Abstract—High associativity is important for level-two cache designs [9]. Implementing CAM-based Highly Associative Caches (CAM-HAC), however, is both costly in hardware and exhibits poor scalability. We propose to implement the CAM-HAC in macro-blocks to improve scalability. Each macro-block contains 128-row and 8-column of cache blocks. We name it Two dimensional Cache, or T-Cache. Each macro-block has an associativity equivalent to 128×8=1024-way. Twelve bits of the T-Cache's tags are implemented by using CAM, while the remaining tag uses SRAM. Furthermore, random replacement is used in rows to balance cache sets usage while LRU is used in columns to select the victim from a row. The hardware complexity for replacement is reduced greatly compared to a traditional CAM-HAC using LRU solely. Experimental results show that the T-Cache achieves a 16% miss rate reduction over a traditional 8-way unified L2 cache. This translates into an average IPC improvement of 5% and as high as 18%. The T-Cache exhibits a 4% total memory access-related energy savings due to the reduction to applications’ execution time.

I. INTRODUCTION

High associativity is important for level-two cache designs [9][4]. Highly associative caches, however, increase hardware cost due to extra comparators and wire routing compared with low associative caches. Content Addressable Memory (CAM) based Highly Associative Caches (CAM-HAC) store tag in CAM cells to facilitate tag comparison, however, CAM cells are power costly and consume large area. CAM-HACs also depend on complex replacement policies, which either need future information [1] to achieve the lowest miss rate or more status counters for replacement. The widely employed Least Recently Used (LRU) replacement is only implemented in low associative caches. On the other hand, random replacements need trivial hardware but endure the penalty of higher miss rates than LRU.

Substantial research has been conducted to reduce either hardware complexity of CAM-HAC or miss rate and miss penalties of low associative caches. Instead of using a hardware-based replacement, a software-controlled fully associative cache [4] takes advantage of the flexibilities of software to manage fully associative caches. The V-Way cache [9] increases the associativity of an 8-way 256 KB cache by dynamically allocating more ways to a particular set by borrowing ways from other sets. Our work focuses on implementing an inexpensive highly associative L2 cache. The major contributions of our work are as follows:

1. We propose to divide L2 cache blocks into rows and columns, thus forming Two-dimensional Caches (T-Cache).
2. We propose to use a combined replacement in the T-Cache. A random replacement is used within the rows to balance the accesses to the cache, while the LRU is used within the columns to pick a victim block from the row selected by the random policy. The combined replacement can achieve a comparable miss rate reduction with much less hardware complexity compared with traditional highly associative caches using solely LRU.
3. We observe that high associativity is important to L2 cache designs; however, it does not mean a fully associative cache is imperative for any capacity of L2 caches. We observe that an associativity of 512-way is sufficient for most applications. The benefit of maintaining a constant associativity for large L2 caches is that we can improve the scalability of CAM-HAC by using a macro-block based design, where each macro-block is a T-Cache.
4. We observe that traditional CAM-HACs overuse the CAM tag and propose a simplified CAM-HAC implementation, where the tag is partially implemented by using CAM, while the remainder tag is implemented by using SRAM.
5. The proposed T-Cache also provides a platform for novel requestor aware replacement to be developed on chip multi-processors, where large L2/L3 cache can be shared by multiprocessors. We leave requestor aware replacements as natural future work for the T-Cache design.

Using execution driven simulations, we demonstrated that the T-Cache achieves an average miss rate reduction of 16% over 16 benchmarks from the SPEC2K [5] suite compared to an 8-way 1 MB L2 cache. This translates into an Instruction Per Cycle (IPC) improvement up to 18% and an average IPC improvement of 5%. The T-Cache consumes 9% more power per access but exhibits a 4% total memory access-related energy savings.

The rest of the paper is organized as follows. Section II describes the organization of the T-Cache. Experimental methodology and results are presented in Section III. Section IV analyzes the cost of the T-Cache. Performance and power analysis are presented in Section V. Related work is discussed in Section VI and concluding remarks are given in Section VII.

II. THE T-CACHE

A. Organization

Figure 1(a) shows the organization of a 128 KB macro-block T-Cache with a line size of 128 bytes. The address is assumed to be 48 bits. For the ease of comparison, Figure 1(b) also shows the organization of a same sized traditional 8-way cache. The T-Cache replaces the original 7×128 decoders with the 12-bit CAM based decoder. The CAM decoder serves as both the row decoders and partial tag comparison. The T-Cache has eight extra signals, CAM hit_1, ..., and CAM hit_8. An assertion of the signal CAM hit_j indicates a CAM tag hit in column j. These signals are used in cache miss handling to
The proposed T-Cache macro-block can be used to implement two-dimensional large caches. For example, a 512 KB L2 cache needs four T-Cache macro-blocks and a 1MB L2 cache needs eight macro-blocks. A macro-block decoder is required and can be indexed by using high order bits of the address. For example, a 3×8 traditional decoder can be used to locate the desired macro-block for a 1MB cache. Only one macro-block is accessed during a cache visit, therefore, the power overhead of the T-Cache due to CAM cells will not be increased as the cache capacity increases.

**B. Partial CAM Tag as Set Decoders**

In a traditional CAM-HAC, tag memory is implemented by using CAM to facilitate searching for the desired tag. We observed that the traditional CAM-HAC overuses power and area costly CAM cells. To reduce the number of CAM cells, we divide the tag memory into SRAM tag and CAM tag for the proposed T-Cache, as shown in Figure 1 (a).

The function of the CAM tag in a traditional CAM-HAC, is threefold. First, the CAM tag serves as a full tag comparison and verifies a cache hit. Second, the CAM tag serves as a decoder, whose output drives one cache line when there is a cache hit. Lastly, the CAM tag makes it possible to choose the best victim, since any cache line can be the candidate for the victim during a cache miss.

The entire tag is required to verify a cache hit. However, this does not require the whole tag to be implemented in CAM. For the second function of the CAM tag, decoding 128 cache blocks does not need all 41 tag bits. In fact, seven bits of CAM cells are sufficient to distinguish the 128 cache blocks. It should be noted that the tag addresses stored in the CAM must be different from each other to maintain unique decoding. For the third function, a victim can be selected from all the cache lines for a cache miss. When the cache size is increased, so is the associativity of a fully associative cache. Increased associativity requires redesigning the control circuit. Therefore, the scalability for the traditional CAM based fully associativity is poor. We propose to implement large L2 caches with constant associativity. When the capacity of L2 caches is increased, the associativity will remain unchanged. The constant associativity is implemented in one macro-block. Each macro-block is a T-Cache with desired associativity. Large L2 caches are implemented using multiple macro-blocks without increasing the associativity. Therefore, the scalability can be improved compared to a traditional CAM fully associative cache. We determine the associativity of the T-Cache macro-block through experiments as discussed in Section III.C.3.

**C. Hardware Reduction through Combined Replacements**

The two widely used replacement policies are random replacement and the LRU. The random policy requires the least amount of hardware to implement but exhibits a higher miss rate than the LRU. To achieve both a low miss rate and low hardware complexity, a better solution would combine the two policies to get the low hardware complexity of the random and the low miss rate of the LRU.

The proposed T-Cache divides the cache blocks into rows and columns and provides opportunities to exploit new replacements. We propose to use the random policy among the rows in each column to balance the accesses to the cache rows and the LRU within the columns to select the best victim from the columns of each row. Therefore, the total hardware complexity required for replacement of the T-Cache approaches traditional 8-way caches using the LRU.

**D. Scalability Improvement**

The cache hit signal of a fully associative cache is generated from all cache blocks. When the cache size is increased, so is the associativity of a fully associative cache. Increased associativity requires redesigning the control circuit. Therefore, the scalability for the traditional CAM based fully associativity is poor. We propose to implement large L2 caches with constant associativity. When the capacity of L2 caches is increased, the associativity will remain unchanged. The constant associativity is implemented in one macro-block. Each macro-block is a T-Cache with desired associativity. Large L2 caches are implemented using multiple macro-blocks without increasing the associativity. Therefore, the scalability can be improved compared to a traditional CAM fully associative cache. We determine the associativity of the T-Cache macro-block through experiments as discussed in Section III.C.3.

**E. The Operation of the T-Cache**

The operation of the T-Cache is divided into two categories, searching for the hit block for a cache hit and searching for the
Performance. We determine the processor simulator [2] to collect the L2 cache miss rate and block is chosen from the two unmatched columns, which are blocks based on the column replacement policy.

Searching for the hit block. One macro-block, which is determined through the block decoder, is probed to search the desired block. All the columns are searched concurrently. The set is determined through the CAM decoder (tag). Each column may have only one candidate for hit block depending on the tag addresses stored in the CAM. The row replacement of the T-Cache guarantees that there is only one CAM tag hit block in each column. The final hit block is determined from the eight candidates from these eight columns.

Searching for the victim. There are three different situations when searching for the victim block. Figure 2 (a) shows the situation in which no CAM tag matches the desired address. The victim block will then be searched for in the following two steps: first, a victim row is selected based on the row replacement policy and secondly the victim block will be chosen from the eight blocks in the selected row based on the column replacement policy.

Figure 2 (b) shows the situation in which each column has one (the replacement guarantees that there cannot be more than one) CAM tag that matches the desired address. Under this situation, the victim must be chosen from these hit CAM blocks, since each column can only have one CAM tag hit to maintain the unique CAM tag decoding. One row from these CAM hit rows is chosen based on the row replacement policy. The victim block is then chosen from the CAM hit blocks in the selected row. For this example, row two is chosen as the victim row. In this row, there are two columns, C1 and C5, which have CAM tag hits. The victim block is chosen from these two blocks based on the column replacement policy.

Figure 2 (c) shows the situation in which several columns have CAM tag hits while the remaining columns do not have CAM tag hits. In this example, there are six columns that have one block matching the CAM tag, while the other two columns do not match the CAM tag. The victim will be chosen from these two unmatched columns. The victim row is chosen based on the row replacement policy, which is row R0 in this example. Based on the column replacement policy, the victim block is chosen from the two unmatched columns, which are C0 and C4.

III. Experimental Methodology and Results

We use cache misses as the primary metric to measure the T-Cache effectiveness. We used a four-issue out-of-order processor simulator [2] to collect the L2 cache miss rate and performance. We determine the T-Cache parameters through experimentation. Overall performance improvement is discussed in Section V.A.

A. Cache Hierarchy

The T-Cache is designed to have miss rates comparable to a fully associative cache but with hardware complexity similar to an 8-way traditional cache. Therefore, we chose the baseline as an 8-way unified L2 cache. To compare with related work [9], a 1M KB L2 cache is chosen. Experiments on 2 MB and 4 MB are also reported. The replacement policy of the baseline is the LRU and the hit latency of the L2 cache is 10 cycles. The L1 cache parameters are kept constant for all experiments. Table 1 shows the parameters of the processor used in our experiments, which are similar to [9].

B. Benchmarks

We ran all 26 SPEC2K benchmarks using the SimpleScalar tool set [2]. The benchmarks were pre-compiled for the Alpha ISA. The benchmarks that use the ref input set were fast-forwarded for fifteen billion instructions and executed for two billion instructions afterwards. For benchmarks bzip2, gcc, mcf, and vpr, a slice of two billion instructions from the ref input were unable to capture the behavior of the benchmarks [9], so the experiments were run with the test input set from start to completion. The benchmark ammp uses the test input set but halts at one billion instructions. We report results of 16 memory intensive benchmarks out of the 26 SPEC2K benchmarks as did in [9]. Other 10 benchmarks from the SPEC2K do not benefit from the techniques we proposed, however, our results show that the proposed techniques do not impact the IPC inadvertently.

C. Experimental Results

In this section, we first show that maintaining a high associativity is important for L2 caches. Second, we show how to determine the row and column associativity. Last, we determine the CAM tag width, which approximates the high associativity in the columns with the goal of reducing the number of CAM cells while maintaining a low misses.

III.C.1 High Associativity is Important to L2 Cache

The T-Cache is designed to have miss rates comparable to a fully associative cache but with hardware complexity similar to an 8-way traditional cache. Therefore, we chose the baseline as an 8-way unified L2 cache. To compare with related work [9], a 1M KB L2 cache is chosen. Experiments on 2 MB and 4 MB are also reported. The replacement policy of the baseline is the LRU and the hit latency of the L2 cache is 10 cycles. The L1 cache parameters are kept constant for all experiments. Table 1 shows the parameters of the processor used in our experiments, which are similar to [9].

### Table 1: Baselines and the T-Cache processor configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch/Issue/Retire Width</td>
<td>4 instructions/cycle, 4 functional units</td>
</tr>
<tr>
<td>Instruction Window Size</td>
<td>128 instructions</td>
</tr>
<tr>
<td>Branch Predictor</td>
<td>Default predictor from SimpleScalar</td>
</tr>
<tr>
<td>L1 cache</td>
<td>16 KB, 64B line size, 2-way, LRU replacement</td>
</tr>
<tr>
<td>L2 Unified Cache</td>
<td>256KB, 128B line size, 8-way, 6 cycles hit</td>
</tr>
<tr>
<td>Main Memory</td>
<td>Infinite size, 100 cycle access</td>
</tr>
</tbody>
</table>
Figure 3: L2 cache miss rate reductions of traditional 32, 128, 512, 1024-way, and the fully associative cache (FA-LRU) over the baseline.

Figure 3 shows the miss rate reductions of the L2 cache at associativities of 32-way, 64-way, 128-way, 256-way, 512-way, 1024-way, and fully associative using the LRU replacement (FA-LRU) compared to the baseline. From these results, we made the following observations.

The first observation we made is that 11 out of 16 benchmarks benefit from the high associativity. These applications include ammp, apsi, craft, facerec, gcc, mcf, mesa, parser, perlbench, vortex, and vpr. For benchmarks apsi, facerec, and mcf, associativity lower than 512-way would not result in any miss rate reduction. On the other hand, associativity higher than 512-way is not important, either. The miss rate reduction of the fully associative cache is 18% higher than the baseline.

III.C.2 Column Associativity

We determined the column associativity based on our expectation that the T-Cache has similar hardware complexity as a same sized 8-way traditional cache. Therefore, we chose 8-way as the column associativity.

III.C.3 Row Associativity

The effective associativity of the T-Cache is row-associativity × column-associativity. Row associativity lower than 64-way may not be useful, since the effective associativity of the T-Cache will be lower than 64×8 = 512-way. Figure 4 shows the miss rate reductions when row associativity is 64-way, 128-way, and 256-way with column associativity of 8-way. The replacement policies used within the rows and columns are both LRU. We can see that row associativity 128-way (the effective T-Cache associativity is 128×8 = 1024) is good enough for all the benchmarks. Only benchmark, ammp, exhibits significant miss rate reduction when the row associativity is increased from 128-way to 256-way, while other benchmarks remain almost unchanged when the row associativity is increased from 64-way to 512-way. Therefore,

Figure 5: The T-Cache miss rate at CAM width of 10, 12, 14, and 16 bits and fully associative cache (FA-LRU).

we chose the row associative to be 128-way. It should be noted that the effective associativity of the T-Cache is 1024-way, which is higher than the 512-way we obtained from Section 4.3.1. This is because the T-Cache loses some benefits of the LRU due to the partial CAM tag decoder. Remember that the T-Cache trades the replacement for less CAM cells.

III.C.4 CAM Tag Width

The row associativity is implemented in each column of the T-Cache. We reduce the CAM cells in the original fully associative cache. We determine the CAM tag width through experimentation. Figure 5 shows the T-Cache miss rate for CAM tag widths of 10, 12, 14, and 16 bits and a traditional fully associative cache. From Figure 5, we can see that when the CAM tag width is 12-bit, the miss rate difference between the T-Cache and the fully associative cache is less than 0.4%. Considering both the performance and the CAM tag overhead, we choose a CAM tag width of 12 bits.

III.C.5 T-Cache Replacement Policy

The T-Cache divides the blocks into rows and columns, so the victim row and block can be selected independently using different replacement policies. Figure 6 shows the miss rate reductions of the T-Cache when the rows and columns use four different replacement policy combinations of random and LRU. There are four possible configurations for row and column replacement policies: random-random, random-LRU, LRU-random, and LRU-LRU.

From Figure 6 we can see, on average, the T-Cache exhibits the lowest miss rate using the replacement combination of LRU-LRU, where both the rows and columns use the LRU. The highest miss rate occurs when both the rows and columns use random policies. The T-Cache exhibits a similar miss rate reduction with replacement combinations of LRU-random and random-LRU. Section IV.A analyzes the hardware complexities of these two different replacements. The combination of LRU-
LRU needs more counter bits than that of the random-LRU combination. Since the performance difference is not significant using LRU-LRU, we determined to use the random-LRU combination for the T-Cache, where the random policy is used within the rows and the LRU is used within the columns.

IV. COST ANALYSIS

A. Storage Overhead

We compare the hardware complexity of the T-Cache in terms of the tag and status counters for replacement to a traditional 8-way cache and a fully associative cache. Table 3 shows the hardware overhead at cache size of 1MB with a line size of 128 bytes and 48-bit address.

<table>
<thead>
<tr>
<th></th>
<th>256 KB</th>
<th>512 KB</th>
<th>1 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-way</td>
<td>200</td>
<td>1.80</td>
<td>341</td>
</tr>
<tr>
<td>FA</td>
<td>671</td>
<td>5.73</td>
<td>1122</td>
</tr>
<tr>
<td>T-Cache</td>
<td>218</td>
<td>1.99</td>
<td>368</td>
</tr>
</tbody>
</table>

B. Power Overhead

The power overhead of the T-Cache over traditional 8-way caches comes from the extra CAM tag cell in the cache memory. The power per cache access of the T-Cache is lower than the same sized fully associative cache and higher than the same sized baseline. Compared to the 8-way baseline, the power per access is increased due to the 12-bit CAM tag since CAM consumes 5-10 times the power of the same size RAM [4]. However, the power consumption is also decreased by the removal of traditional decoders (as shown in Figure 4 (c)), less SRAM tags, and corresponding comparators. Table 2 shows the power consumption (including both tag and data store) at 70nm technology calculated using CACTI v.4.2 model, which is updated to suit advance technologies considering both dynamic and static energy consumption. The power consumption of the T-Cache is 5% higher than original 8-way cache at size of 1 MB, because only one macro-block of the T-Cache is accessed and the static energy of L2 caches will contribute the major portion of the total energy consumption for large caches.

C. Timing Analysis

Accesses to the T-Cache can proceed sequentially or in parallel. We replace the traditional decoders with the CAM decoders. The CAM decoders need longer time than the traditional decoders. However, the T-Cache reduces both the tag width of the SRAM and the corresponding traditional comparators by 4 bits, thus reducing the tag comparison time. We compute the access time of the traditional 7×128 decoder, the CAM decoder, SRAM tag, and comparators (the length of both SRAM tag and comparators are reduced by 4=12-8 bit), using CACTI-4.2. The results are summarized in Table 2. The T-Cache has a faster access time than the traditional fully associative cache, since the T-Cache uses fewer CAMs. At 70nm technology, the T-Cache needs one more cycle to access than the baseline at sizes of 1 MB, 2 MB, and 4 MB.

V. ANALYSIS

A. Overall Performance

Table 1 shows the processor configuration for both the baseline and the proposed T-Cache.

Figure 8 shows the performance improvements measured in IPC over the processor equipped with an 8-way baseline cache. The greatest performance improvement is seen in ammp, where the IPC increases by 18 %. The performance of the T-Cache is on average 4.8% higher than the 8-way baseline.

The miss rate reduction and IPC improvement of the T-Cache approaches that of a fully associative cache. On the other hand, the miss rate reduction of the T-Cache is only 2% less than the fully associative cache.

B. Overall Energy Analysis

We consider both dynamic and static in energy dissipation our energy evaluation. Energy consumption due to accessing off-chip memory is also considered, since fetching instruction and data from off-chip memory is energy costly because of the high off-chip capacitance and large off-chip memory storage. Also, when accessing the off-chip memory, the microprocessor stalls while waiting for the instruction and/or data and this waiting still consumes some energy. Thus, our equation for computing the total energy due to memory accesses is shown as in Figure 7.

The underlined terms are those we obtain through measurements or simulations. We compute L2_access and L2_miss by running SimpleScalar [5] simulations. We compute E_L2 of the original cache and the T-Cache by using CACTI 4.2. The E_offchip_access value is the energy of accessing off-chip memory and the E_uP_stall is the energy consumed when the microprocessor is stalled while waiting for the memory system to provide data. E_L2_refill is the energy to refill an L2 cache block. The two terms are highly dependent on the particular memory and microprocessor being used. We redefined E_miss as shown in Figure 7 and considered the situations of k_miss_E equal to 20 and 200.
Finally, cycles is the total number of cycles for the benchmark to execute, as computed by SimpleScalar. $E_{\text{static}}$ is the total static energy consumed per cycle. In this paper, all energy plots use $k_{\text{miss\_energy}} = 100$ and $k_{\text{static}} = 30\%$. We discuss the impact of the larger values for those constants, while the plots for those larger values are not shown due to space limit. Figure 9 shows the energy of the $T$-Cache normalized to the baseline. On average, the $T$-Cache consumes 3.4\% less energy than the baseline. The greatest energy reduction is seen in apsi where the energy is reduced by 60\%. This significant reduction comes from the miss rate reduction, and hence, the accesses to the off-chip memory, which is time consuming and power costly. The energy consumption is increased when L2 cache misses rate cannot be further reduced by using the $T$-Cache. The benchmark twolf exhibits the highest energy overhead, which is 3.8\% over the baseline.

VI. RELATED WORK

The Indirect Index Cache (IIC) [4] implemented a fully associative cache through a software based replacement scheme. The tag comparison and data look up are performed in serial. A forward pointer is required to identify the corresponding data store. A collision chain is used to provide a platform for other optimizations such as novel cache replacements to reduce the effective access time of large L2/L3 cache blocks for CMPs. Future work includes evaluating the impact of novel requestors aware replacement policies by exploiting the two dimensional organizations of the cache blocks for CMPs.

VII. CONCLUSIONS

We proposed a highly associative L2 cache design by using CAM decoders and dividing the associativity into rows and columns to reduce the hardware complexity. The $T$-Cache provides a platform for other optimizations such as novel cache replacements to reduce the effective access time of large L2/L3 caches for future NUCA for CMPs. Future work includes evaluating the impact of novel requestors aware replacement policies by exploiting the two dimensional organizations of the cache blocks for CMPs.

VIII. REFERENCES


Figure 8: IPC improvements of the $T$-Cache and traditional fully associative cache over the baseline.

Figure 9: Overall energy reduction of the $T$-Cache over the baseline.

<table>
<thead>
<tr>
<th>1MB</th>
<th>Tag</th>
<th>Replace</th>
<th>DEC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-way</td>
<td>8192-31</td>
<td>8192-3-24576</td>
<td>9.512</td>
</tr>
<tr>
<td>FA</td>
<td>8192-41</td>
<td>8192-13-106496</td>
<td>N/A</td>
</tr>
<tr>
<td>T-Cache</td>
<td>8192×12 CAM</td>
<td>column-lru: 8192-3-24576</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>8192×29 SRAM</td>
<td>row-lru: 8×128×7×7-1168</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 3: Tag memory and status counters for replacement of the baseline, a fully associative cache and the $T$-Cache.