electronics

MDPI

Article

An L2 Cache Architecture Supporting Bypassing for Low Energy and High Performance

Jungwoo Park [1,†], Soontae Kim [2] and Jong-Uk Hou [3,*]

[1] Samsung Electronics, Gyeonngi-do 18448, Korea; jw.cs.park@samsung.com
[2] Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, Korea; kims@kaist.ac.kr
[3] School of Software, Hallym University, Chuncheon 24252, Korea
* Correspondence: juhou@hallym.ac.kr
† Current address: 1-1 Samsungjeonja-ro, Hwaseong-si, Gyeonggi-do 18448, Korea.

Abstract: Conventional 2-level cache architecture is not efficient in mobile systems when small programs that do not require the large L2 cache run. Bypassing the L2 cache for those small programs has two benefits. When only a single program runs, bypassing the L2 cache allows to power it down removing its leakage energy consumption. When multiple programs run simultaneously on multiple cores, small programs bypass the L2 cache while large programs use it. This decreases conflicts in the L2 cache among those programs increasing overall performance. From our experiments using cycle-accurate performance and energy simulators, our proposed L2 cache architecture supporting bypassing is shown to be effective in reducing L2 cache energy consumption and increasing overall performance of programs.

Keywords: low-power computing; cache; computer architecture; memory; multicore system

Citation: Park, J.; Kim, S.; Hou, J.-U. An L2 Cache Architecture Supporting Bypassing for Low Energy and High Performance. *Electronics* **2021**, *10*, 1328. https://doi.org/10.3390/electronics10111328

Academic Editors: Giovanni Agosta, Carlo Brandolese and Stefano Cherubin

Received: 6 May 2021
Accepted: 28 May 2021
Published: 1 June 2021

1. Introduction

Multicore processors become popular in mobile systems such as smartphones and smartpads. They have a big potential to be able to improve performance/throughput and to reduce energy consumption. To support multiple programs running simultaneously, they include large L2 caches. The L2 caches are shared when many programs are running simultaneously on different cores while they are monopolized when only a single program is running. Although the L2 caches can improve performance, they may increase energy consumption, especially leakage energy, because they are implemented using a huge number of transistors.

To reduce leakage energy consumption of L2 caches, several schemes have been proposed. One of them is *gated-Vdd* [1], which adds sleep transistors to cut down leakage power but loses data stored in SRAM cells. *Drowsy cache* [2] scheme is proposed to reduce leakage power without losing data stored in SRAM cells. Several architectural techniques have been proposed to reduce leakage energy consumption of L2 caches using *gated-Vdd* and *drowsy cache* schemes. They generally try to detect dead cache lines and put them in a low-leakage mode, which, however, requires applying the *gated-Vdd* and *drowsy cache* schemes at cache line granularity, increasing area overhead very much. Due to this reason, recent application processors employed in smartphones support a low-leakage mode at larger granularity, for example at half size of L2 cache granularity [3].

In this paper, we propose a new L2 cache architecture for reducing the L2 cache leakage energy consumption and for improving performance by bypassing an L2 cache. Bypassing the L2 cache has two advantages. First, the L2 cache leakage energy consumption can be reduced by putting it into a low-leakage mode when a single program is executing and its working set is small, thus bypassing the L2 cache affects performance little. Second, shared use of the L2 cache is not optimal in terms of performance when the sharing incurs

a large number of conflict misses. In some situations, this can be alleviated by letting some programs to bypass the L2 cache, which can decrease conflict misses and unnecessary accesses to the L2 cache.

We implemented an L2 cache architecture supporting bypassing on top of the Zesto [4] simulator and simulated MiBench [5], CSiBE [6] and SPEC2000 [7] benchmarks. Our experimental results show that our proposed architecture is effective in reducing the leakage energy consumption of the L2 cache when a single program runs. Our proposed architecture is also beneficial in some conditions when multiple programs run on different cores simultaneously. Especially, it improves performance when large working set programs use the L2 cache and small working set programs bypass the L2 cache, which reduces conflicts in the L2 cache. The L2 cache leakage energy consumption can be reduced entirely when a single program runs and harmonic mean performance can be improved by up to 41.7% when multiple programs run simultaneously.

This paper is organized as follows. In Section 2, we present related work. Section 3 describes our proposed bypassing architecture in detail and experimental environment and results are discussed in Sections 4 and 5, respectively. Finally, Section 6 concludes the paper.

2. Related Work

There are many studies about reducing leakage energy of cache memory. The most well-known previous technique is gated-V_{dd} by Powell et al. [1]. This technique is used in many cache designs [8–10], but data loss occurs. Drowsy cache [11] is proposed to reduce leakage energy without data loss. Cache blocks not used for long time change their state to drowsy mode. Drowsy mode can reduce leakage energy significantly and can be restored to an accessible state quickly. Thus, its performance degradation is small. It is the most ideal leakage power managing mechanism for caches, but its hardware complexity is too high to be adopted in commercial systems. The third technique is DRG-cache [2], which also reduces leakage energy without data loss. This technique is similar to drowsy cache in terms of data retention mode of unused portion. It also degrades performance, but it effectively reduces leakage energy with relatively low hardware overhead. (Unlike drowsy cache, additional transistor is not needed for each cell). We used this technique when we assume that the data must be retained. The reason why we choose DRG-cache is its lower hardware complexity than drowsy cache. Samsung electronics announced the power management unit of a next-generation application processor that can shut down each core and L2 cache at half-size granularity [3]. We determined the granularity of the techniques to reduce leakage energy to a quarter of the total cache capacity.

Many papers are published using above or similar circuit techniques. The cache decay [9] technique proposed the concept of dead block to turn off unused cache blocks to reduce leakage power. The concept of dead blocks has stimulated many research endeavors about performance and cache energy reduction [12,13]. David H. Albonesi proposed selective cache ways [14], which disables a subset of cache ways in a set associative cache to reduce dynamic and leakage energy consumption. In single core, our proposed scheme is similar to this technique, but we measure entire system energy unlike other similar work. Our program classification does not need special profiling mechanism.

Some stuides for cache energy optimization for multicore systems have been published. Weixun et al. proposed a dynamic cache reconfiguration mechanism for energy optimzation in a real-time multicore system [15]. Xing et al. proposed a two-level (core, L2 cache) utilization control solution for energy efficiency in a multicore real-time system [16]. Hardy et al. used bypassing to tighten the worst case execution time for multicore system with a shared instruction cache [17]. Sato et al. proposed a voting-based working set assessment scheme for dynamic cache resizing mechanisms [10]. These works have overhead of dynamic configuration or profiling. Our work does not need a special profiling method, and it also does not require an initiating procedure and its overhead.

Taeho Kgil et al. proposed new architecture for a tier 1 server called PicoServer [18]. It suggested a flattened memory hierarchy that elides intermediate caches. They have shown that eliding the L2 cache is good for reducing leakage energy and also improving performance in particular systems such as a single-chip server. Prateek et al. compared the bandwidth, latency, and energy filtering of PicoServer with and without an intermediate caches [19].

3. Proposed L2 Cache Bypassing

3.1. Bypassing

Mobile systems such as smartphones adopted large L2 caches to improve performance and to accommodate multiple programs running simultaneously on multiple cores [3]. Generally, data read from the memory are placed onto the shared L2 cache and then onto per-core L1 caches. On each memory access, a per-core L1 cache is probed first; then, the shared L2 cache is accessed on an L1 cache miss. However, this conventional 2-level cache hierarchy management, which is shown in Figure 1a, is not efficient in terms of energy consumption of the L2 cache and overall performance. This is because typical mobile programs are small and their L1 cache misses are low; thus, memory accesses will increase little. Bypassing the L2 cache for the small programs is beneficial in two situations. First, when a small program bypasses the L2 cache, it can be placed into a low-leakage mode without affecting performance, reducing large leakage energy consumption in the L2 cache. Second, when multiple programs run simultaneously, small programs bypass the L2 cache while large programs use the L2 cache, which reduces conflicts in the L2 cache and, consequently, increases overall performance of those programs. Our 2-level cache memory hierarchy supporting L2 cache bypassing is shown conceptually in Figure 1b.

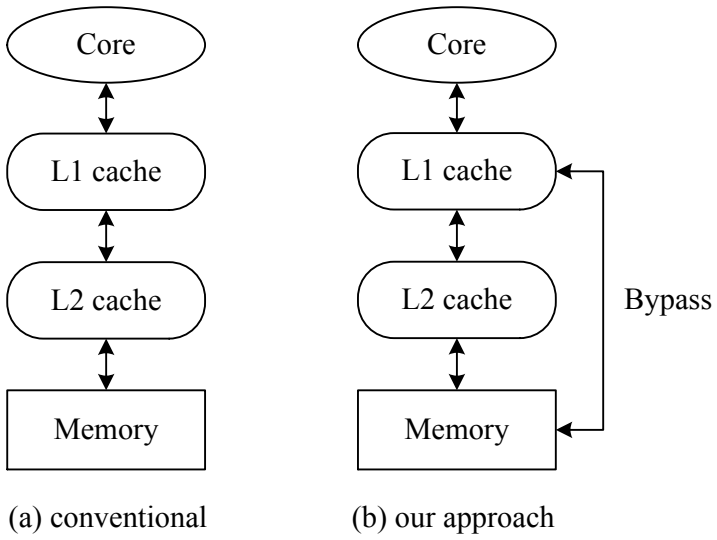


Figure 1. Conventional and our proposed 2-level cache architecture.

3.2. Program Classification

Deciding the optimal L2 cache size/bypassing is based on the history of execution times for L2 cache sizes. In mobile systems, small modules for mobile services or small programs like email are executed repeatedly. This means that a few programs run again and again with high probability [20]. We decided to predict the optimal L2 cache size/bypassing for each program by using the history of past executions instead of online prediction. This approach is simple and does not require additional hardware. Instead, the operating system stores execution time information for each L2 cache configuration. When a program runs first, scan the execution time information of the program and check if execution time is recorded or not. Scan sequence is fully used, a half, a quarter, and bypass. Recording sequence is the same as scan sequence. The algorithm of choosing the optimal L2 cache

capacity follows. If there is no information about execution time, the program runs with full size L2 cache. After its execution, its execution time is recorded into a corresponding slot. When the program runs again, run it with half-size L2 cache and record its execution time. Repeat this step with quarter-size L2 cache and bypassing.

After four total executions, we obtain the execution time information of the four L2 cache configurations. We found some patterns of execution times for these configurations. First, four executions do not show noticeable performance differences. This means that the capacity of L2 cache does not affect performance considerably. This is because the cache footprint of a program is small such that L1 cache can cover the footprint. In this case, L2 cache bypassing is beneficial because we can reduce leakage energy consumption of the L2 cache by turning it off. Programs showing this behavior are classified as a type 1. This type information is recorded into the registry of OS for later use. Registry is a database that stores configurations for each application. Second, L2 cache bypassing requires relatively many cycles over other configurations and there is no outstanding difference among a quarter, a half, and full use of L2 cache. This means that a quarter-size L2 cache is enough to cover cache the program footprint. Programs showing this behavior is classified as type 2. Third, for execution times, they show noticeable differences. This means that the footprint of this program is too large to reduce L2 cache size. In this case, full size of L2 cache should be used for performance. Programs belonging to this case are classified as type 3.

We experimented with various benchmarks and got threshold values to classify their types. If normalized execution times of the three configurations to the baseline of full L2 cache size is between 1 and 1.1, programs are classified as type 1. Programs are classified as type 2 if the normalized execution times of the quarter-size and the half-size configurations are between 1 and 1.1 and the normalized execution time of the bypass configuration is bigger than 1.1. Other programs not belonging to type 1 and type 2 are classified as type 3. When there is an execution request for a program of which type is already classified, run it with the corresponding L2 cache configuration using its type information. Operating system finds this information from registry and sends it to the cache bypass unit. The cache bypass unit configures the L2 cache with the information. We selected power-gating and data retention gated-ground of DRG-cache as energy reduction techniques for unused (bypassed) L2 cache capacity.

3.3. Proposed Cache Architecture

Figure 2 shows our proposed architecture supporting L2 cache bypassing. Its major difference from the conventional cache architecture is in the added circuits between the L1 and L2 caches. They are called bypass units and each core has its own bypass unit. This unit has the information about how much portion of the L2 cache is needed for each program. If a program does not require the L2 cache entirely, it sends memory requests directly to the memory without accessing the L2 cache. On context switches, this information must be stored and restored back as a part of the context of the program. This will not cause noticeable performance and memory overheads because the information is small. One output of this unit goes to the L2 cache on not bypassing the L2 cache and the other output line goes to the memory on bypassing the L2 cache, in which data read from the memory goes to the L1 cache directly. Output direction is directed by the capacity information received from the operating system. In this architecture, we can selectively utiliize the L2 cache fully, partially (a half or a quarter of the L2 cache), or never. Partial capacity use of the L2 cache is only supported when a single program runs in our current architecture.

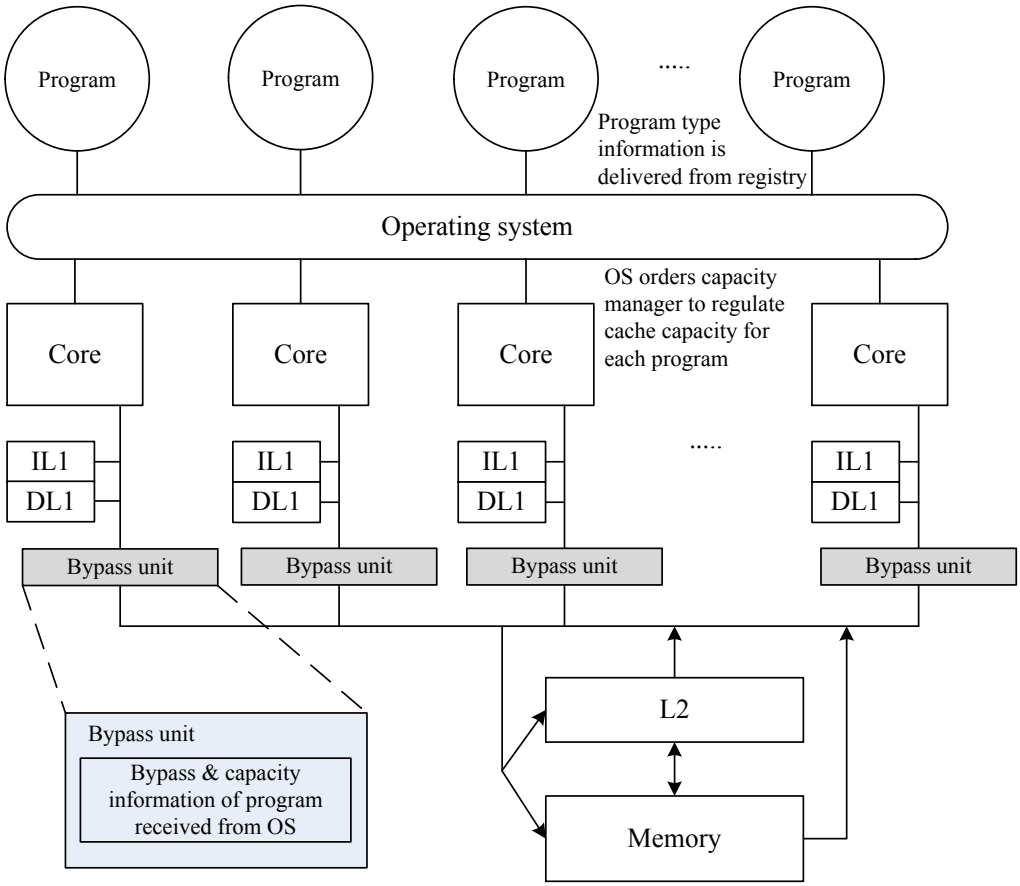


Figure 2. Proposed cache architecture. Bypass units have information about required cache capacity for each program, which is delivered from operating system.

3.4. Multicore Scheduling

Conventional scheduling in multicore systems allocates all programs into the L2 cache, which is, however, not always optimal, especially in mobile systems. We can utilize bypassing for better performance. When there are scheduling requests for type 3 and type 1 (or 2) programs, type 1 program can bypass the L2 cache to increase overall performance because type 1 program will not disturb the cache blocks of type 3 program. Type 3 program is sensitive to L2 cache capacity, so conflicts by the type 1 program can affect its performance largely. We developed a few rules about bypassing the L2 cache based on our extensive experiments. The first one is that type 3 programs do not bypass the L2 cache because these programs require large L2 cache capacity. The second one is that type 1 programs bypass the L2 cache because this type of programs shows little performance degradation due to bypassing and because other programs will have benefits by the bypassing. Finally, type 2 programs do not bypass the L2 cache if the result of multiplication of L1 data cache miss rate by the ratio of data requests per instruction is larger than 0.005. Otherwise, they bypass the L2 cache. Programs whose IPC are low typically show high data requests ratio. However, if their locality is high, high data requests ratio will not cause noticeable performance degradation. Thus, we take into account both data requests ratio per instruction and L1 data cache miss rate. This value represents memory intensiveness of the program. By various experiments, we get value 0.005 to separate type 2 programs. This scheduling algorithm is described as a flowchart in Figure 3.

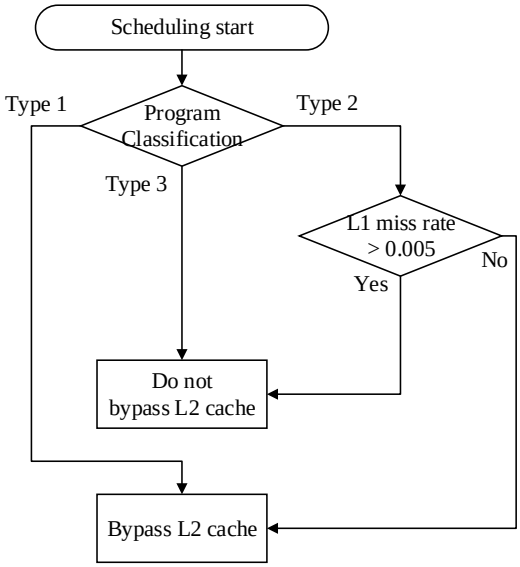


Figure 3. Multi-core scheduling algorithm.

4. Experimental Methodology

We use Zesto [4] simulator for experiments. Table 1 shows the detailed configuration of our experimental environment. We tried to simulate a typical mobile system similar to Cortex A9 [21,22]. The core energy consumption is measured using McPAT [23]. We referenced XIOSim [24] to modify Zesto in order to print out related parameters for McPAT. The energy consumption of cache memory is measured using CACTI 6.5 [25]. We used DRAMSim2 [26] with Zesto to measure the detailed energy and performance of main memory. We use some benchmarks of MiBench suite [5] and CSiBE [6] and SPEC2000 [7]. We run those benchmarks on Zesto using DRAMSim2 with and without cache bypassing. We use four cache configurations; bypass, 256 KB, 512 KB, and 1024 KB. Each cache configuration corresponds to bypassing L2 cache, quarter size, half size, and full size of L2 cache capacity. Cache configuration is changed by adjusting the number of active cache ways. For example, the 512 KB configuration has eight active ways and the other eight ways are disabled. We use two circuit-level cache power control techniques. The first one is power-gating to unused cache area. If a program is type 1, so it bypasses the L2 cache, there is no L2 cache leakage or dynamic power consumption. The other one is data retention technique to unused area using DRG-cache [2]. In DRG-cache with Gsize 1.0, normalized leakage energy is 54% of that of the conventional cache. We calculated leakage energy for data retention mode of unused area. For example, in L2 cache bypassing, the total L2 cache energy is 54% of leakage energy of the conventional cache. Data retention experiment can reflect leakage energy reduction in case data must be retained for later use on context switches. We measure total system energy considering the main memory in single-core experiments. However, in the case of multicore experiments, we could not measure the energy consumption of the main memory because of simulation errors with DRAMSim2. Instead, we include normalized memory accesses to guess the increase of memory energy consumption.

Table 1. Base configuration.

Core	3-Way out of Order, 1 GHz
L1 I-cache	32 KB, 64 bytes per line, 2-way associativity, 2 cycles
L1 D-cache	32 KB, 64 bytes per line, 8-way associativity, 3 cycles
Unified L2 cache	1 MB, 64 bytes per line, 16-way associativity, 12 cycles
Main memory	1 GB x16 DDR3-667

5. Results and Analysis

Table 2 shows detailed energy consumption results for the L2 cache, the main memory, and a core for representative benchmarks of each type. We assume power-gating is used for the unused area. The *adpcm* benchmark is a good example of a type 1 benchmark. Its execution time variations among different L2 cache configurations are almost zero. As L2 cache capacity increases, the L2 cache energy consumption also increases but other components does not show energy increases. This means that the L2 cache capacity variation does not affect memory and core operations. However, the L2 cache is the most significant energy consumer especially for the 1024 KB configuration (almost 57%). Using our classifying scheme, *crc32* and *dijkstra* are classified as type 2 programs. Execution time variations among 256 KB, 512 KB, and 1024 KB are small, but bypassing shows large performance degradation. The *bzip2* benchmark shows large execution time differences among the four configurations. The energy consumption of the memory decreases when L2 cache capacity increases. This is because of the decrease of memory accesses (dynamic energy) and execution time (leakage energy). The energy consumption of the core also decreases because of reduced execution time.

Table 2. Normalized execution time, energy consumption, and data requests per instruction, L1 miss rate and global miss rate of representative benchmarks for each L2 cache configuration with power-gating for unused area (millijoule).

	L2 Usage	Time	Data Request per ins.	L1 Miss Rate	Global Miss Rate	L2 Energy	Memory Energy	Core Energy	Total Energy
adpcm (type 1)	Bypass	1.00	0.40	0.00%	0.00%	0.00	87.58	125.61	213.19
	256 KB	1.00			0.00%	70.65	87.63	125.61	283.90
	512 KB	1.00			0.00%	141.29	87.63	125.61	354.53
	1024 KB	1			0.00%	284.62	87.63	125.61	497.86
dijkstra (type 2)	Bypass	1.55	0.38	1.27%	1.27%	0.00	8.49	3.69	12.18
	256 KB	1.07			0.09%	2.11	5.86	2.54	10.52
	512 KB	1.05			0.09%	4.14	5.43	2.49	12.05
	1024 KB	1			0.09%	7.97	4.63	2.38	14.98
bzip2 (type 3)	Bypass	2.48	0.97	0.79%	0.79%	0.00	211.84	22.83	234.67
	256 KB	1.59			0.41%	21.77	101.81	14.68	138.26
	512 KB	1.22			0.24%	33.47	68.98	11.26	113.71
	1024 KB	1			0.14%	55.34	47.67	9.21	112.24

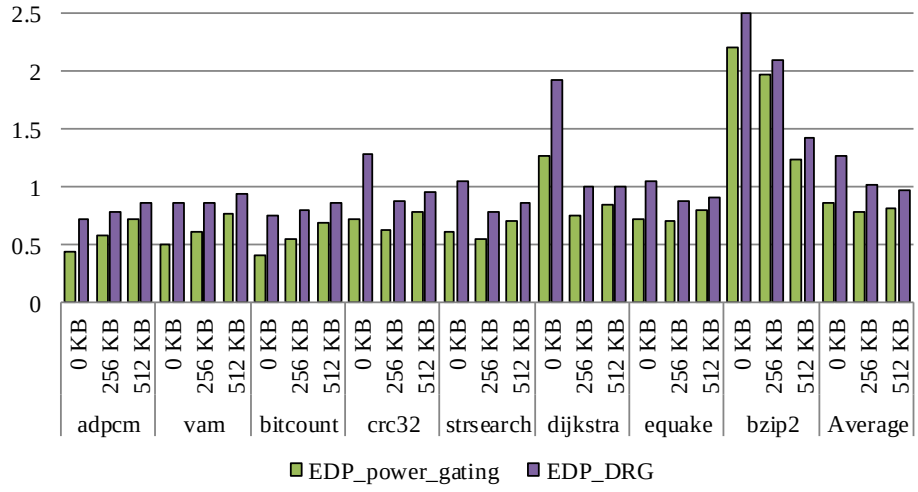
We also measured the execution times and EDPs (Energy-delay product) for each configuration of more benchmarks. Figure 4a shows EDPs with power-gating and EDPs with the DRG-cache scheme supporting data retention. Figure 4b shows normalized energy consumption breakdowns and execution time of the optimal L2 cache configuration for each benchmark. The reason why we do not show execution time results of the DRG-cache

scheme is because it shows similar behavior to power-gating. All values are normalized to the results of the 1024 configuration. The first three benchmarks show similar execution times among the four configurations. EDPs of these benchmarks show gradual increase with increasing L2 cache size. These benchmarks are classified as type 1, where L2 cache bypassing is able to reduce EDP. The total energy reduction with L2 cache bypassing and power-gating for these benchmarks is 57.2% on average. With DRG-cache, the average total energy reduction is 24.9%. The *crc32*, *stringsearch*, *dijkstra*, and *equake* benchmarks show similar execution times among the 256 KB, 512 KB, and 1024 KB configurations but bypassing shows large execution time increase. These benchmarks are classified as type 2, where the 256 KB configuration shows good results. Energy reduction is 35.4% with power-gating and 10.4% with DRG-cache on average. The *bzip2* benchmark shows gradual decrease of execution time with increasing L2 cache capacity. The differences of execution times among the four configurations are large. Reducing L2 cache capacity directly degrades its performance. EDP increases as L2 cache capacity decreases because of increasing execution time although L2 cache leakage power decreases. In terms of execution time and EDP, the 1024 KB configuration is the optimal configuration for this benchmark. Overall, EDPs with power-gating and DRG-cache show similar behavior.

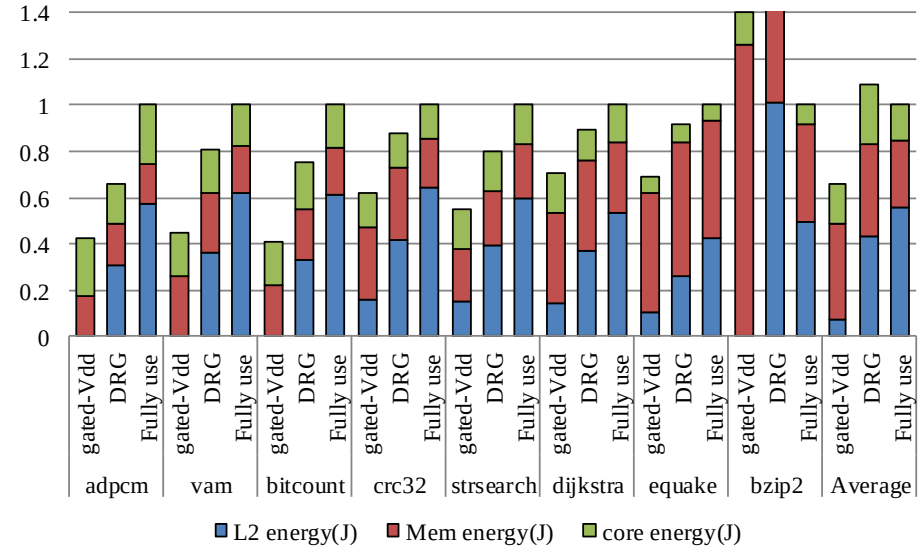
We experimented with multicore system for combinations of benchmarks. Table 3 shows normalized harmonic mean and geometric mean of IPC, total memory accesses, core energy consumption, L2 cache energy consumption, and instruction progress rate of each core. We simulated for same number of cycles for the same benchmark combinations. In the baseline configuration, all benchmarks share the L2 cache. We use instruction progress rate as an important metric to evaluate performance of our multicore scheduling algorithm for the L2 cache. It is a normalized value of the number of committed instructions per core for the same number of cycles. It represents how many instructions are completed for the same time. It effectively shows performance improvement for each program when multiple programs run simultaneously. With our program classification method, the *mcf* and *bzip2* benchmarks are classified as type 3. The *adpcm* and *bitcount* benchmarks are classified as type 1. The *crc32*, *equake*, and *dijkstra* benchmarks are classified as type 2. The L1 data cache miss rates of *crc32*, *equake*, and *dijkstra* are 0.35%, 2.21% and 2.50%, respectively. The ratios of data requests per instruction of *crc32*, *equake*, and *dijkstra* are 0.62, 0.74 and 0.38, respectively. Thus, the multiplied values of the two numbers are 0.0021, 0.0163 and 0.9406 for *crc32*, *equake*, and *dijkstra*, respectively. In our approach, *adpcm*, *bitcount*, and *crc32* bypass the L2 cache while *bzip2*, *mcf*, *equake*, and *dijkstra* utilize the L2 cache, based on our logic in Section 3.4.

The first five experiments show dual-core experiments. In first experiment, *adpcm* and *crc32* are classified as L2 cache bypassing programs. Performance degradation due to bypassing all of them is almost negligible. In this case, the L2 cache can be in a low-power state to reduce leakage energy. When only *bzip2* or *mcf* uses the L2 cache and *bitcount* or *adpcm* or *crc32* bypasses it, overall performance increases. In the fourth and fifth experiments, the instruction progress rates of *bitcount* and *adpcm* show no decrease while the progress of *mcf* shows a 15% increase. The total memory accesses increases by an average of 16%. The main reason for this is increased progress rate of *mcf*. The core energy shows 1% increase and L2 cache energy increases by 2% with 12% increase in the harmonic mean of IPC. There is no main memory energy consumption information but we can guess there is no noticeable memory energy increase considering memory accesses increase with higher instruction progress rates. The sixth and seventh results show triple-core experiments. Bypassing only *adpcm* shows the best performance. Note that *bzip2* and *dijkstra* are determined to utilize the L2 cache. The eighth and nineth combinations show quad-core experiments. If only *equake* uses the L2 cache in the fourth experiments, it shows 31% increase in harmonic mean of IPC. If only *bzip2* and *equake* use the L2 cache in the fifth experiments, harmonic mean of IPC increases by 30%. It is not the best case, but our approach shows performance increase in most cases. It is very difficult to predict runtime interactions among programs. The last combination in Table 3 shows no

performance increase in any combinations. In many high-performance cases, the number of memory accesses is decreased. It is because of the decrease of the global cache miss rate. For example, *equake&bitcount&crc32* configuration of *equake&adpcm&bitcount&equake&crc32* experiment, the number of memory accesses is decreased to 56% from normal configuration. *equake*'s global miss rate is decreased to 0.61% from 0.91%. *bitcount* and *crc32* do not show a noticeable change. *adpcm*'s global miss rate is changed to 0.0027% from 0.0018%, but it does not affect performance critically because it does not access memory frequently. Reduction of memory accesses is good at performance and energy reduction. Leakage power of core, cache, and memory can be reduced since programs can be run earlier. Dynamic energy of memory can be reduced because it will reduce read and write activation power.



(a)



(b)

Figure 4. (**a**) shows normalized EDP of each L2 cache configuration for each benchmark using gated-V_{dd} and DRG-cache. (**b**) shows normalized total energy consumptions of the optimal cache configuration and the baseline, and execution time of the baseline for each benchmark. The optimal cache configuration of *bzip2* is 1024. Execution time is measured when power-gating is applied to the L2 cache.

Table 3. Experimental results of multi-core experiments. The first column shows the benchmarks combinations used in each experiment. The second column shows which benchmarks which use L2 cache. It means other benchmarks bypass L2 cache. The remaining column shows normalized harmonic mean of IPC of total cores, total memory accesses, consumed energy of core and L2 cache, instruction progress of each core. Baseline is the case every job shares L2 cache.

Benchmark Combinations	Bench. L2 Cache Using	HM IPC	MEM acc.	Core en.	L2$ en.	Core 0 Progress	Core 1 Progress	Core 2 Progress	Core 3 Progress
bzip2 & bitcount	bzip2								
bzip2 adpcm	bzip2								
mcf & bitcount	mcf	1.12	1.18	1.01	1.02	1.15	1		
mcf & adpcm	mcf	1.13	1.14	1	1	1.14	1		
bzip2 & adpcm & dijkstra	bzip2	0.98	1	1	1	0.93	1.84	0.46	
	bzip2 & adpcm	1	1	1	1	0.93	1.84	0.48	
	bzip2 & dijkstra	1.28	0.7	1.09	1	1.05	1.91	0.98	
adpcm & bitcount & equake & crc32	equake	1.31	0.56	1.02	1.01	1	0.95	1.74	0.99
	adpcm & equake	1.31	0.56	1.02	1.01	1	0.96	1.75	0.99
	bitcount & equake	1.32	0.56	1.03	1.01	1	0.99	1.76	0.99
	equake & crc32	1.31	0.56	1	1.01	1	0.96	1.75	0.99
	adpcm & equake & crc32	1.31	0.56	1.02	1.01	1	0.94	1.74	0.99
	adpcm & bitcount & equake	1.32	0.56	1.02	1.01	1	0.99	1.76	0.99
	bitcount & equake & crc32	1.41	0.56	1.03	1.01	1	1	1.76	0.99
bzip2 & adpcm & bitcount & equake	bzip2	1.02	1.03	0.99	1	0.96	1	0.93	1.05
	bzip2 & adpcm	1.02	1.03	0.99	1	0.96	1	0.94	1.05
	bzip2 & bitcount	1.03	1.03	1	1	0.96	1	1	1.07
	bzip2 & equake	1.3	0.73	1	1	0.75	1	0.93	2.03
	bzip2 & adpcm & bitcount	1.03	1.03	1	1	0.96	1	1.01	1.07
	bzip2 & adpcm & equake	1.29	0.73	0.99	1	0.75	1	0.9	2.02
	bzip2 & bitcount & equake	1.31	0.72	1.01	1	0.76	1	0.99	2.05
mcf & bitcount & adpcm	mcf	0.99	1.01	0.97	1	0.99	0.98	1	
	mcf & bitcount	1	1	1	1	1	1	1	
	mcf & adpcm	0.99	1.01	0.99	1	0.99	0.98	1	

We only experimented bypassing without considering partial use of L2 cache space in the multicore environment. Our work can be extended to develop online scheduling algorithms considering cache partitioning and bypassing with our program classification information. However, this is complex and left as a future work.

6. Conclusions

We propose a new architecture supporting selective bypassing of the L2 cache. We classify programs into three types to customize L2 cache configuration for each program. This simple history based classification method does not require a special complex hardware. The classification information can be used when a single small program runs to reduce the L2 cache leakage energy consumption by powering down the L2 cache. This information is also helpful, when multiple programs run in different cores, in increasing their overall performance because bypassing the L2 cache for small programs can alleviate L2 cache conflicts. Using a cycle accurate simulator coupled with a detailed power model, we found that the total system energy consumption can reduce up to 42.8% in case of a single program run and that overall performance can improve by up to 41.7% in the case of multiple programs run. This work can be extended to consider online scheduling by including both L2 cache partitioning and bypassing. This is reserved as our future work.

Author Contributions: Conceptualization, S.K.; Methodology: J.P.; Writing—original draft, J.P.; Writing—Review and Editing, J.-U.H.; Funding acquisition, J.-U.H. All authors have read and agreed to the published version of the manuscript.

Acknowledgments: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2020R1C1C1013433). Much of this work was conducted when Jungwoo Park was a Ph.D. student at KAIST.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Powell, M.; Yang, S.-H.; Falsafi, B.; Roy, K.; Vijaykumar, T.N. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In Proceedings of the 2000 International Symposium on Low Power Electronics and Design, Rapallo, Italy, 25–27 July 2000.
2. Agarwal, A.; Li, H.; Roy, K. Drg-Cache: A Data Retention Gated-Ground Cache for Low Power. In Proceedings of the 2002 Design Automation Conference (IEEE Cat. No.02CH37324), New Orleans, LA, USA, 10–14 June 2002.
3. Lee, S.; Lee, J.Y.; Cho, J.; Lee, H.-J.; Cho, D.; Heo, J.; Cho, S.; Shin, Y.; Yun, S.; Kim, E.; et al. A 32 nm High-k Metal Gate Application Processor with GHz Multi-CORE CPU. In Proceedings of the 2012 IEEE International Solid-State Circuits Conference, San Francisco, CA, USA, 19–23 February 2012.
4. Loh, G.H.; Subramaniam, S.; Xie, Y. Zesto: A cycle-level simulator for highly detailed microarchitecture exploration. In Proceedings of the 2009 IEEE International Symposium on Performance Analysis of Systems and Software, Boston, MA, USA, 26–28 April 2009.
5. Guthaus, M.R.; Ringenberg, J.S.; Ernst, D.; Austin, T.M.; Mudge, T.; Brown, R.B. Mibench: A Free, Commercially Representative Embedded Benchmark Suite. In Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538), Austin, TX, USA, 2 December 2001.
6. Csibe: Gcc Code-Size Benchmark Environment. Available online: http://www.inf.u-szeged.hu/csibe/ (accessed on 31 May 2021).
7. Spec cpu2000. Available online: http://www.spec.org/cpu2000/ (accessed on 31 May 2021).
8. Abella, J.; Gonzalez, A.; Vera, X.; O'Boyle, M.F.P. Iatac: A Smart Predictor to Turn-off l2 Cache Lines. *Acm Trans. Archit. Code Optim. (TACO)* **2005**, *2*, 55–77. [CrossRef]
9. Kaxiras, S.; Hu, Z.; Martonosi, M. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In Proceedings of the 28th Annual International Symposium on Computer Architecture, Gothenburg, Sweden, 30 June–4 July 2001.
10. Sato, M.; Egawa, R.; Takizawa, H.; Kobayasi, H. A voting-based working set assessment scheme for dynamic cache resizing mechanisms. In Proceedings of the 2010 IEEE International Conference on Computer Design, Amsterdam, The Netherlands, 3–6 October 2010.
11. Flautner, K.; Kim, N.S.; Martin, S.; Blaauw, D.; Mudge, T. Drowsy Caches: Simple Techniques for Reducing Leakage Power. *ACM Sigarch Comput. Archit. News* **2002**, *30*, 148–157. [CrossRef]
12. Liu, H.; Ferdman, M.; Huh, J.; Burger, D. Cache Bursts: A New Approach for Eliminating Dead Blocks and Increasing Cache Efficiency. In Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture, Como, Italy, 8–12 November 2008.

13. Kim, S.; Vijaykrishnan, N.; Irwin, J.; John, L.K. On Load Latency in Low-Power Caches. In Proceedings of the 2003 International Symposium on Low Power Electronics and Design, Seoul, Korea, 25–27 August 2003.
14. Albonesi, D.H. Selective Cache Ways: On-Demand Cache Resource Allocation. In Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture, Haifa, Israel, 16–18 November 1999.
15. Wang, W.; Mishra, P.; Ranka, S. Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems. In Proceedings of the 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC), San Diego, CA, USA, 5–9 June 2011.
16. Fu, X.; Kabir, K.; Wang, X. Cache-Aware Utilization Control for Energy Efficiency in Multi-Core Real-Time Systems. In Proceedings of the 2011 23rd Euromicro Conference on Real-Time Systems, Porto, Portugal, 5–8 July 2011.
17. Hardy, D.; Piquet, T.; Puaut, I. Using Bypass to Tighten WCET Estimates for Multi-Core Processors with Shared Instruction Caches. In Proceedings of the 2009 30th IEEE Real-Time Systems Symposium, Washington, DC, USA, 1–4 December 2009.
18. Kgil, T.; Saidi, A.; Binkert, N.; Reinhardt, S.; Flautner, K.; Mudge, T. Picoserver: Using 3D Stacking Technology to Build Energy Efficient Servers. *ACM J. Emerg. Technol. Comput. Syst. JETC* **2008**, *4*, 1–34. [CrossRef]
19. Tandon, P.; Chang, J.; Dreslinski, R.G.; Ranganathan, P.; Mudge, T.; Wenisch, T.F. Picoserver Revisited: On the Profitability of Eliminating Intermediate Cache Levels. WDDD, June 2012. Available online: https://web.eecs.umich.edu/~twenisch/papers/wddd12.pdf (accessed on 31 May 2021).
20. Bao, P.; Pierce, J.; Whittaker, S.; Zhai, S. Smart Phone Use by Non-Mobile Business Users. In Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services, Stockholm, Sweden, 30 August—2 September 2011.
21. 7-Zip LZMA Benchmark. Available online: http://www.7-cpu.com/cpu/Cortex-A9.html (accessed on 31 May 2021).
22. The gem5 Simulator System: A Modular Platform for Computer System Architecture Research. Available online: http://www.m5sim.org/Main_Page (accessed on 31 May 2021).
23. Li, S.; Ahn, J.H.; Strong, R.D.; Brockman, J.B.; Tullsen, D.M.; Jouppi, N.P. Mcpat: An Integrated Power, Area, and Timing Modeling Framework for Multi-Core and Manycore Architectures. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA, 12–16 December 2009.
24. Kanev, S.; Wei, G.-Y.; Brooks, D. Xiosim: Power-Performance Modeling of Mobile x86 Cores. In Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, Redondo Beach, CA, USA, 30 July–1 August 2012.
25. Muralimanohar, N. *Cacti 6.0: A Tool to Model Large Caches*; HP Laboratories: Palo Alto, CA, USA, 2009; pp. 1–24 .
26. Rosenfeld, P.; Cooper-Balis, E.; Jacob, B. Dramsim2: A cycle accurate memory system simulator. *Comput. Archit. Lett.* **2011**, *10*, 16–19. [CrossRef]