An Energy-Oriented Evaluation of Buffer Cache Algorithms Using Parallel I/O Workloads

Jianhui Yue, *Student Member*, *IEEE*, Yifeng Zhu, *Member*, *IEEE*, and Zhao Cai, *Student Member*, *IEEE*

Abstract—Power consumption is an important issue for cluster supercomputers as it directly affects running cost and cooling requirements. This paper investigates the memory energy efficiency of high-end data servers used for supercomputers. Emerging memory technologies allow memory devices to dynamically adjust their power states and enable free rides by overlapping multiple DMA transfers from different I/O buses to the same memory device. To achieve maximum energy saving, the memory management on data servers needs to judiciously utilize these energy-aware devices. As we explore different management schemes under five real-world parallel I/O workloads, we find that the memory energy behavior is determined by a complex interaction among four important factors: 1) cache hit rates that may directly translate performance gain into energy saving, 2) cache populating schemes that perform buffer allocation and affect access locality at the chip level, 3) request clustering that aims to temporally align memory transfers from different buses into the same memory chips, and 4) access patterns in workloads that affect the first three factors.

Index Terms—Memory energy consumption, cache replacement algorithms, parallel I/O, cluster storage.

1 Introduction

As the computing capacity increases rapidly in large-scale cluster computing platforms, power management becomes an increasingly important concern. For example, the power density of Google clusters with low-tech commodity PCs exceeds 700 W/ft^2, while the typical cooling capability in data servers lies between 70 and 120 W/ft^2 [2], [3]. A large power consumption in a cluster not only increases its running cost, but also raises its components' temperature through rapid heat dissipation, accordingly reducing the reliability and increasing the maintenance cost. The recent trend toward very-large-scale clusters, with tens of thousands of nodes [4], will only exacerbate the power consumption issue.

Scientific applications usually need to input and output large amounts of data from secondary storage systems [5]. In order to alleviate the I/O bottleneck, cluster supercomputers usually use high-end storage servers with large capacity of main memory. For example, the IBM Blue Gene at LLNL has 32 TB memory [6] and up to 2TB memory can be installed on a single server [7]. Many previous studies [7], [8], [9] have shown that main memory is one of major sources of power consumption. The energy breakdown measured on a real server shows that the memory energy consumption is 41 percent of the total and is 50 percent more than the processors [9]. As the memory capacity continues to increase rapidly in order to bridge the ever-widening gap between disk and processor speeds, memory energy efficiency becomes an increasingly important concern.

• *The authors are with the Department of Electrical and Computer Engineering, University of Maine, Barrows Hall 101, Orono, ME, 04469. E-mail: {jyue, zhu, zcai}@eece.maine.edu.*

Manuscript received 1 Oct. 2007; revised 4 Apr. 2008; accepted 13 June 2008; published online 20 June 2008.
Recommended for acceptance by I. Ahmad, K. Cameron, and R. Melhem.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2007-10-0345.
Digital Object Identifier no. 10.1109/TPDS.2008.109.

In storage servers, most memory space is used as buffer cache. Accordingly, buffer cache management policies heavily influence the overall memory energy efficiency. In particular, under the same workload, different cache placement and replacement algorithms often create significantly different data physical layouts across all memory chips involved. Data physical layouts, however, determine not only access and utilization patterns of each individual memory chip but also the opportunities for each chip to save energy through emergent memory technologies such as power-mode scheduling and multiplexing DMA access.

In this paper, we focus on the evaluation of the memory efficiency of high-end data servers used for parallel applications. We develop a detailed trace-driven memory simulator and use five real-world parallel I/O workloads to compare the relative energy efficiency of eight replacement algorithms, including *LRU*, *Belady*, *LIRS*, *ARC*, *2Q*, *MQ*, *LRFU*, and *LRU2*. We demonstrate that the interplay among cache performance, clustering capability, and cache populating schemes appears to be the most important factors in improving memory energy efficiency. In particular, we have the following conclusions:

- A cache replacement algorithm may directly translate the performance gain in terms of cache hit rates into energy saving. But, it may yet exhibit inferior capability in clustering memory accesses to a minimum number of memory chips and thus waste energy unnecessarily. We show that a good trade off can be achieved if a replacement algorithm can accurately retain both short-term and long-term hot blocks in the same chips.
- The strategies used to allocate buffers before the cache is full, called cache populating schemes, also affect memory energy efficiency. For all replacement algorithms, sequential placement can potentially consume less energy than random placement. This is similar to the conclusion in [8], which advocates

sequential placement in virtual memory under nonscientific workloads. In buffer cache, the energy gain of sequential placement is particularly significant for workloads with mainly sequential or large looping patterns. However, the energy benefits of sequential placement are little for workloads predominated by random accesses or small-looping accesses.

The rest of this paper is organized as follows: Section 2 briefly describes the background, including power-aware memory chips, DMA overlapping and buffer cache replacement algorithms. Section 3 presents our evaluation methodology and simulation results. Section 4 discusses prior related work and Section 5 concludes the paper.

2 Background

2.1 Cache Replacement Policies

The buffer cache performance is theoretically bounded by the optimal *Belady* replacement algorithm [10] that replaces the block whose next reference is farthest in the future. In real systems, *LRU* algorithm or its variants have been widely used. In the past two decades, many new algorithms have been proposed to improve the performance of LRU. These algorithms are described below.

LRU-K dynamically records the Kth backward distance of every block x, which is defined as the number of references during the time period from the last Kth reference to x to the most recent reference to x [11]. A block with the maximum Kth backward distance is dropped to make space for missed blocks. LRU-2 is found to best distinguish infrequently accessed (cold) blocks from frequently accessed (hot) blocks. The time complexity of LRU-2 is $O(\log_2 n)$, where n is the number of blocks in the buffer.

2Q is proposed to perform similarly to LRU-K but with considerably lower time complexity [12]. It achieves quick removal of cold blocks from the buffer by using a FIFO queue $A1_{in}$, an LRU queue Am, and a "ghost" LRU queue $A1_{out}$ that holds no block contents except block identifiers. A missed block is initially placed in $A1_{in}$. When a block is evicted from $A1_{in}$, this block's identifier is added to $A1_{out}$. If a block in $A1_{out}$ or $A1_{in}$ is rereferenced, this block is promoted to Am. The time complexity of 2Q is $O(1)$.

LRFU endeavors to replace a block that is both least recently and least frequently used [13]. A weight $C(x)$ is associated with every block x, and a block with the minimum weight is replaced.

$$C(x) = \begin{cases} 1 + 2^{-\lambda} C(x) & \text{if } x \text{ is referenced at time } t; \\ 2^{-\lambda} C(x) & \text{otherwise,} \end{cases} \quad (1)$$

where λ, $0 \le \lambda \le 1$, is a tunable parameter and initially $C(x) = 0$. LRFU reduces to LRU when $\lambda = 1$ and to LFU when $\lambda = 0$. By controlling λ, LRFU represents a continuous spectrum of replacement strategies that subsume LRU and LFU. The time complexity of this algorithm ranges between $O(1)$ and $O(\log n)$, depending on the value of λ.

MQ uses $m+1$ LRU queues (typically, $m = 8$), $Q_0, Q_1, \ldots, Q_{m-1}$ and Q_{out}, where Q_i contains blocks that have been referenced at least 2^i times but no more than 2^{i+1} times recently, and Q_{out} contains the identifiers of blocks evicted from Q_0 in order to remember access frequencies [14]. On a cache hit in Q_i, the frequency of the accessed block is incremented by 1, and this block is promoted to the most recently used position of the next level of queue if its frequency is equal to or larger than 2^{i+1}. MQ associates each block with a timer that is set to $currentTime + lifeTime$. $lifeTime$ is a tunable parameter that is dependent upon the buffer size and workload. It indicates the maximum amount of time a block can be kept in each queue without any access. If the timer of the head block in Q_i expires, this block is demoted into Q_{i-1}. The time complexity of MQ is $O(1)$.

LIRS uses the distance between the last and second-to-the-last references to estimate the likelihood of the block being rereferenced [15]. It categorizes a block with a large distance as a cold block and a block with a small distance as a hot block. A cold block is chosen to be replaced on a cache miss. LIRS uses two LRU queues with variable sizes to measure the distance and also provides a mechanism to allow a cold block to compete with hot blocks if the access pattern changes, and this cold block is frequently accessed recently. The time complexity of LIRS is $O(1)$. Clock-pro [16] is an approximation of LIRS.

ARC uses two LRU lists L_1 and L_2 for a cache with a size of c [17]. These two lists combinatorially contain c physical pages and c identifiers of recently evicted pages. While all blocks in L_1 have been referenced only once recently, those in L_2 have been accessed at least twice. The cache space is allocated to the L_1 and L_2 lists adaptively according to their recent miss ratios. More cache space is allocated to a list if there are more misses in this list. The time complexity of ARC is $O(1)$. CAR [18] is a variant of ARC based on clock algorithms.

TABLE 1
Power States and Transition Delay of a RDRAM Chip

Power State/Transition	Power (mW)	Delay
Active	300	-
Standby	180	-
Nap	30	-
Powerdown	3	-
Active → Standby	240	1 memory cycle
Active → Nap	160	8 memory cycles
Active → Powerdown	15	8 memory cycles
Standby → Active	240	+6 ns
Nap → Active	160	+60 ns
Powerdown → Active	15	+6000 ns
Standby → Nap	160	+4 ns
Nap → Powerdown	15	~0 ns

2.2 RDRAM Memory Chips

In the RDRAM technology, each memory chip can be independently set to a proper state: active, nap, standby, and powerdown. In the active state, a chip can perform reading or writing and consumes full power. In the other states, the chip powers off different components to conserve energy. In these states, the chip cannot service any read/write requests before it becomes active. The transition from a lower power state to a higher one requires some time delay. Note that RDRAM keeps refreshing all memory cells in lower states, and thus, all data are still accessible after switching to the active state. Table 1 summarizes the power

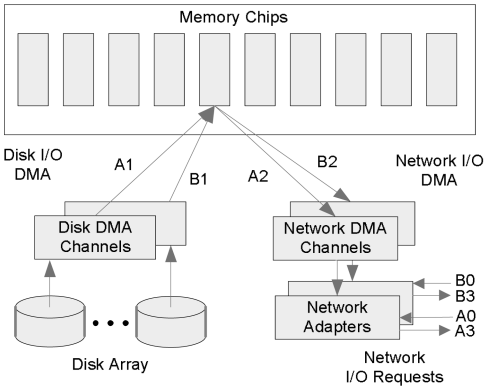


Fig. 1. I/O path during cache read misses in a typical storage server.

consumption rate of each state and the time delay needed to transition among these states.

There are two classes of techniques to control the power state of a memory chip: static and dynamic. Static techniques always set a chip to a fixed low-power state. The chip is transitioned back to full-power state only when it needs to service a request. After the request is serviced, the chip immediately goes back to the original state, unless there is another request waiting. In contrast, dynamic techniques change current power state to the next lower power state only after being idle for a threshold amount of time. The thresholds are dynamically adjusted according to the variation of memory access workload. Since previous studies have shown that dynamic techniques are more energy efficient by avoiding unnecessary power-up and power-down transitions [7], [8], this paper thus only focuses on dynamic ones in energy evaluation.

2.3 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 disk-network data path for two cache misses A and B, following 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 they are cached, 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. For write requests, the data paths are similar but flow in the reverse direction.

On a storage server, recent DMA controllers such as Intel's chipset E8870 and E7500 [19] 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 RDRAM chips [20] and DDR SDRAM are up to 3.2 Gbytes/s and 2.1 Gbytes/s, respectively, while a typical PCI-X bus only gives a maximum rate of 1.064 Gbytes/s, and the second-generation SATA disk DMA throughput is only 300 Mbytes/s.

Multiplexing various slow disk and network I/Os to the same memory chip can reduce the waste of active memory cycles and hence save memory energy. Most DMAs move a large amount of data, usually containing multiple 512-byte disk sectors or 4-KBytes memory pages. Without multiplexing, a memory chip is periodically touched during a DMA transfer, and such access period is too short to justify the transition to a low-power mode [7]. 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 chip, such energy waste can be reduced. For example, when the concurrent requests A and B in Fig. 1 are directed to the same memory chip, the DMA transfers $A1$, and $B1$ can overlap with each other in time and accordingly one of them takes a "free ride" and consumes zero energy, without causing any performance penalty. Similarly, $A2$ and $B2$ can also overlap with each other.

TABLE 2
Summary of I/O Traces

Trace	Data Size (GB)	Working Set (GB)	Run Time (Sec)	Num. of Req.
f1	334.96	80	265	5,488,106
m1	454.30	8.6	265	456,005
ior2	32.28	16	24	528921
mpiBlast	1.12	0.9	7.5	110,327
DB2	229.28	5.2	2,873	3,756,636

3 Energy Evaluation

This section presents the energy evaluation of eight popular buffer cache management algorithms through trace-driven experiments in the light of new memory technologies.

In order to provide a fair comparison between these algorithms, we assume that all buffer cache blocks are physically allocated to a dedicated set of memory chip devices to avoid the disturbances from other competing nonfilesystem memory accesses. In fact, the experiments based on an implementation in Linux presented in [21] show that separation of the buffer cache from the system memory actually improves the overall memory energy efficiency. Similarly, the research on energy-aware virtual memory management strongly recommends to allocate an application's pages into the same chips to minimize the number of chips utilized [7]. In addition, this paper limits our investigation scope only to the energy consumed by all memory chips used for buffer caching and ignore other hardware components, although energy scavenging for the whole system is very important.

3.1 File System I/O Traces

Two sets of I/O traces are used in this study. The first set includes scientific computing I/O trace [22], and the second set includes commercial applications I/O traces [23]. All traces consist of file system I/O calls made by applications such as file operations of open, seek, read, write, and close. Table 2 summarizes the statistics of these traces.

3.1.1 Scientific Computing I/O Traces

Two sets of parallel I/O traces are used in this study.

The set of traces include three parallel scientific applications, *ior2*, *m1*, and *f1* are collected from large supercomputer clusters with more than 800 dual-processor nodes at the Lawrence Livermore National Laboratory (LLNL) [22]. These traces are collected in a parallel file system that runs on multiple data servers. The traces consist of file system calls made by all client processes of a given scientific application. In our experiments, we replay these traces on a large RAID storage server. We faithfully preserve system call numbers, call parameters, and their call order for all client processes of a given scientific application. We believe that this approach can effectively emulate a hypothetical scenario that these application run on a centralized storage system. A detailed description to these scientific applications is given in [22]. The following summarizes the trace characteristics.

ior2 is a parallel file system benchmark suite developed at LLNL [24]. Based on typical data access patterns of scientific parallel applications, this benchmark suite includes three separate benchmarks: *ior2-fileproc*, *ior2-shared*, and *ior2-stride*. The traces of these benchmarks are collected on a 512-node cluster. The *ior2-fileproc* benchmark assigns a different output file for each node and has the best write performance. It achieves 150,000 write requests per second, resulting in an aggregate throughput of 9 Gbytes per second. While *ior2-fileproc* uses a model of one file per node, *ior2-shared* and *ior2-stride* takes the shared-region and shared-stride data access models, respectively. All the nodes simultaneously access a shared file sequentially in *ior2-shared* and noncontiguously with a varying stride between successive accesses in *ior2-stride*.

f1 is a large-scale physics simulation running on 343 nodes. This application has two I/O-intensive phases: the restart phase and the result-dump phase. In the first phase, data are retrieved from a shared file independently by all involved computing nodes. Thus, read operations dominate in this phase. In the result-dump phase, a small set of nodes periodically gather a large amount of simulated results from the others and concurrently save collected results into a shared file. This phase is dominated mostly by writes. The corresponding traces collected are named as *f1-restart* and *f1-write*. The *f1* trace has representative I/O accesses pattern existed in scientific applications: a master node periodically collects and saves intermediate results generated by other computation nodes [22].

m1 is an ever-larger physics simulation that runs on 1,620 nodes. This application uses an individual output file for each node. Similar to the previous application, it also has a restart phase and a result-dump phase. The corresponding traces are referred to as *m1-restart* and *m1-write*. Compared with *f1*, *m1* has a similar yet different I/O behavior. Similar to *f1*, *m1* is also divided into two phases, write and restart. Different from *f1*, all nodes write roughly the same amount of data, and there are also significant amounts of write requests in *m1-restart*.

BLAST [25] is one of the most widely used tools in computational biology for a sequence similarity search. Given a query sequence and a sequence database as inputs,

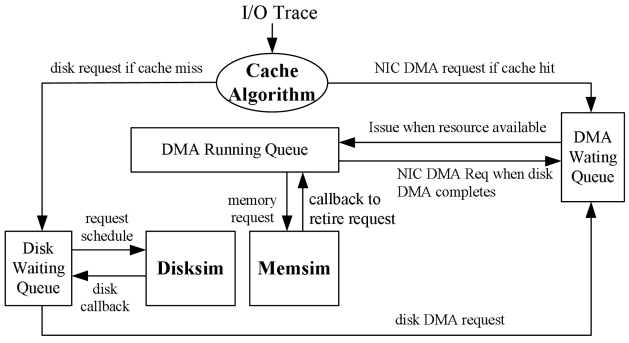


Fig. 2. Event-driven simulation framework.

BLAST searches all entities in the database for those with high-scoring gapped alignment to the given query, where the deletion, insertion, and substitution are allowed in sequence comparison, and the alignment scores are determined statistically and heuristically based on expert-specified scoring matrix. In order to parallelize BLAST program, the sequence database is divided into a number of segments stored on different nodes. The parallel BLAST (*mpiBLAST*) programs consist of master and a number of workers [26]. Each worker excuses the NCBI *blastall* to searches its local segment using the query assigned by the master. The sequence database BLAST *nt* has 1.76 million sequences stored at files with total size of 2.3 Gbytes. The mpiBLAST trace was collected from eight workers searching against eight nt database fragments and consists of 144 I/O accesses. These I/Os includes both small and large reads with a few small writes. Eighty-nine percent operations were reads with data size ranging from 14 bytes to 220 Mbytes, and their average size was 31.29 Mbytes. Eleven percent operations was writes, and the maximal and minimal write size was 778 and 50 bytes, respectively, with a mean of 690 bytes [27].

3.1.2 Commercial Applications I/O Traces

The traces of *DB2* collected in an 8-node IBM DB2 Parallel Edition database system [23]. These *DB2* traces have a total of 3.7 millions of I/O references and 229.28 Gbytes data traffic. The total database size is 5.2 Gbytes, and it was stored in 831 files. Since the traces were collected on old machines in 1997, we choose to scale up the traces by reducing the runtime with a factor of 1,000 in this study.

3.2 Simulation Framework

Our simulation framework is composed of three major components: cache simulator, disk array simulator, and memory simulator. Disksim [28], a well-validated disk array simulator, is incorporated into our framework to precisely emulate the timing of disk I/O traffic. These three components interact with each other through three event queues shown in Fig. 2. We use Disksim APIs with callback functions to generate disk DMA requests. The memory simulator and the cache simulator coordinates with each other to determine the physical chip address for each block. Before the cache is full, the memory simulator resolves the chip address for each missed block based on the cache populating schemes. The populating schemes are discussed

later in Section 3.6. After the buffer cache is full, the cache simulator determines logically the victim block and the chip address of the new block will simply be that of the victim block. In addition, for each chip, the memory simulator maintains the power state transitions based on a time-out mechanism used in [8], and also, the DMA overlapping operations are simulated under the help of the DMA running queue. The total energy consumption, defined as products between power consumption rates and time, is increased during the simulation based on the power state transitions of each chip.

It has been a great challenge for us to validate our simulation framework. The main reason is that it is very difficult to accurately measure the memory energy consumption [29]. First of all, the trace collector's memory accesses cannot be easily separated out physically, thus generating significant measurement errors. Second, due to the complexities of memory accesses, it is almost impossible to manually validate the energy for a small memory access stream. As a result, to our best knowledge, there is no validation for all memory energy simulators expect IBM's proprietary memory simulator [30], which is validated against a proprietary RTL simulator containing 1.5-million lines of VHDL codes developed at IBM. To overcome these challenges, Lee et al. [21] use the product of memory access 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.

3.3 Simulation Environment

We have developed a detailed trace-driven simulator that can accurately emulate network DMA and disk DMA operations and report the energy consumption of memory chips. In storage servers, both DMAs are heavily involved. Through disk DMAs, data missed in the cache or dirty blocks are exchanged between memory chips and disk drives. Through network DMAs, the requested data are sent to clients from the memory through NICs. With new technology introduced, multiple DMAs on different buses can simultaneously access the same chip in a multiplexing way. The simulated data sever is configured with six network adaptors and 12 disks. Each device has its own independent DMA channel with a bandwidth of 200 Mbytes/s Disksim [28], a well validated disk array simulator, is incorporated into our simulator to precisely emulate the timing of disk I/O traffic.

The simulator adapts the RDRAM memory chips, whose parameters are given in Table 1. Each chip capacity is 32 Mbytes and can support up to 16 concurrent DMA operations (3.2 Gbytes/s). The simulator models the chip's power state transition, the DMA operation contention and queuing processes. The time-out thresholds for powerdown, nap, and standby are based on the breakeven time given in [8]. While the simulation results reported in this paper are based on RDRAM memory systems, our simulator is also applicable to DDR SDRAM technologies, where we can treat entire DDR modules as we do single RDRAM chips.

We simulate the traces by replaying all I/O events at predetermined times specified in the traces, independent of the performance of memory hierarchy. This approach is used mainly because all traces that we have access to do not record the dependence among request completion and subsequent I/O arrivals. Such dependence exists at both inter- and intraprocess levels. However, the dependence at the interprocess level does not widely exist in many scientific and database workloads. For parallel scientific applications, Purakayastha et al. [31] conclude that files, particularly write-only files, are often not shared between jobs, and Pasquale and Polyzos [32] shows that application codes are highly structured and often utilize carefully formatted data sets without overlap. Similarly, in parallel database applications, I/O requests issued from independent processes running on different machines [23] are also likely be independent. The dependence at the intraprocess level cannot be easily extracted from a system and recorded in the traces [33]. The traces used in this paper do not provide I/O dependence information, and our simulations fail to preserve I/O dependence, which is one of the limitations of this study.

3.4 Energy Comparisons under Sequential Placement

From the operating systems' point of view, the energy consumption of buffer cache is mainly influenced by the following three factors:

1. How does the buffer cache get populated with blocks? There are two well-known policies, including sequential first-touch policy and random placement. The former allocates buffers in the order they are accessed, filling an entire RDRAM chip before moving on to the next one. The latter is to allocate buffers randomly with respect to chip selection. The buffer cache management module in most operating systems uses random placement to populate the cache, without considering which chips the requested buffers are physically located.
2. How well does the cache algorithm capture temporal locality? A higher hit rate helps reduce the total number of memory accesses made by disks. Such performance gain often translates into lower power consumption by reducing runtime.
3. What is the cache algorithm's efficiency in clustering memory accesses to a minimum number of active chips? Clustering memory access to a small set of chips helps save energy from two aspects. Not only does it decrease the average number of memory chips that are simultaneously active during the runtime, but also increase the level of concurrency between multiple DMA transfers from different I/O buses to the same memory chip.

In this section, we assume that the buffer cache is initially populated by using the sequential first-touch policy since this approach has the best energy efficiencies. We discuss the impact of populating policies in the following section. Hence, we only focus on the study of the first two factors in this section. For the convenience of comparisons, all energy measurements and the completion time are rated to their corresponding values of the buffer cache configuration that has the least cache size and is managed by the *Belady* algorithm.

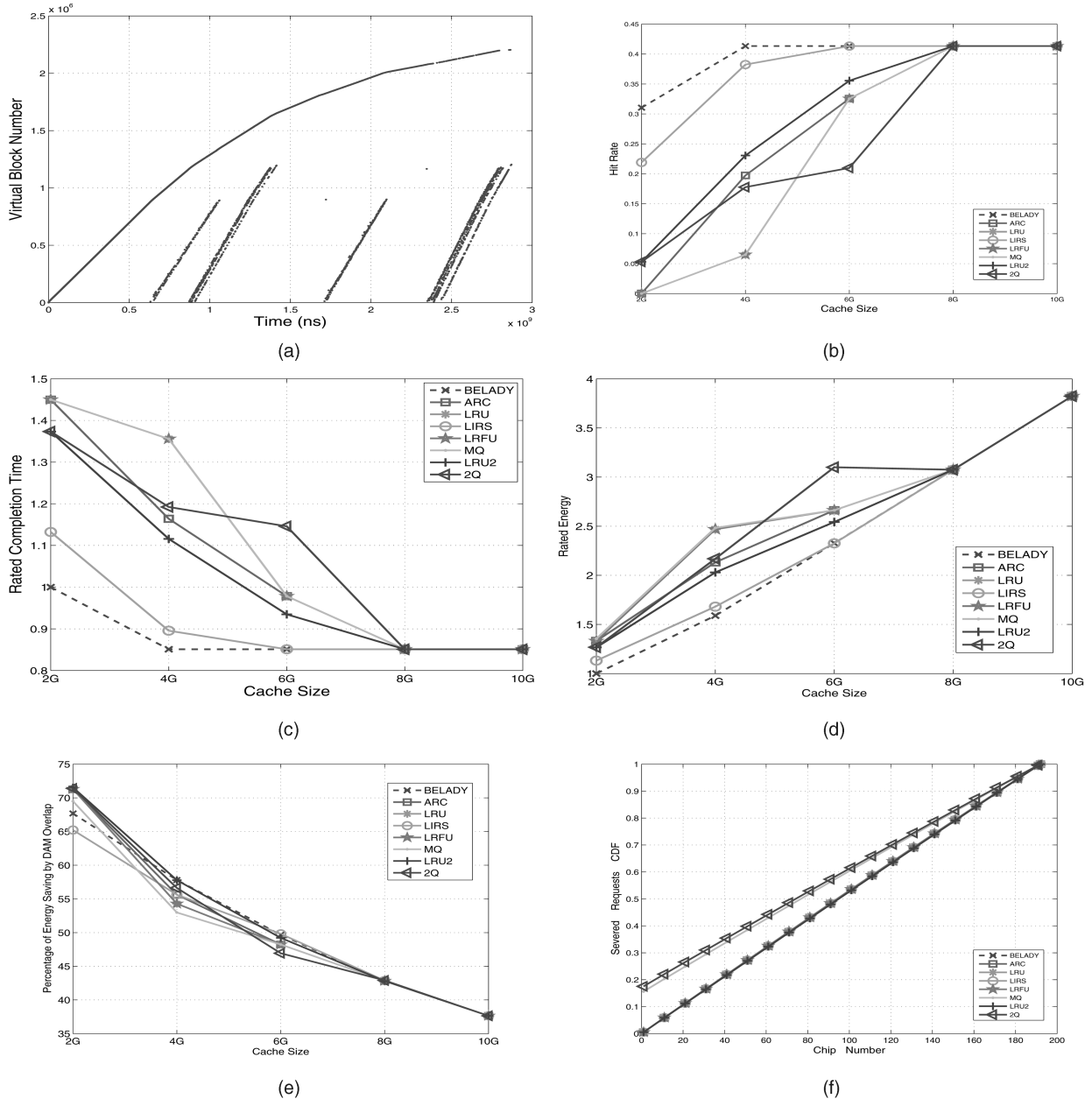


Fig. 3. Comparison of replacement algorithms in workload ***DB2***. (a) *DB2* trace (sampling period: 1,000). (b) Hit rate. (c) Runtime. (d) Energy consumption. (e) Percentage of energy saving by DMA overlap. (f) CDF of requests among banks (Cache size: 6 Gbytes).

3.4.1 Parallel Database Applications DB2

The experimental results of *DB2* are presented in Fig. 3. The energy measurements in Fig. 3d are normalized to the energy consumed in a 2-Gbytes cache that is managed by *Belady*. Similarly, the runtimes in Fig. 3c are normalized to the runtime of a 2-Gbytes cache that is managed by *Belady*. The percentage of energy saving by DMA overlapping, as shown in Fig. 3e, is defined as the ratio of energy saved through DMA overlapping to the total energy consumed when DMA overlapping is disabled. Fig. 3f presents the cumulative distribution function (CDF) of requests among chips. A point (x, y) in the cumulative distribution curve indicates that x memory chips service y percent of the total memory DMA transfers.

We observe that in *DB2*, the cache algorithms can directly translate performance gain into energy saving. The *DB2* trace, as shown in Fig. 3a, have a combination of large sequential and looping accesses. Figs. 3b and 3d show that the memory energy is mainly determined by cache hit rates. For all cache sizes, the energy consumptions of different algorithms are almost inversely proportional to their hit rates. Under the same cache size, a higher hit rate leads to a shorter runtime and a smaller energy consumption. For example, the *Belady* has 24.3 percent higher hit rates than *2Q* on the average, resulting in an average of 24.5 percent energy saving. In addition, all overlapping benefits decrease nearly proportionally as the memory size increases (see Fig. 3e). Intuitively, as there exist more memory chips in systems, memory transfers have less chances to overlap due to reduced utilization on each chip. Furthermore, the percentages of energy saving by DMA overlapping exhibits no significant differences among

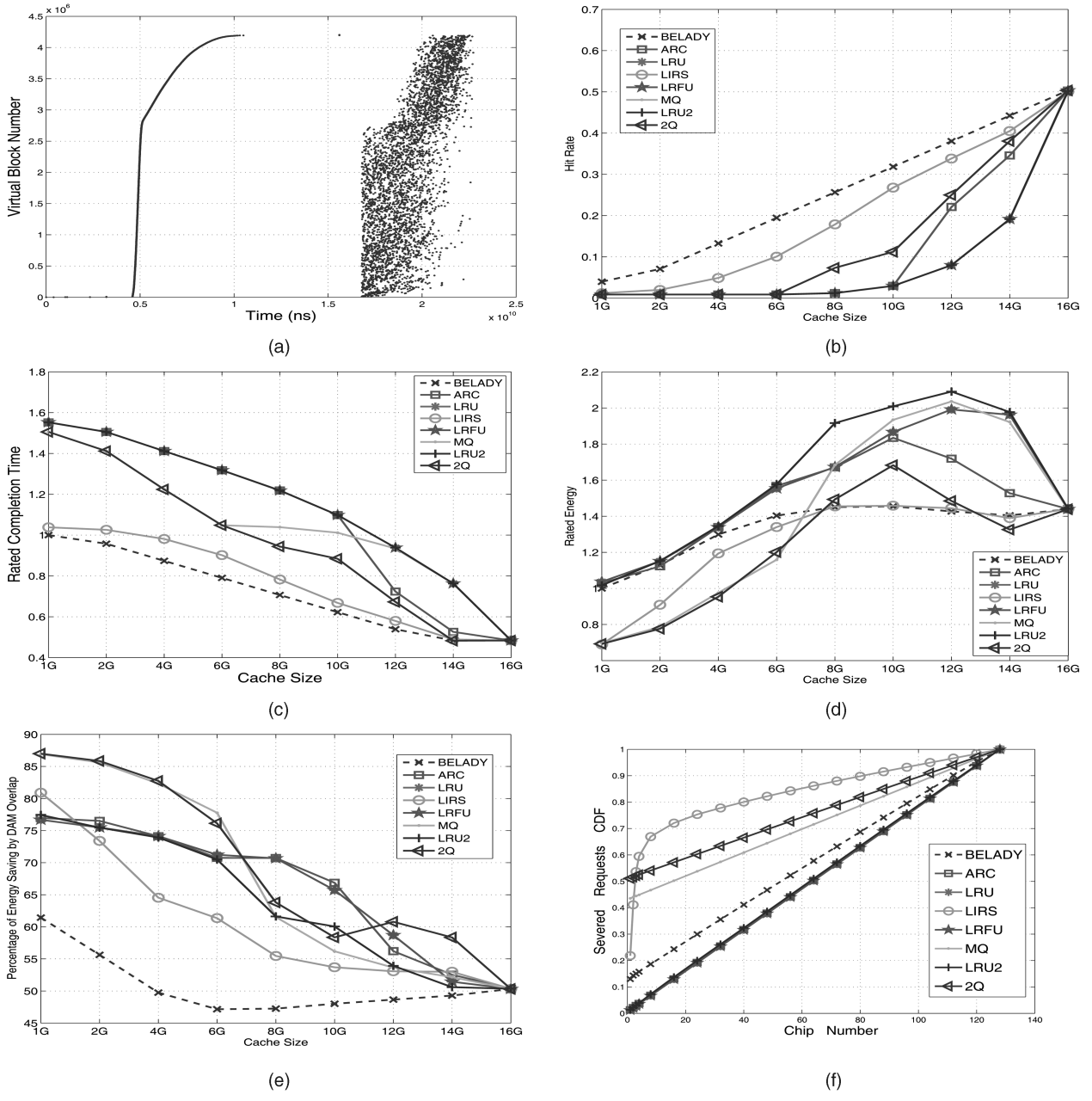


Fig. 4. Comparison of replacement algorithms in workload ***ior2***. (a) *ior2* trace (sampling period: 1,000). (b) Hit rate. (c) Runtime. (d) Energy consumption. (e) Percentage of energy saving by DMA overlap. (f) CDF of requests among banks (Cache size: 4 Gbytes).

different replacement algorithms, which indicates that the effect of overlapping in all algorithms are almost identical. This explains why only cache performance determines the memory energy efficiency in this workload.

3.4.2 Parallel I/O Benchmark ior2

The parallel I/O benchmark *ior2* [24] aims to emulate the I/O behaviors of data-intensive scientific applications. The *ior2-shared* benchmark used in this study has both sequential accesses and random accesses (see Fig. 4a). The random access pattern is created by 512 interleaved parallel I/O streams. The following observations are made under workload *ior2*.

First, *LIRS* is more energy efficient than *Belady*. For example, when the cache size is 1 Gbyte, the active energy of *LISR*, *MQ*, and *2Q* is only 43 percent, 55 percent, and 55 percent of *Belady*'s active energy, respectively. This can explain why *Belady* consumes more energy even though it has a shorter runtime. When the cache size is larger than 8 Gbytes, the total energy consumption of all algorithms, except *LIRS* and *arc*, starts to decrease due to reduced runtimes. When the cache size reaches 16 Gbytes, slightly exceeding the working set of *ior2*, all algorithms converge to the same values since no cache replacement occurs.

Second, the energy efficiency of *2Q* and *MQ* is due to their better capability of I/O clustering. Both algorithms are more likely to retain long-term hot data blocks in the same cache chips. Fig. 4f plots the cumulative distribution curve when the cache size is 4 Gbytes. It shows that a single chip absorbs 51 percent and 43 percent data accesses in *2Q* and

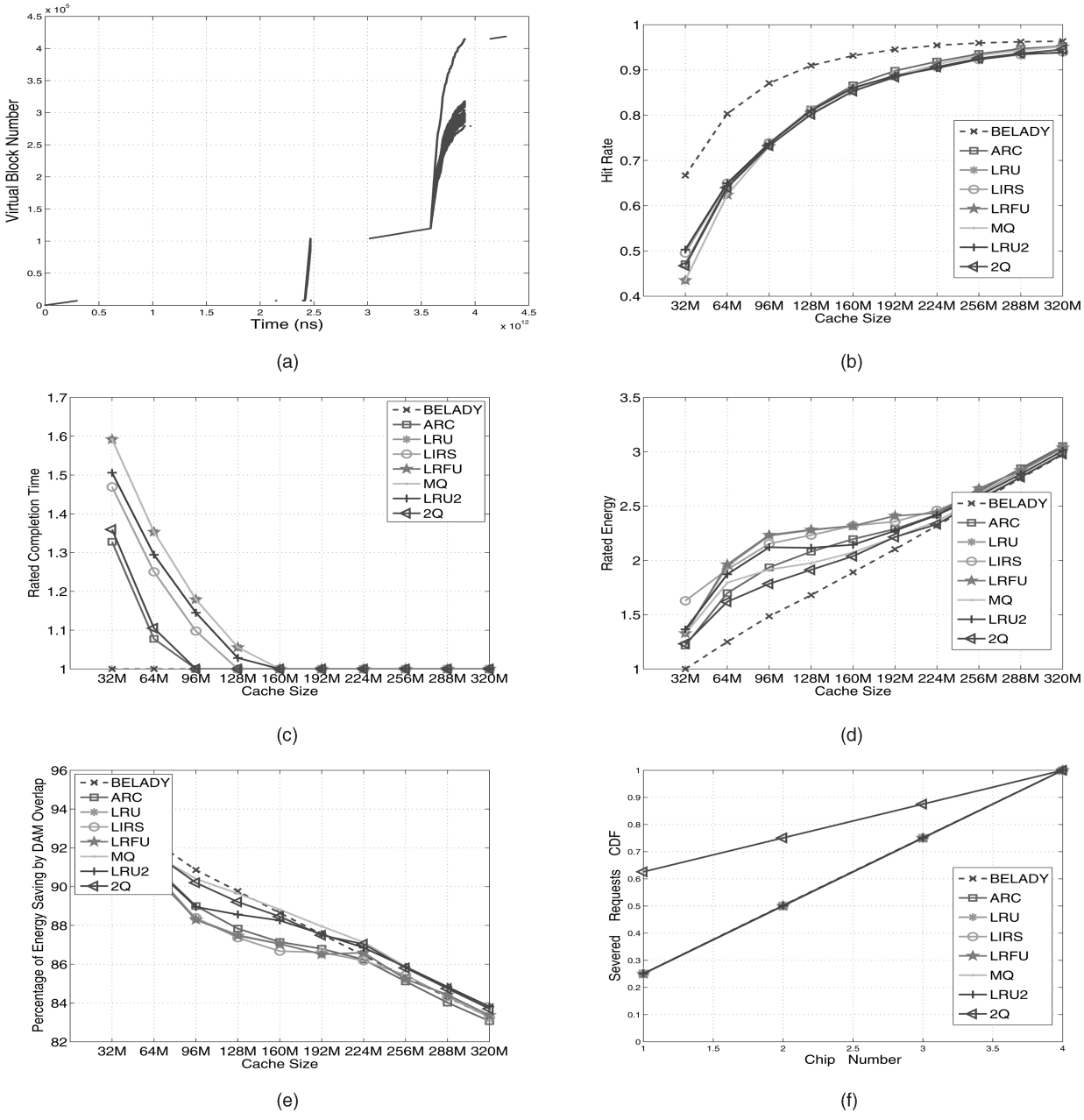


Fig. 5. Comparison of replacement algorithms in workload ***f1***. (a) *f1* trace (sampling period: 1,000). (b) Hit rate. (c) Runtime. (d) Energy consumption. (e) Percentage of energy saving by DMA overlap. (f) CDF of requests among banks (cache size: 128 Mbytes).

MQ, respectively. That is mainly because *2Q* and *MQ* usually do not evict out hot blocks, and accordingly, these hot blocks are never moved among different chips. Both algorithms use several separate queues to store blocks with different temporal locality. They filter out blocks with high access frequency and promote them to separate queue(s). During a cache miss, the blocks in these queues typically have a higher priority of staying in the cache.

3.4.3 Large-Scale Physics Simulations f1 and m1

One important observation in *f1* is that these replacement algorithms have different energy efficiency even if they have nearly the same hit rates. For example, at the cache size of 128 M, the energy difference among all algorithms, except for *Belady*, is up to 16.2 percent, while their corresponding hit rates and runtimes differ by only up to 1 percent and 5 percent, respectively. Specifically, the energy consumption of *2Q* and *MQ* are smaller than *LRFU* by 16.2 percent and 13.5 percent, respectively. These results show that these algorithms inherently have different effects on temporally aligning memory transfers.

In particular, we find that *2Q* and *MQ* provide better opportunities for temporally aligning memory transfers into the same set of chips. For example, when the cache size is 128 Mbytes, one single chip in *2Q* and *MQ* services 62.6 percent of DMA memory transfers, while a chip in the other algorithms only attracts up to 20 percent (see Fig 5f). Such heavily skewed utilization creates larger chances for *2Q* and *MQ* to save energy. As a result, the percentage of energy saving by access overlapping of *2Q* and *MQ* achieves 13 percent and 10 percent, respectively (see Fig. 5e).

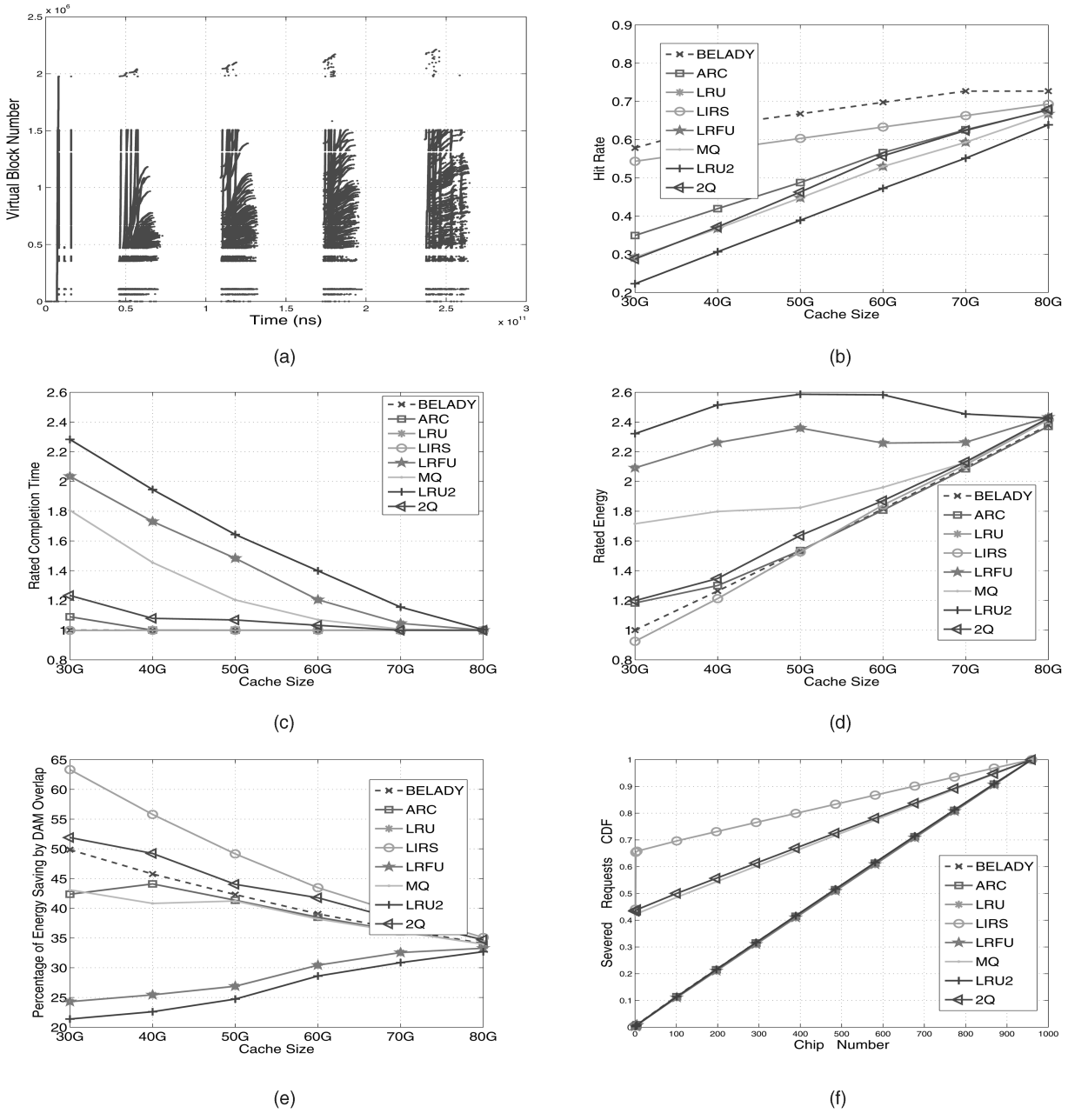


Fig. 6. Comparison of replacement algorithms in workload *m1*. (a) *m1* trace (sampling period: 1,000). (b) Hit rate. (c) Runtime. (d) Energy consumption. (e) Percentage of energy saving by DMA overlap. (f) CDF of requests among banks (cache size: 30 Gbytes).

The *m1* trace exhibits predominantly periodical accesses, as shown in Fig. 6a. In this application, the energy efficiency is mainly determined by the cache performance in terms of cache hit rates. In fact, the total energy consumptions, under different cache sizes as shown in Fig. 6d, is almost inversely proportional to the cache hit rates presented in Fig. 6b. A gain in cache hit rates leads to a decrease of the runtime, as well as the number of memory DMA transfers. Accordingly, this performance gain is directly translated into lower power consumption.

It is interesting to observe that *LIRS* saves slightly more energy than *Belady* that has optimal hit rates. This observation is an exception to the conclusion made above. Fig. 6f indicates *LIRS* clusters 66 percent of memory accesses within four memory chips, while *Belady* distributes all accesses almost evenly across all memory chips. Such a scattered distribution in *Belady* causes an unnecessary amount of memory chips to stay in the active state simultaneously and reduces the opportunity of energy saving through access overlapping.

3.4.4 Parallel Bioinformatics Application mpiBLAST

Under the *mpiBLAST* workload, *LIRS* has lower hit rates but surprisingly is the most energy efficient under most cache size configurations. A further analysis shows that *LIRS* successfully clusters memory accesses to the same chips and achieves more energy saving through DMA overlapping. The *mpiBLAST* trace, as shown in Fig. 7a, is dominated by eight large sequential and looping accesses, and this pattern repeats two times. Fig. 7b classifies these cache algorithms

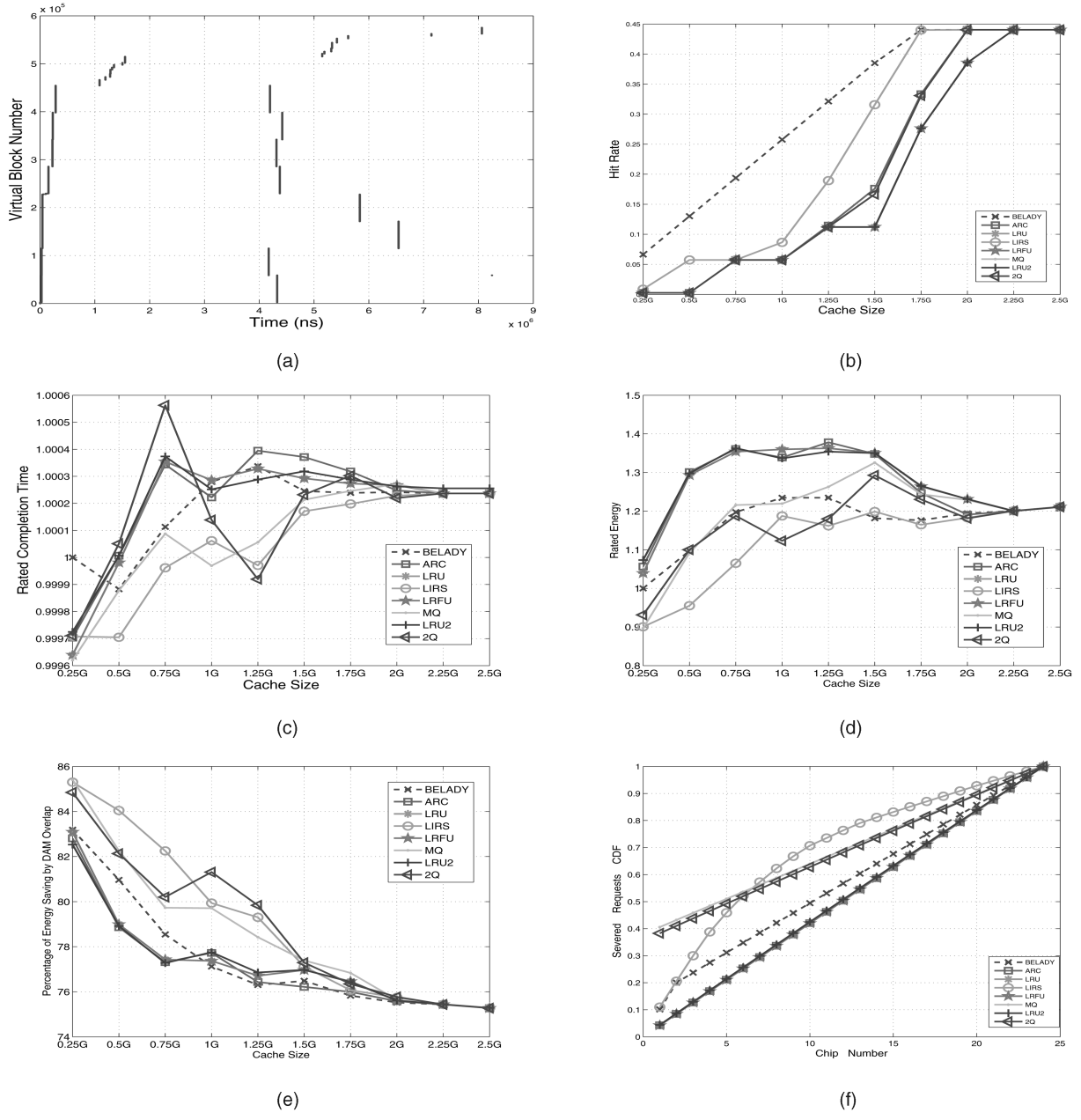


Fig. 7. Comparison of replacement algorithms in workload ***mpiBLAST***. (a) *mpiBLAST* trace (sampling period: 1,000). (b) Hit rate. (c) Runtime. (d) Energy consumption. (e) Percentage of energy saving by DMA overlap. (f) CDF of requests among banks (cache size: 0.75 Gbytes).

into the following three groups: the first group has *Belady*, the second group has *LIRS*, and the remaining algorithms belong to the third group. Fig. 7d shows that the memory energy is mainly determined by cache hit rates. For all cache sizes, the energy consumptions of different algorithms are almost inversely proportional to their hit rates. Under the same cache size, a higher hit rate leads to a smaller energy consumption. For example, the *Belady* has 12.29 percent higher hit rates than *LRU* on the average, resulting in an average of 7.95 percent energy saving. In addition, all overlapping benefits decrease nearly proportionally as the memory size increases (see Fig. 7e). Intuitively, as there exist more memory chips in systems, memory transfers have less chance to overlap due to reduced utilization on each chip. Accordingly, they converge after cache size reaches 2.25 Gbytes. It is also noticed that the percentage of energy saving by DMA overlapping remains as 75.5 percent when the cache size exceeds 2.25 Gbytes. This is because the *mpiBLAST* I/Os are very bursty and include 16 large read operations with an average request size of approximately 200 Mbytes. Such large I/O operations create more chances for I/O overlapping, especially when disk DMA are not involved. Furthermore, the percentage of energy saving by DMA overlapping exhibits no significant differences among different replacement algorithms, which indicates that the effect of overlapping in all algorithms are almost identical. This explains why only cache performance determines the memory energy efficiency in this workload.

3.5 Comparison of Clustering Capabilities

In the previous discussion, we have found that the ability of a given algorithm in clustering hot blocks into the same chips may affect significantly the total memory energy

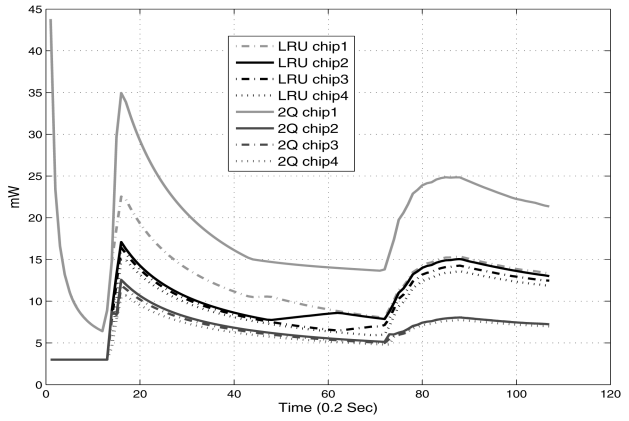


Fig. 8. Power consumption rate of memory chips under workload ***f1*** (cache size $= 128$ Mbytes).

consumption. Specifically, there is a strong correlation between CDF curves and total energy. We use 2Q and LRU in *f1* with a cache size of four memory chips as an example to illustrate this correlation. The reason why this specific example is used is simply that the power consumption rates of each chip, given in Fig. 8, is easy to read in this case. As shown in Figs. 8 and 5f, the busiest chip that serves approximately 60 percent requests in 2Q consumes 38.7 percent much more energy than the busiest chip that serves roughly 25 percent requests in LRU. Additionally, the other three chips serve approximately the same amount of accesses and interestingly their energy consumptions are almost the same too. The main reason for such correlations is that CDF curves not only indicate workload spatial distributions across all memory chips but also imply temporal access locality at the chip level. This motivates us to study CDFs and compare the clustering capabilities of these algorithms.

To measure the clustering ability of a given cache replacement algorithm, we adopt a concept called skewness, defined as the percentage of requests served by the memory that serve the most amount of requests among all chip utilized. In fact, the skewness is simply the percentage of requests served by the first chip in a given CDF curve.

Before discussion, let us first look at the differences between *2Q* and *LIRS*. In order to keep hot blocks longer in the cache and evict cold blocks quickly, both algorithms use ghost caches. In *2Q*, the ghost cache is $A1_{out}$, while the ghost cache is nonresident HIR entries in *LIRS*. However, the ghost cache size in *2Q* is typically half of the physical block entries, but *LIRS*'s ghost cache varies with workloads and can be much larger than *2Q*. The larger size of ghost cache can potentially help reduce inaccuracy in capturing hot blocks and accordingly result in positive effects on the performance. The second important difference between *2Q* and *LIRS* is the size of cache holding cold blocks. *2Q* uses 25 percent while *LIRS* uses only 1 percent. The third difference is that *2Q*'s $A1_{in}$ can filter short-term hot blocks.

The skewness of *2Q* and *LIRS* are 62.6 percent and 25 percent, respectively, in workload *f1*, and 67.5 percent and 25 percent, respectively, in workload *DB2*. Under both workloads, *LIRS* has a similar CDF to the others except *2Q*. *DB2* is dominated by large sequential accesses followed by multiple loops (see Fig. 3a). These loops contain time-varying hot blocks, and they are mostly short term. *LIRS* does not distinguish short-term hot blocks from long-term ones and retains them all in the cache. When old short-term hot blocks are evicted out and then new short-term hot blocks are moved in, the memory layout is more likely to be disturbed, which causes hot blocks to occupy more chips. This explains why *2Q* has a larger skewness than *LIRS* in *DB2*. For the same reason, the large sequential and many short-term hot blocks access patterns in *f1* (see Fig. 5a) make *LIRS* have a weaker capability in retaining long-term hot blocks than *2Q*.

Under workload *ior2*, the *LIRS*'s ability of clustering hot blocks is again inferior to *2Q*. The *ior2* workload is dominated by a large sequential access and many random accesses whose hot blocks change with time. Under such workload, especially at the random access phase, *2Q*'s $A1_{in}$ can prevent short-term hot blocks from being placed in the longer term hot block queue Am. Hence, *2Q* can capture longer term hot blocks in the cache and achieve better clustering capability.

However, under workload *m1*, the *LIRS* CDF is superior to *2Q* (see Fig. 6f). The *m1* shows both large and small looping accesses with different looping periods. Since typically *LIRS* can provide more space to hold hot data than *2Q*, *LIRS* can better identify hot blocks and, thus, avoids unnecessary paging-out and paging-in to hot blocks. Additionally, *LIRS* has a larger ghost cache that also helps accurately identify hot blocks. These two advantages over *2Q* result in better clustering effects.

Under workload *mpiBLAST*, the hit rate of *LIRS* is 6.4 percent lower than *BELADY* on the average, while the *LIRS* energy is 5 percent smaller than *BELADY* on the average. This demonstrates that the factors rather than hit rate can significantly affect memory energy consumption. Fig. 7f shows that the first five chips can serve 46 percent accesses in *LIRS*, while the first five chips can serve only 31.5 percent accesses in *BELADY*. Hence, the *LIRS* CDF curve is above the *BELADY*. This concludes that *LIRS* has stronger capability to cluster hot data to a smaller set chips than *BELADY* under this workload. In addition, the CDF curves of *MQ* and *2Q* are also above *BELADY* shown in Fig. 7f. However, their superior clustering ability is offset by their lower hit rates, resulting in more energy consumptions than *BELADY*.

From the above discussions, we conclude that *LIRS* can better capture hot blocks both in short term and long term, and thus, it typically has superior cache hit ratios than *MQ* and *2Q*. On the other hand, *MQ* and *2Q* only retain well long-term hot blocks. Thus, in *MQ* and *2Q*, these long-term hot blocks may avoid some unnecessary memory paging, and these hot blocks stay in the same chips. As a result, *MQ* and *2Q* can better align memory accesses to the same chips even though they may be inferior in hit rates.

3.6 Sequential Placement versus Random Placement

This section examines the benefits of two buffer cache populating strategies: sequential placement and random placement. While the former allocates buffers in the order they are accessed, filling an entire chip before moving to the

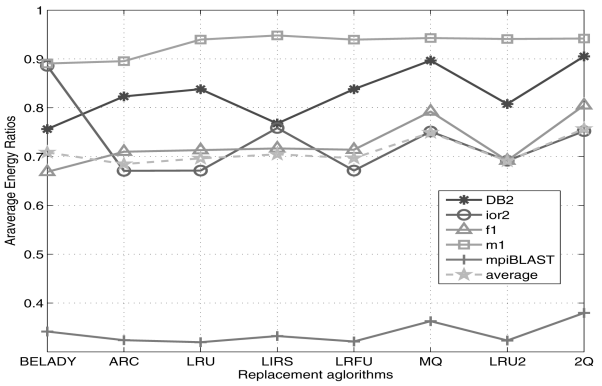


Fig. 9. Average energy consumption ratios of sequential placement to random placement under the same configuration.

next, the latter randomly selects a chip to allocate buffers. We normalize the energy consumption of sequential placement normalized to random placement under the same cache size and replacement algorithm. The energy of random placement is averaged over three repeated experiments.

We conclude that sequential placement is more energy efficient than random placement in all parallel I/O traces studied. As shown in Fig. 9, the average normalized energy across different cache sizes for *Belady*, *ARC*, *LRU*, *LIRS*, *LRFU*, *MQ*, *LRU2*, and *2Q* is 29.0 percent, 31.5 percent, 31.4 percent, 29.6 percent, 30.7 percent, 25.2 percent, 31 percent, and 23.3 percent, respectively. The average saving across all algorithms is 28.96 percent. Our observation is consistent with the conclusion made in the literature on conventional non-file-I/O workloads. For example, Lebeck et al. [8] report that sequential placement achieves 12 percent to 30 percent energy saving. Currently, operating systems widely used in HPC, such as BSD variants, Solaris, and Linux, allocate memory frames, especially buffer and page caches, with little considerations of chip selection. Consequently, contiguous memory regions often become fragmented. Our experimental results build a compelling reason for HPC designers to incorporate energy-aware data placement into the buffer cache management unit.

Another interesting observation is that sequential placement benefits energy-efficiency more significantly in workloads dominated with sequential access or large looping patterns. In *m1* that is dominated with small local looping accesses, the average energy saving of all algorithms is only 7 percent. However, in *f1* with long sequential accesses and *mpiBLAST* with large accesses, the average saving achieves 27.4 percent and 65.7 percent, respectively. *mpiBLAST* has large I/O requests, with an average of 220 Mbytes. Such workload with large requests greatly favors for sequential placement due to fact that large I/Os generate a large sequential cache area victimized for new I/Os in the sequential placement. Since the parallel I/O patterns can often be characterized as large, striping, and concurrent accesses [27], we conjecture that sequential placement under in HPC systems would present a larger energy benefit than it would in conventional systems.

4 Related Work

Until recently, power consumption was an issue primarily in embedded or portable computer systems. However, energy efficiency is becoming an increasingly important concern in the high-performance computing (HPC) community. References [34], [35], [36], [37], and [38] aim to reduce the CPU energy consumption in a cluster environment by using dynamic voltage scaling to slow down the CPU speed. Lawson and Smirni [39] propose an energy-saving scheme that dynamically adjusts the number of processors in a parallel system that operates in "sleep" mode. There are also studies in optimizing disk energy efficiency for scientific applications [40].

On individual servers, many research studies have been conducted to save memory energy. It is proposed in [41], [42], [43], and [44] to adaptively control the memory power states, instead of relying on simple threshold mechanisms. It is proposed in [6], [7], [8], and [45] to save energy in memory management by judiciously allocating or migrating memory pages to cluster an application's pages into a minimal number of chips. Li et al. [46] and Cai and Lu [47] aims to optimize the overall energy efficiency of both memory chips and disk drives. While almost all the research work mentioned above is designed for virtual memory, very little research work has been done for buffer cache. Pandey et al. [7] propose two schemes to save energy in data servers: temporally aligning DAM transfers to the same memory chips through buffering and migrating data among chips to minimize the number of active chips. Zhu and Zhou [48] propose a new buffer cache replacement algorithm to reduce the disk energy consumption.

5 Conclusion

We have developed a detailed trace-driven simulator that emulates the behavior of different cache management schemes. This simulator allows us to quantify the energy impact of eight different cache replacement algorithms including *ARC*, *Belady*, *LRU*, *LIRS*, *LRFU*, *MQ*, *LRU2*, and *2Q*. Under the same workload, the interplay among the following three important factors appears to be the most important: 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. In particular, when the total size of hot blocks is smaller than the size of the cache partition holding hot blocks, *2Q* can save more energy than *LIRS*. Otherwise, *LIRS* is more energy efficient than *2Q*. In large-looping access workloads, a gain in cache rates can be directly translated into better energy efficiency. However, this observation cannot to be applied generically to workloads with more complex access patterns. Additionally, sequential placement can potentially save more energy than random placement in all replacement algorithms, especially under the large size I/O operations. However, such energy benefit diminishes for workloads with mainly random accesses and small-looping accesses. By quantifying and, thus, prioritizing the many factors that may impact the overall energy consumption, we see this study as a first step toward modifying existing replacement algorithms or designing a new one that can optimize the energy saving by striking the optimal trade-off among the important factors. This study also allows us to better understand the performance and energy efficiency of these cache replacement algorithms under parallel I/O workloads.

Limitations and future work. We have had great challenges in validating our simulators due to the lack of customized hardware for memory energy measurements. In our study, our study is limited to buffer cache memory only and ignore all other components. The I/O traces also impose two limitations: all memory-mapped I/O accesses were not collected in the traces, and the dependence between I/O accesses was not recorded in the trace, and thus, our simulator could not faithfully emulate such dependence. Our future work focus on addressing these limitations, studying the impacts of page migrations, and developing analytical energy models and new power-aware buffer cache replacement algorithms.

ACKNOWLEDGMENTS

This work is partially supported by a UMaine Startup Grant, US National Science Foundation (NSF) Grants (CCF 0621493, 0754951, CNS 0723093, and DRL 0737583), a NASA Maine Space Grant, and an equipment grant from SUN. This paper is extended from a conference publication [1].

REFERENCES

[1] J. Yue, Y. Zhu, and C. Zhao, "Evaluating Memory Energy Efficiency in Parallel I/O Workloads," *Proc. IEEE Int'l Conf. Cluster Computing (Cluster '07),* pp. 21-30, Best Paper Award, Sept. 2007.

[2] L.A. Barroso, J. Dean, and U. Holzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro,* vol. 23, no. 2, pp. 22-28, 2003.

[3] B. Moore, "Take the Data Center Power and Cooling Challenge," *Energy User News,* Aug. 2002.

[4] H. Meuer, E. Strohmaier, J. Dongarra, and H.D. Simon, *Top 500 Supercomputers,* http://www.top500.org, 2005.

[5] Y. Zhu and H. Jiang, "CEFT: A Cost-Effective, Fault-Tolerant Parallel Virtual File System," *J. Parallel and Distributed Computing,* vol. 66, no. 2, pp. 291-306, 2006.

[6] M.E. Tolentino, J. Turner, and K.W. Cameron, "An Implementation of Page Allocation Shaping for Energy Efficiency," *Proc. Third Workshop High-Performance, Power-Aware Computing (HP-PAC '07),* Apr. 2007.

[7] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini, "DMA-Aware Memory Energy Management for Data Servers," *Proc. 10th Int'l Symp. High-Performance Computer Architecture (HPCA),* 2006.

[8] A.R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power Aware Page Allocation," *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '00),* pp. 105-116, 2000.

[9] 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.

[10] L.A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer," *IBM Systems J.,* vol. 5, no. 2, pp. 78-101, 1966.

[11] E.J. O'Neil, P.E. O'Neil, and G. Weikum, "The Lru-K Page Replacement Algorithm for Database Disk Buffering," *Proc. ACM SIGMOD '93,* pp. 297-306, 1993.

[12] T. Johnson and D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," *Proc. 20th Int'l Conf. Very Large Data Bases (VLDB '94),* pp. 439-450, 1994.

[13] D. Lee, J. Choi, J.-H. Kim, S.H. Noh, S.L. Min, Y. Cho, and C.S. Kim, "On the Existence of a Spectrum of Policies That Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies," *Proc. ACM Sigmetrics '99,* pp. 134-143, 1999.

[14] Y. Zhou, J. Philbin, and K. Li, "The Multi-Queue Replacement Algorithm for Second Level Buffer Caches," *Proc. General Track: Usenix Ann. Technical Conf.,* pp. 91-104, 2001.

[15] S. Jiang and X. Zhang, "LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance," *Proc. ACM Sigmetrics '02,* pp. 31-42, June 2002.

[16] S. Jiang, F. Chen, and X. Zhang, "CLOCK-Pro: An Effective Improvement of the CLOCK Replacement," *Proc. Usenix Ann. Technical Conf.,* Apr. 2005.

[17] N. Megiddo and D.S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache," *Proc. Second Usenix Conf. File and Storage Technologies (FAST '03),* pp. 115-130, Mar. 2003.

[18] S. Bansal and D.S. Modha, *CAR: Clock with Adaptive Replacement,* pp. 187-200, Mar. 2004.

[19] Intel, *Server and Workstation Chipsets,* http://www.intel.com/products/server/chipsets/, 2008.

[20] R. Inc., *Rambus Memory Chips,* http://www.rambus.com, 2008.

[21] M. Lee, E. Seo, J. Lee, and J. Kim, "PABC: Power-Aware Buffer Cache Management for Low Power Consumption," *IEEE Trans. Computers,* vol. 56, no. 4, 2007.

[22] F. Wang, Q. Xin, B. Hong, S.A. Brandt, E.L. Miller, D.D.E. Long, and T.T. McLarty, "File System Workload Analysis for Large Scale Scientific Computing Applications," *Proc. 20th IEEE/11th NASA Goddard Conf. Mass Storage Systems and Technologies (MSST '04),* http://ssrc.cse.ucsc.edu/Papers/wang-mss04.pdf, Apr. 2004.

[23] M. Uysal, A. Acharya, and J. Saltz, "Requirements of I/O Systems for Parallel Machines: An Application-Driven Study," technical report, 1997.

[24] R. Hedges, B. Loewe, T. McLarty, and C. Morrone, "Parallel File System Testing for the Lunatic Fringe: The Care and Feeding of Restless I/O Power Users," *Proc. 22nd IEEE / 13th NASA Goddard Conf. Mass Storage Systems and Technologies (MSST '05),* pp. 3-17, 2005.

[25] *National Center for Biotechnology Information (NCBI),* N. L. of Medicine and N. I. of Health, ftp://ftp.ncbi.nih.gov/, 2005.

[26] A.E. Darling, L. Carey, and W. chun Feng, "The Design, Implementation, and Evaluation of mpiBLAST," *Proc. Cluster World Conf. and Expo,* June 2003.

[27] Y. Zhu, H. Jiang, X. Qin, and D. Swanson, "A Case Study of Parallel I/O for Biological Sequence Analysis on Linux Clusters," *Proc. IEEE Int'l Conf. Cluster Computing (Cluster '03),* pp. 308-315, Dec. 2003.

[28] J.S. Bucy, G.R. Ganger et al., *The Disksim Simulation Environment Version 3.0 Reference Manual,* http://www.pdl.cmu.edu/DiskSim, 2008.

[29] F. Rawson, "Mempower: A Simple Memory Power Analysis Tool Set," technical report, http://www.research.ibm.com/arl/publications/papers, 2004.

[30] I. Hur, "Enhancing Memory Controllers to Improve Dram Power and Performance," PhD dissertation, Univ. of Texas at Austin, http://www.cs.utexas.edu/~lin/papers/ibrahim.pdf, 2006.

[31] A. Purakayastha, C.S. Ellis, D. Kotz, N. Nieuwejaar, and M. Best, "Characterizing Parallel File-Access Patterns on a Large-Scale Multiprocessor," *Proc. Ninth Int'l Parallel Processing Symp. (IPPS '95),* pp. 165-172, 1995.

[32] B.K. Pasquale and G.C. Polyzos, "Dynamic I/O Characterization of I/O Intensive Scientific Applications," *Proc. Conf. Supercomputing (Supercomputing '94),* pp. 660-669, 1994.

[33] W.W. Hsu and A.J. Smith, "The Performance Impact of I/O Optimizations and Disk Improvements," *IBM J. Research and Development,* vol. 48, no. 2, pp. 255-289, 2004.

[34] C. hsing Hsu and W. chun Feng, "A Power-Aware Run-Time System for High-Performance Computing," *Proc. ACM/IEEE Conf. Supercomputing (SC '05),* p. 1, 2005.

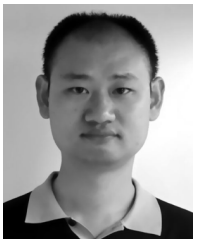
[35] E. Pinheiro, R. Bianchini, E.V. Carrera, and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems," *Proc. Workshop Compilers and Operating Systems for Low Power (COLP '01),* http://research.ac.upc.es/pact01/colp/paper04.pdf, Sept. 2001.

[36] V.W. Freeh and D.K. Lowenthal, "Using Multiple Energy Gears in Mpi Programs on a Power-Scalable Cluster," *Proc. 10th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP '05),* pp. 164-173, 2005.

[37] N. Kappiah, V.W. Freeh, and D.K. Lowenthal, "Just in Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in Mpi Programs," *Proc. ACM/IEEE Conf. Supercomputing (SC '05),* p. 33, 2005.

[38] R. Ge, X. Feng, and K.W. Cameron, "Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters," *Proc. ACM/IEEE Conf. Supercomputing (SC '05),* p. 34, 2005.

[39] B. Lawson and E. Smirni, "Power-Aware Resource Allocation in High-End Systems via Online Simulation," *Proc. 19th Ann. Int'l Conf. Supercomputing (ICS '05),* pp. 229-238, 2005.
[40] K. Coloma, A. Choudhary, A. Ching, W.K. Liao, S.W. Son, M. Kandemir, and L. Ward, "Power and Performance in I/O for Scientific Applications," *Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp. Workshop 10 (IPDPS '05),* p. 224.2, 2005.
[41] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M.J. Irwin, "Scheduler-Based Dram Energy Management," *Proc. 39th Conf. Design Automation (DAC '02),* pp. 697-702, 2002.
[42] H. Huang, P. Pillai, and K.G. Shin, "Design and Implementation of Power-Aware Virtual Memory," *Proc. Usenix Ann. Technical Conf.,* pp. 57-70, citeseer.ist.psu.edu/article/huang03design.html, 2003.
[43] M.E. Tolentino, J. Turner, and K.W. Cameron, "Memory-Miser: A Performance-Constrained Runtime System for Power-Scalable Clusters," *Proc. Fourth Int'l Conf. Computing Frontiers (CF '07),* pp. 237-246, 2007.
[44] B. Diniz, D. Guedes, W. Meira Jr., and R. Bianchini, "Limiting the Power Consumption of Main Memory," *Proc. Int'l Symp. Computer Architecture (ISCA '07),* pp. 290-301, June 2007.
[45] V.D.L. Luz, M. Kandemir, and I. Kolcu, "Automatic Data Migration for Reducing Energy Consumption in Multi-Bank Memory Systems," *Proc. 39th Conf. Design Automation (DAC '02),* pp. 213-218, 2002.
[46] X. Li, Z. Li, Y. Zhou, and S. Adve, "Performance Directed Energy Management for Main Memory and Disks," *Trans. Storage,* vol. 1, no. 3, pp. 346-380, 2005.
[47] L. Cai and Y.-H. Lu, "Joint Power Management of Memory and Disk," *Proc. Conf. Design, Automation and Test in Europe (DATE '05),* pp. 86-91, 2005.
[48] Q. Zhu and Y. Zhou, "Power Aware Storage Cache Management," *IEEE Trans. Computers,* vol. 54, no. 5, pp. 587-602, May 2005.



Jianhui Yue received the MS degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2003. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering, University of Maine. His research interests include energy-aware memory systems, storage system, computer architecture, and operating system. He is a student member of the IEEE and the Usenix Association.

Yifeng Zhu received the BSc degree in electrical engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1998 and the MS and PhD degrees in computer science from the University of Nebraska, Lincoln, in 2002 and 2005, respectively. He is currently an assistant professor in the Department of Electrical and Computer Engineering, University of Maine. His research interests include parallel I/O storage systems, supercomputing, energy-aware memory systems, and wireless sensor networks. He served as the program chair of IEEE NAS '09 and SNAPI '07, the guest editor of a special issue of the *International Journal of High Performance Computing and Networking*, and the program committee of various international conferences, including ICDCS, ICPP, and NAS. He received Best Paper Award at IEEE CLUSTER '07 and several research and education grants from the US National Science Foundation HECURA, ITEST, REU, and MRI. He is a member of the ACM, the IEEE, the IEEE Computer Society, and the Francis Crowe Society.

Zhao Cai received the BSc degree in energy resource and power engineering and the MS degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2000 and 2003, respectively. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering, University of Maine. His research interests include storage system, supercomputing, energy-aware memory systems, database, and mobile computing. He is a student member of the IEEE and the Usenix Association.