Runtime 3-D Stacked Cache Data Management for Energy Minimization of 3-D Chip-Multiprocessors

†Seunghan Lee, ‡Kyungsu Kang, †Jongpil Jung, and †Chong-Min Kyung
†Smart Sensor Architecture Lab (SSAL), KAIST and ‡LSI Lab, EPFL
†{shlee04,jpjung08}@vslab.kaist.ac.kr and kyung@kaist.ac.kr and ‡kyungsu.kang@gmail.com

***Abstract*—In a 3-D processor-memory system, multiple cache dies can be stacked onto multi-core die to reduce latency and power of the on-chip wires connecting the cores and the cache, which finally increases the power efficiency. However, there are two challenging issues. The first is the high power density (resulting from multiple die stacking) that incurs many temperature-related problems including temperature-dependent leakage power. The second is the processor-cache traffic congestions that occur at through-silicon vias (TSVs) shared by multiple stacked caches. In this paper, a runtime cache data mapping is proposed for 3-D stacked L2 caches to minimize the overall energy of 3-D chip multiprocessors (CMPs). The proposed method considers both temperature distribution and memory traffic of 3-D CMPs. Experimental result shows that the proposed method achieves up to 22.88% energy reduction compared to an existing solution which considers only the temperature distribution.**

I. Introduction

Three-dimensional integrated circuits (3-D ICs), where two or more layers of active electronic components are integrated vertically into a single chip, significantly reduces the on-chip wire length that often becomes a major bottleneck of performance and/or power dissipation in 2-D ICs [1]. Particularly, 3-D memory stacking has received a great attention since it resolves the memory bandwidth challenges of 2-D ICs by stacking cache memory onto a multi-core die. However, the high power density resulting from multiple (memory) die stacking may lead to the temperature-related problems in reliability (*e.g.,* NBTI), power, performance, and cooling cost. Especially, the exponential dependence of leakage power on temperature, in conjunction with the large amount of cache stacked onto a multi-core die, might aggravate the energy efficiency of 3-D processor-memory systems [2], when considering that on-chip SRAM cache often consumes almost half of total energy in a microprocessor system [3][4].

Dynamic cache reconfiguration (DCR) is an effective method to reduce cache energy by configuring capacity, line size, and associativity of cache according to workload characteristics, and turning off unused parts of the cache. For example, the amount of turned on cache blocks (i.e., capacity) can be optimally determined and assigned to each core based on the memory access demands of applications [5] and, then, the unassigned cache blocks can be turned off to reduce the operating temperature and the temperature-induced leakage energy [6][7]. However, excessive power gating of cache blocks may incur performance degradation due to the increase in cache misses [7].

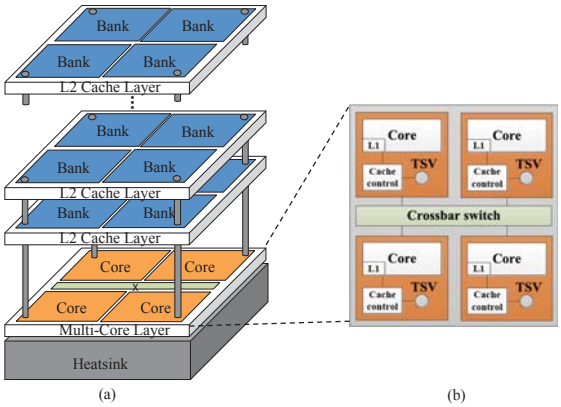

Fig. 1. (a) 3-D CMP consisting of a multi-core layer and multiple stacked L2 cache layers. (b) Multi-core layer consisting of multiple cores with a crossbar switch. Each core has a L2 cache controller connected to its own TSVs

Fig. 1 shows an abstract structure of 3-D CMPs where multiple layers of multi-banked L2 cache are stacked onto a multi-core layer[1]. Each core has a low-latency and large-bandwidth access to the cache banks directly stacked on it (i.e., local cache banks) through through-silicon vias (TSVs). The core is also able to access cache banks stacked on the other cores (i.e., remote cache banks), but the corresponding memory transaction has to be done through the crossbar switch with longer latency. Since cache banks directly stacked on a core share the same TSVs, traffic collision might occur even when cores access different cache banks if the cache banks are directly stacked on the same core (and share the same TSVs).

In this paper, we propose a dynamic cache reconfiguration (DCR) scheme that minimizes the energy consumption of 3-D CMPs with temperature and time-to-deadline constraints. Given the time-varying temperature profile of cores and L2 cache banks, the proposed solution determines (1) the number of L2 cache banks (i.e., the amount of cache capacity) logically allocated to each core and (2) the physical placement of the allocated L2 cache banks, considering both temperature distribution and memory traffic of the 3-D CMPs. To the best of our knowledge, this is the first work on online DCR schemes for real-time 3-D CMPs that considers both temperature and memory traffic. Considering both temperature distribution and cache traffic congestion gives more energy reduction than

[1]Throughout the paper, we call it 3-D CMP that multiple dies of L2 cache are stacked onto the single multi-core die.

considering only one without the other, as shown in the motivational examples in Section III. Since workload characteristics such as memory access behavior change dramatically at runtime, online adaptive configuration of cache memory is paramount for energy reduction. We also investigate the impact of non-uniform cache access latency, cache traffic congestion, and temperature distribution on the energy consumption of 3-D CMPs.

The rest of this paper is organized as follows. Section II presents related works. Section III gives a motivational example of our work. Section IV presents the problem definition. Section V presents analytical formulations to solve the defined problem. Section VI explains how to apply the analytical formulations at runtime. Section VII presents the experimental setup and results followed by conclusion in Section VIII.

II. Related Works

A key challenge in 3D memory stacking is the heat generated from the 3D chip with its increased power density. In case of memory-stacked CMP systems, temperature of each core directly affects the temperature of cache memory blocks stacked on the core. There are prior works on the temperature-aware management for 3D CMP. Yun *et al.* [8] proposed a dynamic voltage and frequency scaling scheme for 3D-stacked L2 DRAM with taking account of both DRAM error-rate and temperature-induced power consumption. Zhao *et al.* [9] proposed thermally aware thread migration among processor cores to reduce temperature variance and peak temperature of stacked DRAM. In [10], to reduce energy consumption, heavily communicating tasks are allocated within the same vertical stack by taking account of shorter interconnect distance between vertical adjacent cores.

Another challenge in 3D memory stacking is the increase in the processor-memory traffic congestion in the bus network with the increased memory capacity. This issue has not been properly dealt with in earlier works. In this paper, we propose a dynamic cache reconfiguration (DCR) solution considering both the bus traffic congestion while keeping the system temperature under the maximum temperature limit. To consider workload characteristics such as memory access behavior changes during runtime, we propose a run-time solution to minimize the system energy consumption (including the energy consumption of core, cache, and off-chip memory), using a multi-core system with stacked L2 cache as an example.

III. Motivation

In this section, we explain the motivation of our work with an example of 3-D CMPs consisting of two cores and four stacked L2 cache layers, as shown in Fig. 2. Each cache layer consists of two cache banks. The capacity of each cache bank is 256KB, assuming a cache bank has the same area/shape as that of a core. Cache banks directly stacked on a core (i.e., local cache banks) are connected to the core through shared TSVs. Cores are also able to access cache banks stacked on the other cores (i.e., remote cache banks) through the crossbar switch with longer latency. The core layer is located

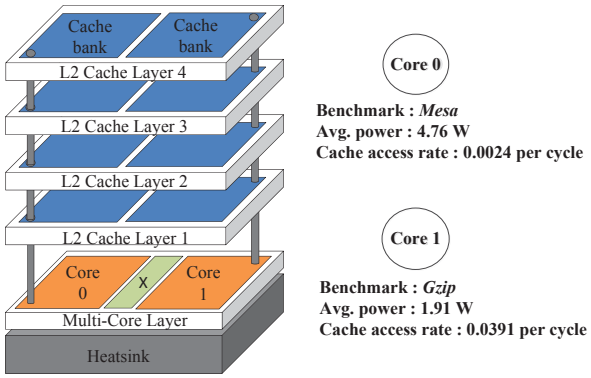

Fig. 2. Motivational example of 3D CMPs and brief descriptions of two programs (*i.e., Mesa* and *Gzip*) mapped onto cores

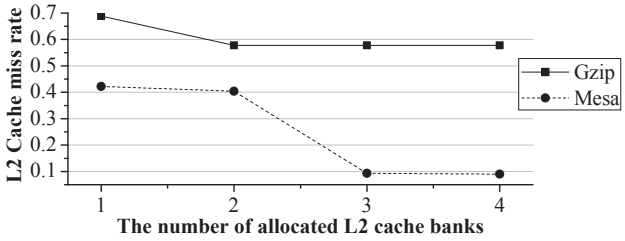

Fig. 3. L2 cache-miss rate with respect to the assigned cache capacity.

next to the heat sink. Let us assume that dynamic power gating is performed at the granularity of cache bank. Two threads, *e.g., Mesa* and *Gzip* in SPEC2000 benchmark [11], are mapped onto Core 0 and Core 1, respectively. The average power consumption of *Mesa* and *Gzip* are 4.76W and 1.91W, respectively. The numbers of L2 cache accesses per cycle of *Mesa* and *Gzip* are 0.0024 and 0.0391, respectively.

Fig. 3 shows the average L2 cache-miss rate of the two threads (i.e., *Mesa* and *Gzip*) with respect to the allocated cache capacity from 256KB (one bank) to 1MB (four banks) when the two threads are executed separately. Larger cache capacity reduces the number of cache misses as shown in Fig. 3, which, in turn, reduces the stall time spent by a core waiting for external memory data. Note that *Mesa* shows a larger drop of cache-miss rate than *Gzip* as more cache capacity is allocated.

Fig. 4 shows brief results (i.e., the results of cache data placement, average temperature distribution, and energy consumption) of three different DCR schemes (i.e., DCR [5], TA (Temperature-Aware)-DCR [7], and TCA (Temperature- and Congestion-Aware)-DCR). In case of DCR, cache banks are allocated to each core such that the total stall time spent by cores waiting for memory data accesses is minimized in a power-efficient manner. As shown in Fig. 4 (a), three cache banks are allocated to Core 0 and two cache banks to Core 1 so that cache-miss rates of the two threads (*Mesa* and *Gzip*) are converged (shown in Fig. 3), while the rest of cache banks are turned off to reduce the energy. Also, all the cache banks are allocated to cores such that only access of local cache banks

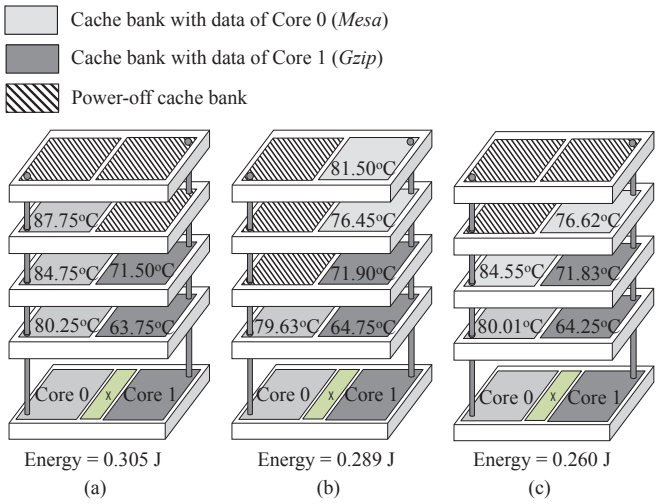

Fig. 4. Results of cache bank placement and energy consumption for the three differnet DCR schemes; (a) DCR (b) TA-DCR (c) TCA-DCR (proposed).

is allowed, which removes the unnecessary communication overhead through the crossbar switch. Note that this scheme does not consider temperature distribution within the 3-D CMP, which is applicable for 2-D CMPs where temperature variation on cache memory is relatively small.

In case of TA-DCR, determining the amount of cache capacity (i.e., the number of cache banks) is the same as DCR scheme, however, cache data is mapped onto the physical cache banks with considering temperature distribution of 3-D CMPs. In the motivational example, Core 1 (running *Gzip*) dissipates lower power than Core 0 (running *Mesa*), which results in lower temperature of Core 1 than Core 0. In 3-D CMPs, power consumption of a core can strongly influence the temperatures of cache banks directly stacked on the core [12], thus the average temperatures of cache banks stacked on Core 1 is lower than those stacked on Core 0 as shown in Fig. 4 (a). To minimize the temperature-induced leakage energy, TA-DCR tries to map cache data of each core onto the physical cache banks with the lower temperatures as shown in Fig. 4 (b). Due to the temperature consideration on the cache data placement, TA-DCR yields the lowest average temperature. However, placement of cache data onto remote cache banks (i.e., cache banks directly stacked on the other core) increases core's stall time owing to the increased latency through the crossbar-switch communications as well as memory traffic congestion at the shared TSVs, which, in turns, increases leakage energy of both core and cache due to the increased execution time. In this example, TA-DCR leads a lower energy consumption than DCR by 5.25%.

The basic idea of our method, i.e., TCA-DCR, is to exploit a trade-off between the leakage energy induced by the higher temperature and the leakage energy owing to the longer execution time (resulting from core's stall). As shown in Fig. 4 (c), Core 1 has lower operating temperature than Core 0 because of lower power consumption of Core 1. However, Core 1 (running *Gzip*) accesses its cache banks more frequently than Core 0 (running *Mesa*) does. Compared with TA-DCR, TAC-DCR reduces memory traffic congestion at Core 1's TSVs and, thus, reduces the stall time of both cores by mapping Core 0's data onto its two local cache banks [shown in Fig 4 (c)], instead of only one local cache bank [shown in Fig 4 (b)], while sacrificing the temperature distribution that incurs more leakage energy induced by the higher temperature. TCA-DCR yields additional energy reduction of 5.2% compared with TA-DCR. Compared with DCR, TCA-DCR reduces the energy consumption by 13.8%.

IV. Problem Definition And Solution Overview

This paper focuses on a multi-core system where 3-D L2 cache is stacked as shown in Fig. 1. The layer closest to the heat sink consists of multiple cores with its own private L1 cache. Multiple layers of L2 cache, each of which consists of multiple SRAM cache banks, are stacked on the multi-core layer. Each cache bank has the same area/shape as that of a core. In the view of a core, local cache banks can be directly accessed, while remote cache banks (i.e., cache banks stacked on the other cores) can be accessed through the crossbar switch. Off-chip DRAM communicates with cores through I/O interfaces. Each cache bank can be dynamically turned on and off according to whether it is necessary or not.

The problem is to find 1) the number of cache banks allocated to each core and 2) the physical positions of the allocated cache banks at runtime such that the overall system energy consumption, E_{tot}, is minimized, while all the deadline constraints are met and the operating temperature does not exceed the maximum limit, T_{max}. The problem can be defined as follows. Given the number of cores, M, an allocated thread set, the number of stacked L2 cache layers, N, and the core's clock frequency, f, the problem is formulated as follows.

Find b_i and $p_{i,j,k}$; $\forall i, j = 1, .., M, \forall k = 1, ..., N$;

such that $$E_{tot} = \sum_{i=1}^{M} E_i^{core} + E_i^{cache} + E_i^{DRAM} \quad (1)$$

is minimized

subject to $$\frac{X_i^c + X_i^s(b_i, p_{i,j,k})}{f} \leq d_i; \ \ \forall i, j, k \quad (2)$$

$$\sum_{i=1}^{M} b_i \leq M \cdot N \quad (3)$$

$$T_i^{core} \leq T_{max}; \ \ \forall i \quad (4)$$

$$T_l^{bank} \leq T_{max}; \ \ \forall l = 1, ..., M \cdot N \quad (5)$$

where b_i is the number of cache banks allocated to Core *i*. $p_{i,j,k}$ is 1 if cache data of Core *i* is mapped onto the cache bank stacked on Core *j* at k^{th} cache layer. Otherwise $p_{i,j,k}$ is 0. E_i^{core} is the energy consumed by Core *i*, E_i^{cache} the energy consumed by cache banks allocated to Core *i*, and E_i^{DRAM} the DRAM energy consumption due to DRAM accesses from Core *i*. X_i^c is the number of clock cycles executed by Core *i* for ideal CPU operation and X_i^s is the number of clock

Algorithm 1: Overall flow of the proposed method

```
1: for each core do
2:   if end of execution of thread then
3:     Workload profiling and estimation
4:   end if
5: end for
6: for each core do
7:   if start of execution of thread then
8:     Find the energy-minimal number of stall cycles (Section V)
9:     Find b_i and p_{i,j,k} (Section VI)
10:  end if
11: end for
```

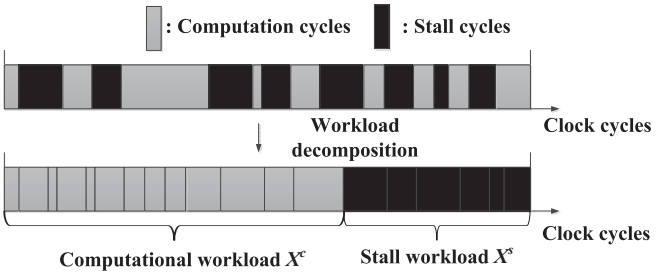

Fig. 5. Execution cycles of a thread consisting of computational workload, X^c, and memory stall workload, X^s, in a core.

cycles stalled in Core *i* waiting for the completion of memory accesses, which is a function of b_i and $p_{i,j,k}$ (explained in Section V). d_i is the time-to-deadline constraint of the thread allocated to Core *i*. T_i^{core} and T_l^{bank} are the temperatures of Core *i* and Cache bank *l*, respectively.

Algorithm 1 shows the overall flow of the proposed method. The proposed method is largely divided into workload prediction (line 1-5) and setting of b_i and $p_{i,j,k}$ (line 6-13), which are invoked at the end/start of every thread execution on each core. In the workload prediction step, we profile runtime information (e.g., the number of cache hits/misses and computational execution clock cycles) of current thread execution and estimate runtime information for the next thread execution. In the b_i and $p_{i,j,k}$ setting step, the number (b_i) and positions ($p_{i,j,k}$) of cache banks for each core is determined based on the estimated runtime information. By adjusting b_i and $p_{i,j,k}$, the number of stall cycles of core *i* is changed, which affects the whole energy consumption of the cores, cache, and DRAM. The energy-optimal number of stall cycles is obtained from our analytic energy model (Section V) and b_i and $p_{i,j,k}$ are adjusted so that the estimated number of stall cycles reaches the given optimal value at runtime (Section VI).

V. Analytic Formulation For Energy-Minimal Cache Configuration

Application workload running on a core is divided into two parts, *i.e.,* computational workload, X^c, and memory stall workload, X^s, as shown in Fig. 5. The memory stall workload, X_i^s, is stall clock cycles of Core i waiting for the completion of memory accesses, which depends on the cache configuration, i.e., b_i and $p_{i,j,k}$ and presented as follows.

$$X_i^s(b_i, p_{i,j,k}) = N_i^{hit}(b_i) \cdot c^{hit} + N_i^{miss}(b_i) \cdot c^{miss} + N_i^{access} \cdot c^{cg}(p_{i,j,k}) \quad (6)$$

where N_i^{hit}, N_i^{miss}, and N_i^{access} are, respectively, the number of L2 cache hits, misses, and accesses from Core i. c^{hit} and c^{miss} are, respectively, the average access latency (in terms of the number of clock cycles) in the cases of cache hit and cache miss, where there is no memory traffic congestion and cores access only their local cache blocks. c^{cg} denotes the average additional cache access latency owing to both the memory traffic congestion at shared TSVs and the crossbar-switch communication. The estimation of c^{cg} will be explained in Section V-A.

Because the number of cache misses per instruction (i.e., cache-miss rate) decreases as more cache capacity is allocated to a core (as shown in Fig. 3), the number of cache misses from Core i, N_i^{miss}, can be modeled as a power law as follows [13].

$$N_i^{miss}(b_i) = \frac{N_i^{miss}(\widetilde{b_i})}{(b_i/\widetilde{b_i})^{\epsilon}} \quad (7)$$

where $\widetilde{b_i}$ represents the number of cache banks allocated to Core *i* in the previous execution of thread. The exponent, ϵ, is directly related to the memory access demand of running threads and typically lies between 0.3 and 0.7. To estimate ϵ at runtime, the monitoring circuits proposed in [5] are adopted. The number of cache hits, N_i^{hit}, is represented as follows.

$$N_i^{hit}(b_i) = N_i^{access} - N_i^{miss}(b_i) \quad (8)$$

Based on the number of computational cycles (X_i^c), the number of memory stall cycles (X_i^s) and operating temperature (T_i^{core}), energy consumption of Core *i*, E_i^{core}, is presented as follows [14].

$$E_i^{core}(T_i^{core}, X_i^s) = X_i^c \cdot e^c(T_i^{core}) + X_i^s \cdot e^s(T_i^{core}) \quad (9)$$

$$e^c(T_i^{core}) = a_s f^{b_s} + a_l(T_i^{core}) \cdot f^{b_l(T_i^{core})} + c(T_i^{core}) \quad (10)$$

$$e^s(T_i^{core}) = \theta \cdot e^c(T_i^{core}) \quad (11)$$

where e^c and e^s represent energy consumption per cycle [15] in the computational and memory stall state of a core, respectively. a_s, a_l, b_s, b_l, and c are empirical constants and functions, and f is core's clock frequency. In Eqn. (10), the first term represents core's switching energy, while the second and third terms together represent core's leakage energy as a function of temperature. θ represents the ratio of energy consumption per clock cycle of memory stall to that of computation (0.6 in our experiments).

Dynamic energy consumed by the cache banks allocated to Core *i*, $E_i^{cache,dyn}$, is represented as follows.

$$E_i^{cache,dyn} = N_i^{hit}(b_i) \cdot e^{hit} + N_i^{miss}(b_i) \cdot e^{miss} \quad (12)$$

where e^{hit} and e^{miss} are dynamic energy consumption per cache-bank hit and miss, respectively. Given the temperatures

of cache banks, the leakage energy consumed by the cache banks allocated to Core i, $E_i^{cache\_leak}$, is presented as follows.

$$E_i^{cache,leak} = \sum_{j=0}^{M}\sum_{k=0}^{N} p_{i,j,k} \cdot e^{leak}(T_{j,k}^{bank}) \cdot (X_i^c + X_i^s) \quad (13)$$

where $T_{j,k}^{bank}$ is temperature of the cache bank directly stacked on Core j at k^{th} cache layer. e^{leak} (T) is the leakage energy per cycle consumed in a cache bank at the temperature of T and is presented as follows.

$$e^{leak}(T) = \alpha \cdot (T)^2 + \beta \cdot T + \gamma \quad (14)$$

where α, β, and γ are extracted from CACTI [16], a cache power estimation tool, with 0.1% of maximum estimation error for all operating temperature range from 50°C to 120°C.

Based on the relation between b_i and X_i^s [shown in Eqn. (6) and (7)], Eqn. (12) and (13) can be represented as follows.

$$E_i^{cache,dyn} = \mu_1 - \mu_2 \cdot X_i^s \quad (15)$$

$$E_i^{cache,leak} = e^{leak}(T_i^{avg}) \cdot \mu_3 \cdot (\mu_4 \cdot X_i^s - \mu_5)^{-\mu_6} \quad (16)$$

where μ_1, μ_2, μ_3, μ_4, μ_5, and μ_6 are positive empirical constants, and T^{avg} is the average temperature of the cache banks allocated to Core i. We omit the derivations of Eqn. (15) and (16) due to the page limit, which are given in our technical note [17].

Off-chip DRAM is accessed when (last-level) L2 cache miss occurs. Thus, the DRAM energy consumed by the accesses from Core *i*, E_i^{DRAM}, is presented as follows.

$$E_i^{DRAM}(X_i^s) = e^{DRAM} \cdot N_i^{miss}(X_i^s) = \mu_7 \cdot X_i^s - \mu_8 \quad (17)$$

where e^{DRAM} is energy consumption per DRAM access, μ_7 and μ_8 are positive empirical constants. (The derivation of Eqn. (17) is also given in the technical note [17].)

Fig. 6 shows the total energy consumption, $E_i^{tot} = E_i^{core} + E_i^{cache} + E_i^{DRAM}$ with respect to the number of stall cycles when *gzip* in SPEC2000 [11] is executed in Core i. The number of stall cycles varies with the cache configuration, i.e., b_i and $p_{i,j,k}$. In Fig. 6, E_i^{tot} is convex with respect to the number of stall cycles, X_i^s. Therefore, $X_{i,opt}^s$ which minimizes E_i^{tot} is obtained by solving the differential equation as follows.

$$\frac{\partial E_i^{tot}(X_i^s)}{\partial X_i^s} = 0 \quad (18)$$

Bisection method [18] is applied to solve Eqn. (18) for all the cores in the 3-D CMP. The computational complexity is $O(\log_2 N_b)$, where N_b is the number of possible b_i values (*i.e., M·N*). Since the number of stall cycles of each core depends on its own cache configuration, the proposed runtime solution determines b_i and $p_{i,j,k}$ so that the number of stall cycles of each core reaches the energy-minimal value, i.e., $X_{i,opt}^s$.

A. Estimation of Additional Stall Cycles

When multiple cores (e.g., Core *a* and Core *b*) simultaneously try to access the cache banks directly stacked on the same core (e.g., Core *c*), memory traffic congestion occurs to get the ownership of the cache banks connected to the same TSVs even though the multiple cores access different cache banks. In addition, when cores communicate with cache banks directly stacked on the other cores (i.e., remote cache banks), the memory transaction has to be through a crossbar switch. These traffics cause additional stall cycles per cache access.

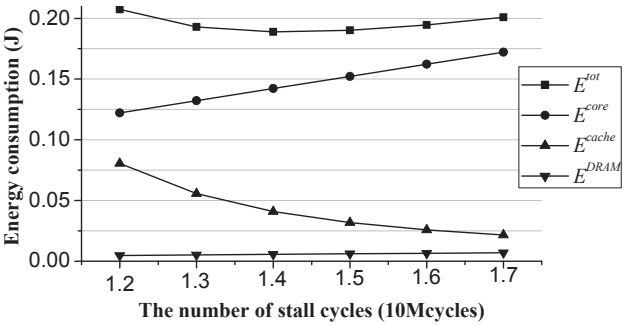

Fig. 6. Energy consumption with respect to the number of stall cycles, X_i^s.

The additional stall cycles of Core *a* due to competing with Core *b* for accessing cache banks directly stacked on Core *c* is denoted as $c_{a,b,c}^{cg}$, which can be computed by summing the conditional expectations of the possible conditions of congestions between the two competing cores, i.e., no congestion and congestion. $c_{a,b,c}^{cg}$, is presented as follows.

$$c_{a,b,c}^{cg} = P_{a,b,c}^0 \cdot E[c_{a,b,c}^{cg}|0] + P_{a,b,c}^1 \cdot E[c_{a,b,c}^{cg}|1] \quad (19)$$

where $P_{i,j,k}^n$ and $E[c_{i,j,k}^{cg}|n]$ are, respectively, the probability of the congestion condition *n* (n = 0 when there is no congestion, otherwise n = 1) and the corresponding expected congestion delay (i.e., additional stall cycles) when Core *a* competes with Core *b* for accessing the cache banks directly stacked on Core *c*. When no congestion occurs (i.e., n = 0), there are no additional stall cycles for cache access owing to the competitions between cores, therefore $E[c_{a,b,c}^{cg}|0]$ becomes zero.

When there exists congestion (i.e., n = 1), the probability, i.e., $P_{a,b,c}^1$, is presented by the portion of time that the cache banks directly stacked on Core c are busy owing to the accesses from Core b thus, not available from Core a. The portion of time can be computed using a M/M/1 queuing model [19][2]. Let us assume that a cache bank has an access latency of t. Then the maximum service rate of the cache bank, i.e., μ, is $1/t$. Let us also assume that the cache banks directly stacked on Core c face an average arrival rate of accesses of $\lambda_{b,c}$ from Core b. Then $P_{a,b,c}^1$ can be presented as follows.

$$P_{a,b,c}^1 = \frac{\lambda_{b,c}}{\mu} = t \cdot \lambda_{b,c} \quad (20)$$

When the two cores (i.e., Core a and Core b) competing the same resource (i.e., the cache banks directly stacked

[2]Let us consider a single server that processes customers at the rate of μ, facing customers at the rate of λ, where services and arrivals are both *Poisson*. Then the utilization of the server is the proportion of time that the server is busy and presented as λ/μ [19].

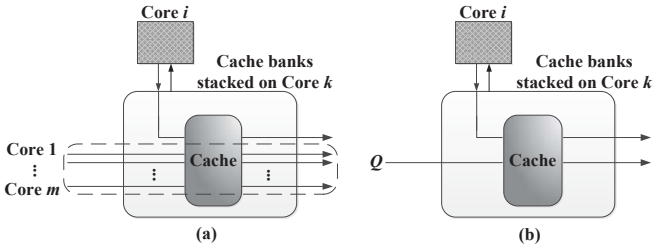

Fig. 7. (a) Memory traffics between Core i and the other m cores onto the cache banks stacked on Core k. (b) Aggregated memory traffic of the other m cores.

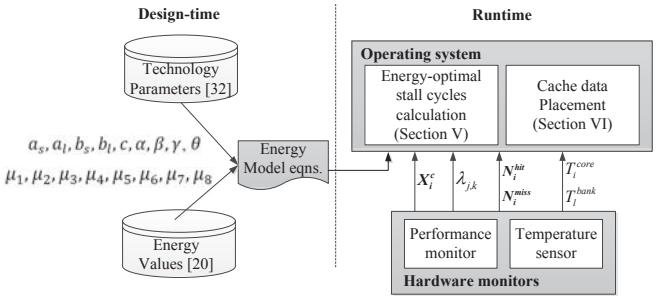

Fig. 8. Overview of runtime cache data placement methodology.

on Core c), the waiting time for getting the ownership is approximately one-half of the latency of the resource, (i.e., $t/2$) [20], therefore, the expected congestion delay, $c^{cg}_{a,b,c}$, is presented as follows.

$$c^{cg}_{a,b,c} = P^1_{a,b,c} \cdot E[c^{cg}_{a,b,c}|1] = \frac{t^2}{2} \cdot \lambda_{b,c} \tag{21}$$

Based on the analysis of the two-core congestion model, we derive general formulas which compute the congestion delay of Core *i* where the other m cores competes with Core i for the banks directly stacked on the same core, i.e., Core k, as shown in Fig. 7 (a). Using the two-core congestion model, the memory traffics from the m multiple cores, which compete with Core *i*, can be aggregated as shown in Fig. 7 (b) with assumptions as follows. We assume that cache access arrivals are *Poisson* with access rate of λ as in many previous studies [21]. Then, for the cache banks stacked on Core k, the mean cache access rate generated by the m cores is $\sum_{j=1}^{m} \lambda_{j,k}$ [19] and the expected congestion delay of Core *i*, $g^{cg}_{i,k}$ is presented as follows.

$$g^{cg}_{i,k} = \frac{t^2}{2} \cdot \sum_{j=1}^{m} \lambda_{j,k} \tag{22}$$

We measured the accuracy of Eqn. (22) from a modified *Tsim* simulator [22] and found that the absolute estimation error of Eqn. (22) is no more than 0.2 clock cycles in 98% of simulation cases and less than 0.3 clock cycles in 80% of the cases. Details of our experiment environment are explained in Section VII.

In conclusion, the additional stall cycles of Core i is estimated as follows. The number of stall cycles of Core *i* due to memory traffic congestion for accessing the cache banks stacked on Core *k* is given by multiplying the number of cache bank accesses, $A_{i,k}$ ($N^{access}_i = \sum_{k=1}^{M} A_{i,k}$) and the average additional cache access cycles owing to memory traffic congestion, $g^{cg}_{i,k}$, i.e., $A_{i,k} \cdot g^{cg}_{i,k}$. Then the total number of additional stall cycles of Core *i*, i.e., the third term in Eqn. (6), is re-written as follows.

$$N^{access}_i \cdot c^{cg}(p_{i,j,k}) = \sum_{k=1}^{M} A_{i,k} \cdot g^{cg}_{i,k} \cdot c^{xbar}_{i,k} \tag{23}$$

where $c^{xbar}_{i,k}$=1 if $i = j$, otherwise $c^{xbar}_{i,k}$=c^{xbar} where c^{xbar} is the additional latency consumed by a crossbar switch.

VI. Runtime Energy-Optimal Cache Configuration

Fig. 8 shows the overview of the proposed runtime cache data placement methodology. The information required for the runtime cache data placement consists of the energy model parameters, the number of clock cycles for each core executing instructions (X^c_i), the number of cache hits/misses (N^{hit}_i, N^{miss}_i), mean arrival rate of cache accesses from each core ($\lambda_{j,k}$), and the current temperature of each core and cache bank (T^{core}_i, T^{bank}_l). We obtained parameters for the energy model at design-time. At runtime, X^c_i, N^{hit}_i, N^{miss}_i, $\lambda_{j,k}$, T^{core}_i, and T^{bank}_l are collected at every 10ms, which is the least time interval in the OS time scheduler.

When the runtime solution is invoked at first, initialization is performed only once such that all the cache data of each core are allocated to local cache banks, that removes the additional stall cycles due to memory traffic congestion and crossbar switch communication. When a thread is started, deallocation followed by reallocation of cache bank is performed for all the cores to find best b_i and $p_{i,j,k}$ while satisfying all the constraints [i.e., Eqn. (2), (3), (4), and (5)]. Fig. 9 shows an example of how to find the best b_i and $p_{i,j,k}$. For all the cores, when Core i has less number of stall cycles than $X^s_{i,opt}$ [i.e., point A in Fig. 9 (a)], deallocation of cache banks allocated to Core i is performed to move the number of stall cycles to the optimal point as long as not exceeding the deadline constraint, i.e., Eqn. (2). For the cores that has larger number of stall cycles than $X^s_{i,opt}$ [i.e., point B in Fig. 9 (a)], the deallocation process is skipped since they needs more cache banks to reduces the off-chip memory accesses and, thus, the number of stall cycles. For all the cores, if the temperature constraints, i.e., Eqn. (4), and (5) are violated, deallocation process is also performed until the temperatures of cores and cache banks keep from the maximum temperature limit, without considering the number of stall cycles.

Cache bank reallocation is invoked after the deallocation process such that the total energy is minimized while all the constraints are met, i.e., Eqn. (2), (3), (4), and (5). Fig. 9 (b) shows the total energy with respect to the number of stall cycles in the views of Core i and Core j, respectively. In Fig. 9 (b), Core j (point D) reduces more energy than Core i (point C) due to one additional cache bank allocation. This shows that allocation of one additional cache bank to each core in descending order of their slopes of the current operating

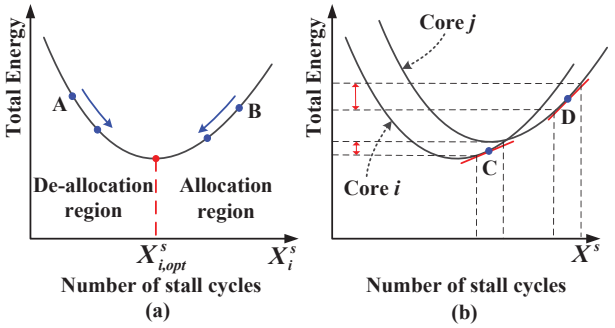

Fig. 9. Energy consumption with respect to the number of stall cycles, showing how to find the best cache data placement.

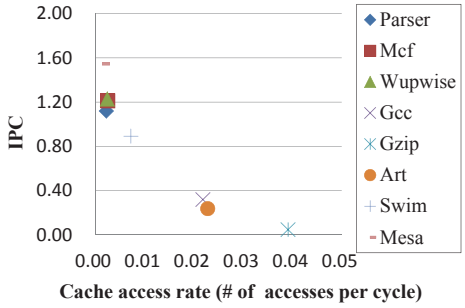

Fig. 10. IPC (instructions per cycle) and cache access rate of programs in SPEC2000.

points minimizes the total energy in 3-D CMPs because of the concavity of the total energy with respect to the number of stall cycles.

VII. Experiment

A. Setups

We performed experiments using a CMP consisting of a 4-core layer and four stacked L2 cache layers each of which consists of four cache banks. The core used in our experiment is Alpha 21264 processor in 65nm process technology. Each core has 32KB L1 instruction and data cache. The size of each core is 2.85mm×2.78mm. The number of cache ways per bank is eight and the capacity of each cache bank is 256KB [16]. Hotspot [1] is used for estimating temperatures of the cores and cache banks in the 3-D CMP. Our experiment was performed with SPEC2000 benchmark programs [11]. Fig. 10 shows the results of instructions per cycle (IPC) and L2 cache access rate of eight programs in SPEC2000. Performance results are obtained from *Simplescalar* simulator [24] based on Alpha21264 processor. From SPEC2000 benchmark programs, we constructed three test cases, i.e., Set A, Set B, and Set C. Set A consists of the programs with high IPC and low cache access rate, *i.e.,* CPU-bound applications (*gzip*, *wupwise*, *mesa*, and *gcc*), whereas Set B consists of the programs with high cache access rate and low IPC, *i.e.,* memory-bound applications (*swim*, *mcf*, *art*, and *parser*) where execution time is dominated by memory access time. Set C consists of programs which are combinations of memory-bound applications (*mcf* and *art*) and CPU-bound applications (*mesa* and *wupwise*).

B. Result

We have evaluated the impact on energy consumption for the following cache data placement policies.

DCR [5]: This technique allocates the energy-minimal number of cache banks to each core without considering time-varying temperature.

TA-DCR [7]: This technique allocates the energy-minimal number of cache banks, considering the temperature distribution and assuming that there is no memory traffic congestion.

TCA-DCR-DT: This technique determines the number and the positions of allocated cache banks at design-time, considering both temperature distribution and memory traffic congestion.

TCA-DCR-RT (proposed): This method determines the number and the positions of allocated cache banks, considering both the temperature, memory traffic congestion at runtime.

Fig. 11 shows the results of energy consumption, which are normalized with respect to that of DCR. Overall, TCA-DCR-RT yields up to 23% further reductions in energy consumption compared with DCR and TA-DCR. Such an improvement is due to the consideration of the temperature distribution and memory traffic congestion together in TCA-DCR-RT. In Fig. 11, in case of Set B (i.e., memory-bound programs), TCA-DCR-RT shows relatively smaller energy reduction than in cases of Set A and Set C. It is because, in Set B, cache-miss rate of each program converges at a larger cache capacity than Set A and Set C. As the cache capacity required by each program increases, it has less possibility to reallocate cache data of each program to exploit both temperature distribution and memory traffic congestion.

In Fig. 11, note that the cache energy consumption in TA-DCR is lower than that in DCR while the core energy consumption in TA-DCR is higher than that in DCR. TA-DCR places cache data of each core by exploiting the temperature distribution of the 3-D CMP, which leads to a decrease in the temperature-induced cache energy. However, TA-DCR may also lead to an increase of stall cycles in cores due to unin-

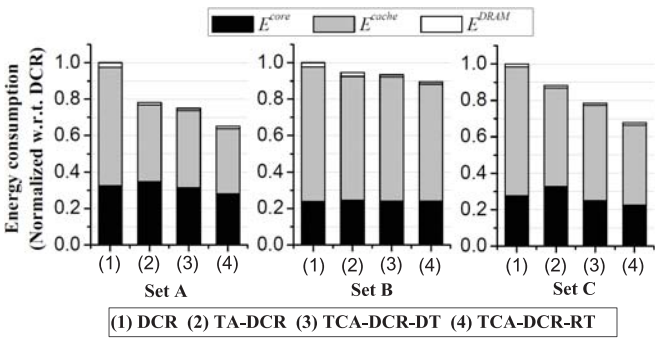

Fig. 11. Comparison of energy consumption for each test case. Set A, Set B, and Set C consist of CPU-bound programs, Memory-bound programs, and combinations of the two programs.

telligent cache data placement by not considering the memory traffic congestion as well as the crossbar switch latency. Therefore, the later offsets the energy reduction resulting from the former, thus TA-DCR yields smaller total energy reduction compared with TCA-DCR-RT. For example, in Set C, even though TA-DCR reduces cache energy consumption by 21% compared to DCR, core energy consumption of TA-DCR increases by 12.0%, therefore TA-DCR yields only 14.1% of total energy reduction compared with DCR.

In order to evaluate TCA-DCR-RT as a runtime solution, we prepared a design-time solution of TCA-DCR-DT. In TCA-DCR-DT, all the runtime information (*i.e.,* cache miss rate, cache access rate, the number of computational clock cycles, and temperature of each core and cache bank) is given as average values. As shown in Fig. 11, TCA-DCR-RT reduces the total energy consumption up to 11.2% compared with the TCA-DCR-DT.

We measured the runtime overhead of TCA-DCR-RT on the modified *TSIM* simulator [22] at 3GHz. For the runtime computation of finding the best cache data placement, the time overhead ranges between $31.5\mu s$ and $126\mu s$, with its corresponding energy overhead ranging from $119\mu J$ to $583\mu J$. After the optimal cache configuration is obtained, there are transition overheads to change the number of cache banks allocated to each core. When the number of assigned cache banks to a core is decreased, dirty cache blocks in the cache banks which will be turned off must be written back to off-chip DRAM for data coherency. The energy overhead for write-back operation was up to $47.52\mu J$. When the number of assigned cache banks to a core is increased, it costs wakeup time to switch cache banks back to the active mode from the power-gating mode. According to [23], the wake-up time is negligible (*i.e.,* four clock cycles). The area overhead of implementing power-gating technique in L2 caches is about 5% [6].

VIII. Conclusion

In this paper, we address the problem of cache data mapping for a multi-core architecture with 3D-stacked L2 cache to minimize the overall system energy. Unlike the classical approaches, our method considers memory traffic congestion as well as temperature distribution in the 3-D CMP. The proposed method places cache data to the specific physical position by exploiting the trade-off between the energy induced by memory traffic congestion and the energy induced by temperature of the cache block. We solved this problem with a runtime solution and the experiment results show that the proposed method yields up to 22.88% improvement in energy reduction compared to an existing runtime cache configuration method which only considers the temperature distribution [7].

IX. Acknowledgement

This work was supported by the Center for Integrated Smart Sensors funded by the Ministry of Science, ICT and Future Planning as Global Frontier Project (CISS-2013066998).

References

[1] B. Black, et al., "Die Stacking (3D) Microarchitecture," in *Proc. the 39th Intl. Symp. on Microarchitecture*, pp. 469-479, Dec. 2006.

[2] W. Liao, L. He, and K. Lepak, "Temperature-aware performance and power modeling," in Technical Report UCLA Engr. 04-250, UCLA, Los Angeles, CA, 2004.

[3] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache for low energy embedded systems," in *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 2, pp. 363-387, May. 2005.

[4] D.H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *Proc. the 32nd Intl. Symp. on Microarchitecture*, pp. 248-259, Nov. 1999.

[5] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proc. the 39th Intl. Symp. on Microarchitecture*, pp. 423-432, 2006.

[6] M. Powell, et al., "Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *Proc. the ACM/IEEE Intl. Symp. on Low Power Electronics and Design*, pp. 90-95, July. 2000.

[7] H. Noori, et al., "Temperature-Aware Configurable Cache to Reduce Energy in Embedded Systems," in *IEICE Trans. Electronics*, vol. 91, no. 4, pp. 418-431, 2008.

[8] W. Yun, et al., "Temperature-Aware Energy Minimization of 3D-Stacked L2 DRAM Cache through DVFS," in *Proc. ISOCC*, 2012, pp. 475–478.

[9] D. Zhao, H. Homayoun, and A. V. Veidenbaum, "Temperature Aware Thread Migration in 3D Architecture with Stacked DRAM," in *Proc. ISQED*, 2013, pp. 80–87.

[10] Y. Cheng, et al., "Thermal-Constrained Task Allocation for Interconnect Energy Reduction in 3-D Homogeneous MPSoCs," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 2, pp. 239-249, Feb. 2013.

[11] *Standard performance evaluation corporation* [Online]. Available: http://www.specbench.org.

[12] A.-C. Hsieh and T. Hwang, "Thermal-aware memory mapping in 3D designs," in *Proc. the Conf. on Design, Automation and Test in Europe*, pp. 1361-1366, 2009.

[13] A. Hartstein, et al., "Cache miss behavior: is it $\sqrt{2}$?," in *Proc. the 3rd Conf. on Computing frontiers*, pp. 313-320, 2006.

[14] S. Kim, J. Lee, and C.-M. Kyung, "3D-stacked L2 Cache Configuration for DVFS-enabled Processor to Minimize Overall Energy Consumption," in *Intl. Conf. on Convergence and Hybrid Information Technology*, Aug. 2010.

[15] J. Kim, Y. Lee, S. Yoo, and C.-M. Kyung, "An analytical dynamic scaling of supply voltage and body bias exploiting memory stall time variation," in *Proc. the 2010 Asia and South Pacific Design Automation Conf.*, 2010.

[16] D. Tarjan, S. Thoziyoor, and N. P. Jouppi, "CACTI 4.0," HP Laboratories, Palo Alto, CA, Tech. Rep. HPL-2006-86, Jun. 2006.

[17] *Technical note*, "Technical note of runtime 3-D stacked cache data management for energy minimization of 3-D CMPs," [Online]. Available: http://ssal.kaist.ac.kr/shlee04/Appendix.pdf.

[18] G. Corliss, "Which root does the bisection algorithm find?," in *SIAM Review*, vol. 19, no. 2, pp. 325-327, 1997. [Online]. Available: http://www.jstor.org/stable/2029507.

[19] A.M. Mood, F.A. Graybill, and D.C. Boes, *Introduction to the Theory of Statics* (McGraw-Hill, NewYork, 1974).

[20] D. Min and M. W. Mutka, "A Framework for Predicting Delay Due to Job Interactions in a 2-D Mesh Multicomputer," in *Proc. Seventh Intl. Parallel Processing Symposium*, pp. 350-357, Apr. 1993.

[21] D. L. Eager, M. C. Ferris, and M. K. Vernon, "Optimized Caching in Systems with Heterogeneous Client Populations," in *Performance Evaluation, Special Issue on Internet Performance Modeling*, vol. 42, no. 2/3, pp. 163-185, Sep. 2000.

[22] S. Cho, et al., "TPTS: A Novel Framework for Very Fast Manycore Processor Architecture Simulation," in *Proc. the 37th Intl. Conf. on Parallel Processing*, pp. 446-453, Sep. 2008.

[23] H. Homayoun, M. Makhzan, and A. Veidenbaum, "Multiple sleep mode leakage control for cache peripheral circuits in embedded processors," in *Proc. the Intl. Conf. on Compilers, architectures and synthesis for embedded systems*, pp. 197-206, 2008.

[24] D. C. Burger, et al., "Evaluating future microprocessors-the simplescalar tool set," Technical Report 1308, University of Wisconsin-Madison Computer Sciences Department, July 1996.