

# Transparent Data-Memory Organizations for Digital Signal Processors

Sadagopan Srinivasan, Vinodh Cuppu, and Bruce Jacob

Dept. of Electrical & Computer Engineering  
University of Maryland at College Park  
College Park, MD 20742  
<http://www.ece.umd.edu/~blj/embedded/>  
{sgopan,ramvinod,blj}@eng.umd.edu

## ABSTRACT

Today's digital signal processors (DSPs), unlike general-purpose processors, use a non-uniform addressing model in which the primary components of the memory system—the DRAM and dual tagless SRAMs—are referenced through completely separate segments of the address space. The recent trend of programming DSPs in high-level languages instead of assembly code has exposed this memory model as a potential weakness, as the model makes for a poor compiler target. In many of today's high-performance DSPs this non-uniform model is being replaced by a uniform model—a transparent organization like that of most general-purpose systems, in which all memory structures share the same address space as the DRAM system.

In such a memory organization, one must replace the DSP's tagless SRAMs with something resembling a general-purpose cache. This study investigates the performance of a range of traditional and slightly non-traditional cache organizations for a high-performance DSP, the Texas Instruments 'C6000 VLIW DSP. The traditional cache organizations range from a fraction of a kilobyte to several kilobytes; they approach the SRAM performance and, for some benchmarks, beat it. In the non-traditional cache organizations, rather than simply adding tags to the large on-chip SRAM structure, we take advantage of the relatively regular memory access behavior of most DSP applications and replace the tagless SRAM with a near-traditional cache that uses a very small number of wide blocks. This performs similarly to the traditional caches but uses less storage. In general, we find that one can achieve nearly the same performance as a tagless SRAM while using a much smaller footprint.

## 1. INTRODUCTION

Traditional memory organizations for DSPs include dual tagless SRAMs to hold important data and DRAM to hold everything else.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
In *Proc. Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'01)*, November 16-17, 2001, Atlanta, Georgia, USA.  
Copyright © 2001 ACM 1-58113-399-5/01/0011...\$5.00.

The dual-channel SRAM design provides a large amount of fast storage while supporting guaranteed single-cycle access to both operands of a typical instruction (e.g. the product terms of a multiply-accumulate) [Lapsley et al. 1994]. The SRAMs are tagless because they are not transparent to the programmer: DSPs offer segmented memory spaces where different physical memory structures (SRAM0, SRAM1, DRAM, ROM, etc.) are explicitly indicated by corresponding sets of data addresses.

The difference is best served by illustration. Figure 1 shows two separate organizations; on the left is a DSP-style tagless SRAM, on the right is a traditional cache. The tagless SRAM is in a separate namespace from the primary memory system; it is *non-transparent* in that a program addresses it explicitly. A datum is brought into the SRAM by an explicit move that does not destroy the original copy. Therefore, two equal versions of the data remain—there is no attempt by the hardware to keep the versions consistent (ensure that they always have the same value) because the semantics of the mechanism suggest that the two copies are not, in fact, copies of the same datum but instead two independent data. If they are to remain consistent, it is up to software. By contrast, the traditional cache uses the same namespace as the primary memory system; it is transparent in that a program addresses main memory to access the cache—a program does not explicitly access the cache or even need to know that the cache exists.

How each of the mechanisms fits into the CPU's memory model is also shown in Figure 2. A general-purpose memory model (using physically indexed caches) has a single namespace that is shared by all memory structures. A DSP memory model explicitly places the system's memory structures at specific disjunct locations in the namespace.

Because they behave differently, traditional caches and tagless SRAMs have different implementations and different ramifications. Traditional caches, because they could potentially hold anything in the space, require tags to indicate which data is currently cached. Tagless SRAMs require no tags. Traditional caches are managed transparently: if a data item is touched, it is brought into the cache. Tagless SRAMs require explicit management: nothing is brought into the SRAM unless it is done so intentionally in software through a load and store. The flip-side of this is interesting: whereas any reference can potentially cause important information to be removed from a traditional cache, data is never removed from a tagless SRAM unless it is done so intentionally. This is why tagless SRAMs have better real-time behavior.

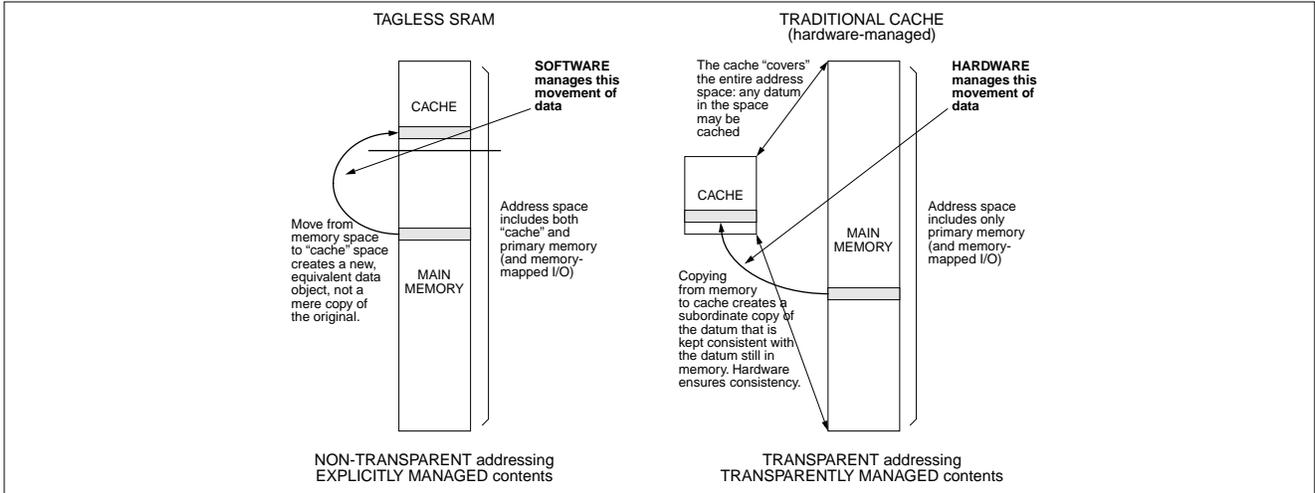


Figure 1: Transparent caches versus tagless SRAMs.

However, the addressing of tagless SRAMs makes them difficult to work with: Contiguity in the address space must be preserved, and distance between objects must be preserved. If two items are adjacent, they must remain so if one of them is to be brought into the SRAM. For example, one could not place into a tagless SRAM only the odd elements of an array without wasting the SRAM storage for all the elements in between. Furthermore, distance between objects must be preserved: The fields of a C structure are referenced by their relative distances, and two fields that are  $n$  bytes apart in the executable file must be  $n$  bytes apart in the tagless SRAM as well. By contrast, it is possible to cache every other element in an array if one uses traditional caches and a uniform memory space (and the element size is large enough), and two items that are  $n$  bytes apart in the executable file need not be  $n$  bytes apart in the cache (in fact, they usually are not, if the data structure is large enough).

Not surprisingly, the industry has decided that the tagless SRAM model is quite difficult to compile for. This is not good, as the trend is increasingly toward programming all embedded systems in high-

level languages. The solution is to provide DSPs with memory structures that use a uniform address space resembling traditional general-purpose caches. We note that several of the newest DSP architectures (for example, the 'C6000 from Texas Instruments [Turley 1997] and the StarCore from Lucent and Motorola [Wolf & Bier 1998]) are moving to a unified memory model to make their architectures more compiler-friendly.

Therefore it is worth investigating different cache organizations for DSPs, as their applications tend to behave slightly different than traditional general-purpose applications. This study looks at traditional cache organizations, and it also looks at a slight modification that exploits some of the behavior seen in DSP applications: One might be able to make good use of large cache blocks, in particular block sizes where a general-purpose application might exhibit high degrees of cache pollution. An obvious downside of doing so is the long fill-time for large blocks. One trend that we can exploit is the appearance of new DRAM architectures that provide an abundance of bandwidth, such as Enhanced SDRAM [ESDRAM 1998], Ram-

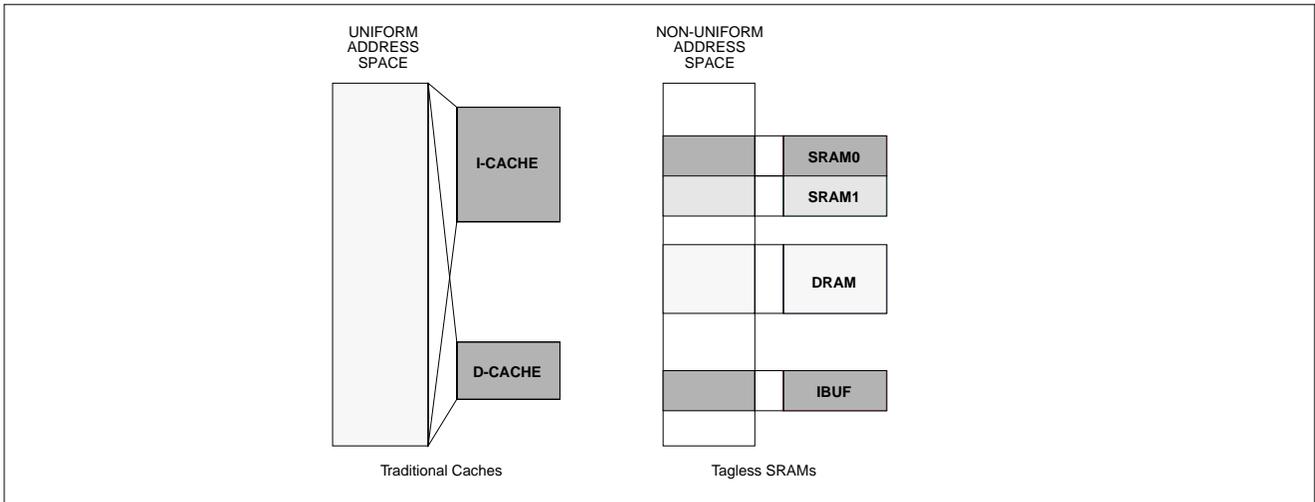


Figure 2: Traditional caches in a uniform space versus tagless SRAMs in a non-uniform space.

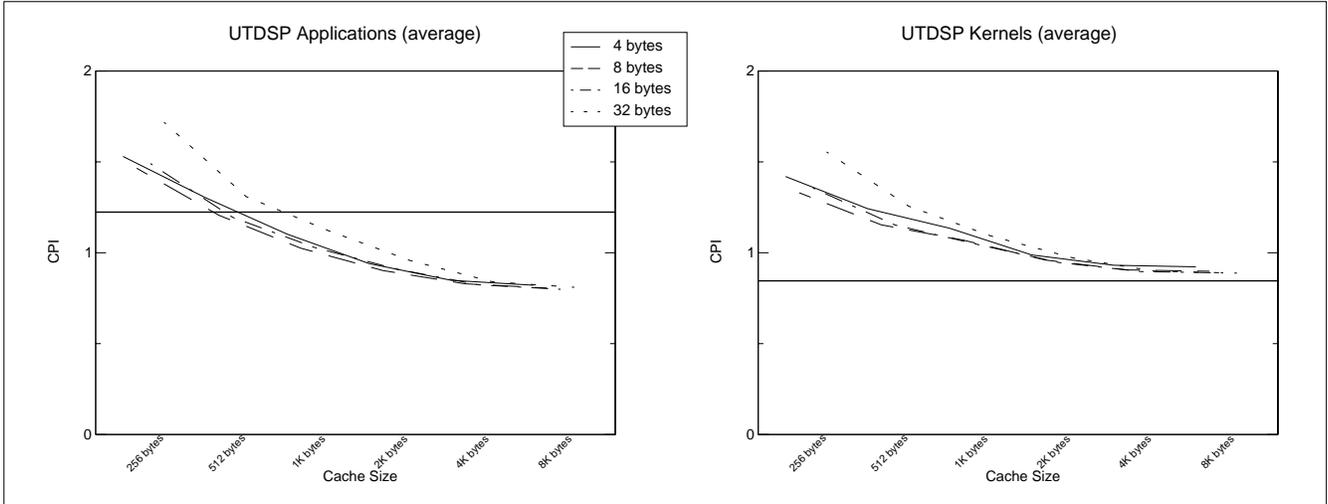


Figure 3: Performance of traditional cache organizations.

bus [Rambus 1998a], and Direct Rambus [Rambus 1998b]. These architectures are improvements over the traditional DRAM architecture; some are newer than others, and the newest members of the set reduce bandwidth overhead by a factor of four compared to the oldest members [Cuppu et al. 1999, Cuppu et al. 2001].

We look at the performance side of the trade-off between reducing the size of the on-chip SRAM array and increasing the bandwidth to the off-chip DRAM array. We find that a small associative on-chip cache can approach and even beat the performance of a large tagless SRAM, and one can meet SRAM performance with as little as 64 total bytes of storage provided one uses an appropriate management policy. Moreover, the performance levels are possible without enormous increases in off-chip memory bandwidth.

## 2. EXPERIMENTAL SET-UP & RESULTS

For our baseline DSP architecture, we model a Texas Instruments TMS320C6000 signal processor. This is an 8-wide VLIW design that both provides a good compiler target and enables single-cycle throughput for multiply-accumulate operations. The DSP core is a load/store architecture and has two load/store units, identified as belonging to side ‘1’ or side ‘2’ of a 2-way clustered design (the load-store units are called .D1 and .D2 in ‘C6000 terminology’). Because it is a load/store architecture, all logic and arithmetic operations are performed through the register file. We use the data-memory system of the ‘C6201 processor. The data-memory side is a 64KB tagless SRAM organized as pseudo-dual-ported with eight independent banks. For this study, we model the instruction memory as perfect. The off-chip memory bus for the baseline DSP is a 400MB/s bus, 32 bits wide, roughly equal to half a PC100 channel.

For our range of traditional cache models, we look at sizes of 256 bytes (one fourth kilobyte) up to 8 Kbytes, with block sizes ranging from 4 bytes to 32 bytes. The caches are pseudo-multiported, using 4 independent banks. The caches are 4-way set associative.

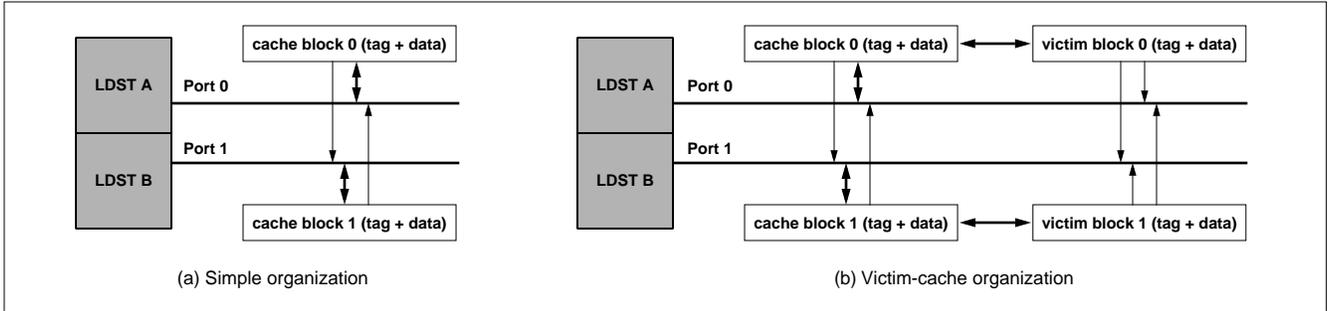
For an alternative on-chip memory structure we chose a small fully associative cache (2 blocks) with block sizes ranging from 4 bytes to 512 bytes. The organization differs from a traditional cache

in its replacement policy: The cache blocks are divided into two partitions, where each partition is dedicated to holding data requested by either the A-side or B-side load/store unit. Though a load/store unit can read from any block, it can only write into half the cache. The write policy is write-allocate/fetch-on-write-miss. We ran the off-chip bus at 400, 800, 1200, and 1600 MB/s.

The benchmarks are taken from the UTDSP benchmark suite (see <http://www.eecg.toronto.edu/~corinna/DSP/>) and include both DSP kernels (e.g. fast Fourier transform, FIR and IIR filters, matrix multiply, etc.) and DSP applications (e.g. linear predictive coding, image compression, an implementation of the G.721 modem protocol, etc.). They are compiled for our DSP simulator using the Texas Instruments’ C6000 C compiler, which has been shown to produce code for the C6000 that (according to studies from TI) performs within 90% of hand-written assembly code.

Figure 3 shows the performance results for the traditional cache organizations, using an 800MB/s memory bus, which performs up to 20% better than a 400MB/s bus. The baseline performance of the SRAM model is also given, and the benchmarks are presented as averages for the ‘applications’ as well as the ‘kernels’ in the benchmark suite. The 800 MB/s bus (as opposed to the SRAM model’s 400 MB/s bus) accounts for part of the reason that the traditional cache ‘beats’ the tagless SRAM design on the Application benchmarks. The rest of the reason is that the data footprint of many applications is just large enough to exceed the 64K data-side SRAM. Because the compiler statically assigns data to either SRAM or DRAM, the SRAM model does not take advantage of moments of significant temporal locality, the way that a traditional cache does. The graphs show that a modest-sized cache can approach and even beat the performance of the SRAM model.

For alternative cache models, we look at two different organizations, illustrated in Figure 4. The first is the very simple organization described above (two blocks, each one dedicated to a different load/store unit); the second adds a single-block victim cache to each of the blocks in the first organization—in this organization, each victim-cache block holds only victims taken from its associated cache block. Note that in all organizations, cache reads may be satisfied out



**Figure 4: Slightly non-traditional cache organizations modeled in this study.** Each load/store unit can read from any block in the cache, but it can only write to the primary block that is assigned to it. Similarly, load misses only fill the cache block assigned to a load/store unit. The victim-cache organization adds two blocks to the cache that are only filled by blocks replaced from the corresponding primary blocks.

of any block in the cache: The tying of a block to a particular load/store unit is simply used in the replacement policy; it does not affect the cache lookup mechanism. Any load/store unit can read a value from any block in the cache, which makes the cache effectively fully associative—this is why we look only at small numbers of blocks in the cache.

Figure 5 gives the performance of the “simple” organization. Performance does not really approach that of the SRAM model, though it is surprisingly not far off, given the very small amount of storage being used.

The last study looks at the slightly more advanced victim-cache organization. Figure 6 gives the performance. The graphs show that the organization does significantly better than the “simple” organization and in fact approaches and, in some cases beats, the performance of the SRAM model. For the same amount of on-chip storage (designs up to 512 total bytes of memory), it can even beat the performance of the traditional cache model.

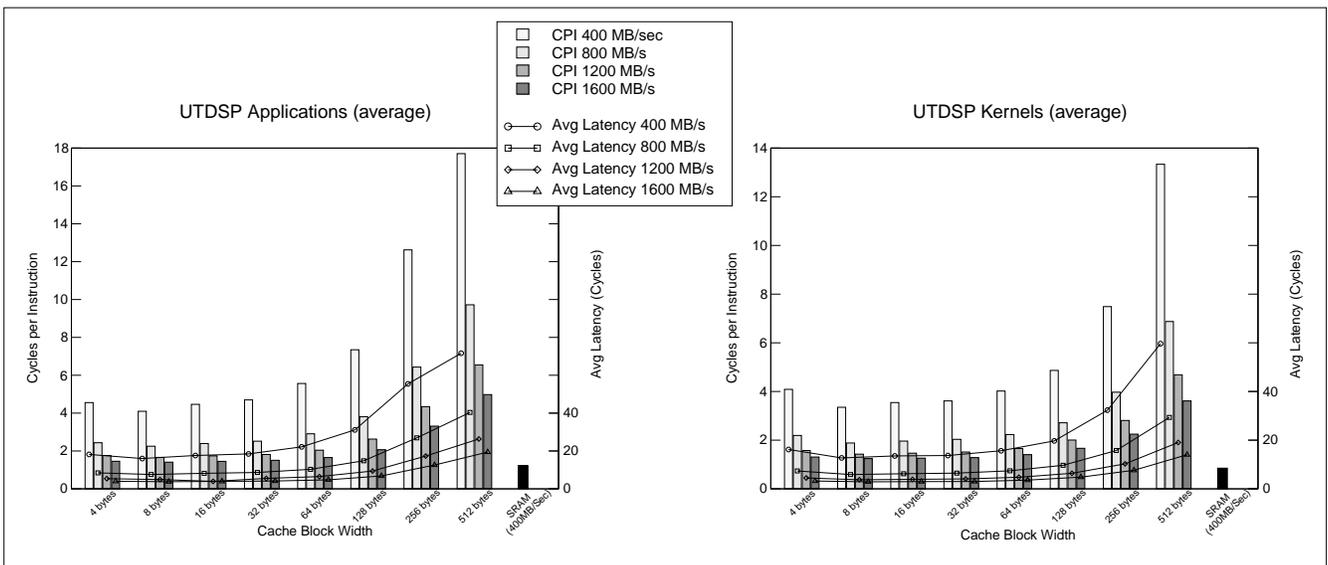
In general, though the traditional cache performs better than all other schemes on some benchmarks and at larger sizes (8 Kbytes), the “victim-cache” buffer organization can approach the same performance as these other models while using significantly less on-chip

storage: The best organization, with 64-byte buffers, uses a total of 256 bytes of storage. However, to approach the performance of the better schemes, it does require a significantly higher bandwidth to main memory. This represents a trade-off that a DSP system engineer would need to make: System cost in higher DRAM bandwidth versus CPU cost of a larger DSP core (more on-chip storage).

### 3. CONCLUSIONS

We look at several different traditional cache organizations and variations thereupon to be used in the new breed of DSPs that use a transparent memory model. We find that relatively small amounts of on-chip storage are required to approach and, in some cases, exceed the performance of a DSP’s non-uniform memory model that makes extensive use of tagless SRAMs.

We also investigate an alternative cache organization in which a small number of large cache blocks are used, and we find that it presents an interesting trade-off between chip costs, in the form of on-chip storage, and system costs, in the form of memory bandwidth. Like the traditional cache organization, the alternative cache organization approaches and, in some cases, exceeds the performance of



**Figure 5: Performance of “simple” organization.**

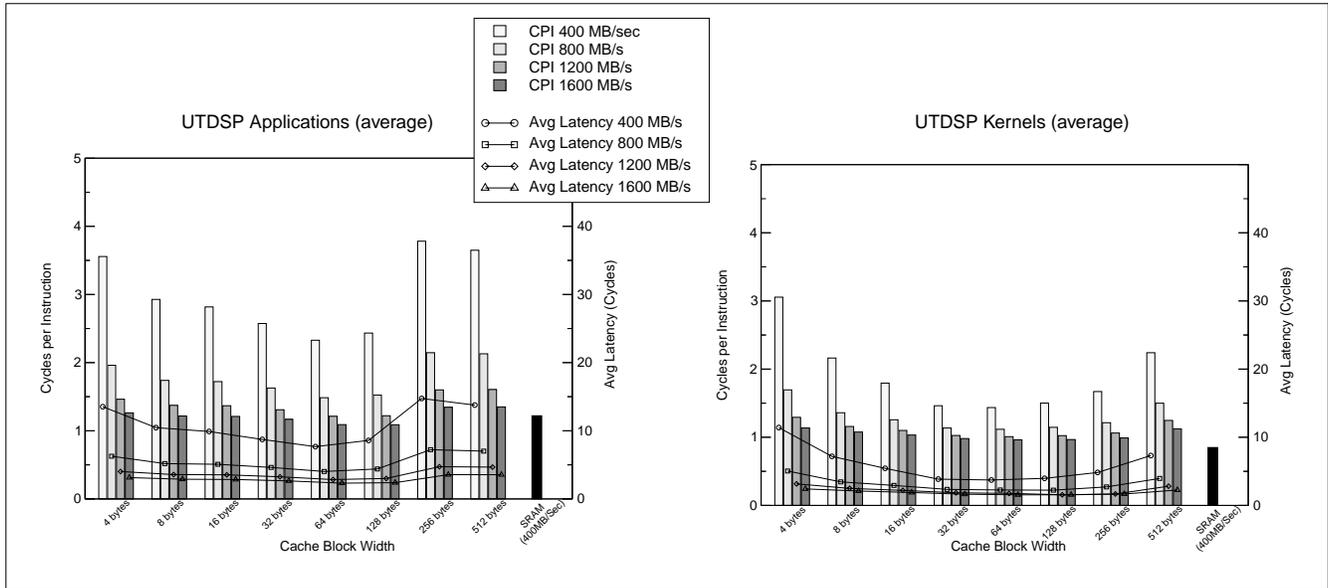


Figure 6: Performance of the “victim-cache” organization.

the DSP using large tagless SRAMs. In general, if one is willing to pay for more bandwidth, one can get away with significantly less on-chip storage and reach the same levels of performance. That one can approach the performance of the traditional DSP memory organization with such a simple structure is likely due to the fact that many DSP applications stream data, obviating large cache structures and making stream buffers more attractive.

## ACKNOWLEDGMENTS

We thank Corinna Lee for the use of her UTDSP benchmark suite (see <http://www.eecg.toronto.edu/~corinna/DSP/>), a large collection of DSP programs ranging from small kernels to full-blown signal-processing applications.

The work of Sadagopan Srinivasan was supported in part by NSF CAREER Award CCR-9983618 and NSF grant EIA-0000439. The work of Vinodh Cuppu was supported in part by NSF CAREER Award CCR-9983618. The work of Bruce Jacob was supported in part by NSF CAREER Award CCR-9983618, NSF grant EIA-9806645, NSF grant EIA-0000439, DOD award AFOSR-F496200110374, and by Compaq and IBM.

## REFERENCES

V. Cuppu, B. Jacob, B. Davis, and T. M. Mudge. “A performance comparison of contemporary DRAM architectures.” In *Proc. 26th*

*Annual International Symposium on Computer Architecture (ISCA-26)*, Atlanta GA, May 1999, pp. 222–233.

V. Cuppu, B. Jacob, B. Davis, and T. Mudge. “High performance DRAMs in workstation environments.” *IEEE Transactions on Computers*, vol. 50 no. 11. November 2001. (TC Special Issue on High-Performance Memory Systems)

ESDRAM. Enhanced SDRAM 1M x 16. Enhanced Memory Systems, Inc., [http://www.edram.com/products/datasheets/16M\\_esdram0298a.pdf](http://www.edram.com/products/datasheets/16M_esdram0298a.pdf), 1998.

P. Lapsley, J. Bier, A. Shoham, and E. A. Lee. *DSP Processor Fundamentals*. Berkeley Design Technology, Inc., 1994.

Rambus. 16/18Mbit & 64/72Mbit Concurrent RDRAM Data Sheet. Rambus, <http://www.rambus.com/docs/Cnctds.pdf>, 1998.

Rambus. Direct RDRAM 64/72-Mbit Data Sheet. Rambus, <http://www.rambus.com/docs/64dDDS.pdf>, 1998.

Texas Instruments. *TMS320C6000 peripherals reference guide*, TI Inc., <http://www-s.ti.com/sc/psheets/spru190d/spru190d.pdf>.

J. Turley. 1997. “TI’s new ‘C6x DSP screams at 1,600 MIPS.” *Microprocessor Report*, 11(2):14–18.

O. Wolf and J. Bier. 1998. “StarCore launches first architecture: Lucent and Motorola disclose new VLIW-based approach.” *Microprocessor Report*, 12(14):22–55.