Circuit Design of a Dual-Versioning L1 Data Cache for Optimistic Concurrency

Azam Seyedi[†‡] Adrià Armejach[†‡]
Adrián Cristal[†◇] Osman S. Unsal[†] Ibrahim Hur[†] Mateo Valero[†‡]
[†]BSC-Microsoft Research Centre [‡]Universitat Politècnica de Catalunya
[◇]IIIA - Artificial Intelligence Research Institute CSIC - Spanish National Research Council
{azam.seyedi, adria.armejach, adrian.cristal, osman.unsal, ihur, mateo.valero}@bsc.es

ABSTRACT

This paper proposes a novel L1 data cache design with dual-versioning SRAM cells (dvSRAM) for chip multi-processors (CMP) that implement optimistic concurrency proposals. In this new cache architecture, each dvSRAM cell has two cells, a main cell and a secondary cell, which keep two versions of the same data. These values can be accessed, modified, moved back and forth between the main and secondary cells within the access time of the cache. We design and simulate a 32-KB dual-versioning L1 data cache with 45-nm CMOS technology at 2GHz processor frequency and 1V supply voltage, which we describe in detail. We also introduce three well-known use cases that make use of optimistic concurrency execution and that can benefit from our proposed design. Moreover, we evaluate one of the use cases to show the impact of the dual-versioning cell in both performance and energy consumption. Our experiments show that large speedups can be achieved with acceptable overall energy dissipation.

Categories and Subject Descriptors

B.3.2 [**Memory Structure**]: Design Styles

General Terms

Performance, Design, Experimentation, Verification

Keywords

Data cache design, optimistic concurrency, parallelism

1. INTRODUCTION

Tremendous progress in architecture has made chip multi-processors increasingly common. To benefit from the additional performance offered by these multi-core chips, various novel architecture implementations have been proposed, such as speculative multithreading [10], lock elision [13], or hardware transactional memory [9]. All of these proposals leverage optimistic concurrency, by assuming that conflicting data accesses will not occur; in case a conflict occurs all tentative data updates have to be undone. Thus, multiple versions of the same data have to be maintained by the L1 data cache. However, the circuit design of such a cache which supports all these different proposals has not been proposed yet.

In Section 2 we introduce a detailed circuit design of a novel 32-KB dual-versioning SRAM (dvSRAM) L1 data cache with 4-way set associativity, 64B data lines, 2 clock cycle access time and 45-nm Predictive Technology Model [1] at 2 GHz processor frequency and 1V supply. The main characteristic of the dvSRAM cache is the inclusion of two bits in each cell, primary value and secondary value. These two values can be accessed separately or synchronously, modified and exchanged within the cache access time using defined operations supported by the cache. We simulate both the dvSRAM and the typical SRAM array with Hspice 2003, design the layouts [2], and calculate dynamic and static power consumptions and access times for all the operations.

In Section 3 we discuss how the aforementioned optimistic concurrency based systems can benefit from the efficient dual-versioning provided by the dvSRAM. Moreover, we evaluate one of the systems using state-of-the-art baseline and benchmarking suite and show that significant speedups are achieved with an acceptable overall energy consumption.

In Section 4 we disscus the related work, and finally we conclude in Section 5.

2. dvSRAM DESIGN DETAILS

In this section, we start with dvSRAM cell design. We describe the cell circuit, and we introduce the available operations of the dvSRAM array structure. Then we describe the different blocks that form the dvSRAM array structure in detail.

2.1 Dual-Versioning Cell Design

Figure 1 depicts the structure of our proposed dvSRAM cell, which is composed of two typical standard 6T SRAM cells [12]: the main cell and the secondary cell. Each of them keeps different versions of the same data. These two cells are connected via two exchange circuits. The idea of exchange circuit design is based on utilizing tri-state inverters [12] and transistor stacks [16]. We want to completely separate the main and the secondary cells from each other. The only time they are not isolated is when the exchange circuits are active, and the data of the main cell stores to the secondary cell, or the data of the secondary cell restores to the main cell. When these exchange circuits are not active and signals dvRstr and dvStr are low; more than one 'off' transistor in the nMOS or pMOS stacks in the exchange circuits leads to a significant reduction in leakage current [16].

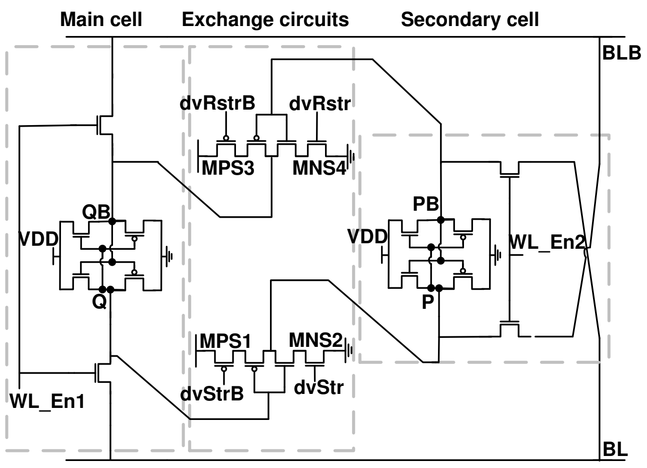

Figure 1: Circuit schematic of the proposed dvSRAM cell: a main cell with a secondary cell and exchange circuits.

Operation	Description
Write	Writing to a main cell by activating WL_En1
Read	Reading from a main cell by activating WL_En1
dvStr	∼Q→P: Storing the main cell to the secondary cell by activating dvStr
dvRstr	∼PB→QB: Restoring the secondary cell to the main cell by activating dvRstr
dvWrite	Writing to the secondary cell by activating WL_En2
dvRead	Reading from the secondary cell by activating WL_En2
dvBWrite	Writing to both cells simultaneously by activating WL_En1 and WL_En2
dvStrAll	Storing all main cells to their secondary cells simultaneously

Figure 2: Brief description of the dvSRAM cell operations.

In Figure 2, we briefly explain dvSRAM cell operations. Read and Write act like read and write operations in a typical SRAM cell. Other operations are created based on our goal in this paper. When dvStr is high, transistors MPS1 and MNS2 are turned on, and the exchange circuit acts as an inverter by inverting Q to P. The secondary cell keeps the value of P when dvStr is low, and it inverts P to PB, so that PB has the same value as Q. Similarly, when dvRstr is high, PB in the secondary cell is inverted to QB and converted to Q, so that previously saved data in the secondary cell can be recovered. Note that dvStr operation acts just for a line; when we need to store all the cells to their secondary cells simultaneously, we use dvStrAll operation. We use dvRead to read the data from the secondary cell and dvWrite to write to the secondary cell. Finally, we use dvBWrite to write to both main and secondary cells simultaneously. In Section 2.2.2, we describe how these signals are generated.

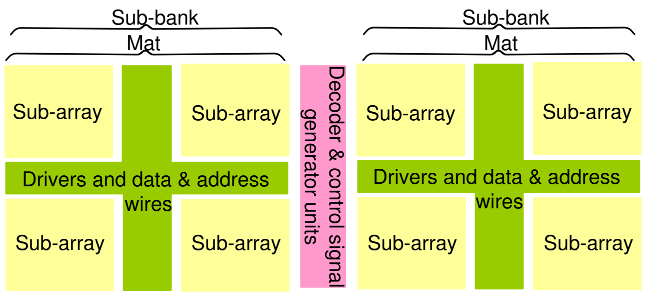
Sub-bank
Mat
Sub-array
Sub-array
Drivers and data & address wires
Sub-array
Sub-array
Decoder & control signal generator units
Sub-bank
Mat
Sub-array
Sub-array
Drivers and data & address wires
Sub-array
Sub-array

Figure 3: dvSRAM array with two sub-banks, one mat in each of them with four identical sub-arrays. Decoder and control signal generator units are in the middle part.

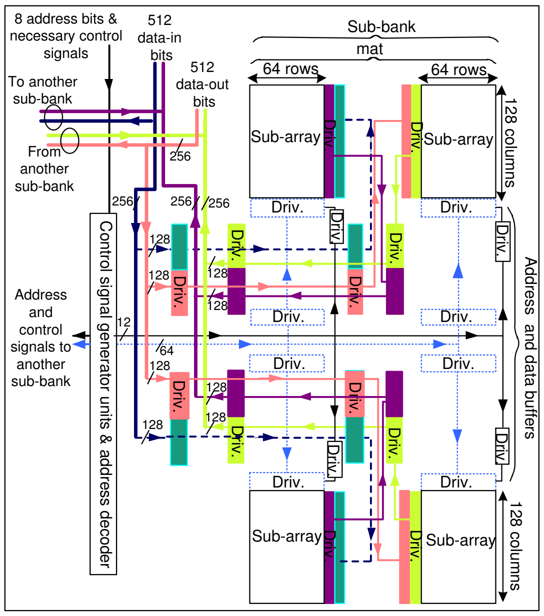

Figure 4: A view of one dvSRAM sub-bank and the middle part of array: four identical sub-arrays, each with 64 rows and 128 columns.

2.2 dvSRAM Array Structure Design

In this subsection we describe the high-level organization of the dvSRAM array, considering each part of its structure.

2.2.1 *Brief description of the whole array structure*

We use Cacti 5 [14] to determine the optimal number and size of the array components. For the L1 data cache configuration we assume 32-KB, 4-way, 64-byte lines, 2 clock cycle access time; and we use the 45-nm Predictive Technology Model (high performance model) [1] with 1V supply voltage. For a one bank array, Cacti suggests two identical sub-banks, one mat for each sub-bank and four sub-arrays in each mat as can be seen in Figure 3. The address decoder and control signal generator units are placed in the middle part of the array and necessary drivers and data and address wires in middle part of each sub-bank.

Figure 4 shows a view of one sub-bank of the dvSRAM array and the middle part, decoder and control signal generator units. Our sub-bank design is based on Cacti sug-

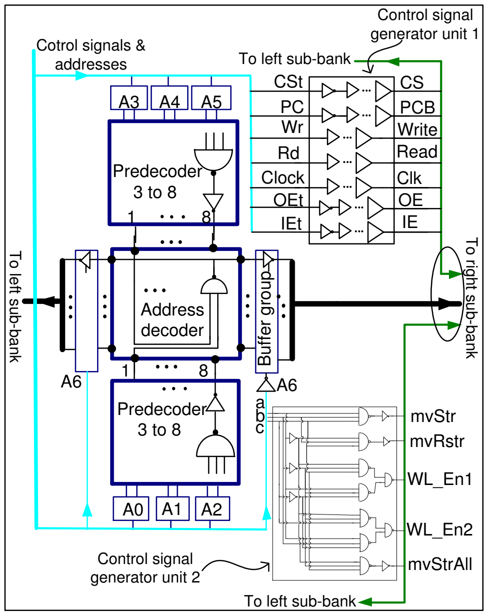

Figure 5: Middle part of dvSRAM array: address decoders, control signal units, and tri-state buffers. Boolean functions: dvStr $= abc$**, dvRstr** $= \bar{a}bc$**, WL_En1** $= a\bar{b}\bar{c} + \bar{a}\bar{b}c$**, WL_En2** $= \bar{a}\bar{b}c + ab\bar{c}$**, dvStrAll** $= \bar{a}\bar{b}\bar{c}$**.**

gestions and the SRAM configuration that Wang et al. [15] proposed. Considering the cache structure of our system, i.e., 32KB size, 64B lines and four-way set associativity, we need seven address bits to address a line in the array, so these seven bits and the necessary control signal bits which we describe later, enter to the up side of Figure 4. During an access, only one of the two sub-banks is activated; hence the numbers of bits that deal with decoder and produce word-line addresses are six; so 64 word-line addresses and 12 control signals enter to the sub-bank from the left side as can be seen in Figure 4. 512 data-in and 512 data-out bits are routed from the up side of the sub-bank. The four identical sub-arrays of a mat are activated during an access, therefore each sub-array holds a part of the cache line, so 128 of the 512 data bits (64B) are distributed to each sub-array. We put necessary optimized drivers (chain of two series inverters) in paths to reach their related loads in the sub-arrays. Each wire and its related drivers have same greyscale in this Figure. We describe the details later in this section.

2.2.2 *Middle part of the array*

Figure 5 depicts the address decoder, control signal units, and tri-state buffers that reside in the center of the dvSRAM array. To generate 64 word-line addresses from six address bits, A0…A5, we design a two-level decoder similar to the design of Amrutur [4]. Two 3-to-8 pre-decoders produce partially decoded products, and the main decoder generates 64 word-line addresses from their outputs. We use the highest value bit, A6 bit, as an enable signal to select the sub-bank which is activated during the access time. 64 word-line addresses are connected to both sub-banks via two tri-state buffer groups with enable signals $A6$ and $\overline{A6}$. Control signal units are responsible to generate the necessary signals for executing the operations and have a synchronous

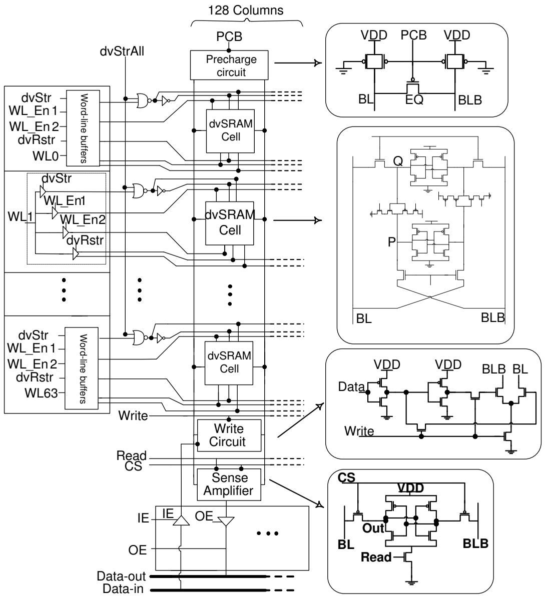

Figure 6: The structure of one sub-array.

flow-thru SRAM [3]. Control signal generator unit 1 consists of some optimized inverter chains that regulate the signals to control the dvSRAM sub-array circuits such that main cell operations, typical read and write can be operated correctly. Control signal generator unit 2 is a 3-to-8 decoder, generating dvStr, dvRstr, WL_En1, WL_En2 and dvStrAll signals according to its input signals level, a, b and c.

2.2.3 *Sub-array structure description*

Figure 6 shows the structure of one sub-array that consists of 2D matrix of memory cells, data-in, data-out, word-line address buffers, and associated peripheral circuitry. On the left side of Figure 6, we can see the word-line address buffers and a set of NOR and inverter gates. Each word-line address enters to its corresponding line via one of four buffers. Control signal generator unit 2 generates signals dvStr, WL_En1, WL_En2, and dvRstr to enable word-line address buffers; it also generates dvStrAll. When dvStrAll is activated, all the main cells of whole sub-array are stored to their secondary cells simultaneously. Data-in and data-out buffers are typical data buffers that isolate a sub-array with interconnected wires and reduce injected noise effects. We use typical precharge [4] and write circuits [17], and sense amplifiers [17]; the control signal generator unit 1 generates signals to control these circuits. Similar to Thoziyoor et al.'s [14] implementation, a layer of multiplexing can be added, before output buffers, at the outputs of sense amplifiers; but, for simplicity, we waive this option, and the signal CS (from control signal generator unit 1, form Figure 5) in the sense amplifiers selects the correct cache set. We show the simulation waveform of dvSRAM for all operations later.

2.2.4 *Data and address signals distribution*

Figure 7 shows the distribution of address, data and control signals, the equivalent wire resistance and capacitance and necessary optimized drivers for one sub-array. 64 lines of word-line addresses, 128 data-in, 128 data-out, and 12 control signal wires are routed to the sub-array from the left side of the Figure. These are long wires with considerable

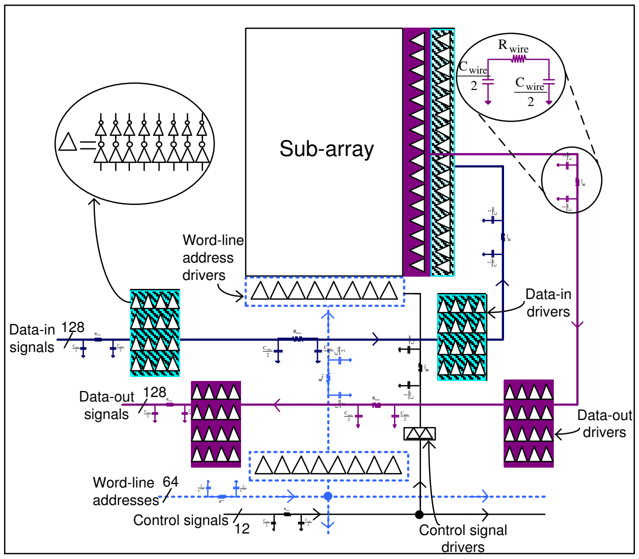

Figure 7: The distribution of address and data drivers in a sub-array. The equivalent resistance and capacitance of each wire is shown in lumped Π model. Each triangle in the driver groups is representing eight drivers (two series inverters).

RC delay constant and big propagation delay. For compensating the voltage drop, reflection effect and for driving big word-line capacitances, we divide the wires to shorter length and put drivers (two series inverters) at the end of each piece of wire. For instance, for each data-in (data-out) wire, we put two stages of drivers in the path to reach each sub-array and one driver just in the sub-array (to drive the bit-line capacitances) as can be seen in this Figure. We optimize the sizes of these drivers by considering resistance and capacitance of wire sections and sub-array circuit sizes. For 64 word-line address and 12 control signal wires, we put one stage of drivers in the path to reach each sub-array and one another driver just in sub-array as can be seen in this Figure.

We can place the drivers of each stage in one line as Cacti suggests [3] but it leads to high area and un-optimized area floor-plan so we use an optimal allocation of each stage drivers that we explain as follows for both data-in and data-out wires. Instead of locating all the drivers (512 drivers) of one stage in one line, we divide them into four groups of 128 drivers, each group for one sub-array and then we put these 128 drivers in four lines and rotate them 90 degree under clockwise as can be seen in Figure 7. Setting word-line address and control signal drivers is easier and we locate them in one line at each stage. In the Figure, for simplicity, all related drivers and wires have the same greyscale.

2.3 Design Methodology and Analysis

We construct, for one array of dvSRAM and one array of a typical SRAM, Hspice transistor level net-lists that include the complete decoder, control signal units, and two sub-banks with necessary drivers and equivalent capacitances and resistances of wires. The high-level structure of a typical SRAM array is the same as of dvSRAM array, but with 6T typical SRAM cells, related word-line address buffers and control signal generator units. We simulate and optimize both the dvSRAM and typical SRAM array with Hspice

Operation	Energy (pJ)		Access time (ps)	
	SRAM	dvSRAM	SRAM	dvSRAM
Read	90.4	112.4	480	616
Write	80.2	99.3	736	832
dvRead	-	114.4	-	374
dvWrite	-	101.1	-	401
dvBWrite	-	122.6	-	370
dvStr	-	114.9	-	670
dvRstr	-	112.9	-	832
dvStrAll	-	1123.2	-	371
Static	35.1	51.4	-	-

Figure 8: Typical SRAM and dvSRAM energy consumption and access time per operation.

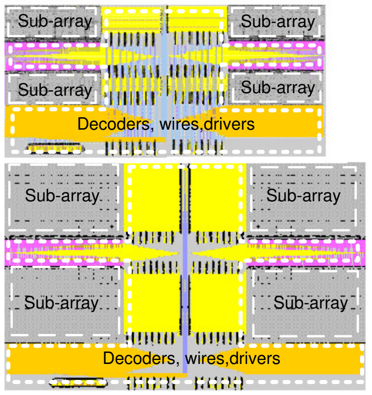

Figure 9: Typical SRAM (top) and dvSRAM (down) layouts. Showing one sub-bank, address decoders.

2003.03 using HP 45-nm Predictive Technology Model for VDD=1V, 2 GHz processor clock frequency and T=25°C. We calculate the access time, dynamic energy and static energy per access for all operations in dvSRAM and SRAM and present it in Figure 8. Our analysis indicates that our dvSRAM design meets, as the typical SRAM, the target access time requirement of two clock cycles and acceptable power and area increase. Figure 9 shows the layouts [2] for both dvSRAM and the typical SRAM arrays. They include one sub-bank, address decoders, address and data wires, and the control signals; the second symmetric sub-bank is omitted due to space constrains. After adding PADs and I/O interfaces [7], the area increase of the dvSRAM compared to the typical SRAM is 46%. The area devoted to L1Ds in state-of-the-art CMPs is small compared to the entire die, for example, in Power 7 [11] it is less than 1%, therefore the dvSRAM area overhead in die size is modest.

Figure 10 shows the simulation waveform for a sequence of eleven operations that take 11ns for the first cell at the last raw of one sub-array. Q63, P63 and Out63 are internal nodes and dvStr63, dvRstr63, WL63_En1 and WL63_En2 are the outputs of the last word-line address buffers. Signals PC, CS, CLK, OE, IE, a, b, c are omitted for space concerns and their behaviors are similar to the signals in a typical SRAM array.

3. USE CASES

In this section we discuss several use cases for the proposed dvSRAM. Moreover, a short evaluation for one of the use cases follows to show the impact of the dvSRAM.

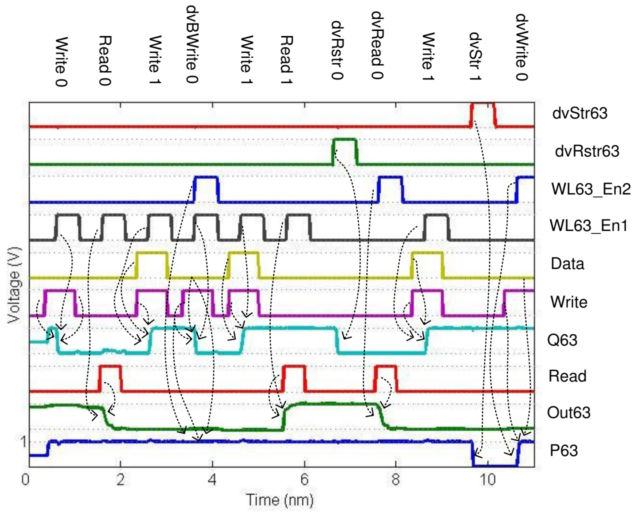

Figure 10: The simulation waveform of dvSRAM array for a sequence of eleven operations for the first cell of the last raw of one sub-array. The operations are shown on top.

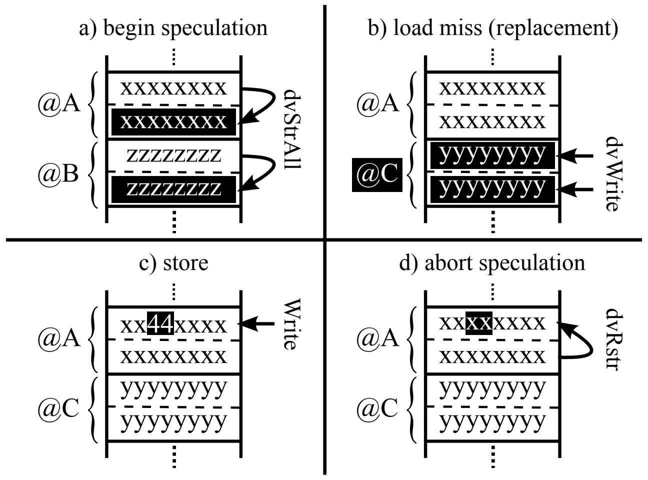

Figure 11: Simple usage example of the dvSRAM in an optimistically concurrent system. Black background indicates state changes.

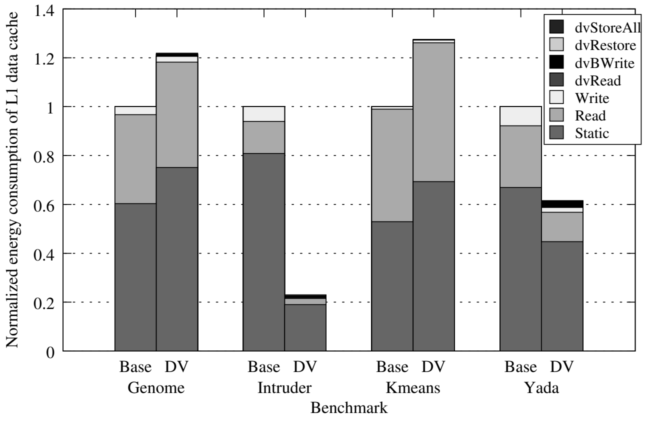

Figure 12: Normalized energy consumption breakdown of L1 data cache.
Base — Baseline-HTM; **DV** — Dual-Versioning-HTM

Speculative Multithreading (SpMT): SpMT [10] is a concurrency mechanism that attempts to speedup sequential executions by partitioning the workload in two threads. The second thread executes the bottom half of the sequential program optimistically. By using a mechanism like the dvSRAM, each thread can use one of the independent cells, providing an easy and efficient versioning management and a fast in-place conflict detection mechanism with state bits.

Transactional Memory (TM): TM [9] is a promising technique that helps with parallel program development by abstracting away the complexity of managing shared data. TM uses optimistic concurrency, assuming that conflicting data accesses will not occur; in case a conflict occurs then one or more transactions must abort, undoing all tentative data updates. This requires a multiversioning mechanism to restore previous state. By using the dvSRAM, a partial snapshot of the past state of the system can be maintained at L1D level, leveraging a fast mechanism to restore previous state, and rely in slower mechanisms only if it is strictly necessary.

Speculative Lock Elision (SLE): In SLE [13] the system tries to avoid waiting in a lock when it might not be necessary by predicting potential non conflicting executions, allowing several threads to execute the same critical section optimistically. Similarly to TM, SLE has to deal with speculative updates and can benefit from the efficient dual-versioning management of the dvSRAM.

In Figure 11 we illustrate how the dvSRAM is used in an optimistically concurrent system like the ones described above. The example shows one possible way to use the dvSRAM, even though others might be more convenient depending on the applicability. As we can see, when an optimistic execution (speculation) starts, the system creates copies in the secondary cells by using the `dvStrAll` operation (Figure 11a). During the speculative section, new lines can be added to the dvSRAM upon a miss (Figure 11b), and modifications are done in the *main cell* (Figure 11c). If the system has to abort due to a conflict, it rolls-back its state by using fast `dvRstr` operations (Figure 11d). A comprehensive proposal of the TM use case is discussed in [5].

3.1 Results: Transactional Memory

State-of-the-art HTM proposals [18] can be adapted to work with the dual-versioning cache. In particular, we evaluate in a full-system simulation environment a state-of-the-art log-based HTM as baseline (labeled *Base*), and a modified version of the baseline that uses our proposed dvSRAM and logs as a fall-back mechanism only for cachelines that overflow the L1D cache (labeled *DV*). We selected four applications of the STAMP [6] benchmark suite, which represent different amounts of contention, to evaluate the dvSRAM TM applicability in both performance and power consumption. In general, highly contended benchmarks scale poorly due to a larger number of aborts. This is the case of Intruder which performs 6.31× better when using the dvSRAM. Yada with moderate contention also shows a significant speedup of 2.24×, while low contention benchmarks like Genome and KMeans show 1.15× and 1.09× speedups respectively. This is achieved by completely avoiding the use of software logs in a large percentage of transactions (over 90% on average).

Regarding power consumption, as can be seen in Figure 12, two applications are less energy efficient compared to a typical SRAM cache, while the other two are more energy efficient due to larger execution time speedups. In addition, for all the applications, the amount of energy spent in specific dual-versioning operations is not significant compared to the usual (Read, Write) operations and the static energy. Note that the energy results are considering only L1 energy consumption, and the L1 data cache accounts for a very small fraction of an entire processor, as discussed in Section 2.3. Thus, the energy impact considering an entire processor would be palliated.

4. RELATED WORK

Ergin et al. [8] proposed similar work using a shadow-cell SRAM design for checkpointed register files. In that technique, each bit-cell has a shadow-copy cell to store the temporal value which can be recovered later. They use two cells, consisting of two back to back inverters, connected to each other using two pass transistors and two inverters. Even when both "check point" and "recover" signals (the enable signals copy the data to the shadow-copy cell and recover it later) are inactive, pMOS transistors or nMOS transistors of these inverters are always "on" leading to power consumption increase. Moreover, if we want to modify this structure for our purpose, we should make upper inverters of cells bigger to mitigate the imbalance of this structure. While the situation is a bit better when we replace the inverters and pass transistors with each other, we still have imbalance problem which cause instability issues. All these problems lead to design a new cell with more capabilities that we can use in L1 data cache for optimistic concurrency. In our dvSRAM structure, both cells, the main cell and the secondary cell are isolated from each other with two exchange circuits and this leads to lower static power consumption. Also, there are no unnecessary 'on' transistors in the active mode. We calculate and compare the static power for our dvSRAM cell and Ergin's proposed cell to prove our discussion. The static power of the dvSRAM cell is 28.35μW compared to the 51.08μW of Ergin's cell [8].

We can use single-electron transistors and implement L1 data cache with multi-valued SRAM [19] which has much smaller area compared to our circuit; but this technique is too complex for our purpose because of complicated input and output circuitries for each cell.

5. CONCLUSIONS

In this paper, we propose a new L1 data cache with dual-versioning SRAM cells (dvSRAM) that aims to overcome the data versioning problem present in optimistic concurrency mechanisms. We present the design details and its available operations. We calculate the power consumption, access time and design the layout. We introduce three use cases that can benefit from our proposed design. We evaluate one of them, Hardware Transactional Memory, to show our design impact in terms of performance and energy consumption. Our experiments show that the dvSRAM allows for significant performance gains with acceptable costs in power, delay and area.

6. ACKNOWLEDGMENTS

This work is supported by the cooperation agreement between the Barcelona Supercomputing Center and Microsoft Research, by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contracts TIN2007-60625 and TIN2008-02055-E, by the European Network of Excellence on High-Performance Embedded Architecture and Compilation (HiPEAC) and by the European Commission FP7 project VELOX (216852).

7. REFERENCES

[1] Predictive technology model. `http://ptm.asu.edu/`.
[2] The Electric VLSI Design System. `http://www.staticfreesoft.com`.
[3] Understanding Static RAM Operation. Technical Report IBM Application Notes, IBM, 1997.
[4] B. S. Amrutur. *Design and Analysis of Fast Low Power SRAMs*. PhD thesis, 1999. Stanford University.
[5] A. Armejach, A. Seyedi, et al. ShadowHTM: Using a dual-bitcell L1 Data Cache to Improve Hardware Transactional Memory Performance. Technical Report UPC-DAC-RR-2010-49, UPC, 2010.
[6] C. Cao Minh et al. STAMP: Stanford transactional applications for multi-processing. In *IISWC*, 2008.
[7] S. Cosemans et al. A Low-Power Embedded SRAM for Wireless Applications. *IEEE JSSC*, 2007.
[8] O. Ergin et al. Early Register Deallocation Mechanisms Using Checkpointed Register Files. *IEEE Trans. Computers*, 2006.
[9] T. Harris et al. Transactional Memory, 2nd edition. *Synthesis Lectures on Computer Architecture*, 2010.
[10] V. Krishnan et al. A Chip-Multiprocessor Architecture with Speculative Multithreading. *IEEE Trans. Computers*, 1999.
[11] J. Pille et al. A 32kB 2R/1W L1 data cache in 45nm SOI technology for the POWER7TM processor. In *ISSCC*, 2010.
[12] J. M. Rabaey et al. *Digital integrated circuits - A design perspective*. Prentice Hall, 2nd edition, 2004.
[13] R. Rajwar and J. R. Goodman. Speculative lock elision: Enabling highly concurrent multithreaded execution. In *MICRO*, 2001.
[14] S. Thoziyoor et al. CACTI 5.1. Technical Report HPL-2008-20, HP Laboratories, Palo Alto, 2008.
[15] Y. Wang et al. A 1.1 GHz 12 μA/Mb-Leakage SRAM Design in 65 nm Ultra-Low-Power CMOS Technology With Integrated Leakage Reduction for Mobile Applications. *IEEE JSSC*, 2008.
[16] Y. Ye, S. Borkar, and V. De. A new technique for standby leakage reduction in high-performance circuits. In *Symp. on VLSI Circuits*, 1998.
[17] A. F. Yeknami. Design and Evaluation of A Low-Voltage, Process-Variation-Tolerant SRAM Cache in 90nm CMOS Technology. Master's thesis, 2008. Linköping University, Sweden.
[18] L. Yen et al. LogTM-SE: Decoupling hardware transactional memory from caches. In *HPCA*, 2007.
[19] Y. Yu et al. Multi-valued static random access memory (SRAM) cell with single-electron and MOSFET hybrid circuit. *Electronics Letters*, 2005.