Accelerating Non-volatile/Hybrid Processor Cache Design Space Exploration for Application Specific Embedded Systems

Mohammad Shihabul Haque, Ang Li, Akash Kumar, Qingsong Wei*
National University of Singapore and *Data Storage Institute (DSI) Singapore
{matmsh, angli, akash}@nus.edu.sg, *WEI_Qingsong@dsi.astar.edu.sg

Abstract— In this article, we propose a technique to accelerate non-volatile/ hybrid of volatile and non-volatile processor cache design space exploration for application specific embedded systems. Utilizing a novel cache behavior modeling equation and a new accurate cache miss prediction mechanism, our proposed technique can accelerate NVM/hybrid FIFO processor cache design space exploration for SPEC CPU 2000 applications up to 249 times compared to the conventional approach.

1. INTRODUCTION

Presence of Non-Volatile Memory (NVM) cell in processor cache is no longer a science fiction. In the June of 2014, Toshiba Corporation revealed their non-volatile Perpendicular STT-MRAM cell based L2 processor cache that overwhelmed all the advantages of conventional volatile SRAM caches [20]. Toshiba Corporation also offer budget friendly STT-MRAM+SRAM hybrid cache [19]. Several commercially implementable designs came out recently to use non-volatile Phase Change Random Access Memory (PCRAM) cells with SRAM cells for low power hybrid processor caches (e.g. [17]). Moreover, extensive researches are going on to utilize the other NVM cell technologies (such as Resistive Random Access Memory, NAND Flash, etc.) in different levels of the processor cache hierarchy. The features that made NVM cells so attractive for processor caches; especially in energy-performance-area critical embedded systems, are (i) Higher storage density [13], (ii) Lower area occupancy [22] and (iii) Significantly lower energy consumption [14] over SRAM cells.

Besides being energy-performance-area critical, embedded systems are usually application specific. For application specific systems, in SRAM cache design space exploration [1], trace-driven single-pass cache simulators are widely used to quickly find the total number of cache misses during execution of the application on different cache configurations [24, 9]. We call this phase as the cache performance evaluation phase. Once the cache performance (i.e. number of cache misses) is known, analytical models (such as the one in [12]) can be used to calculate the amount of energy and area consumption by each cache configuration simulated. The cache configuration that best suits the performance-energy-area occupancy criteria is chosen for the final system design. Available single-pass cache simulators are ill-suited to fulfill the requirements of the cache performance evaluation phase in NVM/hybrid cache design space exploration. Two such requirements are discussed below:

- Almost every type of NVM cell is prone to wear out when written heavily (i.e. loaded with data/updated frequently) [15, 29]. Therefore, in a heavily written cache that deploys NVM cells/cache lines (from here we use the words "cell" and "cache line" interchangeably) with limited write endurance, cache configuration as well as performance and energy consumption may change over time due to cache line wear out. Change in cache performance and energy consumption can be fatal in systems such as application specific real-time embedded systems. Therefore, the system designer must deploy a safety mechanism to prevent the system from being used when its cache configuration reaches to an unsuitable state. To assist the system designer in designing a safety mechanism or to identify the unsuitable states in a cache during design space exploration, the cache performance evaluation phase needs to evaluate the performance in every initially deployable cache configuration as well as in every other configuration it generates due to line wear out.
- Besides limited write endurance, any type of NVM cell is far more expensive than SRAM cell. Moreover, writing time is significantly slower in some types of NVM cells compared to SRAM cells (e.g. PCRAM write latency is 5ns where SRAM write latency is only 1ns [17]). Due to one or few of these reasons, instead of deploying NVM cell in every cache line, hybrid cache that deploys different types of cells (such as PCRAM, STT-MRAM, SRAM, etc.) is used. To meet budget, cache performance and/or cache lifespan constraints, a hybrid cache should deploy different types of cells in such a combination that (i) price of the cache remains reasonable, (ii) application executes at a reasonable speed, (iii) NVM cells/lines are not used in the heavily written lines, and/or (iv) cache performance and energy efficiency does not degrade significantly when couple of NVM lines wear out. To assist in finding the most suitable combination of cells for a hybrid cache configuration during design space exploration, the cache performance evaluation phase needs to report the number of writes per cache line and per cache set.

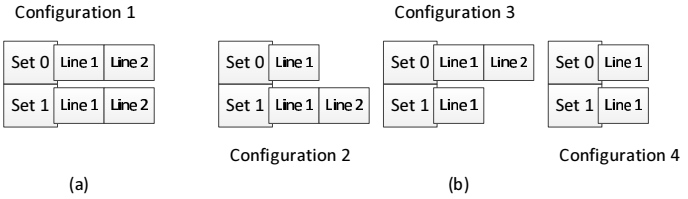

Figure 1: An NVM Cache's Configurations Due to Line Wear Out

Let us explain with an example how these new requirements make the available single-pass cache simulators infeasible for NVM/hybrid processor cache design space exploration. Figure 1 (b) shows all the three additional configurations that may generate from the NVM/hybrid FIFO cache configuration shown in Figure 1 (a) when the cache becomes unusable after the last active line in any cache set wears out. That means, if the set size was 8 and associativity was 64 in the cache configuration of Figure 1 (a), it was necessary to simulate 281,474,976,710,655 configurations in the existing single-pass cache simulators. Simulating so many configurations will take few years for sure by the exiting single-pass cache simulators. Moreover, to record the number of misses per line in each of these cache configurations, enormous amount of storage is required.

In this article, we propose a trace-driven resource generous technique "Breakneck Cache Performance Evaluation Method" ("BCPEM") to accelerate the cache performance evaluation phase in NVM/hybrid processor cache design space exploration for application specific embedded systems. "BCPEM" is exclusively designed for First-In-First-Out (FIFO) replacement policy as FIFO is very popular in embedded processor caches [9]. "BCPEM" can accelerate efficiently as long as (i) line wear out in one cache set does not influence the number of misses in other cache sets in an NVM/hybrid cache and (ii) line size is same in all the cache configurations to analyze. To accelerate, "BCPEM" utilizes the following approaches:

- Instead of simulating each cache configuration, "BCPEM" evaluates performance of each cache set separately. For example, if the

[1]The process of finding the most suitable processor cache configuration. Combination of the following cache parameters is defined as cache configuration: (i) set size: number of cache sets, (ii) Associativity: number of storage cells/cache lines per set, (iii) Line size: amount of data storable in each storage cell, and (iv) Replacement Policy: policy to select a storage cell to load new data

978-1-4799-7792-5/15/$31.00 ©2015 IEEE

set size is 8 and associativity is 64 in a cache initially, "BCPEM" evaluates performance of each of the 8 cache sets separately for associativities 64, 63, 62,...1 when line wear out can reduce associativity to 1. Therefore, only 512 ***cache set*** configurations are necessary to evaluate rather than 281,474,976,710,655 ***cache configurations***. This approach reduces storage requirement significantly.

- Per simulated cache set, by simulating few of the associativities, "BCPEM" predicts the number of cache misses for all the associativities possible due to line wear out. "BCPEM" utilizes "Breakneck Modeling Equation" ("BME") and "Breakneck Prediction Mechanism" ("BPM") to predict the number of misses in the associativities which are not simulated. Once the number of cache misses is known for a given application for each of the associativities possible (e.g 64 to 1) in each cache set (e.g. each of the 8 sets), "BCPEM" can calculate the number of cache misses in all the cache configurations possible in the NVM/hybrid cache (e.g. total of 281,474,976,710,656) in a split of a second.

This article discusses "Breakneck Cache Performance Evaluation Method" ("BCPEM") as follows: Section 2 discusses some related works; Section 3, 4 and 5 discuss "BME", "BPM", and "BCPEM" respectively, Section 6 shows the efficiency of "BCPEM" with empirical evidences and Section 7 concludes the paper.

Throughout the article, we made the following assumptions:

- One faulty bit makes a cache line completely unusable.
- When all the lines wear out in a set, the set as well as the entire cache dies/wears out.
- Line size is same in all the cache configurations.

2. RELATED WORK

Among the available cache simulation techniques (such as system simulation [28], instruction set simulation [16], etc.), application's memory access trace-driven single-pass cache simulation is known to be the fastest and the most resource generous. In a single-pass cache simulator, a trace file that indicates when and which data blocks were accessed by the processor during execution of an application is used as the input. By reading one data block access at a time from the trace file, the single-pass simulator checks whether the requested data block is available in the simulated cache configurations. Cache configurations are represented by an array or a list in single-pass simulation. Therefore, without spending a large amount of time in simulating the exact hardware behavior (unlike system simulation [28] and instruction set simulation [16]), single-pass cache simulators can quickly and accurately estimate the number of cache misses for a particular application on a group of cache configurations.

To mimic the hardware behavior minimally and to reduce the need for extensive computing resources, additional mechanisms, such as special data structures [8], trace compression [18, 27], running the simulation on parallel hardware [10, 25], inclusion properties [24], etc. are applied in single-pass simulation (Point to note, not all of these mechanisms can be used for every cache replacement policy). The state-of-art, single-pass FIFO cache simulator is "CIPARSim" [9]. Like the other existing cache simulators, "CIPARSim" is also not fast enough to fulfill the requirements of NVM/hybrid FIFO processor cache design space exploration for application specific systems.

Modeling the effect of caching capacity on cache misses is mathematically challenging. However, having such an analytical model can help to estimate the number of cache misses during an application without performing any time consuming simulation. Therefore, such an analytical model can be the perfect choice to replace single-pass cache simulators in the cache performance evaluation phase during NVM/hybrid FIFO processor cache design space exploration. Analytical modeling possibilities have been studied widely but without much success. Earlier works in this attempt found the necessity to adapt simple memory access models [1, 21] and/or focus on specific replacement policies [7, 4]. However, they neither take into account the interaction among processes and with the kernel [11], nor the changes in the memory access pattern [3]. Considering the previous limitations, Tay et al. [23] proposed the following analytical model:

$$n = (1/2) \times (H + \sqrt{H^2 - 4}) \times (n^* + n_0) - n_0 \qquad (1)$$

where $H = 1 + \frac{(M^* - M_0)}{(M - M_0)}$ and $M < M^*$. In this equation, n is the total number of cache misses and it varies when the fully-associative cache's caching capacity M (for which the number of misses has to be estimated) changes. M^* is the optimal caching capacity of a fully-associative cache that does not require any reloading of any data block for a given application. M_0 represents the amount of cache space that cannot be used due to some sort of locking mechanism or scenario-specific reasons. n^* is the number of cold misses for an application. n_0 is a correction factor independent of M. M^*,M_0,n^* and n_0 are fixed for a given application, operating system and hardware. **This modeling equation is not efficient for a cache configuration on which an application's n is close to n^* or cache hit rate is too low.**

Equation 1 is called "Page Fault Equation" (referred to as "PFE"). This equation can model the effect of caching capacity on the number of cache misses efficiently (for real workloads on Linux and Windows 2000 that include different replacement policies, compute, IO and memory intensive benchmarks, multiprogramming and user input. Performance of "PFE" has never been evaluated for SPEC CPU 2000 and 2006 benchmark applications) when the values of M^*,M_0,n^* and n_0 are provided for an application. To obtain/approximate the values of M^*,M_0,n^* and n_0, simulations are performed to know the number of misses for all the caching capacities (Ms) possible in a fully-associative cache and the results are used in regression analysis. The less number of caching capacities are simulated, the less the efficiency of "PFE". Therefore, "PFE" is not helpful enough to be utilized as a cache miss prediction method to avoid cache simulation in the cache performance evaluation phase in design space exploration. In addition, as there was no known fast method to calculate the actual values of M^* and n^*, it was impossible to verify the definition of M^* and n^* used in "PFE" and accuracy of their values found via regression analysis. Moreover, "PFE" does not consider cache set size, and no proposal was made on how to adapt "PFE" on modern set-associative processor caches. Therefore, this equation could not help much either in SRAM or in NVM/hybrid cache design space exploration.

Several system-level analytical models are available to collect information for architectural studies such as the effect of associativity on the memory fetch time, effect of cache line size on cycle time, etc. CACTI [26] is one such famous system-level model for SRAM caches. For NVM, NVSim [5] is a popular system-level model based on CACTI. However, these system-level analytical models are not capable to model the effect of associativity or caching capacity on the number of cache misses for a given application.

3. BREAKNECK MODELING EQUATION

Realizing the potential of analytical modeling and inspired by the success of "PFE" [23], we became interested to analyze "PFE" in search for a modeling equation that can speed up the cache performance evaluation phase in NVM/hybrid FIFO processor cache design space exploration. Here, we discuss our findings that lead us to "Breakneck Modeling Equation" ('BME"). BME is the basis of "BCPEM".

To have a better understanding, we started with verifying the definition of each parameter and variable used in "PFE". As "PFE" does not take cache set size into account, it was intuitive that "PFE" should be applicable to each set separately in a set-associative cache. Therefore, M **should be the associativity and M^* should be the optimal associativity to avoid reloading of any data block for a given application**. We also designed our own algorithm to find the actual values of M^* and n^* quickly and accurately (discussed in Section 5). With the set-associative cache specific definition of M and actual values of M^* and n^*, we discovered that, even when ns for all the possible Ms are known and used in regression analysis for "PFE", the approximate values do not quite match with the actual values of n^* and M^*. Moreover, when we provide the actual values of M^* and n^* in regression analysis to approximate n_0 and M_0 only, "PFE" modeling efficiency degrades. It indicates a problem with the definition or use of n_0, M_0, M^* and n^* in "PFE".

To identify the root of the problem, we decided to analyze n_0 as the starting point. We noticed that n_0 changes significantly when the number of known data points (associativities for which cache misses are known via simulation) in regression analysis changes. It suggests that n_0 **should be replaced with a variable R_0 which is dependent on associativity** (i.e. when M changes, R_0 value should change for a given application. In the original proposal of PFE, n_0 was fixed for a given application). Below, we provide a mathematical proof to support our claim.

Proof Against Fixed n_0 in "PFE": In equation 1 of Section 2, $(1/2) \times (H + \sqrt{H^2 - 4}) = V_1$ is a variable dependent on associativity M and n^* is fixed/constant for the given application.

Therefore, in equation 1,

$$\begin{aligned} n &= (V_1 \times (n^* + n_0)) - n_0 \\ or, n_0 &= (V_1 \times (n^* + n_0)) - n \\ or, n_0 &= (V_1 \times n^*) + (V_1 \times n_0) - n \\ or, n_0 \times (1 - V_1) &= (V_1 \times n^*) - n \\ or, n_0 &= \frac{(V_1 \times n^*) - n}{1 - V_1} \end{aligned} \quad (2)$$

By replacing V_1 *with* $(1/2) \times (H + \sqrt{H^2 - 4})$ *in equation 2*,

$$n_0 = \frac{((1/2) \times (H + \sqrt{H^2 - 4}) \times n^*) - n}{1 - ((1/2) \times (H + \sqrt{H^2 - 4}))} \quad (3)$$

Equation 3 suggests, *as H and n are variables dependent on M, and M_0 is fixed for the given application, we can never find a value for n_0 that does not change with the change in M. Hence, n_0 should be replaced with a variable R_0 dependent on associativity (M).*

Involvement of M_0 in H raised the second concern in our analysis. No cache line is considered to be blocked in the available trace-driven cache simulators [6, 12, 24]. Therefore, even without M_0, the modeling equation should be able to estimate the total number of cache misses (n) for a given associativity (M). With these changes, the equation to model the effect of M on n should look like the following:

$$n = (1/2) \times (H + \sqrt{H^2 - 4}) \times (n^* + R_0) - R_0 \quad (4)$$

where $H = 1 + \frac{(M^*)}{(M)}$ and $M < M^*$. We named equation 4 as the "Breakneck Modeling Equation" ("BME"). like "PFE", "BME" is also not efficient in modeling for a cache configuration on which an application's n is close to n^* (i.e. difference between n and n^* is less than 20%) and/or 80% or more memory accesses by the processor generate cache miss.

4. BREAKNECK PREDICTION METHOD

When associativity decreases in a fully-associative cache/set, usually number of misses (n) increases for a particular application. If the number of misses (n) for associativities M_x,$M_{x-1} = M_x - 1$,$M_{x-2} = M_{x-1} - 1$,...M_1 are $\frac{n_1}{2^{x-1}}$, $\frac{n1}{2^{x-2}}$, $\frac{n_1}{2^{x-3}}$, ...n_1, the ns for M_x to M_1 are certainly not collinear. Similarly, if the number of misses (n) for associativities M_x to M_1 are $\frac{n_1}{1.5^{x-1}}$, $\frac{n_1}{1.5^{x-2}}$, ...n_1, the ns for M_x to M_1 are not collinear; however, the curve generated by the ns is more linear compared to the previous case. In fact, it is very easy to verify using Microsoft excel or similar software that, to be considered collinear, ns for all the associativities in between and including M_x and M_1 must not differ by more than 9% compared to the n of M_x. **When collinear, by knowing ns for associativities M_x and M_1 only, it is possible to calculate/predict ns for all the associativities in between M_x and M_1 using linear interpolation. This method can save a huge amount of time in the cache performance evaluation phase.**

To find the range of associativities that have collinear ns (such as M_x to M_1), we can use "BME". For all the associativities in between and including M_x and M_1, if Hs are very close (i.e. does not differ significantly compared to the H of M_x) and can be considered equal/fixed/constant, then equation 4 can be written as

$$\begin{aligned} n &= C_1 \times (n^* + R_0) - R_0 \\ where\ constant\ C_1 &= (1/2) \times (H + \sqrt{H^2 - 4}) \\ or, n &= (C_1 \times n^*) + (C_1 \times R_0) - R_0 \\ or, n &= (C_1 - 1) \times R_0 - C_2\ where\ constant\ C_2 = (C_1 \times n^*) \\ or, n &= C_3 \times R_0 - C_2\ where\ constant\ C_3 = (C_1 - 1) \end{aligned} \quad (5)$$

Equation 5 is a straight line equation and, therefore, ns for M_x to M_1 are collinear. However, the question remains, when can we consider the Hs for associativities M_x to M_1 as constant? Even though we do not know the answer, the following two information are enough to design an efficient cache miss prediction method: (i) Hs for M_1 to M_x must not differ significantly, and (ii) ns for M_1 to M_x must not differ by more than 9% compared to the n of M_x.

Now, let us explain how the prediction method should work. Initially, it is necessary to calculate H values for all the associativities (M) possible, due to line wear out, in the target cache set for the given application. After that, a single-pass cache simulator have to be used to simulate those pairs of Ms within which Hs of any M does not change by 20% or more compared to the H of the largest M in the pair [2]. If for any simulated associativity pair $\{M_i, M_j\}$, there exists no simulated associativity M_k such as $M_i < M_k < M_j$, and ns for M_i and M_j varies by 10% or more, a new associativity $M_k = \frac{(M_i + M_j)}{2}$ has to be simulated. This process is repeated unless between simulated associativities M_i and M_j, there exists no simulated associativity M_k, and n for M_i is not larger than 9% or more compared to the n for M_j. For example, assume that the single-pass cache simulator simulated associativities (M) 1,16 and 44 for set 0 in a cache with four sets and cache line size 16 bytes. It means, between associativity 1 and 16 (as well as between associativities 16 and 44) Hs do not change by 20% or more. If n for associativity 1 is 10% larger than associativity 16, simulator needs to simulate associativity 8. If n for associativity 1 is 10% larger than associativity 8, simulator needs to simulate associativity 4 and so on. When between any simulated associativities M_i and M_j, there exists no simulated associativity M_k, and ns for M_i and M_j varies by less than 10%, linear interpolation can be used to calculate/predict the number of cache misses in all the associativities in between M_i and M_j. As a result, simulation time can be reduced significantly. We name this prediction method as "Breakneck Prediction Method" ("BPM").

A point to note, instead of linear interpolation in "BPM", "BME" can be used to predict the ns for all the associativities in between M_1 and M_x when their ns are collinear. However, use of "BME" to predict n is slightly time consuming compared to linear interpolation. This is because, to predict the ns using "BME" for all the associativities in between M_1 and M_x, R_0s of M_1 and M_x have to be calculated using "BME" first. After that, using R_0s of M_1 and M_x in regression analysis or linear interpolation (as R_0s for associativities M_x to M_1 should be collinear according to Equation 5), R_0s for all the associativities in between M_1 and M_x have to be approximated/predicted. When n^*, H and R_0 for an associativity (M) are known for a given application, "BME" can be used to predict/estimate the number of cache misses for that M to avoid simulation.

5. BREAKNECK CACHE PERFORMANCE EVALUATION METHOD

Breakneck Cache Performance Evaluation Method ("BCPEM") is an application specific, trace-driven technique which is aimed to speed up the cache performance evaluation phase in NVM/hybrid, set-associative, FIFO processor cache design space exploration, when line wear out in one cache set does not influence the number of misses in other cache sets. "BCPEM" utilizes the fact, when line wear out in a cache set does not influence the number of cache misses (n) in other cache sets, every cache set can be considered as an independent fully-associative cache. That means, in Figure 1 (b), set 0 in both Configuration 2 and Configuration 4 will generate the same number of cache misses for a given application. Therefore, by collecting n in set 0 associativity 1, set 1 associativity 1 and set 1 associativity 2 separately, and by combining the results, n of Configuration 2 and Configuration 4 can be estimated accurately. It is an wastage of time to collect n of set 0 in both of Configuration 2 and Configuration 4. Due to this wastage of time, existing single-pass cache simulators are infeasible to deploy when the NVM/hybrid cache has large number of sets and each set can generate large number of different associativities due to line wear out.

"BCPEM" cache performance evaluation flow is presented in Figure 2. "BCPEM" collects n of each cache set in a cache for different associativities that may generate due to line wear out. For this purpose, "BCPEM" collects n^* and M^* for each cache set first. After that, for each cache set, "BCPEM" collects n for the associativities possible to generate due to line wear out, by simulating only few of those associativities. Let us explain these steps in details:

Step1: n^* calculation - Total number of cold misses (n^*) for a given application in a cache set is the total number of unique data blocks loaded in that cache set. A cache memory loads data blocks containing multi-

[2]Definition of H in Section 3 suggests, when maximum associativity is 64 and optimal associativity is up to 10 million, 20% difference between the Hs of M_1 and M_x can help to skip simulation of up to ten consecutive associativities. It is not a good idea to skip simulation of more than 10 consecutive associativities for which we want to predict ns using linear interpolation

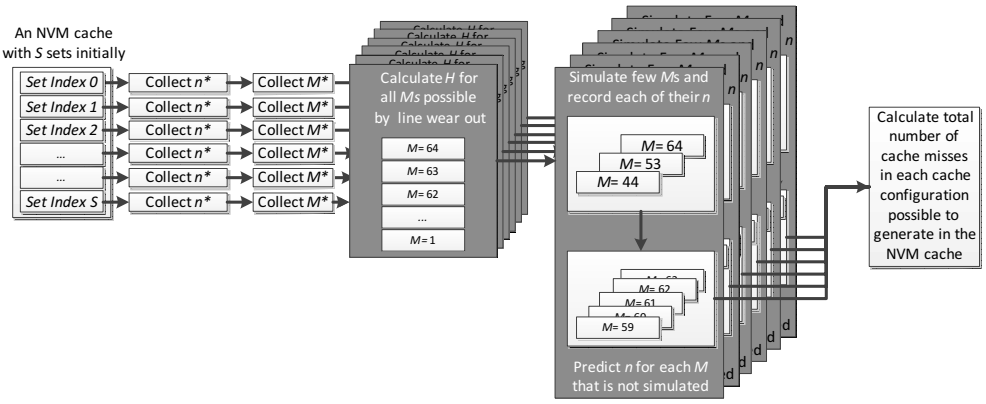

Figure 2: BCPEM Cache Performance Evaluation Flow

ple bytes, but processor requests specific bytes from the loaded blocks. Therefore, trace file records byte addresses that are requested by the processor. However, one specific portion of a byte address indicates the block address. Therefore, number of unique block addresses loaded in a cache set during execution of a given application can be calculated without any additional cost during trace file generation.

Alternatively, by analyzing the byte address trace, "BCPEM" can generate a trace of block addresses destined to each cache set. After that, "BCPEM" can find n^* for each cache set very quickly using the "sort -u <trace of block addresses>|wc -l" command in linux operating system. For a given application, n^* is necessary to calculate only once per set in an NVM cache. Note that this alternative method to calculate n^* may incur large amount of time depending on the size of the byte address trace.

n^* value is necessary in "BCPEM" to estimate the optimal associativity (M^*) per cache set for a given application.

Step 2: M^* ***calculation -*** M^* is the minimum associativity in a cache set required to have no more misses other than cold misses. M^* changes depending on the replacement policy. Without any optimization, M^* calculation can be complicated and time consuming. As our target replacement policy is FIFO and we did not have any optimized algorithm available to calculate M^* in a feasible amount of time, we designed our own fast algorithm for M^* calculation and deployed in "BCPEM". The algorithm is presented below in Algorithm 1.

Algorithm 1: Algorithm for M^* Calculation

```
1  lineCount= 0;
2  countAssoc= 0;
3  currentTime= 0;
4  M*= 0;
5  AssocLimit= n*;
6  while NOT the last line in
   File_containing_unique_addresses_sorted_on_first_access_time do
7      LRS= Read the Next Least Recently Started Memory Address from file
       File_containing_unique_addresses_sorted_on_first_access_time;
8      finisTime= (LRS's last access time from
       File_containing_unique_addresses_sorted_on_last_access_time) + 1;
9      while lineCount < AssocLimit do
10         countAssoc++;
11         lineCount++;
12         currentTime= Read the Next Memory Address's starting time from file
           File_containing_unique_addresses_sorted_on_first_access_time;
13         if ((currentTime >= finishTime) and (countAssoc > M*)) then
14             countAssoc--;
15             M*=countAssoc;
```

Basically, the algorithm works with two input files containing unique block addresses (extracted from the byte trace file destined to a cache set for a given application. These input files are also created at the time of trace generation) along with their first access time in the first file and last access time in the second file. The block addresses are sorted on the first access time and the last access time in the first and second file respectively. Using these input files, the algorithm checks how large a round robin (FIFO) list should be to store unique block addresses without evicting any stored block address before its last access. As an example, let's assume that the first file has addresses 'A', 'B' and 'C' with starting time 1, 5 and 9AM respectively. The second file has addresses 'B', 'C' and 'A' with last access time 6,10 and 11AM respectively. So, the algorithm will read 'A' from the first file first and get the last access time of 'A' from the second file. Then it will check how many other unique block addresses are accessed after the first access to 'A' and their accessing is not over before the last access to 'A'. As 'B' started after 'A' but finished before 'C' was accessed, and 'C' was also accessed for the last time before 'A' was accessed for the last time, only two storage locations are needed to accommodate 'A','B' and 'C' without evicting any address before its last access. The maximum number of storage needed in the FIFO list is the M^*. M^* must be less than or equal to n^* for a given application and a particular cache set. The complexity of this algorithm is only O(n^*). For a given application, M^* is necessary to calculate only once per set in an NVM/hybrid cache.

Step 3: H ***calculation -*** In this step, "BCPEM" calculates the H values for all the associativities possible in the target cache set due to line wear out.

Step 4: Collecting total number of misses (n) ***for each cache set-*** For each cache set, "BCPEM" simulates the associativities (M) suggested by the prediction method "BPM" and collects n. Any single-pass cache simulator can be used in this phase. For the remaining associativities, ns are predicted using linear interpolation on the simulated Ms and their corresponding ns.

Step 5: Collecting total number of misses (n) ***for each cache Configuration-*** When performance/number of cache misses is known for each associativity possible in each cache set of an NVM/hybrid cache, by combining the results, BCPEM can calculate the total number of cache misses in a split of a second in each of the configurations possible in the target NVM/hybrid cache.

6. EXPERIMENTAL RESULTS

We implemented "BCPEM" using the C programming language. We re-implemented "CIPARSim" using the C programming language following the specifications provided in [9]. "CIPARSim" is also used for the cache simulation part in "BCPEM". We conducted our experiments on a machine with Hexa-core processor, 32 GB main memory and 12 MB L2 cache.

Our experiment was divided into three parts. In the first part, we checked the accuracy and efficiency of "BCPEM". This test is referred to as 'T3'. Accuracy of "BCPEM" for a given application and associativity (M) is measured by the deviation in the predicted number of cache misses compared to the number of cache misses estimated in "CIPARSim" via simulation. The bigger the deviation, the lower the accuracy. Efficiency of "BCPEM" for a given application is measured by the number of associativities in a cache set for which cache misses can be predicted. We were not sure whether "BME" can model the effect of M on n accurately in a cache set. As the success of linear interpolation based prediction method "BPM" is dependent on the accuracy of modeling in "BME", we were also not sure whether the number of cache misses (i) for different associativities in a cache set can be predicted accurately, and (ii) can be predicted for significant number of associativities in a cache set to reduce design space exploration time. Therefore, test 'T3' was necessary. To ensure that "BME" was not better than linear interpolation when used in "BPM", accuracy and efficiency of "BCPEM" was also checked in this experiment by replacing linear interpolation with "BME" (see Section 4 for details). This test is referred to as 'T2'. Test 'T2' also helped to show the accuracy of modeling in "BME". As "BME" is formulated based on "PFE", to ensure that "BME" is indeed more useful than

bwaves		M	9	11	12	14	15	17	18	19	21	22	23	24	26	27	28	29	31	32	33	34	35	37	38	39	40	41	42	43	45	46	47	48	49	50	51	52	54	55	56	57	58	59	60	61	62	63
	Dev%	T3	0	0	0	0	0	2	-4	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	-4	-3	-3	-2	-2	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		T1	1	1	2	3	3	4	-3	-3	-2	-2	-1	-1	-1	0	0	0	1	1	1	1	1	-2	-2	-2	-2	-2	-2	-1	-1	-1	0	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
		T2	1	0	0	0	0	2	-3	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	-4	-3	-2	-2	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bzip2		M	9	11	12	14	15	17	21	22	23	24	29	31	32	33	34	35	37	38	39	40	41	42	43	45	46	50	51	52	54	55	56	57	58	60	61	62	63									
	Dev%	T3	0	-1	-1	3	-3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	0	2	0	0	1	0	0	-1	0									
		T1	-9	-7	-5	2	-4	2	-14	-10	-8	-6	-4	1	3	5	8	10	14	17	19	21	23	25	27	32	34	-26	-21	-14	3	C	C	C	C	C	C	C	C									
		T2	0	-1	-1	3	-3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	0	1	0	0	1	0	0	-1	0									
Soplex		M	9	14	15	17	18	19	21	22	23	24	26	31	34	35	37	39	41	42	43	45	46	47	48	49	50	51	52	54	55	56	57	58	59	60	61	62	63									
	Dev%	T3	0	0	0	-1	0	1	1	1	0	1	0	0	0	0	-1	0	-1	-1	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									
		T1	9	-16	-11	-5	-1	2	7	9	10	13	14	-4	-8	-9	-11	-8	-6	-4	-2	7	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C									
		T2	0	0	0	0	0	1	1	1	0	1	1	0	1	0	-1	0	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									
Omnetpp		M	17	18	19	21	22	23	24	26	27	28	29	31	32	33	34	35	37	38	39	40	41	42	43	45	46	47	48	50	51	52	54	55	60	61	62	63										
	Dev%	T3	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	-2	-1	-1	0	0	1	0	0	2	0	2	0	0	-1	0	-1	0										
		T1	-21	-18	-16	-11	-9	-8	-6	-3	-3	-2	0	2	3	4	5	5	7	7	7	6	6	7	8	9	10	9	9	10	7	8	6	6	-5	-4	-4	-3										
		T2	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	-2	-2	-1	-1	0	1	0	0	2	0	2	0	0	-1	0	-1	0										
Libquantum		M	14	15	17	18	19	23	24	28	29	31	32	33	34	35	37	41	42	43	45	46	47	48	49	50	51	52	54	55	56	57	58	59	60	61	62	63										
	Dev%	T3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										
		T1	9	13	19	22	25	-3	0	-22	-19	-14	-12	-10	-7	-5	-1	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C										
		T2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										
Milc		M	9	11	12	14	15	17	18	19	21	22	23	24	26	27	28	29	31	32	33	34	35	39	41	52	54	55	57	60	61																	
	Dev%	T3	0	1	1	-3	-1	2	-1	-1	0	1	0	0	0	0	0	0	1	2	3	4	-1	0	0	0	0	0	0	0	0																	
		T1	-8	-5	-3	-6	-4	-1	-3	-2	2	4	4	6	9	11	13	15	19	21	23	24	22	15	20	-102	-77	-64	-34	-19	C																	
		T2	-1	1	1	-3	-2	1	-1	-1	0	1	0	0	0	0	0	0	1	2	3	4	-1	0	0	0	0	0	0	0	0																	

Table 1: Accuracy of Predicted Cache Misses in "PFE" (T1), "BME" (T2), "BCPEM" (T3)

"PFE" in NVM cache performance evaluation, accuracy and efficiency of "BCPEM" was also checked in this experiment by replacing linear interpolation with "PFE". This test is referred to as 'T1'.

To conduct the tests, we chose an NVM cache configuration with two sets, associativity 64, line size 64 Bytes and FIFO replacement policy. The cache is an instruction cache. Line wear out can bring the associativity down to 8 in any cache set. Associativity below 8 makes a cache set dead. Therefore, total 3,249 configurations can be generated in this cache due to line wear out in different cache sets. Initially, we used six SPEC CPU 2006 applications (i) 401.bzip2, (ii) 471.omnetpp, (iii) 462.libquantum, (iv) 410.bwaves, (v) 433.milc, and (vi) 450.soplex. No embedded system benchmark application was used in our experiment as "BCPEM" has nothing to do with the features tested through those applications. "BCPEM" is equally efficient for embedded applications as well as general purpose applications. "BCPEM" relates to embedded systems due to its application specific nature and its relationship with NVM/hybrid caches which are preferred for embedded systems. SPEC CPU 2006 applications were executed to complete the first 600 Million instructions in Intel instrumentation tool pinatrace [2] to generate the processor's byte access trace files. During this process, input files for "BCPEM" were also generated. As SPEC CPU2006 applications can run upwards of a few billions of instructions, to reduce our experimentation time, we took only the first 600 Million instructions.

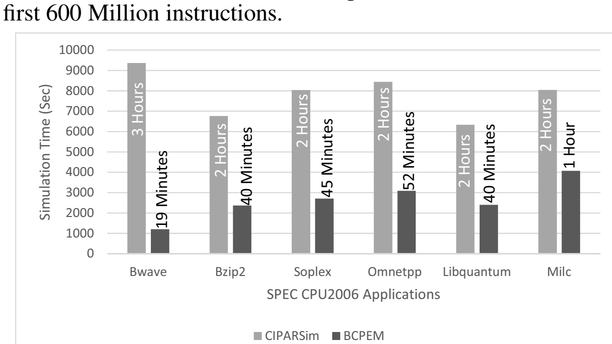

Figure 3: Single Cache Set Performance Evaluation Time

In Table 1, we present the variations observed in the predicted ns in 'T1', 'T2' and 'T3' compared to the actual ns (found in "CIPARSim" simulation) for the six SPEC CPU 2006 applications. The results in this table are collected from cache set index 1 (when index 0 is the first set). In Table 1, the first column lists the applications tested. For each application segment, in the second, third and fourth to the top row (the row that shows the associativities (M)), the percentage of deviation observed in the predicted ns compared to the actual ns are shown for 'T3', 'T1' and 'T2' respectively. For example, for application bwaves and $M = 9$, the n predicted by 'T1' was 1% smaller than the actual n. That is why, percentage of deviation is 1 (i.e. when predicted n is bigger than actual n, percentage of deviation is a negative number). The Ms that are not presented in this table are the simulated Ms whose ns are provided as the known data points in 'T1', 'T2' and 'T3'. Note that "PFE" is just a modeling equation to model the effect of associativity on the number of cache misses in a cache set. Therefore, without simulating all the associativities and without using the simulation results in regression analysis, it was not possible to reasonably approximate the values of n^*, n_0, M^* and M_0 in 'T1'. As we were using "PFE" for cache miss prediction in 'T1' to avoid simulation for some of the associativities possible in an NVM cache set, we did not have the freedom to simulate all the associativities possible in that cache set. To be fair, we used the same known data points (which are suggested by "BPM") in all of these three tests (i.e. 'T1', 'T2' and 'T3'). Moreover, 'C' value indicates that 'T1' failed to predict n for the corresponding M.

Form Table 1, it can be seen that 'T2' and 'T3' managed to predict n accurately in almost every case. Very rarely, up to $\pm 4\%$ difference is observed, which is insignificant in cache performance evaluation. To make these accurate predictions, minimum 11 (for application bwaves) and maximum 28 (for application Milc) known data points were used to obtain ns for 57 data points (associativities M=64 to 8) in cache set index 1. However, for 'T1', the situation is completely opposite. Very frequently failure and 34% to -102% differences are observed among the predicted and actual ns for 'T1'. From the experiment results, it is proven that "BME" can model the effect of M on n in a cache set almost accurately. However, in terms of accuracy and efficiency, "BME" is neither better nor worse compared to linear interpolation when used in "BPM". Therefore, due to less complexity and shorter execution time, linear interpolation is the right choice in "BPM". "PFE" is not useful at all in "BPM". Utilizing linear interpolation based "BPM", "BCPEM" can complete performance evaluation of cache set index 1 efficiently and accurately without simulating all the 57 associativities possible. If "CIPARSim" was used to evaluate the performance of the same cache set, 57 time consuming simulations were necessary to perform. Therefore, "BCPEM" has a great potential to accelerate cache performance evaluation phase resource generously in design space exploration.

In the second part of our experiment, we were interested to know how much time can be reduced by "BCPEM" compared to "CIPARSim" to collect n in all the possible configurations of a single NVM/hybrid cache set. This experiment was necessary to ensure that simulating cache sets separately (rather than entire cache configuration) is not playing the major role in acceleration, but "BPM" is the main cause of acceleration in "BCPEM" over "CIPARSim". For this purpose, we recorded the total time consumed by "BCPEM" and "CIPARSim" to collect n for all the 57 associativities (M=64,63,62,...8) in cache set index 1 in the previous experiment. The results are presented in Figure 3. From Figure 3, it can be seen that "BCPEM" completed the task 2 (for application Milc)

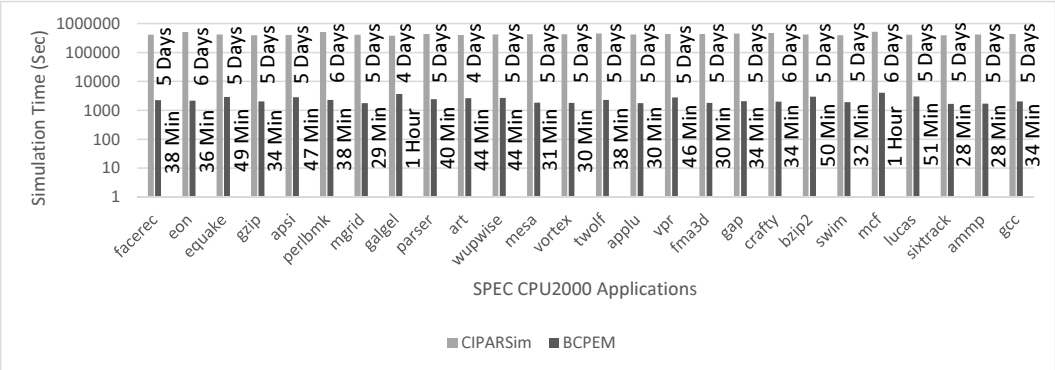

Figure 4: Entire Cache Performance Evaluation Time for SPEC CPU 2000 Applications

to 8 (for application bwaves) times faster than "CIPARSim". "BCPEM" could do the job so quickly because, cache misses were predicted for a large number of associativities using "BPM". Due to the use of "BPM", "BCPEM" will always be faster than "CIPARSim" no matter how small the application is.

The first two parts of our experiment made it evident that, as (i) the same cache simulator is used to simulate less and simpler configurations (i.e. cache set rather than an entire cache) and (ii) n^* and M^* estimation time is ignorable, time consumed by "BCPEM" can never exceed the time consumed by "CIPARSim". Therefore, in the third part, to figure out how much time can be reduced by "BCPEM" compared to "CIPARSim" for an entire benchmark suite, we chose SPEC CPU 2000 suite. To complete our experiment in reasonable time, we used (i) the cache configuration of the previous experiment but with reduce line size of 4bytes and (ii) trace of the first 100 Million instructions for each application. Figure 4 compares the total time consumed by "BCPEM" and "CIPRASim" to collect n of each SPEC CPU 2000 application on the 3,249 configurations possible in the target cache. For these applications too, "BCPEM" is 104 (for application "galgel", "BCPEM" took 1 hour and "CIPARSim" took 4 days) to 249 (for application "ammp", "BCPEM" took 28 minutes and "CIPARSim" took 5 days) times faster than "CIPARSim". For SPEC CPU 2000 applications, accuracy of predicted misses in "BCPEM" were within +6% and -5%. Therefore, "BCPEM" is an efficient acceleration technique for NVM/hybrid FIFO processor cache design space exploration.

7. CONCLUSION

Trace-driven single-pass cache simulators are not as successful in accelerating non-volatile/hybrid of volatile and non-volatile processor cache design space exploration as they are in SRAM processor cache design space exploration for application specific embedded systems. In this article, we propose a technique to accelerate the phase which is conventionally accelerated by single-pass cache simulators in processor cache design space exploration for application specific embedded systems. Utilizing a novel cache behavior modeling equation and a new accurate cache miss prediction mechanism, our proposed technique can accelerate NVM/hybrid FIFO processor cache design space exploration for SPEC CPU 2000 applications up to 249 times compared to the conventional approach.

8. REFERENCES

[1] A. V. Aho, P. J. Denning, and J. D. Ullman. Principles of optimal page replacement. *J. ACM*, 18(1):80–93, Jan. 1971.

[2] S. Berkowit. Pin - a dynamic binary instrumentation tool. *http://software.intel.com/en-us/articles*, 2013.

[3] J. Casmira, J. Fraser, D. Kaeli, and W. Meleis. Operating system impact on trace-driven simulation. In *Annual Simulation Symposium*, page 76, 1998.

[4] A. Dan and D. Towsley. An approx. analysis of the lru and fifo buffer replacement schemes. *SIGMETRICS Perf. Eval. Rev.*, 18(1):143–152, 1990.

[5] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 31(7):994–1007, 2012.

[6] J. Edler and M. D. Hill˙Dinero iv trace-driven uniprocessor cache simulator. *http://www.cs.wisc.edu/ markhill/DineroIV/*, 2004.

[7] P. A. Franaszek and T. J. Wagner. Some distribution-free aspects of paging algorithm performance. *J. ACM*, 21(1):31–39, Jan. 1974.

[8] M. S. Haque, J. Peddersen, A. Janapsatya, and S. Parameswaran. Dew: A fast level 1 cache simulation approach for embedded processors with fifo replacement policy. In *DATE'10*.

[9] M. S. Haque, J. Peddersen, and S. Parameswaran. Ciparsim: cache intersection property assisted rapid single-pass fifo cache simulation technique. In *ICCAD*, pages 126–133, 2011.

[10] P. Heidelberger and H. S. Stone. Parallel trace-driven cache simulation by time partitioning. In *WSC' 90*, pages 734–737.

[11] S. Irani, A. R. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. In *SODA*, 1992.

[12] A. Janapsatya, A. Ignjatović, and S. Parameswaran. Finding optimal l1 cache configuration for embedded systems. In *ASP-DAC '06*.

[13] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das. Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps. DAC '12.

[14] C. H. Kim, J.-J. Kim, S. Mukhopadhyay, and K. Roy. A forward body-biased low-leakage sram cache: device and architecture considerations. In *ISLPED*, 2003.

[15] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. *SIGARCH Comput. Archit. News*, 37(3):2–13, June 2009.

[16] M. Loghi, M. Poncino, and L. Benini. Cache coherence tradeoffs in shared-memory mpsocs. *ACM Trans. Embed. Comput. Syst.*, 5(2):383–407, May 2006.

[17] P. Mangalagiri, K. Sarpatwari, A. Yanamandra, V. Narayanan, Y. Xie, M. J. Irwin, and O. A. Karim. A low-power phase change memory based hybrid cache architecture. GLSVLSI '08.

[18] A. Milenković and M. Milenković. An efficient single-pass trace compression technique utilizing instruction streams. *ACM Trans. Model. Comput. Simul.*, 17(1), Jan. 2007.

[19] H. Noguchi et al. Ultra low power processor using perpendicular-stt-mram/sram based hybrid cache toward next generation normally-off computers. *Journal of Applied Physics*, 111, 2012.

[20] H. Noguchi et al. Highly reliable and low-power nonvolatile cache memory with advanced perpendicular stt-mram for high-performance cpu. *Symposia on VLSI Technology and Circuits*, 2014.

[21] E. J. O'Neil, P. E. O'Neil, and G. Weikum. An optimality proof of the lru-k page replacement algorithm. *J. ACM*, 46(1):92–112, Jan. 1999.

[22] S. Sheu. A 4mb embedded slc resistive-ram macro with 7.2ns read-write random-access time and 160ns mlc-access capability. In *ISSCC'11*.

[23] Y. C. Tay and M. Zou. A page fault equation for modeling the effect of memory size. *Perform. Eval.*, 63(2):99–130, Feb. 2006.

[24] N. Tojo, N. Togawa, M. Yanagisawa, and T. Ohtsuki. Exact and fast l1 cache simulation for embedded systems. In *ASP-DAC'09*.

[25] W.-H. Wang and J.-L. Baer. Efficient trace-driven simulation method for cache performance analysis. *SIGMETRICS Perform. Eval. Rev.*, 18(1):27–36, Apr. 1990.

[26] S. J. E. Wilton and N. P. Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31:677–688, 1996.

[27] Z. Wu and W. Wolf. Iterative cache simulation of embedded cpus with trace stripping. In *CODES*, pages 95–99, 1999.

[28] XTMP. The xtensa modeling protocol and xtensa systemc modeling for fast system modeling and simulation. *www.tensilica.com*.

[29] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA'09*.

This work was supported by Agency for Science, Technology and Research (A*STAR), Singapore under Grant No. 112-172-0010.