Energy and Thermal Aware Buffer Cache Replacement Algorithm

Jianhui Yue, Yifeng Zhu, Zhao Cai, Lin Lin

Department of Electrical and Computer Engineering,
University of Maine, Orono, USA
{jyue, zhu, zcai, llin}@eece.maine.edu

***Abstract*—Power consumption is an increasingly impressing concern for data servers as it directly affects running costs and system reliability. Prior studies have shown most memory space on data servers are used for buffer caching and thus cache replacement becomes critical. Temporally concentrating memory accesses to a smaller set of memory chips increases the chances of free riding through DMA overlapping and also enlarges the opportunities for other ranks to power down. This paper proposes a power and thermal-aware buffer cache replacement algorithm. It conjectures that the memory rank that holds the most amount of cold blocks are very likely to be accessed in the near future. Choosing the victim block from this rank can help reduce the number of memory ranks that are active simultaneously. We use three real-world I/O server traces, including TPC-C, LM-TBF and MSN-BEFS to evaluate our algorithm. Experimental results show that our algorithm can save up to 27% energy than LRU and reduce the temperature of memory up to 5.45^oC with little or no performance degradation.**

***Index Terms*—Buffer cache, memory energy, data servers**

I. Introduction

In order to bridge the ever-widening gap between disk and processor speeds, high-end storage servers often require a large-capacity main memory. For example, the IBM Bluegene at LLNL has 32 TB [1] and up to 2 TB can be installed on a single server [2]. Previous studies [3], [2], [4] have shown that main memory is one of major sources of power consumption. It is believed that memory energy consumption is 50% more than processor power on many production servers [5]. In addition, memory can also consume more power than hard disks. For example, in a representative high-end server with 3.8 TB memory and 115 TB local disks [6], the disks consume 6KW while the memory takes 19KW, according to a validated power estimation tool [7]. Large energy consumption not only increases running costs, but more importantly, raises the temperature of server components and thus reduces their reliability. As the memory capacity continues to increase rapidly to alleviate the I/O bottleneck, its energy efficiency becomes a pressing concern, especially for high-density racks and blade servers.

Buffer cache replacement algorithms play an important role in conserving memory energy, since buffer cache takes more than 77% of the total available memory on a PC desktop frequently and much more on a server [8]. Replacement algorithms influence the memory energy from two sometimes conflicting aspects: (1) algorithms with high hit rates help shorten the overall running time and directly reduce the energy; (2) algorithms decide which data blocks to be evicted and thus determine the access sequence and utilization of individual memory chips, which eventually influences the opportunities of power saving for each chips. The power saving techniques considered in this paper includes (1) dynamically setting a memory chip to a low-power mode if idling longer than some idle-time limit and (2) DMA overlapping that allows a memory chip to perform multiple DMA I/Os simultaneously to improve its throughput and reduce its active time.

Most existing replacement algorithms only aim to maximize cache hit rates and ignore the current power status of memory chips when selecting victim blocks upon cache misses. Energy saved due to higher hit rates and shorter running time may not offset the extra energy cost when keeping more memory chips active simultaneously. This observation motivates us to study new replacement algorithms that optimize the tradeoff between cache hit rates and memory energy-efficiency.

In this paper, we propose a new energy- and thermal-aware buffer cache replacement algorithm that saves energy and reduces chip temperature with no or little sacrifice to cache hit rates. The basic idea of our algorithm is to select victim blocks from the chip that very likely holds the maximum number of least recently used blocks. As a result, our algorithm can naturally cluster most I/O requests to a small set of active memory chips and hence increase the opportunity for the other chips to power down. At the same time, it makes no or little degradation to hit rates when comparing with LRU. Instead of evicting out the block that will be accessed potentially in the farthest future, our algorithm evicts the one that are suboptimal in terms of hit rates but potentially has a larger energy saving. Our algorithm differs from existing studies [2], [4], [8], [9], [10], [11], [12] in that ours does not rely on data migration and thus has much less bookmarking and running overheads.

The rest of this paper is organized as follows. Section II briefly describes the background of power-aware memory chips, Fully Buffer DIMM, and DMA overlapping. Section III presents our energy-efficient buffer cache management algorithm. Section IV gives our evaluation methodology and simulation results. The related works are discussed in Section V. Section VI concludes the paper.

II. Background

A. DDR2 DRAM Memory

In order to reduce DRAM power consumption, nowadays DDR2 chips offer a number of energy states, including read/write, precharge, power down and self refresh. A DRAM system consists of a set of ranks and each rank can be independently set to one of these states. In the read/write state, the data is accessed and a rank in this state has the largest power consumption rate. In order to activate a rank with a row to reading or writing, the rank must be precharged so that the read or write can start at the next clock edge. In the power down state, the rank powers off I/O buffer, sense amplifiers and row/column decoders to conserve energy and thus it cannot access data before switching back to the precharge state. In addition, DRAM must periodically refresh stored data to prevent data loss due to charge leakage. The transition from a lower power state to a higher one causes some time delay for synchronization.

B. FB-DIMM (Fully Buffered DIMM)

FB-DIMM is a widely used DRAM technology to increase reliability, speed and density of memory systems. It introduces an advanced memory buffer (AMB) between the memory module and the memory controller including write link and the other read link. AMB acts as a pass-through switch. With this architecture, each data/command frame targeted to one specific memory module pass through every AMB in this channel, which makes AMB consume extra power.

C. DMA Overlapping

Direct Memory Access (DMA) has been widely used to transfer data blocks between main memory and I/O devices including disks and network. Fig. 1 gives an example of disk-network datapath on a storage server when cache misses occur for two external I/O requests A and B received from the network. The datapath consists of four steps from 0 to 3. When a read request arrives through a network interface (NIC), the server first performs data address translation and then checks whether desired data blocks are stored in the main-memory buffer cache. If the target data are in the memory, the host processor on the storage server initiates a network DMA operation to transfer the data out directly from the main memory through NIC. If they are not, the processor first performs a disk DMA transfer to copy the data from disks to the main-memory buffer cache, and then the processor conducts a network DMA transfer to send the data out to the client applications. For write requests, the datapaths are similar but flow in the reverse direction.

On a storage server, recent DMA controllers, such as Intel's chipset E8870 and E7500 [13], allow multiple DMA transfers on different buses to access the same memory module simultaneously in a time multiplexing fashion. Typically, the peak transfer rate of a memory chip can be a multiple factor of the bandwidth of the PCI bus. For example, the transfer rate of most recent DDR2 SDRAM is up to 5.3GB/s, while a typical PCI-X bus only gives a maximum rate of 1.064GB/s and the second-generation SATA disk DMA throughput is only 300 MB/s.

Multiplexing various slow disk and network I/Os to the same memory rank can reduce the waste of active memory cycles and hence save memory energy. Most DMAs move

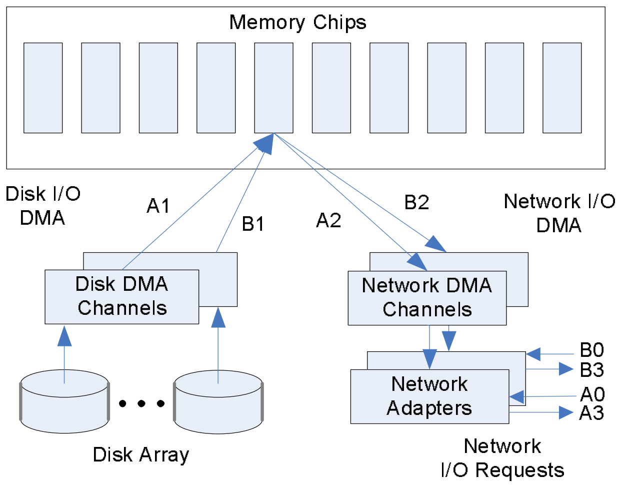


Fig. 1. I/O datapaths when cache misses occur for two read requests A and B on a storage server (following steps from 0 to 3)

a large amount of data, usually containing multiple 512-byte disk sectors or several KByte memory pages. Without multiplexing, a memory rank is periodically touched during a DMA transfer and such access period is too short to justify the transition to a low-power mode [2], [4], [14] As a result, significant amount of active energy is wasted. However, when DMAs on different I/O buses are coordinated to access the same memory rank, such energy waste can be reduced. For example, when the concurrent requests A and B in Fig. 1 are directed to the same memory rank, the DMA transfers $A1in$ and $B1$ can overlap with each other in time and accordingly one of them takes a "free ride" and consumes zero background energy, without causing any performance penalty. Similarly, $A2$ and $B2$ can also overlap with each other if they use different DMA channels.

III. Energy and Temperature Aware Replacement Algorithm

The Least Recently Used (LRU) algorithm or its clock-based approximation is used in most operating systems to manage buffer cache due to the advantages of simple implementation and constant space and time complexity. Accordingly, this paper focuses on improving the energy efficiency based on LRU, without losing its simplicity. Specifically, we design a new variant of LRU that optimizes the tradeoff between cache hit rates and memory energy efficiency. Our basic idea is generic and can be applied to many other cache replacement algorithms too.

Our previous study [15], [16] shows that the interplay among the following three factors appears to be the most important for memory energy saving: the cache performance in terms of hit rates, the cache's ability to temporally align memory accesses to the same chips, and the cache populating schemes to allocate buffers. The cache hit rates can directly translate performance gain into energy saving by reducing the running time. The cache populating schemes perform buffer allocation and affect access locality at the chip level.

The strategies used to allocate buffers before the cache is full, called cache populating schemes. Ref. [17] advocates sequential placement in virtual memory. However, high-end data servers usually run data-intensive applications and page allocation schemes then have little impacts to the memory energy efficiency. This is because few page allocations are performed after the buffer cache is full. As a result, the cache performance in terms of hit rates and the clustering capability are the most important factors for data servers.

For a given sequence of memory I/O stream, the order in which memory ranks are accessed are in fact determined by the cache replacement algorithm. Upon a cache miss, LRU replaces the block whose last reference was the earliest among all cached blocks. The key weakness of LRU is that it tends to access memory ranks in a random order. This leads to energy inefficiency from two aspects: (1) too many memory ranks are accessed simultaneously and few have chances to enter to a low power state; (2) less DMA overlap operations can be performed since the target data of different DMAs tend to be located in different memory ranks.

We design a new algorithm based on LRU to better save memory energy. The basic idea is to limit cache replace operations to a small set of memory ranks, without decreasing much hit rates. When a cache miss occurs, instead of replacing the block that is the least likely to be accessed in the future, we choose to replace a block that meets two requirements: (1) the block is not likely to be accessed very soon and (2) victim rank has the most cold blocks at decision time. Our solution is to first identify those blocks that are not accessed very frequently, i.e., cold blocks, and then evict cold block from same victim rank until no sufficiently stale blocks remain it.

Our algorithm, as summarized in Alg. 1, consists of two steps: first identify a victim memory rank and then choose a victim block to be replaced from this memory bank. During the first step, this algorithm scans $search\_window$ blocks starting from the LRU bottom and chooses the rank with the largest number of blocks as the victim rank. Here $search\_window$ is a predefined constant. Among all blocks that are located in the victim rank and are among the predefined bottom window of the LRU stack (i.e., $threshold\_window$), the least recently accessed one is chosen as the victim block. Note that the same victim rank is repeatedly chosen to be evicted until this rank has no more data blocks within the bottom LRU stack window. In another word, the array $rank\_cnt$ is not updated for every cache miss; it is only get updated if $last\_victim\_cnt$ reaches zero.

Compared with LRU, our algorithm selects the victim rank much less randomly and thus avoids touching a large set of memory ranks simultaneously. This creates a larger opportunity for the other memory banks to enter low power modes to conserve energy and reduce temperature. In order to minimize the reduction in cache hit rates, this algorithm wisely chooses the victim rank as the one that has the largest number of blocks within a predefined distance from the bottom of the LRU stack. After the victim rank is identified, the victim block will be the least recently used block within the victim rank. We call this algorithm $Bottom\_Most$.

Algorithm 1 *Bottom_Most*

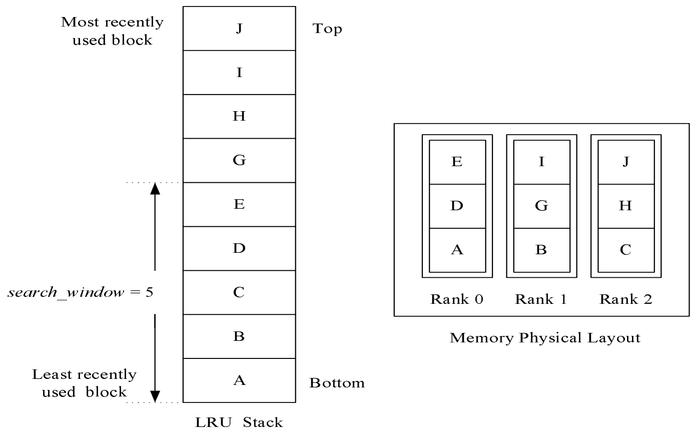
```
/* Global variables : */
/* $last\_victim\_rank$: last victim rank */
/* $last\_victim\_cnt$: # of un-replaced blocks in last victim
rank that are in the bottom $search\_window$ of the LRU
stack */
/* $stack$: LRU stack */
/* $rank\_cnt[i]$: # of blocks in bank $i$ that are in the bottom
$search\_window$ of the LRU stack */
/* $LRU\_block$: the least recently used block in the stack*/

/* Local variables */
$R$: the total number of memory ranks
$N$: the total number of blocks in the LRU stack

/*Step 1: choosing the victim rank */
**if** $last\_victim\_cnt == 0$ **then**
  /*scan $search\_window$ blocks starting from the LRU
  bottom and find $victim\_rank$ that has the largest number
  of blocks within this window */
  **for** $i = 1$ to $R$, **do** $rank\_cnt[i] \Leftarrow 0$, **end for**
  **for** $i = 1$ to $search\_window$ **do**
    $item \Leftarrow stack[N - i]$
    $rank\_cnt[item.rank] + +$
  **end for**
  $victim\_cnt \Leftarrow \max(rank\_cnt[i]),\ 0 \leq i \leq R$
  $victim\_rank$ is the $i$ that achieves the maximum above
  $last\_victim\_rank \Leftarrow victim\_rank$
  $last\_victim\_cnt \Leftarrow victim\_cnt - 1$
**else**
  $victim\_rank \Leftarrow last\_victim\_rank$
**end if**

/*Step 2: choosing the victim block */
**for** $i = 0$ to $ThresholdWindow$ **do**
  $b \Leftarrow stack[N - i]$
  **if** $b.rank == victim\_rank$ **then**
    $victim\_block \Leftarrow b$
    $last\_victim\_cnt \Leftarrow last\_victim\_cnt - 1$
    **return** $victim\_block$
  **end if**
**end for**
/* Otherwise, degrade to standard LRU */
**return** $LRU\_block$
```

Fig. 2 gives an example that compares $Bottom\_Most$ and LRU when the access sequence is $\{X, Y, Z\}$. For the simplicity, assume that the memory has only three ranks and each rank can store only three blocks. We also assume that the interval between the request for block X and the request for block Y is shorter than DRAM power down threshold. In this example, both search window length and thresh window are 5. LRU chooses victim blocks A, B and C, respectively for the missed request X, Y and Z only taking their recency



(a)

	LRU		Bottom-Most	
Requst-blk	Victim-blk	Victim-rank	Victim-blk	Victim-rank
X	A	0	A	0
Y	B	1	D	0
Z	C	2	E	0
Total # of touched ranks	3		1	

(b)

Fig. 2. A simple example compares LRU and $Bottom\_Most$. Assume the memory has three ranks and each rank can hold three blocks. The LRU stack has nine blocks. The table compares the total number of memory ranks accessed for a given access sequence $\{X, Y, Z\}$.

into accounting. As a result, three memory ranks are accessed during this access sequence. In the $Bottom\_Most$ algorithm, A is chosen as the victim block for two reasons: (1) rank 0 has more blocks in the LRU bottom window than rank 1 and rank 2, and (2) A is the least recently accessed among all blocks of rank 0. When Y is missed, the victim block is D because rank 0 because it still has two blocks in the search window and D is the least recently accessed block in rank 0. In sum, LRU activates three ranks while $Bottom\_Most$ activates only one rank in this example.

IV. Performance and Energy Evaluation

This section presents the evaluation of our algorithm via trace-driven simulations.

A. Simulation framework

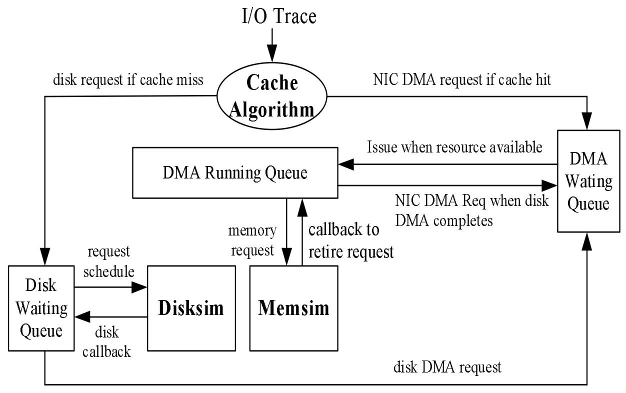


Fig. 3. Event-driven simulation framework

The simulation framework is composed of three major components: cache simulator, disk array simulator, and memory simulator. Disksim [18], a well validated disk array simulator, is incorporated to precisely emulate the timing of I/O traffic. These three components interact with each other through two event queues and two request queues shown in Fig. 3.

Disksim APIs with callback functions are used to generate disk I/O events and place these events into the disk event queue waiting for DMA operations. The memory simulator and the cache simulator coordinate with each other to determine the chip address. Before the cache is full, the memory simulator resolves the chip address of each missed block based on populating schemes. When the buffer cache is full, the cache simulator determines logically the victim block. For each chip the memory simulator simulates the DMA overlapping operations and maintains the power state transitions based on a timeout mechanism used in Ref. [4].

When a request is issued from the I/O trace, the cache simulator generates a disk access event, sends it to the disk request queue if the cache is missed. A network DMA request event is then generated and sent to the DMA request queue. A DMA request is served if the target DMA channel is available and the target memory chip is active. The DMA request queue emulates DMA overlapping and contentions.

This paper assumes that a separate set of memory chips are dedicated for kernel codes, process pages, stacks and heaps. The energy of these dedicated chips is not studied in this paper. Our memory simulator only calculates the energy of memory chips used as buffer cache, which typically takes a large portion of memory on data servers. We share this assumption with previous studies on memory energy [4], [8].

It was a challenge for us to validate our simulation framework since it is very difficult to accurately measure the memory energy consumption in real systems [8], [19]. First of all, the trace collector's memory accesses cannot be easily separated out and thus interfere the native memory traffic. Second we do not have specifically designed servers that allow the direct measurement of energy consumed by memory chips. Third, no RTL-level memory simulator is publically available for us to validate our simulation framework. IBM's memory simulator [19] is validated against a proprietary RTL simulator that contains 1.5 million lines of VHDL codes. To overcome these challenges, Ref. [8] use the product of memory access

Parameter	value
rank powerdown delay	3 memory cycles
rank powerup delay	10 memory cycles
rank powerdown threshold	30 memory cycles
tRP: Row Precharge time	15ns
tRCD: Row active to row active delay	15ns
tRAS : Row Activation time	45ns
tCAS: Delay to access a certain column	15ns
IDD0: Active precharge current	80mA
IDD2P: Precharge powerdown current	7mA
IDD2N: Precharge standby current	45mA
IDD3N: Active standby current	55mA
IDD3P: Active powerdown current	240mA
IDD4R: Burst read current	145mA
IDD4W:Burst write current	140mA
IDD5: Burst refesh current	170mA
IDD6: Self refresh current	7mA
AMB_Power1: the last AMB static power	2W
AMB_Power: other AMB static power	2.55W
AMB dynamic power for local traffic	3.55W
AMB dynamic power for forwarding traffic	2.8W
AMB read bandwidth	6.4GB/s
AMB write bandwidth	3.2GB/s
# of ranks per DIMM	2
# of DIMMS per channel	8
Rank capacity	256MB
Vdd: Supply voltage	1.8v

TABLE I
DRAM PARAMETERS

time duration and the requested data size to approximate the real energy consumption. However, this approach cannot be used in our study since it ignores the energy saved through DMA overlapping.

B. Simulation parameters

The data server simulated in this paper is configured with 6 network adaptors (NIC) and 12 disks with exception for the TPC-C trace which has 305 disks. All disks and NICs are attached to one 133 MHz PCI-X bus. A DMA request is performed on the corresponding PCI-X bus that has the target device. Due to the lack of client information in the I/O traces, we assign each incoming I/O request to a NIC in a round-robin fashion. The server response a request through assigned NIC. The simulator emulates FB-DIMM with DDR2 memory whose parameters are given in Table I. The simulator precisely models the memory access timing and power consumption complying with DRAM power module published by Micron Technology Inc.

We follow the method and parameters give in Ref. [20] to model the dynamics of DIMM and AMB temperature. Equation (1) and (2) are used to compute the stable temperatures for AMB and DRAM, respectively [21]. Besides influenced by the ambient temperature, both AMB and DRAM power dissipation can affect each other's temperature by mutual thermal resistance ψ_{AMB_DRAM} and ψ_{DRAM_AMB}.

We use the same method presented in Ref. [22], [23] to dynamically update all temperatures (see Equation (3)). Basically, all temperatures are treated in the same way as the voltage in an electrical RC circuit. In our simulator, the thermal resistance from DRAM and AMB to the ambient is $11.2^oC/W$ and $4.9^oC/W$, respectively. The thermal resistance is $4.3^oC/W$ from DRAM to AMB and $5.3^oC/W$ from AMB to DRAM. The ambient temperature to set to be 85^oC in all experiments.

Trace	Working Set(GB)	Traffic(GB)	Duration(Sec)
TPC-C	66.53	465.3	360
LM-TBE	143.9	205.5	3604
MSN-BEFS	10.58	18.05	603

TABLE II
TRACE STATISTICS

$$\begin{aligned} T_{AMB} &= T_A + P_{AMB} * \psi_{AMB} \\ &\quad + P_{DRAM} * \psi_{DRAM\_AMB} \qquad (1) \\ T_{DRAM} &= T_A + P_{AMB} * \psi_{AMB\_DRAM} \\ &\quad + P_{DRAM} * \psi_{DRAM} \qquad (2) \\ T(t+\Delta t) &= T(t) + (T_{stable} - T(t)) * (1 - e^{-\frac{\Delta t}{\tau}}) \qquad (3) \end{aligned}$$

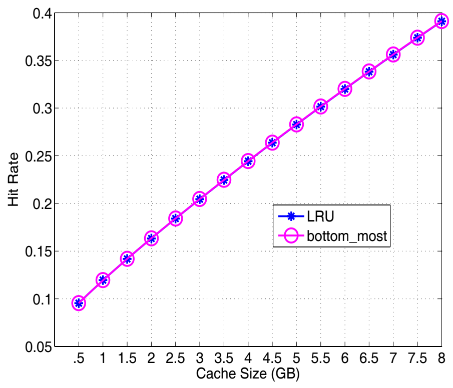
where T_{AMB} and T_{DRAM} are the temperature of AMB and DRAM; T_A is the ambient temperature; ψ_{AMB} is thermal resistance from the AMB to ambient; ψ_{DRAM} is thermal resistance from DRAM to ambient.

We simulate the traces by replaying all I/O events at pre-determined time instants specified in the traces, independent of the performance of the memory hierarchy. This approach is used mainly because all traces that we have access to do not record the dependence between request completion and subsequent I/O arrivals. This is one of the major limitations of this study. Three real-life data server traces are used, including TPC-C, LM-TBE, and MSN-BEFS [24]. All these traces were collected in 2008. TPC-C is transaction processing workload. The LM-TBE is I/O traces of image tile back-end server in the Live Map. The MSN-BEFS trace is collected for the MSN storage server which provides Live data services. Table II summaries the statistics of these three traces. In our experiments, we make workload more intensive by reducing arrival intervals by a factor of ten.

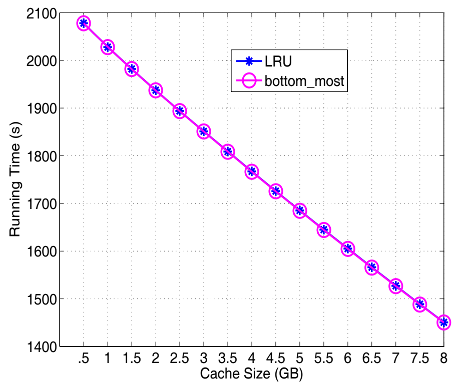
The default $search\_window$ and $threshold\_window$ in the bottom-most replacement algorithm are 1024. The default data block size is 64KB, a typical value in NFS3 file systems on data servers.

C. TPC-C Workload

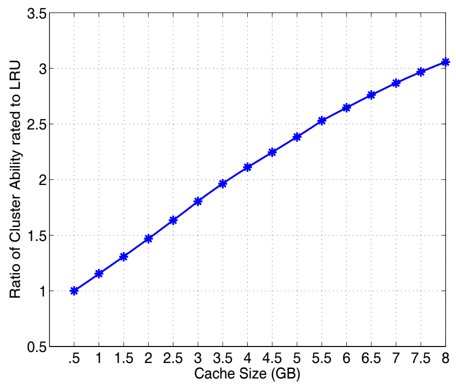
Under the TPC-C workload, we conclude that our algorithm can achieve significant energy savings with very little scarification to the performance in terms of hit rates. As shown in Fig. 4(d), the average energy saving across different cache sizes is 22% compared with LRU. The maximum energy saving over LRU can be as much as 27% at cache size of 4.5GB. It is interesting that the energy saving curve reaches the lowest point when the cache size is 4.5 GB. There are two conflicting factors to effect the $Bottom\_Most$ algorithm's energy efficiency. As the number of memory ranks increases, our algorithm has a larger opportunity to choose a better victim rank. However, increasing memory capacity makes memory ranks spend more time in power down mode and hence the percentage of active energy reduction achieved by our algorithm becomes smaller. As a result, as the cache size



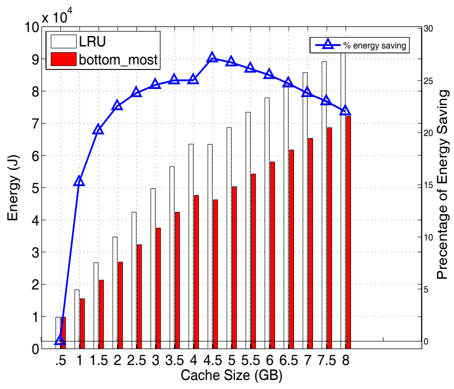
(a) Cache Hit Rates



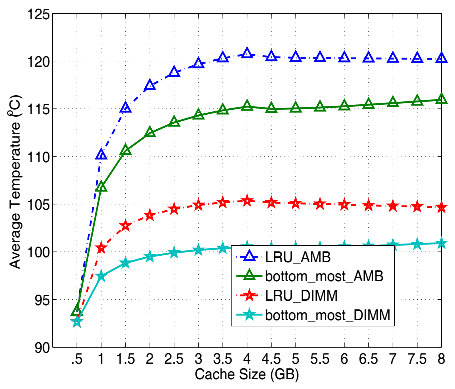
(b) Running Time



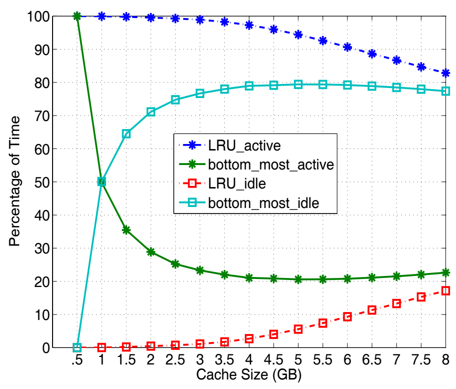
(c) Cluster Ability Rated to LRU



(d) Total Energy Consumption

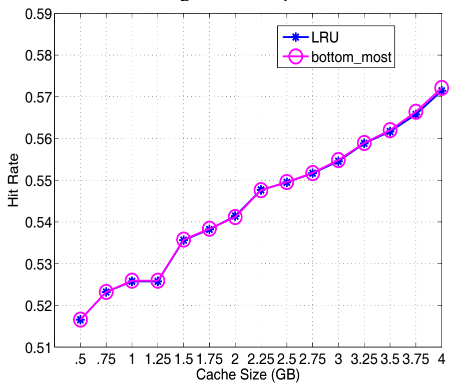


(e) Average Temperature

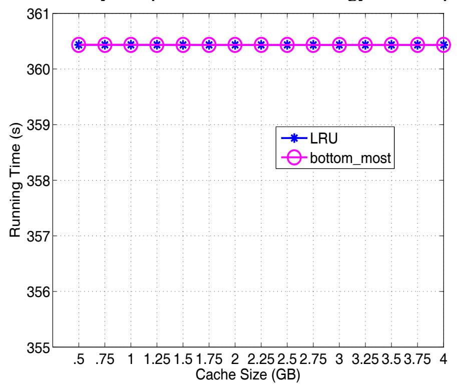


(f) Time Components

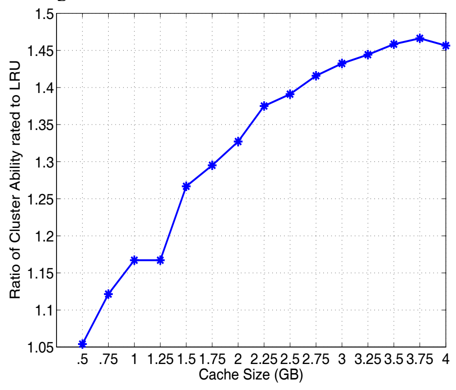
Fig. 4. Comparison of cache hit rates,memory temperature and total energy consumption using workload *TPC-C* .



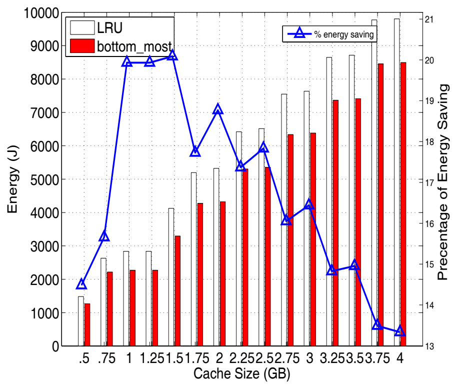
(a) Cache Hit Rates



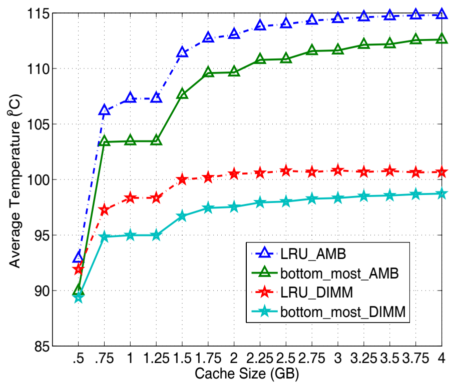
(b) Running Time



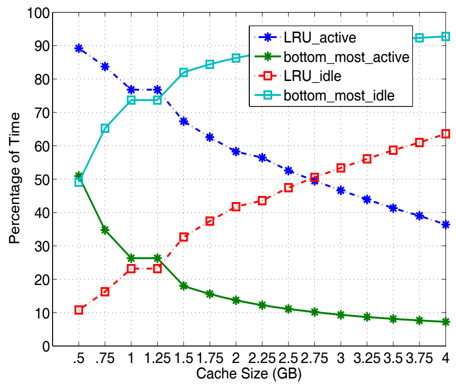
(c) Cluster Ability Rated to LRU



(d) Total Energy Consumption

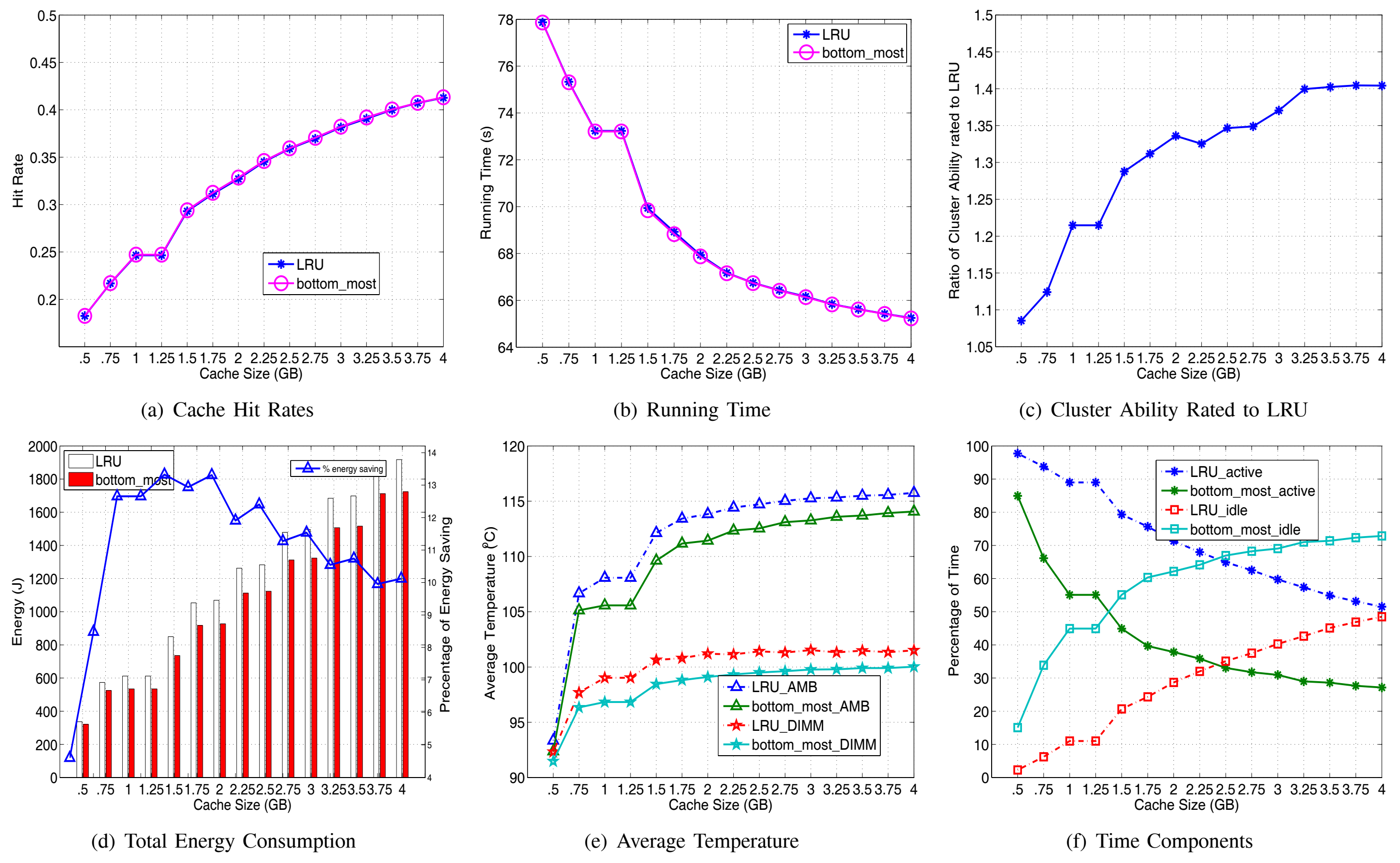


(e) Average Temperature



(f) Time Components

Fig. 5. Comparison of cache hit rates, memory temperature and total energy consumption using workload *LM-TBE* .



(a) Cache Hit Rates

(b) Running Time

(c) Cluster Ability Rated to LRU

(d) Total Energy Consumption

(e) Average Temperature

(f) Time Components

Fig. 6. Comparison of cache hit rates, memory temperature and total energy consumption using workload *MSN-BEFS* .

increases, such two conflicting factors interplay with each year and the energy efficiency achieve an optimal value under some certain cache size.

Due to large energy saving and accordingly less hit dissipation, the temperature of AMB and DIMM is decreased significantly (see Fig. 4(e)). The maximum temperature reduction for DIMM and AMB is 5.45^oC and 4.82^oC, respectively, while the average reduction is 4.59^oC and 4.03^oC. The reason why our algorithm reduces more temperature for the DIMM than AMB is that every memory request packet is relayed through AMBs in other DIMM within one channel. And for the same reason, AMB temperature is higher than DIMM.

When comparing the average hit rate across all experiments under different cache sizes, $bottom\_most$ is only inferior than LRU by 0.03% (see Fig. 4(a)). The completion time of LRU and $bottom\_most$ are almost the same under all experiments. This confirms that our algorithm does not cause any slowdown or performance degradation for *TPC-C* workload.

We use the overlap time to measure the capability of clustering I/O onto the same rank for better opportunities of DMA overlapping. For a given workload, a larger overlapping time means stronger clustering capability. The overlap ability is defined here as the ratio of two algorithms' overlap time and larger value means stronger overlap ability. Fig. 4(c) shows that the overlap ability of our algorithm against LRU. The results confirm that our algorithm overlap ability is always stronger than LRU under all experiments under TPC-C workload. Our algorithm clusters consecutively missed blocks to one victim rank and thus creates more DAM overlap operations than LRU.

Fig. 4(f) shows the percentage of active time and idle time replacement algorithms spend. We have the following observations. First, as the number of memory ranks increases, the idle time increases but the active time decreases. Second, our algorithm's active time is smaller than LRU's active time, and our idle time is larger than LRU's idle time. Clustering consecutively missed blocks into a wisely chosen victim rank contributes to reduced active time and increased idle time. This time breakdown also confirms $Bottom\_Most$ algorithm outperforms LRU in terms of energy efficiency.

D. *LM-TBF Workload*

The energy simulation results under the *LM-TBF* workload are given in Fig. 5(d). The energy saving of bottom_most over LRU can reach up to 20% at cache size of 1.5GB, with an average of 18% over all experiments. We find that when the memory size is larger than 1GB, the energy efficiency of $Bottom\_Most$ is better in experiments when the number of ranks is even. This is because two ranks share one AMB and the memory with odd number ranks consumes extra energy in AMB.

Fig. 5(a) and Fig. 5(b) show that there is no performance degradation in *LM-TBF*. The maximum average temperature reduction are 3.83^oC and 3.36^oC respectively for AMB and DIMM. The average of average temperature reduction are 2.8^oC and 2.45^oC respectively for AMB and DIMM. The overlap ability comparison in Fig. 5(c) and the time component comparison in Fig. 5(f) are consist with the energy results.

It is also noted that the completion time under different cache sizes does not change, as shown in Fig. 5(b). This is because we use the open model to issue requests from trace due to being lack of data dependency information in the original applications. The *LM-TBF* trace's requests are not burst. As a result, the simulation completion time is roughly the same as the arrival time of the very last request in the trace.

E. *MSN-BEFS Workload*

In the *MSN-BEFS* workload, the same conclusions can be obtained as in *TPC-C* and *LM-TBF*. Compared to LRU, $bottom\_most$ achieves an average of 11.0% energy saving, with a maximum of 13.3% at cache size of 1.5GB (see Fig. 6(d)). *bottom_most* achieves the same hit rates and completion time of as LRU, as shown in Fig. 6(a) and Fig. 6(b). The maximum temperature reduction is 2.49^oC and 2.19^oC respectively for AMB and DIMM, with average reduction of 1.98^oC and 1.73^oC (See Fig. 6(e)).

V. Related works

Existing research works of reducing memory energy consumptions can be classified into the software and hardware approaches.

A. *Software approaches to reduce memory power consumption*

The memory energy management can be done in three different areas, including OS, compiler, and database. In the area of operating systems, the power-aware page allocation [4] uses the sequential first touch policy to allocate pages according to their access order and only activates ranks where the running process' pages reside. PAVM [25] clusters a process pages into as a smaller set of ranks as possible by specifying the rank which pages are allocated from. After the process scheduler has chosen the next-to-run process, the PVAM puts this process' rank set to be active state while putting other ranks to low power state for achieving power reduction and hiding latency caused by powering up low power state rank. PABC [8] exploits spatial locality in the page access that (1) a file's buffer caches are usually accessed together and (2) files used by a process are most likely accessed together. PABC clusters buffer cache associated with a file to a small set of ranks and it also places a process's user space pages and buffer pages belonged files accessed by this process to the same set ranks. Taking advantages of slack in memory demand, the memory miser [26] power down memory ranks with no data. It migrates pages between memory ranks and also uses a classic PID control algorithm to compute the number of high power ranks serving memory requests.In the compiler area, Ref. [27] reduces the loop energy consumption which dominates the memory energy consumption in the embedded applications. It clusters the data with similar lifetime pattern to same bank to achieve memory energy saving.In the database area, Ref. [28] clusters each table's data into a rank to achieving energy saving. In addition, we propose two memory energy efficient buffer cache management schemes in previous research works [29], [30].

B. *Hardware approaches to reduce memory power consumption*

The hardware approaches not only exploit exiting low power states but also investigate new memory organizations to save energy. Ref. [31] proposes to add a small prefetching cache in the AMB in order to not only improve performance but also reduce energy consumption. This energy reduction comes from the number of row access operations is reduced since high prefetch cache hit rate saves row access operations. Mini-rank [32] advocates to increase the bus width of DRAM devices so that each memory rank actives less DRAM devices simultaneously. By placing a small cache in the DRAM device, Ref. [33] cannot only improve performance but also reduce DRAM energy consumption due to the cache effect. Since delaying write operations won't give much penalty to the overall performance, Ref. [34] proposes to add a write buffer between the processor and DRAM to improve row buffer hit rate and thus to reduce DRAM power consumption. Ref. [35] reduces memory energy saving through DRAM command scheduling. Each memory cycle, the memory controller checks each active rank's activities against all scheduled commands and issues the power-down command to ranks which won't be accessed by the scheduled commands. In order to avoid consuming the peak-power, Ref. [12] limits the memory power consumption without exceeding the power budget. The memory chips are organized as a LRU list according to their access order and inactive chips will be put into deeper power states after their idle durations exceed the break-even time. Ref. [36] proposes changes the interleave degree according to workload to consume as lower energy as possible without exceeding the performance degradation.

VI. Conclusion and Future Work

As increasingly more memory is used on data servers for buffer caching to bridge the processor/disk speed gap, the memory energy consumption becomes a pressing concern. To the best of our knowledge, this paper is the first to study directly on power aware buffer caching replacement algorithms without using data migration.

This paper proposes a new generic power- and thermal-aware algorithm. This algorithm limits cache replacements to a small set of memory ranks without decreasing much hit rate. Thus choosing this rank as a victim rank help make memory chips less randomly touched. This eventually reduces the total number of ranks that are simultaneously active. While our strategies can be incorporated many conventional non-power aware replacement algorithms, we integrate these strategies into LRU, the most popular algorithm in real systems.

We use three real-world I/O server traces, including TPC-C, LM-TBF and MSN-BEFS, to evaluate our algorithm. Experimental results show that our power aware algorithm can save up to 27% energy, comparing with LRU, and reduce the temperature of memory up to 5.45^oC without performance degradation. It is important to emphasize that our algorithm is generic and they provide useful heuristics and insights to improve the energy efficiency of many other state-of-the-art replacement algorithms.

We do not consider data migration in this paper. Periodically migrating data blocks between memory banks can reshape the memory data layout and may help reduce power. In the future, we will study migration algorithms to further improve the memory energy efficiency and reduce the memory runtime temperature.

Acknowledgements

This work is supported by a UMaine Startup Grant,NSF IIS-0916663, NSF CCF-0937988, and NSF CCF-0621493.

References

[1] M. E. Tolentino, J. Turner, and K. W. Cameron, "An implementation of page allocation shaping for energy efficiency," in *Proceedings of 3rd Workshop on High-Performance, Power-Aware Computing*, April 2007.

[2] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini, "DMA-aware memory energy management for data servers," in *The Proceedings of the 10th International Symposium on High-Performance Computer Architecture (HPCA'06)*, 2006.

[3] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[4] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," in *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*. New York, NY, USA: ACM Press, 2000, pp. 105–116.

[5] G. Papadopoulos., "Impacts and importance of energy efficiency: industry viewpoint," http://www.energystar.gov/ia/products/downloads/GPapadopoulos_Keynote.pdf.

[6] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, and D. D. E. Long, "File systemworkload analysis for large scale scientific computing applications," in *MSST '04: Proceedings of the 21nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'04)*. Washington, DC, USA: IEEE Computer Society, 2004.

[7] A. Leventhal, "Flash storage memory," *Communication of The ACM*, vol. 51, no. 7, 2008.

[8] M. Lee, E. Seo, J. Lee, , and J. Kim, "Pabc: Power-aware buffer cache management for low power consumption," *IEEE Transactions on Computers*, vol. 56, no. 4, 2007.

[9] P. Ramamurthy and R. Palaniappan, "Performance-directed energy management using bos," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 1, pp. 66–77, 2007.

[10] V. D. L. Luz, M. Kandemir, and I. Kolcu, "Automatic data migration for reducing energy consumption in multi-bank memory systems," in *DAC '02: Proceedings of the 39th conference on Design automation*. New York, NY, USA: ACM Press, 2002, pp. 213–218.

[11] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *Proceedings of the Workshop on Compilers and Operating Systems for Low Power COLP'01*, September 2001. [Online]. Available: http://research.ac.upc.es/pact01/colp/paper04.pdf

[12] B. Diniz, D. Guedes, W. M. Jr., and R. Bianchini, "Limiting the power consumption of main memory," in *Proceedings of the International Symposium on Computer Architecture ISCA*. ACM Press, June 2007, pp. 290–301.

[13] Intel, "Server and workstation chipsets," http://www.intel.com/products/server/chipsets/.

[14] X. Li, Z. Li, Y. Zhou, and S. Adve, "Performance directed energy management for main memory and disks," *ACM Trans. Storage*, vol. 1, no. 3, pp. 346–380, 2005.

[15] J. Yue, Y. Zhu, and Z. Cai, "Evaluating memory energy efficiency in parallel I/O workloads," in *Proceedings of IEEE International Conference on Cluster Computing, Best Paper Award*, Austin, TX, 2007, pp. 21–30.

[16] J. Yue, Y. Z. Zhu, and Z. Cai, "An energy-oriented evaluation of buffer cache algorithms using parallel i/o workloads," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1565–1578, 2008.

[17] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," *SIGOPS Oper. Syst. Rev.*, vol. 34, no. 5, pp. 105–116, 2000.

[18] J. S. Bucy, G. R. Ganger, and et al., "The disksim simulation environment version 3.0 reference manual," www.pdl.cmu.edu/DiskSim.

[19] I. Hur, "Enhancing memory controllers to improve dram power and performance," Ph.D. dissertation, University of Texas at Austin, 2006, http://www.cs.utexas.edu/ lin/papers/ibrahim.pdf.

[20] J. Lin, H. Zheng, Z. Zhu, H. David, and Z. Zhang, "Thermal modeling and management of dram memory systems," in *The 34th International Symposium on Computer Architecture*, 2007.

[21] K. Man, "Bensley fb-dimm performance/thermal management," Intel Developer Forum, Tech. Rep., 2006.

[22] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management," in *HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2002, p. 17.

[23] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," *SIGARCH Comput. Archit. News*, vol. 31, no. 2, pp. 2–13, 2003.

[24] S. Kavalanekar, B. L. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *The 4th International Symposium on Workload Characterization*, 2008.

[25] H. Huang, P. Pillai, and K. G. Shin, "Design and implementation of power-aware virtual memory," in *USENIX Annual Technical Conference*, 2003, pp. 57–70. [Online]. Available: citeseer.ist.psu.edu/article/huang03design.html

[26] M. E. Tolentino, J. Turner, and K. W. Cameron, "Memory miser: Improving main memory energy efficiency in servers," *IEEE Trans. Comput.*, vol. 58, no. 3, pp. 336–350, 2009.

[27] V. Delaluz, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Energy-oriented compiler optimizations for partitioned memory architectures," in *CASES '00: Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems*. New York, NY, USA: ACM, 2000, pp. 138–147.

[28] J. Pisharath, A. Choudhary, and M. Kandemir, "Energy management schemes for memory-resident database systems," in *ACM Thirteenth Conference on Information and Knowledge Management (CIKM'05)*. Washington, DC, USA: ACM Press, Nov 2004.

[29] J. Yue, Y. Zhu, and Z. Cai, "Energy efficient buffer cache replacement," in *Proceedings of 16th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'08)*, Baltimore, MD, USA, 2008.

[30] J. Yue, Y. Zhu, and C. Zhao, "Impacts of indirect blocks on buffer cache energy efficiency," in *ICPP'08: Proceedings of the 2006 International Conference on Parallel Processing*. Portland, Oregon, USA: IEEE Computer Society, 2008.

[31] n. Z. Z. n. Z. Z. H. D. Jiang Lin, null Hongzhong Zheng, "Dram-level prefetching for fully-buffered dimm: Design, performance and power saving," in *ispass '07: 2007 IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE Computer Society, address = Washington, DC, USA,, 2007.

[32] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive dram architecture for improving memory power efficiency," in *MICRO '08: Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 210–221.

[33] N. AbouGhazaleh, B. Childers, D. Mosse, and R. Melhem, "Near-memory caching for improved energy consumption," in *ICCD '05: Proceedings of the 2005 International Conference on Computer Design*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 105–110.

[34] S. Liu, S. O. Memik, Y. Zhang, and G. Memik, "A power and temperature aware dram architecture," in *DAC '08: Proceedings of the*

45th annual Design Automation Conference. New York, NY, USA: ACM, 2008, pp. 878–883.

[35] C. Hur, I. Lin, "A comprehensive approach to dram power management," in *HPCA '08: IEEE 14th International Symposium on High Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 305–316.

[36] H. S. Khargharia, B. and M. S. Yousif, "An adaptive interleaving technique for memory performance-per-watt management." *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 7, pp. 1011–1022, 2009.