SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

January 2018, Vol. 61 012105:1–012105:14
doi: 10.1007/s11432-016-0408-1

Toward multi-programmed workloads with different memory footprints: a self-adaptive last level cache scheduling scheme

Jingyu ZHANG, Minyi GUO*, Chentao WU & Yuanyi CHEN

Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240*, China*

Received 15 June 2016/Accepted 21 September 2016/Published online 14 July 2017

Abstract With the emerging of 3D-stacking technology, the dynamic random-access memory (DRAM) can be stacked on chips to architect the DRAM last level cache (LLC). Compared with static random-access memory (SRAM), DRAM is larger but slower. In the existing research papers, a lot of work has been devoted to improving the workload performance using SRAM and stacked DRAM together, ranging from SRAM structure improvement, to optimizing cache tag and data access. Instead, little attention has been paid to designing an LLC scheduling scheme for multi-programmed workloads with different memory footprints. Motivated by this, we propose a self-adaptive LLC scheduling scheme, which allows us to utilize SRAM and 3D-stacked DRAM efficiently, achieving better workload performance. This scheduling scheme employs (1) an evaluation unit, which is used to probe and evaluate the cache information during the process of programs being executed; and (2) an implementation unit, which is used to self-adaptively choose SRAM or DRAM. To make the scheduling scheme work correctly, we develop a data migration policy. We conduct extensive experiments to evaluate the performance of our proposed scheme. Experimental results show that our method can improve the multi-programmed workload performance by up to 30% compared with the state-of-the-art methods.

Keywords 3D-stacking technology, cache architecture, cache scheduling, multi-programmed workloads, memory system, performance optimization

Citation Zhang J Y, Guo M Y, Wu C T, et al. Toward multi-programmed workloads with different memory footprints: a self-adaptive last level cache scheduling scheme. Sci China Inf Sci, 2018, 61(1): 012105, doi: 10.1007/s11432-016-0408-1

1 Introduction

The on-chip capacity of static random-access memory (SRAM) is limited by the chip area and expensive price. Whereas with the development of 3D integration technology [1–3], the 3D-stacked dynamic random-access memory (DRAM) can break the above barrier, achieving hundreds of megabytes or more [4]. In general, SRAM is faster but smaller, while DRAM is larger but slower. Since both DRAM and SRAM have their advantages and disadvantages, a lot of efforts are devoted to improving workload performance using DRAM and SRAM together, ranging from SRAM structure improvement [5, 6], to cache tag and data access optimization [7–9]. Most of recent researches on hybrid SRAM and DRAM caches focus mainly on enhancing the overall performance of SRAM (resp., DRAM) by utilizing the merit of DRAM (resp., SRAM). There are also many papers devoted to investigating workload performance:

* Corresponding author (email: guo-my@cs.sjtu.edu.cn)

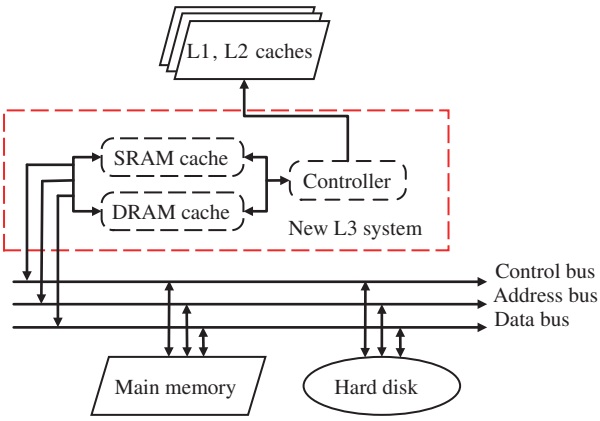

Figure 1 (Color online) Illustration of the architecture.

(1) For multi-programmed workloads, prior work discussed the issues of relieving memory contention [10, 11], workload balance [12, 13] and power related optimization [14]; (2) To improve the performance of memory-intensive workloads, many solutions (e.g., architecture design [15–17], OS level method [18–20] and feedback control [21, 22]) have also been proposed; (3) In the cache system, the improved cache architectures [4, 9, 23, 24] and 3D-stacked DRAM technologies [25–27] are used to achieve better workload performance; and so on (a broader overview of related work will be covered in Section 2). Instead, little attention has been paid to designing a last level cache (LLC) scheduling scheme for multi-programmed workloads with different memory footprints.

Motivated by this, we attempt to design an LLC scheduling scheme to improve the performance of workloads with different memory footprints. In the real scenarios, the multi-programmed workloads should be mixed with different programs (e.g., computational finance applications, computer vision applications, computer animation programs, data mining applications, or data center server workloads), and each program has its own features and memory footprint. Our key observations are: (1) if a program is with large memory footprint and sensitive to the LLC's capacity, then using DRAM as the LLC could bring better performance; (2) otherwise, using SRAM as the LLC could be better (more detailed explanations will be discussed in Section 3). To determine which LLC module (SRAM or DRAM) benefits specific programs with different memory footprints, we formulate the mathematical models which provide us a basic intuition. Then, we introduce a self-adaptive LLC scheduling scheme that integrates both SRAM and DRAM. Figure 1 shows the architecture integrating our scheduling scheme (the system marked by the dashed line is the new L3 system). The core of our scheduling scheme is to selectively use SRAM or DRAM, based on specific workloads.

In a nutshell, our main contributions can be summarized as follows:

- We formulate the mathematical models which provide us a basic intuition for determining which LLC module could be better for the program with a specific memory footprint.

- For multi-programmed workloads, we present a self-adaptive LLC scheduling scheme selectively using SRAM or DRAM as the LLC.

- We conduct extensive experiments based on the widely used benchmarks. The experimental results validate the effectiveness and efficiency of our proposed LLC scheduling scheme.

The remaining paper is organized as follows. Section 2 reviews the previous work. The preliminaries are given in Section 3. We describe the details of our proposed method in Section 4. We experimentally validate the effectiveness and efficiency of our proposed method in Section 5. Finally, Section 6 concludes this paper.

2 Related work

In this section, we review previous work most related to ours. Specifically, Subsection 2.1 reviews hybrid SRAM and DRAM caches. Subsection 2.2 discusses the solutions for multi-programmed workloads. Moreover, we cover memory-intensive workloads, 3D-stacked DRAM, cache architecture design in Subsections 2.3–2.5, respectively.

2.1 Hybrid SRAM and DRAM caches

A lot of efforts have been devoted to employing both SRAM and stacked DRAM for various goals, including SRAM structure improvement [5, 6], and optimization for cache tag and data access [7–9], etc. For example, Hameed et al. [5] proposed a DRAM placement policy to increase the workload performance. In this work, after a cache miss in the primary SRAM L3 cache, a stacked DRAM module will be accessed. Hundal et al. [6] presented an SRAM-DRAM combination memory which is used as the on-chip cache. A stacked DRAM secondary cache is used to improve the SRAM cache performance in this work. In [28,29], via compound-access scheduling, Loh and Hill made hits faster than just storing tags in stacked DRAM caches, where SRAM is used as a hardware to track the presence or absence of cache blocks. Ref. [30] proposed a heterogeneous stacked DRAM chip design that can better exploit spatial locality by tightly integrating a small SRAM cache. Huang et al. [8] proposed a small SRAM tag cache to reduce DRAM cache latency. Hameed et al. [7] introduced a novel concept of small and low latency tag-cache structures that can quickly determine whether an access to the large L3 SRAM/L4 DRAM caches will be a hit or a miss. These papers always use SRAM or DRAM as the primary LLC to improve the workload performance, while our work adaptively employs SRAM and DRAM LLCs.

2.2 Multi-programmed workloads

For multi-programmed workloads, there are numerous papers studied the performance optimization. The prior work covers relieving memory contention [10, 11], workload balance [12, 13], power related optimization [14]. In [11], a memory scheduling algorithm called time-based least memory intensive scheduling was proposed according to the memory contention situation. Based on the previous work, Ref. [10] proposed the adaptive time-based least memory intensive scheduling. Chen et al. [12] proposed a demand-aware work-stealing task scheduler, with which a work-stealing program uses cores according to its realtime demand, then they presented the adaptive version in [13]. In [31], a fast, automated technique was proposed for accurate on-line estimation of the performance and power consumption of interacting processes in a multi-programmed, multi-core environment. Suo et al. [32] investigated system level speedup oriented cache partitioning for multi-programmed workloads, according to current performance status and misses of all the possible partitions. In [14], authors presented a method that extends auto-tuning for multi-programmed workload to reduce the energy-delay product. These designs mainly care about the workload scheduling for multi-programmed workloads to improve the performance or energy consumption, however our work cares about the last level cache scheduling.

2.3 Memory-intensive workloads

Memory-intensive workloads are also known as the workloads with large memory footprints. To improve the performance of memory-intensive workloads, researchers have paid a lot of attention. The directions in this field include architecture design [15–17], OS level method [18–20], and feedback control [21, 22].

Huang et al. [16] proposed a novel high-level synthesis methodology for designing multi-partitioned architectures for memory-intensive workloads. In this work, authors employed an iterative improvement strategy to determine the best way of distributing array data into physical memory. Huang et al. [17] also presented the techniques for memory-intensive applications considering various factors including data access locality, balanced workloads, inter-partition communication. Castellana and Ferrandi [15] proposed an adaptive architecture to exploit the available parallelism for memory-intensive workloads. Yi et al. [20] presented optimization methods including matrix transposition and kernel fusion to improve the

memory-intensive workload's performance on GPGPU. Athanasaki et al. [18] explored the performance limits by evaluating the tradeoffs between instruction level parallelism and thread-level parallelism for memory-intensive workloads. Chun et al. [19] presented a task mapping scheme for memory-intensive software-pipelined workloads. Kirovski et al. [33] developed a new approach for area optimization for memory-intensive workloads. This approach uses basic block relocation in order to reduce the number of cache misses. Qin et al. [22] proposed a feedback control mechanism to enhance the performance of memory-intensive workloads. In [21], another feedback control mechanism was proposed to improve overall performance of a cluster for memory-intensive workloads.

Unlike these papers which mainly focus on the memory-intensive workloads, our work pay the attention for multi-programmed workloads with different memory footprints.

2.4 3D-stacked DRAM

There are many papers investigating the emerging 3D-stacked DRAM. Jevdjic et al. [26] introduced footprint cache, an efficient die-stacked DRAM cache design for server processors. Ref. [34] detailed how to provide reliability, availability, and serviceability support for stacked DRAM cache architectures in a practical and cost-effective manner. Akin et al. [25] presented a two pronged approach for efficient data reorganization using 3D-stacked DRAM. Oskin and Loh [27] analyzed the performance bottlenecks in OS page caches, and proposed two techniques that make the OS approach viable using stacked DRAM. Most of work in these papers did not cover the combination with SRAM and different types of programs, which motivates our work.

2.5 Cache architecture design

A lot of efforts also have been made in the literature to improve cache architecture design. Qureshi and Loh [9] proposed Alloy cache which can eliminate the delay due to tag serialization by streaming tag and data together in a single burst. A tagless cache architecture for large in-package DRAM caches was introduced in [4]. Xiao et al. [24] presented a dual queues cache replacement algorithm based on sequentiality detection to improve the cache design. Jaleel et al. [23] presented directions for further research to maximize performance of exclusive cache hierarchies. Chou et al. [35] proposed CAMEO which not only makes stacked DRAM visible as part of the memory address space but also exploits data locality. Ou et al. [36] proposed a penalty aware memory allocation scheme, which uses impacts on service time to determine where a unit of memory space should be (de)allocated. These papers focus on the cache inherent design, while our work focuses on SRAM/DRAM hybrid LLC scheduling.

3 Preliminaries

3.1 Typical cache hierarchy

As we know, in a typical computer system, the caches (L1, L2 and L3 caches) are mainly used to alleviate the access latency gap between CPU and main memory, since they can provide the requested data to CPU instantly [37] (see Figure 2). The early system used 2 level caches [6,37]. In the recent literature [5,7,29], 3 level and/or 4 level cache hierarchies are widely used. From level 1 to last level, the cache capacity is increasing, and the last level connects the main memory via system bus interface. In this paper, we focus on the last level cache (LLC). For the convenience of discussion, we assume the 3 level cache hierarchy is the default setting, although our scheduling scheme can be applied to the 4 level cache hierarchy.

To measure the cache system performance (note that, cache system performance reflects the workload performance, we may use them interchangeably later), we use the average memory access time to quantify it. Denote by T_{ama}, $H_{\text{L}x}$, $M_{\text{L}x}$, Mp_{LLC} the average memory access time, the level x cache hit time and miss rate, the miss penalty of LLC, respectively. For a classical 3 level cache hierarchy system, we have

$$T_{\text{ama}} = H_{\text{L1}} + M_{\text{L1}} \times (H_{\text{L2}} + M_{\text{L2}} \times (H_{\text{LLC}} + M_{\text{LLC}} \times \text{Mp}_{\text{LLC}})). \tag{1}$$

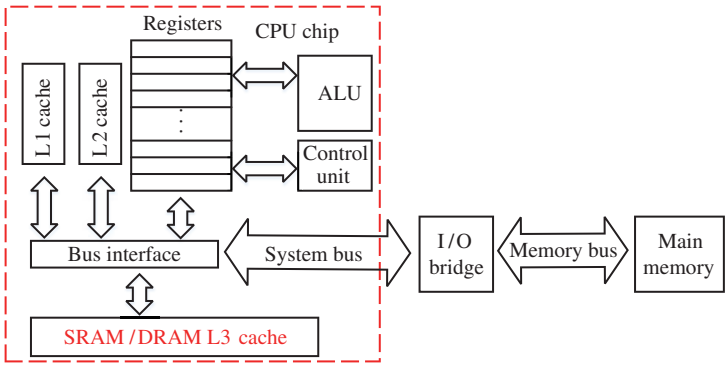

Figure 2 (Color online) Cache system in current computer architecture.

Next we develop our mathematical models based on the expression above.

3.2 Mathematical models

Considering a workload containing n programs with different memory footprints, we next formulate the mathematical models that can provide us some insights. Let $f(x_i)$, $A(x_i)$, $H_{\rm S}$, $M_{\rm S}$, $H_{\rm D}$, $M_{\rm D}$ be the total memory access time of ith processed program, the number of memory access requests, hit time and miss rate of SRAM LLC, hit time and miss rate of DRAM LLC, separately. T_{x_i} denotes the average memory access time of the ith program. Let $H_{\rm L1}(x_i)$, $M_{\rm L1}(x_i)$ be the L1 cache hit time, miss rate of program i. $H_{\rm L2}(x_i)$, $M_{\rm L2}(x_i)$, $H_{\rm LLC}(x_i)$ and $M_{\rm LLC}(x_i)$ have the similar meanings as $H_{\rm L1}(x_i)$ and $M_{\rm L1}(x_i)$. $\mathrm{Mp}_{\rm LLC}(x_i)$ is the LLC miss penalty. By Formula (1), we have

$$\begin{cases} \min \sum_{i=1}^{n} f(x_i) = \min \sum_{i=1}^{n} \sum_{j=1}^{A(x_i)} T_{x_i}, \\ \text{s.t. } H_{\rm S} \leqslant Hi_{\rm LLC}(x_i) \leqslant H_{\rm D}, \\ \quad M_{\rm D} \leqslant Mi_{\rm LLC}(x_i) \leqslant M_{\rm S}, \qquad\qquad i = 1, 2, \ldots, n, \end{cases} \tag{2}$$

$$\begin{aligned} T_{x_i} = {} & H_{\rm L1}(x_i) + M_{\rm L1}(x_i) \times (H_{\rm L2}(x_i) + M_{\rm L2}(x_i) \\ & \times (H_{\rm LLC}(x_i) + M_{\rm LLC}(x_i) \times \mathrm{Mp}_{\rm LLC}(x_i))). \end{aligned} \tag{3}$$

By Formulas (2) and (3), we can get the following:

$$\begin{aligned} \min \sum_{i=1}^{n} f(x_i) &= \min \sum_{i=1}^{n} \sum_{j=1}^{A(x_i)} T_{x_i} \\ &= \min \sum_{i=1}^{n} \sum_{j=1}^{A(x_i)} (H_{\rm L1}(x_i) + M_{\rm L1}(x_i) \times H_{\rm L2}(x_i) \\ &\quad + M_{\rm L1}(x_i) \times M_{\rm L2}(x_i) \times H_{\rm LLC}(x_i) \\ &\quad + M_{\rm L1}(x_i) \times M_{\rm L2}(x_i) \times M_{\rm LLC}(x_i) \times \mathrm{Mp}_{\rm LLC}(x_i)). \end{aligned} \tag{4}$$

The values $H_{\rm L1}(x_i)$, $M_{\rm L1}(x_i)$, $H_{\rm L2}(x_i)$ and $M_{\rm L2}(x_i)$ are constant (as they are related to the L1 and L2 caches). The total number of memory access requests $A(x_i)$ is also a constant value. For Eq. (4), we replace those parts with constant values using A, B, C, and D, respectively. It can be rewritten as follows:

$$\min \sum_{i=1}^{n} f(x_i) = \sum_{i=1}^{n} \sum_{j=1}^{A} (B + C \times H_{\rm LLC}(x_i) + D \times M_{\rm LLC}(x_i)). \tag{5}$$

For the 3D-stacked DRAM cache, it will significantly improve the LLC capacity, so $M_{\rm LLC}$ will be decreased. However, the DRAM cache hurts the $H_{\rm LLC}$. The SRAM cache has the fast LLC hit time $H_{\rm LLC}$, while it will increase the $M_{\rm LLC}$. Eq. (5) gives us a basic intuition, i.e., it is a trade-off between the LLC hit time $H_{\rm LLC}$ and the LLC miss rate $M_{\rm LLC}$.

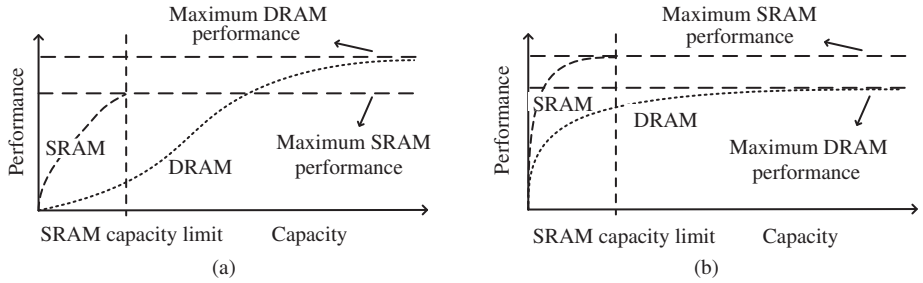

Figure 3 Performance comparison for different L3 caches. (a) and (b) illustrate the performance characteristics for a program with large memory footprint and a program with a small memory footprint, respectively.

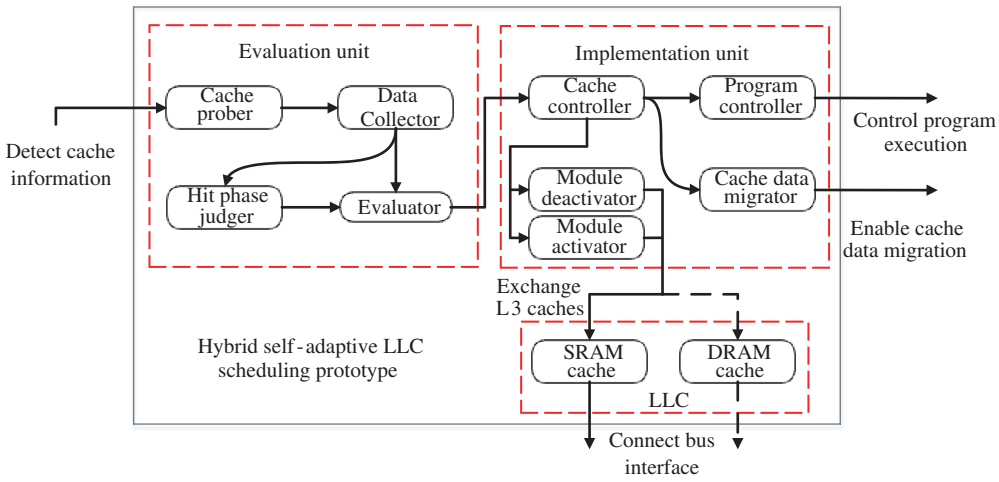

Figure 4 (Color online) Prototype framework.

3.3 Preliminary observations

To help us make an appropriate trade-off, we study some basic characteristics of workloads with different memory footprints. Figure 3 illustrates the general map (extracted from our preliminary experimental results). Specifically, we observe that, for a program (e.g., mcf[1)]) with large memory footprint, the large DRAM LLC outperforms SRAM LLC in terms of program performance. See Figure 3(a). This is mainly because larger 3D-stacked DRAM LLC can reduce dramatically the number of the main memory access, which has significantly impact on the performance of programs with large memory footprints. On the other hand, we observe that, a program with small memory footprint (e.g., cactusADM) achieves higher performance when the small and fast SRAM LLC is used. See Figure 3(b).

The above observations imply that, if one wants to improve the workload performance, it is needed to identify which LLC module (SRAM or DRAM) is appropriate for specific programs with different memory footprints. In the next section, we shall design a last level cache scheduling scheme based on the key observations.

4 Proposed scheme

Our proposed LLC scheduling scheme can be applied to modern computer system seamlessly. Figure 4 illustrates the integrated framework of our prototype based on the scheduling scheme. In brief, this prototype uses both SRAM and DRAM at the last level of cache hierarchy. To coordinate the LLC scheduling, we employ (1) an evaluation unit and (2) an implementation unit. These two units coopera-

1) SPEC CPU 2006. https://www.spec.org/cpu2006/.

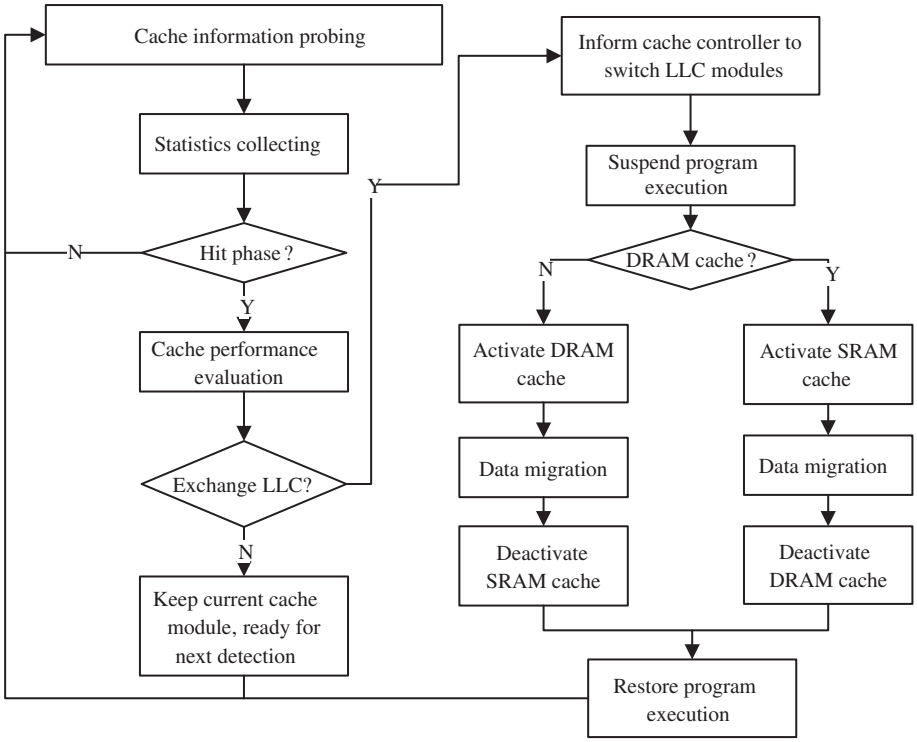

Figure 5 LLC scheduling work flow.

tively contribute to appropriately choosing the LLC module (SRAM or DRAM) through interacting with L1, L2 caches and main memory.

4.1 Overview

The general running process of the LLC scheduling prototype is shown in Figure 5. Evaluation unit and implementation unit work together to implement the LLC scheduling scheme in the computer system.

• **Evaluation unit.** This unit is used to probe caches and collect the cache statistics. It analyzes the cache statistics to decide which LLC module should be selected. It contains four components (see Figure 4): (1) cache prober; (2) data collector; (3) hit phase judger; and (4) evaluator.

• **Implementation unit.** This unit is used to switch the accessible LLC modules according to the messages from evaluation unit. For the function of LLC switching, it employs five parts (see Figure 4): (1) cache controller; (2) program controller; (3) cache data migrator; module (4) activator and (5) deactivator.

Each component in evaluation and implementation units takes less than 1 KB storage space. In total, the storage cost of the evaluation and implementation units can be controlled in 10 KB. In other words, the total storage cost will be less than 10 KB. Next, we detail the function of each component in these two units.

4.2 LLC scheduling

In the proposed prototype, the LLC scheduling includes two processes: (1) LLC estimation and (2) LLC switching. The main scheduling scheme is shown in Algorithm 1 (The major symbols used in this algorithm are listed in Table 1). We discuss these two processes afterwards.

4.2.1 *LLC estimation*

The LLC estimation is the first step of self-adaptive LLC scheduling. To deep understand the LLC estimation policy, it is needed to explain some preliminaries. The execution duration of programs could be divided into two phases: the miss phase and the hit phase. In the miss phase, the cache is warming up

Algorithm 1 LLC scheduling

```
Initialize detection environment, clear states;
// Cache statistics analysis
Retrieve M_c, M_l during the evaluation cycle;
while |M_c − M_l| ⩽ Δ_p do
    // LLC modules scheduling
    Map M_S, M_D, M_L1, M_L2, Mp_LLC, retrieve T_S, T_D;
    if C_D = 1 then
        if T_S < T_D then
            Activate C_c, D_M, C_S, suspend P_c;
            D_d ⇒ D_S;
            Deactivate C_D, D_M, C_c, restore P_c;
        end if
    else
        if T_S > T_D then
            Activate C_c, C_D, D_M, suspend P_c;
            D_S ⇒ D_D;
            Deactivate C_S, D_M, C_c, restore P_c;
        end if
    end if
end while
```

Table 1 Symbolic table

Item	Description
$M_{\rm c}, M_{\rm l}$	LLC miss rate during current and last cache evaluation cycle
$\Delta_{\rm p}$	The predefined percentage change threshold
$M_{\rm L1}, M_{\rm L2}$	L1, L2 cache miss rates
$M_{\rm S}$, $M_{\rm D}$	SRAM and DRAM LLC miss rates
$\mathrm{Mp}_{\rm LLC}$	LLC miss penalty
$T_{\rm S}, T_{\rm D}$	Average memory access time for SRAM and DRAM LLCs
$C_{\rm S}, C_{\rm D}$	Indicators for SRAM and DRAM LLCs
$C_{\rm c}$	Cache controller indicator
$D_{\rm M}$	Cache data migrator indicator
$D_{\rm S}, D_{\rm D}$	Data resided in SRAM and DRAM LLCs
$D_{\rm d}$	Dirty data in current cache module
$P_{\rm c}$	Current running program

and the hit rate is pretty low but is increasing quickly; in hit phase, hit rate will retain at a stable level and it can reflects the actual program characteristics. Our prototype only uses hit phase information to estimate whether need to exchange LLC modules.

With the above concepts in mind, we now explain how the self-adaptive LLC scheduling scheme of our prototype works. During the program execution, the cache prober probes frequently the current working cache system. The obtained cache information will be stored in data collector. Data collector periodically sends the cache statistic messages to hit phase judger. Hit phase judger uses the cache statistics to judge whether the testing program is in hit phase. For a new program to the prototype, when it reaches the hit phase, the cache information will be stored in the data collector. And then the current LLC module will be changed. In the changed environment, after the program enters hit phase, the evaluator will estimate the current LLC performance and choose the corresponding cache scheduling policy. The evaluator makes our self-adaptive LLC scheduling system be able to switch the LLC modules according to current situation. If the LLC needs to be changed, the implementation unit will finish the LLC switching. Otherwise, program will be ready for the next cache evaluation cycle.

4.2.2 *LLC switching*

If cache controller receives the scheduling message, it will inform program controller to suspend the program execution. Then the module activator will enable the target cache module (which is deactivated

now). Then the cache data migrator should migrate cache data from current cache module to the target. For the correctness of switching, We propose a migration policy, which contains two cache data migration modes: (1) switching SRAM to DRAM and (2) switching DRAM to SRAM. The migration operation of the policy are presented as follows.

• Switching SRAM to DRAM: If the current working LLC is the SRAM cache, cache data migrator will directly transmit the data in SRAM cache to the DRAM cache (since the capacity of SRAM cache is smaller than that of DRAM cache);

• Switching DRAM to SRAM: When DRAM cache is current LLC, the cache data transmitter will only write the dirty data in DRAM cache to SRAM cache (for a short transmission time). Since the capacity of DRAM is larger than that of SRAM cache, the data overflow may occur when this migration mode is working. If data overflow happens, the cache data overflow will be written back to main memory.

Depending on different real transmission data bus and the migration data, the program suspending time may vary. After the cache data migration, module deactivator disables current LLC module. Finally, program controller restores the program execution to finish the LLC switching.

5 Experiments

In this section, we first describe the experimental settings (Subsection 5.1), and then report the experimental results (Subsection 5.2).

5.1 Experimental settings

Our experimental platform is a Pin-based system-level architecture simulator, MACSIM[2)]. In our experiments, we use 20000 instructions as the cache evaluation cycle, and Δ_{p} is conservatively set to 1%. The simulation trace files are generated by Pin[3)]. This work simulates five computer system architectures as follows.

• SRAM LLC architecture: In this architecture, there is only one SRAM cache used as LLC. This is a classical computer architecture.

• DRAM LLC architecture: In this computer system, there is only one DRAM cache used as LLC.

• HMP architecture: This is an improved DRAM LLC architecture introduced in [38]. In this architecture, a low-cost hit-miss predictor (HMP) is proposed. For brevity, we call it HMP architecture.

• Self-adaptive LLC scheduling (SLS) architecture: This architecture is based on our self-adaptive LLC scheduling scheme, including both the SRAM and DRAM caches.

• SLS+HMP architecture: This is an improved SLS architecture, using HMP technique for the DRAM cache.

The rough configurations of these five architectures are shown in Table 2. Note that, the DRAM cache doubles both the clock frequency and the number of channels of the main memory [9, 38]. Eight benchmarks from SPEC CPU2006 suite are chosen for our tests, these benchmarks are widely used in the literature [9, 38, 39]. To achieve diverse simulations, all those benchmarks we selected are mixed into 10 multi-programmed workloads. Table 3 shows some details about our workloads. The mixing rule of this table is: each workload should contain both the benchmarks with small memory footprints and the benchmarks with large memory footprints.

5.2 Experimental results

We analyze experimental results from three aspects: (1) the total number of cache write-back, (2) the average memory access time, and finally (3) the system performance. The cache write-back means writing the cache data back to main memory. The cache write-back operation happens when the main memory

2) MACSIM. https://code.google.com/p/macsim/.
3) Pin. https://software.intel.com.

Table 2 Experimental settings

CPU	Type	Intel's Sandy Bridge, x86, Out-of-Order
	Frequency	4 GHz
	Other parameters	Round Robin fetch policy, 3-level cache hierarchy
Caches	L1 cache	64 KB, 64 sets, 8 associates, 3 cycle latency
	L2 cache	256 KB, 256 sets, 8 associates, 8 cycle latency
	SRAM cache	4 MB, 64 Bytes cacheline size, 30 cycles latency
	DRAM cache	64 MB, 1.6 GHz, 16 channels, tCAS-tRCD-tRP 7-7-9
Main memory	Frequency	0.8 GHz
	Scheduling	FRFCFS policy
	Other parameters	4 Bytes bus width, 16 banks, 8 channels, 2 KB row-buffer

Table 3 Multi-programmed workloads

Workload	Benchmark
Mix-1	zeusmp, bzip2, mcf, lbm
Mix-2	cactusADM, zeusmp, mcf, lbm
Mix-3	bzip2, mcf, 2×lbm
Mix-4	mcf, lbm, bwaves, zeusmp
Mix-5	mcf, bwaves, GemsFDTD, cactusADM
Mix-6	lbm, bwaves, mcf, bzip2
Mix-7	mcf, soplex, cactusADM, bzip2
Mix-8	lbm, cactusADM, mcf, zeusmp
Mix-9	zeusmp, cactusADM, mcf, bzip2
Mix-10	2×bzip2, zeusmp, mcf

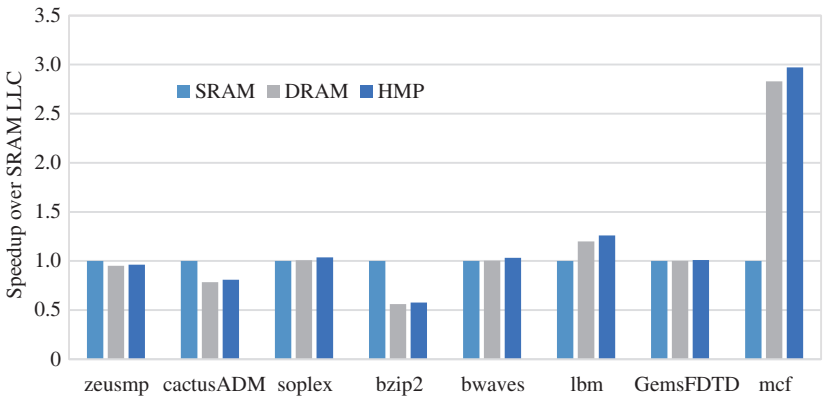

Figure 6 (Color online) Normalized benchmark performance comparison. The baseline is the performance of the SRAM LLC architecture.

data need to be written into cache after a cache miss. The number of cache write-back can indicate the access frequency of main memory. The less number means the less main memory accesses.

5.2.1 *Benchmark impacts*

From Figure 6, we can notice the DRAM LLC-based architectures (DRAM and HMP) improve the performance for benchmarks lbm and mcf which have large memory footprints. On the other hand, the DRAM LLC will slow down some benchmarks with small memory footprints, especially for the bzip2. Next we discuss the reasons from two other aspects.

Figure 7 indicates the DRAM LLC-based architectures decrease the number of cache write-back for all benchmarks. Since DRAM and HMP architectures have the same LLC capacity, their numbers of memory write-back are the same. The DRAM LLC has larger capacity, so more data could reside in it and main memory will be accessed less. Compared with other benchmarks, memory-intensive benchmarks

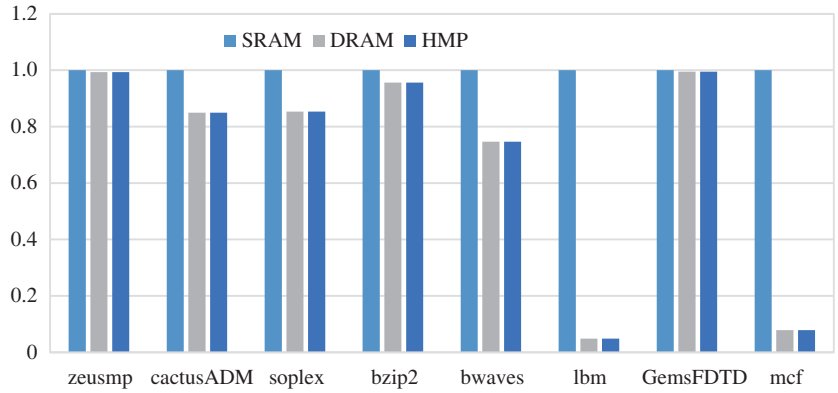

Figure 7 (Color online) Normalized cache write-back comparison. For all benchmarks no matter whether they are memory-intensive ones, the number of cache write-back reduced via using DRAM LLC. The baseline is the number of cache write-back with the SRAM LLC.

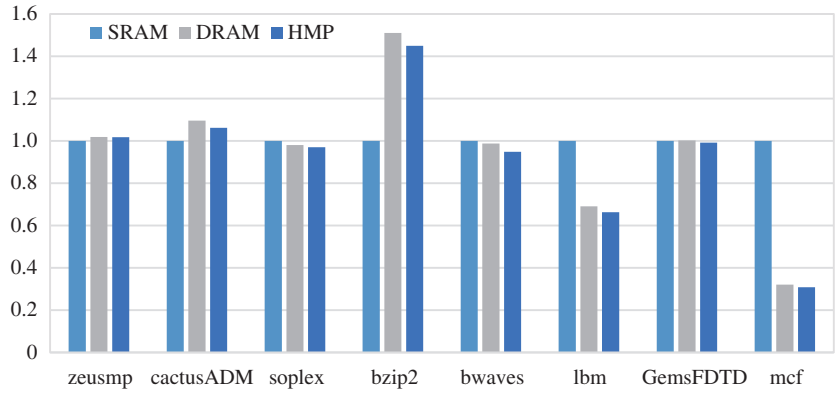

Figure 8 (Color online) Normalized average memory access time comparison. This figure indicates DRAM LLC impacts average cache latency differently. The baseline is the average cache latency of benchmarks with the SRAM LLC.

could achieve a greater reduction in the number of cache write-back. For either lbm or mcf, DRAM LLC can reduce the number of cache write-back by more than 85%. That is why DRAM LLC can greatly improve the performance of these two benchmarks. Therefore, for some benchmarks with small memory footprints, the reduction may be inconspicuous.

According to experimental results in Figure 8, we notice that the DRAM LLC-based architectures decrease the average memory access time for benchmarks lbm and mcf which have large memory footprints. However, for some benchmarks with small memory footprints, the average cache latency could be increased by using DRAM LLC. This is because, these benchmarks do not need a large LLC. The LLC access latency impacts these benchmarks greatly, and large LLC capacity makes no sense for benchmarks with small memory footprints. Meanwhile with the increased LLC access latency, memory requests should take considerable long time to access data resided in the DRAM LLC. It indicates cache access latency plays a relatively important role in computer systems for some programs with small memory footprints.

5.2.2 *Multi-programmed workload performance*

In this section, we first show the impacts on (1) cache write-back and (2) average memory access time. Finally, the workload performance results and analysis are presented.

Figure 9 shows the average memory access time and cache write-back comparisons impacted by different architectures. Sometimes SRAM LLC architecture has lower average memory access time than the DRAM LLC-based architectures, sometimes not. While the SLS-based architectures always achieve the lower average memory access time than the other two relative architectures (e.g., SLS+HMP has lower average memory access time than SRAM and HMP). Since the DRAM LLC-based architectures always use the

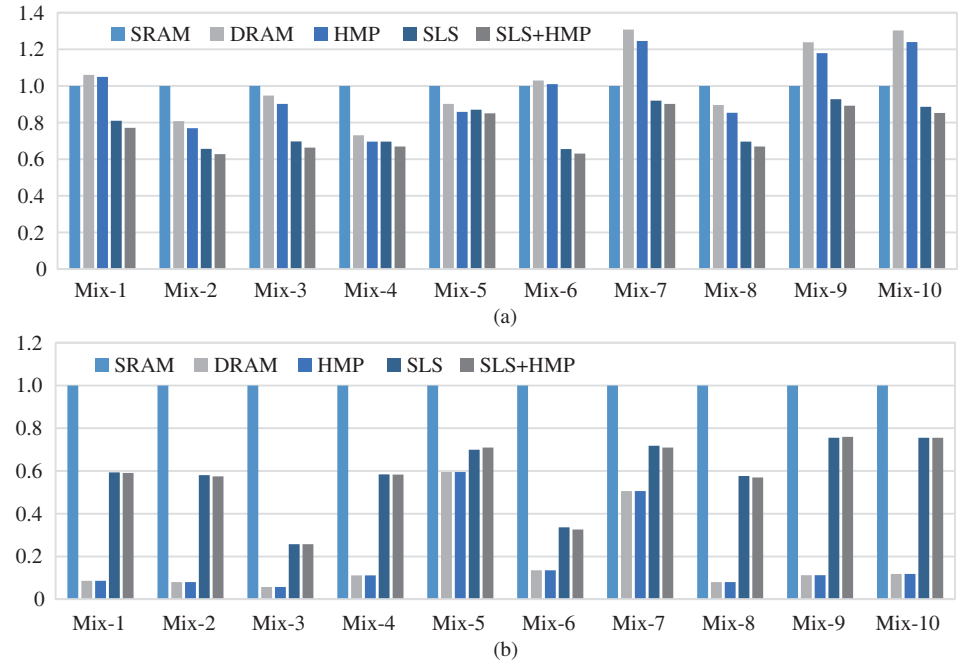

Figure 9 (Color online) Workload impacts. Both (a) and (b) are drawn by normalized experimental data. (a) Average memory access time comparison; (b) cache write-back comparison.

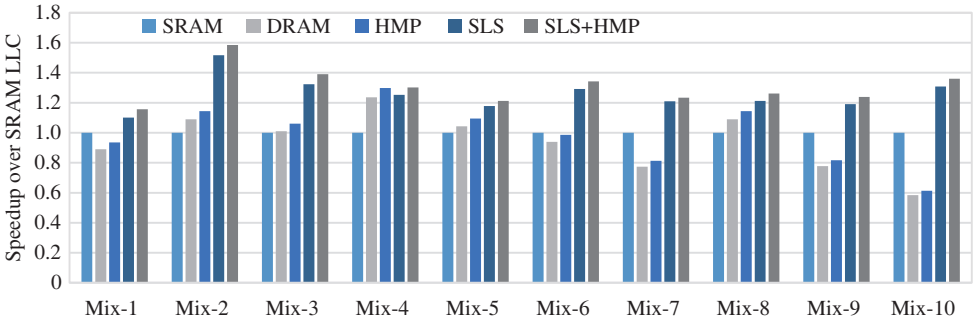

Figure 10 (Color online) Workload performance comparison.

large DRAM LLC, they have least number of cache write-back. The total numbers of cache write-back of the SLS-based architectures are between those of the other two relative architectures, since they can self-adaptively use the different L3 caches.

The SLS-based architectures always achieve better workload performance than SRAM LLC or DRAM LLC-based architectures, as shown in Figure 10. When the multi-programmed workload is mixed with benchmarks with different memory footprints, both SRAM LLC and DRAM LLC-based architectures could not gain better performance than the SLS-based architectures. This is because SRAM LLC architecture could not improve the performance of memory-intensive benchmarks, and the same thing happens to DRAM LLC-based architectures when they deal with benchmarks with small memory footprints. For the multi-programmed workloads, the SLS-based architectures always can find the better way to execute them. Experimental results show that our self-adaptive LLC scheduling scheme can improve the workload performance by up to 30% compared with the state-of-the-art methods.

6 Conclusion

In this paper, we propose the mathematical evaluation models which provide us a basic intuition for determining which LLC module could be better for the program with a specific memory footprint. We then

introduce a self-adaptive LLC scheduling scheme, which allows the LLC module to switch between the SRAM and DRAM caches, according to the specific cache information for multi-programmed workloads. Our experimental results show that the self-adaptive LLC scheduling scheme outperforms than the state-of-the-art methods in terms of the multi-programmed workload performance.

Acknowledgements This work was supported by National Basic Research Program of China (973 Program) (Grant No. 2015CB352403), National Natural Science Foundation of China (Grant Nos. 61261160502, 61272099, 61303012, 61572323, 61628208), Scientific Innovation Act of STCSM (Grant No. 13511504200), EU FP7 CLIMBER Project (Grant No. PIRSES-GA-2012-318939), and CCF-Tencent Open Fund. We would like to acknowledge the anonymous reviewers for their careful work and instructive suggestions. Also, we thank Dr. Zhi-Jie Wang for his warm help and advices.

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Chou C, Jaleel A, Qureshi M K. BEAR: techniques for mitigating bandwidth bloat in gigascale DRAM caches. ACM SIGARCH Comput Arch News, 2016, 43: 198–210
2 Hudec B, Hsu C W, Wang I T, et al. 3D resistive ram cell design for high-density storage class memory—a review. Sci China Inf Sci, 2016, 59: 061403
3 Lun Z Y, Du G, Zhao K, et al. A two-dimensional simulation method for investigating charge transport behavior in 3-D charge trapping memory. Sci China Inf Sci, 2016, 59: 122403
4 Lee Y, Kim J, Jang H, et al. A fully associative, tagless DRAM cache. In: Proceedings of ACM/IEEE International Symposium on Computer Architecture, Portland, 2015. 211–222
5 Hameed F, Bauer L, Henkel J. Adaptive cache management for a combined SRAM and DRAM cache hierarchy for multi-cores. In: Proceedings of Design, Automation and Test in Europe, Grenoble, 2013. 77–82
6 Hundal R, Oklobdzija V G. Determination of optimal sizes for a first and second level SRAM-DRAM on-chip cache combination. In: Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, Cambridge, 1994. 60–64
7 Hameed F, Bauer L, Henkel J. Reducing latency in an SRAM/DRAM cache hierarchy via a novel tag-cache architecture. In: Proceedings of Design Automation Conference, San Francisco, 2014. 1–6
8 Huang C C, Nagarajan V. ATCache: reducing DRAM cache latency via a small SRAM tag cache. In: Proceedings of International Conference on Parallel Architectures and Compilation, Edmonton, 2014. 51–60
9 Qureshi M K, Loh G H. Fundamental latency trade-off in architecting DRAM caches: outperforming impractical SRAM-tags with a simple and practical design. In: Proceedings of IEEE/ACM International Symposium on Microarchitecture, Vancouver, 2012. 235–246
10 Elhelw A S, El-Moursy A, Fahmy H A H. Adaptive time-based least memory intensive scheduling. In: Proceedings of IEEE 9th International Symposium on Embedded Multicore/Manycore Systems-on-Chip, Turin, 2015. 167–174
11 Elhelw A S, Moursy A E, Fahmy H A H. Time-based least memory intensive scheduling. In: Proceedings of IEEE 8th International Symposium on Embedded Multicore/Manycore Systems-on-Chip, Aizu-Wakamatsu, 2014. 311–318
12 Chen Q, Zheng L, Guo M. DWS: demand-aware work-stealing in multi-programmed multi-core architectures. In: Proceedings of International Workshop on Programming Models and Applications on Multicores and Manycores, Orlando, 2014. 131
13 Chen Q, Zheng L, Guo M. Adaptive demand-aware work-stealing in multi-programmed multi-core architectures. J Concurr Comput Prac Exp, 2016, 28: 455–471
14 Roscoe B, Herlev M, Liu C. Auto-tuning multi-programmed workload on the SCC. In: Proceedings of International Green Computing Conference, Arlington VA, 2013. 1–5
15 Castellana V G, Ferrandi F. Abstract: speeding-up memory intensive applications through adaptive hardware accelerators. In: Proceedings of SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, 2012. 1415–1416
16 Huang C, Ravi S, Raghunathan A, et al. Synthesis of heterogeneous distributed architectures for memory-intensive applications. In: Proceedings of International Conference on Computer Aided Design, San Jose, 2003. 46–53
17 Huang C, Ravi S, Raghunathan A, et al. Generation of heterogeneous distributed architectures for memory-intensive applications through high-level synthesis. IEEE Trans Very Large Scale Int Syst, 2007, 15: 1191–1204
18 Athanasaki E, Anastopoulos N, Kourtis K, et al. Exploring the performance limits of simultaneous multithreading for memory intensive applications. J Supercomp, 2008, 44: 64–97
19 Chun K C, Jain P, Kim C H. Logic-compatible embedded DRAM design for memory intensive low power systems. In: Proceedings of IEEE International Symposium on Circuits and Systems, Paris, 2010. 277–280
20 Yi W, Tang Y, Wang G, et al. A case study of SWIM: optimization of memory intensive application on GPGPU. In: Proceedings of International Symposium on Parallel Architectures, Algorithms and Programming, Dalian, 2010. 123–129

21 Qin X, Jiang H, Zhu Y, et al. A Feedback control mechanism for balancing I/O-and memory-intensive applications on clusters. Scal Comput Prac Exp, 2005, 6: 95–107

22 Qin X, Jiang H, Zhu Y, et al. Dynamic load balancing for I/O- and memory-intensive workload in clusters using a feedback control mechanism. In: Proceedings of International Euro-Par Conference, Klagenfurt, 2003. 224–229

23 Jaleel A, Nuzman J, Moga A, et al. High performing cache hierarchies for server workloads: relaxing inclusion to capture the latency benefits of exclusive caches. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture, Burlingame, 2015. 343–353

24 Xiao N, Zhao Y J, Liu F, et al. Dual queues cache replacement algorithm based on sequentiality detection. Sci China Inf Sci, 2012, 55: 191–199

25 Akin B, Franchetti F, Hoe J C. Data reorganization in memory using 3D-stacked DRAM. In: Proceedings of ACM/IEEE International Symposium on Computer Architecture, Portland, 2015. 131–143

26 Jevdjic D, Volos S, Falsafi B. Die-stacked DRAM caches for servers: hit ratio, latency, or bandwidth? have it all with footprint cache. In: Proceedings of ACM/IEEE International Symposium on Computer Architecture, Tel-Aviv, 2013. 404–415

27 Oskin M, Loh G H. A software-managed approach to die-stacked DRAM. In: Proceedings of International Conference on Parallel Architecture and Compilation, San Francisco, 2015. 188–200

28 Loh G H, Hill M D. Supporting very large DRAM caches with compound-access scheduling and MissMap. IEEE Micro, 2012, 32: 70–78

29 Loh G H, Hill M D. Efficiently enabling conventional block sizes for very large die-stacked DRAM caches. In: Proceedings of IEEE/ACM International Symposium on Microarchitecture, Porto Alegre, 2011. 454–464

30 Dong H W, Seong N H, Lee H H S. Pragmatic integration of an SRAM row cache in heterogeneous 3-D DRAM architecture Using TSV. IEEE Trans Very Large Scale Int Syst, 2013, 21: 1–13

31 Chen X, Xu C, Dick R P, et al. Performance and power modeling in a multi-programmed multi-core environment. In: Proceedings of Design Automation Conference, Anaheim, 2010. 813–818

32 Suo G, Yang X. System level speedup oriented cache partitioning for multi-programmed systems. In: Proceedings of IFIP International Conference on Network and Parallel Computing, Gold Coast, 2009. 204–210

33 Kirovski D, Lee C, Potkonjak M, et al. Application-driven synthesis of memory-intensive systems-on-chip. IEEE Trans Comp-Aided Des Int Circ Syst, 1999, 18: 1316–1326

34 Sim J, Loh G H, Sridharan V, et al. A configurable and strong RAS solution for die-stacked DRAM caches. IEEE Micro, 2014, 34: 80–90

35 Chou C, Jaleel A, Qureshi M K. CAMEO: a two-level memory organization with capacity of main memory and flexibility of hardware-managed cache. In: Proceedings of IEEE/ACM International Symposium on Microarchitecture, Cambridge, 2014. 1–12

36 Ou J, Patton M, Moore M D, et al. A penalty aware memory allocation scheme for key-value cach. In: Proceedings of International Conference on Parallel Processing, Beijing, 2015. 530–539

37 Hennessy J L, Patterson D A. Computer Architecture: a Quantitative Approach. 5th ed. Waltham: Morgan Kaufmann, 2012. 72–96

38 Sim J, Loh G H, Kim H, et al. A mostly-clean DRAM cache for effective hit speculation and self-balancing dispatch. In: Proceedings of IEEE/ACM International Symposium on Microarchitecture, Vancouver, 2012. 247–257

39 Begum R, Hempstead M. Power-agility metrics: measuring dynamic characteristics of energy proportionality. In: Proceedings of IEEE International Conference on Computer Design, New York, 2015. 643–650