition. We will consider individual data streams, caching potential, and the applicability of techniques such as stream bypassing and compression. As the instruction reference stream shows high locality and generally contributes little to application bandwidth demands, we will not consider it in detail here. Furthermore, we focus on general cache organization which affect the performance / energy tradeoff, rather than how specific cache architectures affect program performance. Consideration of effects such as line size, associativity, and cache replacement policy can be found in cited works [10].

Data Caching

The most direct approach to reducing memory system demand is to capture more data on the processor itself, exploiting spatial locality. Due to the size of knowledge base data, and the input dependent nature of knowledge base access patterns, such spatial locality is not generally seen in speech recognition applications. Furthermore, data access patterns, while somewhat predictable at the software level, are hidden at the hardware level by the same input dependent characteristics. As the bottleneck is memory bandwidth rather than latency, intelligent prefetching efforts do little to improve performance and only exacerbate the problem.

Despite these access characteristics, we find a multi-level cache hierarchy to be quite effective at reducing memory system demand. Despite the poor reference locality in knowledge base data, a substantial fraction of references occur to program metadata (active lists and such), which shows considerably more locality. Furthermore, caching data for knowledge base elements that are currently under evaluation not only reduces memory demand, but significantly improves performance.

At the level of individual processing elements, we find that a 2k, 4way cache is quite effective at capturing active data and metadata locality. A system with L1 caches tied di-

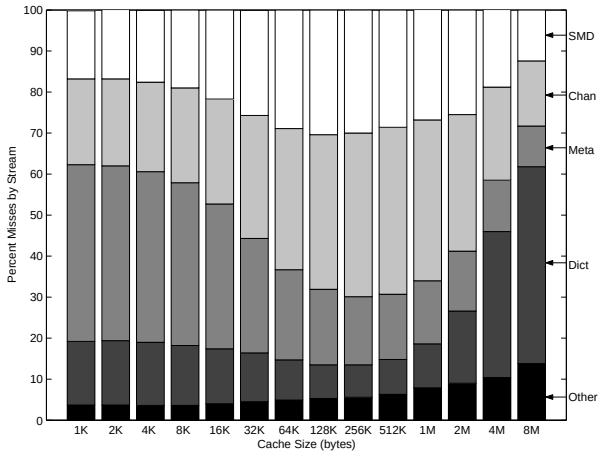

Figure 2: Fraction of L2 misses by data stream for a 4–processor, 4–context system with 2K L1 caches. CHAN, SMD, PDF, LM, and DICT all represent knowledge base data. Note the shift in misses from program metadata (META) to knowledge base data at around 128K.

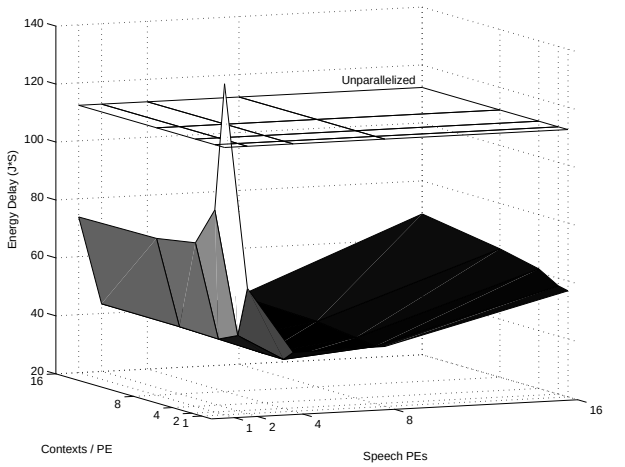

Figure 3: Energy delay product of a system with a 128K L2 cache, 2K per–pipeline L1 caches, and varied processor pipelines and hardware thread contexts per pipe. Note surface representing unparallelized workload on a standard XScale system.

rectly to the memory system, however, still suffers from extremely constrained performance. Adding a 128K–256K L2 cache to back up all L1 caches leads to further dramatic reductions in bandwidth demand and corresponding performance improvement. The overall analysis here is best presented by Figure 2, which depicts the percent of L2 misses at various L2 sizes for each of a number of application data streams. The shift from metadata to knowledge base data (SMD and CHAN, which represent acoustic and linguistic knowledge base data respectively) is clearly seen at around 128k. Figure 3 shows the estimated workload energy delay product (EDP) of a system with a 128k L2 cache, 2k L1 caches, and varied processors and contexts per processor. This is compared to the workload EDP of the unparallelized code running on a standard XScale system. The spike for one processor, one context demonstrates the parallelization overhead of our approach. We focus on EDP because, by combining (multiplying) running time and energy consumption, the resulting metric provides an inherent tradeoff by which to compare disparate configurations. Note that this 128K L2 configurations shows a 50% EDP improvement for greater than two processor configurations when compared to a system without an L2.

Stream Bypassing

Given the benefit of caching metadata, and the stream access characteristics of program knowledge base data, a logical configuration might be to stream knowledge base data around the L2 and/or L1 cache structures. This has the potential of reducing L2/L1 pollution with low–locality data, improving overall performance. We consider such a potential optimization by streaming a number of individual knowledge base streams around both cache components, and find no significant improvement in performance in the general case. The L2, while ineffective at capturing knowledge base working set, *is* effective at backing up currently active data that has been kicked out of the L1 due to actions such as context switches. For smaller L2 configurations, the added bandwidth demand of these references overwhelms the potential benefit of better metadata locality. At larger L2 configurations, the metadata is not unnecessarily evicted anyway, and the performance improvement due to added cache size trades off roughly evenly (in terms of EDP) with energy dissipation.

Data Compression

We consider the potential benefit of maintaining compressed data on–chip, increasing the effective size of the L2 and thereby reducing memory demand. We evaluate the potential for such an optimization by compressing all immutable knowledge base data (the bulk of the knowledge base is immutable, eliminating the need for run–time re–compression) with a basic LZ scheme, and assuming no decompression penalty. In reality, this is highly optimistic not only because of the lack of decompression penalty, but because random access constraints make LZ an overly generous compression algorithm for this data.

Despite this optimistic approach (we achieve 3x compression on average), we find no benefit to maintaining compressed data on–chip. Further analysis shows that knowledge base data elements tend to be multiple cache lines in size on the original system, and remain multiple (if fewer) cache lines when compressed. Thus, comparable numbers of cache misses will occur regardless of compression. By contrast, we find that substantial (10's of cycles) decompression overhead can be incorporated into the lower levels of the memory system with very little effect on overall performance due to the nature of this parallel architecture. Though on–chip decompression does reduce the memory bandwidth demand, this effect is minimal as immutable data make up less than a third of overall traffic on the bus. Thus, while not leading to performance improvements, compression can be used to mitigate large knowledge base sizes with no negative effects.

5. MEMORY SYSTEM COMPONENTS

We now begin a consideration of off–chip components of

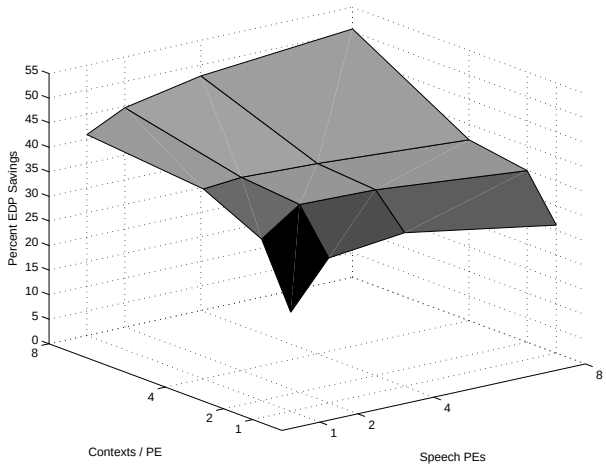

Figure 4: Relative EDP reduction of systems with 128K L2 cache with SDRAM memory over those with DDR memory and no L2 cache.

the memory system. We evaluate direct bandwidth improvement by moving to a DDR based system, and shifting of references through stream partitioning and use of Flash and ROM technology to reduce energy consumption. Clearly, moving from a 100MHz SDRAM system to a 200MHZ DDR based system will increase available memory bandwidth, not only through increased clock rate, but also increased banking (2x). Data from Micron Technologies suggests, however, that DDR memory dissipates power at a higher rate. We find that, while performance improves with a DDR system, a L2 based system achieves better performance improvement at lower power. This is best depicted in Figure 4 which considers the EDP reduction of a L2/SDRAM based system over a no–L2/DDR based system. Obviously, combining a DDR memory and an L2 cache will lead to further performance improvements. We find, however, that due to the added DDR power dissipation, there remains no EDP benefit over comparable L2 only systems.

Flash and ROM

The fact that a large fraction of current and likely future speech recognition knowledge bases is in fact immutable at runtime opens up the possibility of utilizing low power, read–only data structures for their storage. We consider utilizing low–power Flash and ROM components, which dissipate far less overall power than DRAM, but also tend to have longer access latencies. Our analysis suggests that this added latency is far less of a problem than the bandwidth limitations of a single point of access for all static knowledge base data. Implementing the flash or ROM system as a multi–banked or multi–chip solution with the ability to manage overlapping requests internally (even given data transfer constraints along the bus) leads to performance comparable to standard SDRAM systems with the potential for substantial reductions in energy dissipation. Such high performance flash type technologies to support multimedia applications are just beginning to enter the commercial market as demand for such capabilities increases.

Embedded DRAM

We further consider the use of recent advances in embedded DRAM technology to place significant amounts of DRAM on–chip. Rather than simply use this to increase the size of the L2, we consider partitioning the data stream to place mutable program data (a small component relative to immutable knowledge base data, around 2MB for our knowledge bases) in partitioned embedded DRAM, giving each pipeline exclusive access to data that it is responsible for. Accessing embedded DRAM requires longer latency than accessing the L1 cache, leading to lower performance on systems with little parallelism. Further, we discover that a L2 cache is still very useful to capture shared program data and metadata. A system with 2MB of embedded DRAM and a 128K L2 cache, however, is able to achieve 20–30% higher performance than without the embedded DRAM. The energy dissipation of embedded DRAM, however, is larger than SRAM, and connecting embedded DRAM size to mutable knowledge base size may be difficult in a still evolving field.

6. DESIGN SPACE ANALYSIS

We consider the general design space tradeoffs against a cost metric in Figure 5. This figure depicts the EDP of various configurations, 2–8 processors and 2–8 contexts,s L2 cache sizes, and embedded DRAMs against an estimate of on die area. The die area estimate is generated from approximate areas of individual components. Off–chip systems (such as Flash and ROM) are entirely orthogonal to this analysis (assuming sufficient bandwidth), and would serve to uniformly reduce the EDP of all configurations. Given the choice of axis for this graph, it is possible to plot a optimal contour, representing potential configuration points along this design space that provide the best EDP/area ratio for a given tradeoff decision. An analysis of configurations defining this optimal contour tend to include small (4 processing element) systems with small L2 cache configurations.

Consideration of this graphic provides a number of useful insights. We see that excessive caching beyond program metadata does not lead to a favorable tradeoff, and that performance benefits from large pipeline counts are equally ineffective in this regard. Indeed, 4 processor pipelines seems to match memory system bandwidth quite effectively in the general case. While slight EDP improvements are seen with embedded DRAM systems, a substantial increase in chip area is also required. Further details on sub–optimal configurations are available in cited work [10].

7. SUMMARY AND CONCLUSIONS

This paper performs a design space exploration of the memory architecture necessary to support real–time, low–power speech recognition on hand–held devices. We begin by introducing an architectural model to exploit the thread level concurrency found in this domain, focusing the bottleneck on memory system performance. Our analysis leads to a number of insights:

- **Cache Metadata** - While program knowledge base data generally demonstrates low locality and reference predictability, metadata and data elements currently under evaluation show significant caching opportunities. Furthermore, the size of this data component is unlikely to vary as dramatically as knowledge base

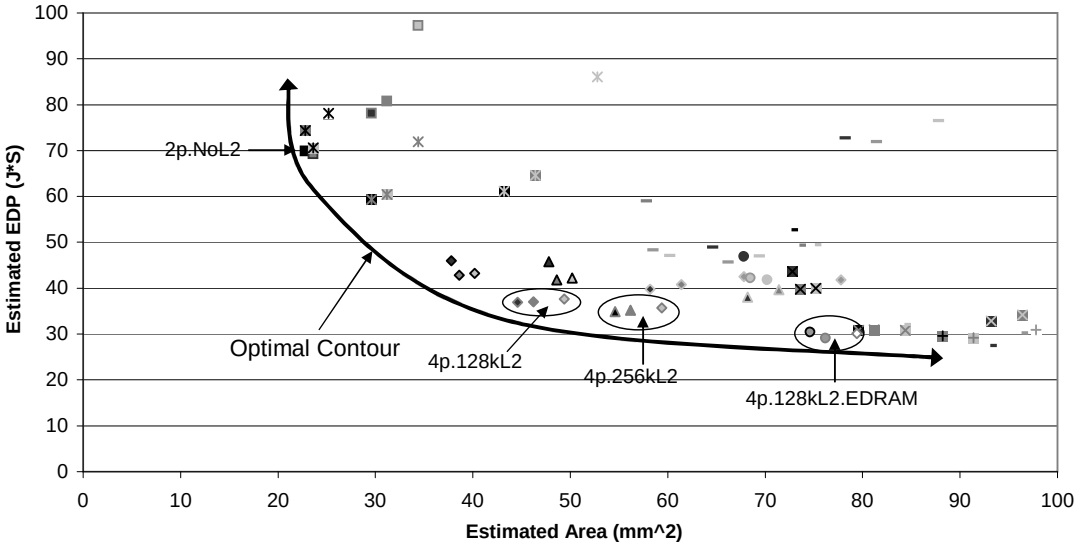

Figure 5: General tradeoff analysis of a number of memory system variants, considering EDP and estimated on–chip area. Solutions falling along the optimal contour (labeled np.mk for n processors, m L2 size, and potentially 2MB of embedded DRAM) tend to fit smaller parallel configurations with L2 caches matching program metadata size.

data. We find that in this reference domain, caching is a far more power efficient approach to improving performance than a more sophisticated memory system (such as DDR).

- **Compress in Memory System** - Compression of static knowledge base data in the memory system can be done with minimum performance penalty. Maintaining compressed data is the L2 does not provide substantial performance benefits, though decompressing on–chip may provide bandwidth benefits.
- **High Bandwidth Flash/ROM** - Use of a flash or ROM system for static knowledge base data has the potential to dramatically reduce energy consumption *if* such a device is banked or otherwise modified to provide substantially higher bandwidth than current implementations generally offer.

Overall, we find that a 4 processor, 4 context configuration with 2K L1 caches per processing element a 128K shared L2 cache, and a standard 100MHz SDRAM system is able to achieve 3-4x performance over unparallelized code running the same workload on an XScale processor. This performance leads to both EDP and workload energy reductions, but added hardware results in a approximately 2x increase in instantaneous power consumption. Given the performance gain, this seems a reasonable tradeoff.

8. ACKNOWLEDGMENTS

We would like to thank the reviewers for their valuable comments. This work is supported under the DARPA / MARCO GSRC and the NSF, grant number CSA-0310511 and ITR grant CCR-0325898. Equipment support was provided by Intel.

9. REFERENCES

[1] K. Agaram et al. A characterization of speech recognition on modern computer systems. In *Proc. of 4th Workshop on Workload Characterization*, December 2001.

[2] T. Austin et al. Simplescalar: An infrastructure for computer system modeling. *IEEE Computer*, Feb 2002.

[3] C.Lai et al. Performance analysis of speech recognition software. In *Proc. of 5th CAECW*, February 2002.

[4] L. Clark et al. An Embedded 32-b Microprocessor Core for Low-Power and High-Performance Applications. *IEEE Journal of Solid–State Circuits*, 36(11), November 2001.

[5] P. Clarkson and R. Rosenfeld. Statistical language modeling using the CMU-Cambridge toolkit. In *EUROSPEECH'97*, pages 2707–2710, 1997.

[6] C.Zhang, F. Vahid, and W. Najjar. A Highly–Configurable Cache Architecture for Embedded Systems. In *ISCA 2003*.

[7] B. Falsafi and D. A. Wood. Parallel dispatch queue: A queue-based programming abstraction to parallelize fine-grain communication protocols. In *HPCA*, 1999.

[8] W. Fornaciari et al. Power Estimation for Architectural Exploration of HW/SW Communication on System-Level Buses. In *7th CODES Workshop*, 1999.

[9] X. Huang et al. The SPHINX-II speech recognition system. *Computer Speech and Language*, 7(2):137–148, 1993.

[10] R. Krishna. *Hardware Architectures to Support Low Power Natural I/O Applications*. PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 2004.

[11] B. Mathew et al. A low-power accelerator for the sphinx 3 speech recognition system. In *CASES 2003*.

[12] J. Mulder et al. An Area Model for On-Chip Memories and its Applications. *IEEE Journal of Solid–State Circuits*, 26(2), February 1991.

[13] M. Ravishankar. *Efficient Algorithms for Speech Recognition*. PhD thesis, Computer Science Department, CMU, May 1996.

[14] P. Shivakumar and N. Jouppi. CACTI 3.0: An integrated cache timing, power, and area model. Technical report, August 2000.

[15] D. Wang and B. Jacobs. MASE DRAM memory simulator manual. http://www.ece.umd.edu/courses/enee759h.S2003 /references/mase_dram.pdf.

[16] B. Yang and L. Kim. A Low–Power ROM using charge recycling and charge sharing techniques. *IEEE Journal of Solid–State Circuits*, 38(4), April 2003.