Increasing Off-Chip Bandwidth in Multi-Core Processors with Switchable Pins

Shaoming Chen, Yue Hu, Ying Zhang, Lu Peng, Jesse Ardonne, Samuel Irving, Ashok Srivastava
Division of Electrical & Computer Engineering, School of Electrical Engineering and Computer Science
Louisiana State University
{schen26, yzhang29, lpeng, jardon2, sirvin1, eesriv}@lsu.edu

Abstract

Off-chip memory bandwidth has been considered as one of the major limiting factors to processor performance, especially for multi-cores and many-cores. Conventional processor design allocates a large portion of off-chip pins to deliver power, leaving a small number of pins for processor signal communication. We observed that the processor requires much less power than that can be supplied during memory intensive stages. This is due to the fact that the frequencies of processor cores waiting for data to be fetched from off-chip memories can be scaled down in order to save power without degrading performance. In this work, motivated by this observation, we propose a dynamic pin switch technique to alleviate the bandwidth limitation issue. The technique is introduced to dynamically exploit the surplus pins for power delivery in the memory intensive phases and uses them to provide extra bandwidth for the program executions, thus significantly boosting the performance.

1. Introduction

As memory-intensive applications such as web servers, database software, and tools for data analysis prevail, the focus of computer architects shifts from Instruction Level Parallelism (ILP) to Memory Level Parallelism (MLP). The term "Memory Wall" was coined to describe the disparity between the rate of core performance improvement and the relatively stagnant rate of off-chip memory bandwidth increase. Additional cores, when integrated on the same die, and supplemental applications serve to widen this gap, since each individual core may generate substantial memory requests that need to be queued and served by the memory subsystem. Obviously, the capability of the off-chip memory system largely determines the per-core or even the overall performance of the entire system. In scenarios where the off-chip memory is insufficiently fast to handle all memory transactions in a timely manner, the system performance is highly likely to be bottlenecked by the slow memory accesses. An intuitive solution to this problem is to increase the off-chip memory bandwidth by enabling more memory channels. Figure 1 illustrates the variation of normalized throughput with the number of memory channels increased from 1 to 4 when 4 *lbm* programs are running on an X86 platform. As can be seen from the figure, enabling more memory channels significantly increases the off-chip bandwidth, which in turn translates to an impressive boost of the system performance. Furthermore, compared to compute-intensive stages, processors consume much less power during memory-intensive phases when cores wait for data to be fetched from main memory.

Motivated by this observation, we propose an innovative technique to mitigate the shortage of off-chip bandwidth during the memory-intensive phases of program executions, in order to enhance the overall performance. Our scheme is built on top of a novel switchable pin design and accurate identifications of memory-intensive phases. Pins can be dynamically altered for power delivery or signal transmission via accessory circuits. These circuits enable pins to deliver quality power or signal with relatively low area overhead. On the other hand, we identify the memory-intensive phases by observing the key performance metrics at runtime. Extra off-chip bandwidth is demanding in phases with high memory intensity. Therefore, by switching the pins and providing additional bandwidth for off-chip memory transactions, the performance of memory-intensive stages can be boosted, thus impressively accelerating the overall execution.

In general, the main contributions of this paper are as follows:

1. We devise a memory controller that can dynamically increase the off-chip bandwidth at the cost of a lower core frequency. Results show a significant increase in throughput for memory-intensive workloads with only a slight hardware overhead.
2. We propose a switchable pin design which can convert a power pin to a signal pin or the other way around. Detailed examinations at both the circuit and architectural level are conducted to validate the feasibility of the proposed design.
3. We examine the performance improvement of our design in various memory configurations. A sensitivity study is conducted to compare the benefit of our design with a different number of channels, buses, banks and ranks.
4. We design Dynamic Switching to alleviate the negative side-effects of pin switching by actively identify-

ing memory-intensive phases and only switching when the condition is satisfied. Without prior knowledge of program characteristics, this policy switches the system to prioritize memory bandwidth or core performance according to the identified phase. Our experiments show that significant performance improvement can be achieved for memory-intensive workloads while maintaining the same performance for compute-intensive workloads as the system without Pin Switching.

2. Related Work

DRAM-Based Memory System: Several papers propose to physically alter the main memory in a DRAM-based memory system to improve the performance and energy efficiency. Zhang et al. propose setting the bus frequency higher than the DRAM module to improve channel bandwidth where the induced bandwidth mismatch is resolved by a synchronization buffer inside the DIMM for data and command [34]. Papers also explore using low power DDR2 (LPDDR2) memory, in place of conventional DDR3, due to its higher energy efficiency [21][33].

To reduce the delay of bank access, thereby increasing memory bandwidth, architects optimize the memory system at the rank and bank level. Zhang et al. subdivides conventional ranks into mini-ranks with a shorter data width. These mini-ranks can be operated individually via a small chip on each DIMM for higher DRAM energy efficiency [35]. Rank sub-setting is also proposed to improve the reliability and performance of a memory system [8].

Inside a DRAM bank, increasing the row buffer hit ratio is key to improving energy efficiency and performance. Kim et al. partition a row buffer into multiple sub-arrays inside a bank to reduce the row buffer miss rate [17]. An asymmetric DRAM bank organization can be used to reduce the bank access latency and improve the system performance [31]. Unlike preceding work, we focus on increasing off-chip bandwidth to boost the performance of the memory system since it is the major bottleneck of memory systems in the multi-core era.

Off-Chip Bandwidth: Rogers et al. have already stressed the significance of off-chip bandwidth [27]. To increase the overall energy efficiency of a memory system, Udipi et al. split a 64 bit data bus into eight 8 bit data buses reducing the queue delay at the expense of data transfer delay [32]. Ipek designs a memory scheduler using principles of reinforcement learning to understand program behaviors and boost performance [15]. Mutlu et al. focus on boosting multi-threaded performance by providing fair DRAM access for each thread in their memory scheduler [24][25]. Our method of adding additional buses to multiply the off-chip bandwidth is orthogonal to the aforementioned methods, which focus on the memory scheduler and bus control.

Tradeoff between core performance and off-chip bandwidth: Architects employ several sophisticated methods to balance core and memory performance [9][11][13]. However, few of them are able to increase the off-chip bandwidth beyond the constraint of static pin allocation.

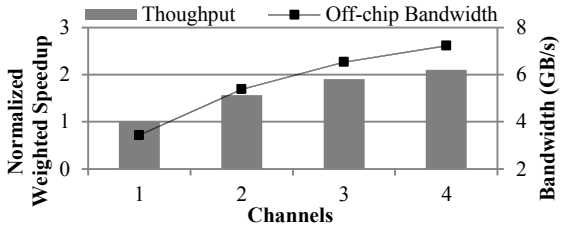


Figure 1. The normalized weighted speedup and off-chip bandwidth of 4 *lbm* co-running on a processor with 1,2,3,4 memory channels

3. Design Overview

Our design aims to boost computer system performance especially for memory-intensive programs. In conventional designs, the performances of these workloads are degraded by a shortage of memory buses which limits off-chip bandwidth. We provide increased memory bandwidth, thereby reducing the average latency of off-chip memory access, at the expense of a lower core frequency. Rather than retaining a fixed number of buses connected to the DRAM (typically one bus per channel), our design dynamically switches buses between signal and power pins (VDD or GND) to reduce the latency for these workloads. This is referred to as *multi-bus mode* henceforth, as opposed to *single-bus mode* similar to conventional processor operation. Switchable pins facilitate changing between these two modes as discussed below. This paper focuses on how to fully exploit the benefits of substituting power pins for I/O pins during memory-intensive programs without interfering with compute-intensive programs.

3.1 Pin Switch

Figure 2 depicts the schematic of two switches and a signal buffer which serve as the basic units for exchanging power pins for signal pins. The signal-to-power switch shown in Figure 2(a) is key to alternate a regular pin between the two modes. As illustrated in this figure, we utilize a dedicated power switch [22] which sits on the power delivery path to minimize the corresponding IR drop and power consumption with its ultra-low switch-on resistance, measuring as low as 1.8mΩ. While in the single-bus mode, the power switch is turned on while two 5 stage tri-state buffers on the signal line are off. Otherwise, the power switch is turned off to block noisy interference from the power line, and the tri-state buffers are turned on in one direction according to whether data is read from the memory or written by the memory controller. To compensate for the parasitic capacitances of the power switch, we place the 5 stage tri-state buffers in signal lines to amplify I/O signals. Between each stage, the buffer size is increased by four times to amplify

Table 1. Pin allocation of an Intel processor i5-4670

V_{DD}	GND	DDR3	Others	Total
153	475	250	272	1150

the signal with small delay. In total, the 5 stage tri-state buffer incurs a 0.9ns delay. On the other hand, the die area of the aforementioned power switch is commensurate to that of 3,000 traditional transistors [22]. The number of signal pins for a DRAM bus could slightly vary depending on different processors (e.g. with or without ECC). We pick up 125 power switches per bus which consists of 64 data pins and 61 address and command pins from the pin allocation of an i5-4670 Intel Processor [4]. The total die area consumes 375,000 (3,000 * 125) traditional transistors. Considering a billion-transistor chip, the area overhead for the 3 buses which will be used in our work is less than 0.12% of the total chip area.

The signal switch shown in Figure 2(b) is employed to guarantee that data in the DRAM can be accessed in two modes. The signal switch uses two pairs of 5 stage tri-state buffers to enable memory devices that can be accessed via two buses. The buffers identical to that in the signal-to-power switch can resist noise from a channel when the other channel is selected. On the other hand, the signal buffers shown in Figure 2(c) also have strong peak-drive current and sink capabilities. They are utilized to amplify the signal in order to offset the effect of the parasitic capacitance.

Processors possess specific pin allocations depending on the package, power consumption, and hardware interface (the number of memory channels). For our experiment, we use the pin allocation of an i5-4670 Intel Processor [4] shown in Table 1. While this processor includes 4 cores and 2 memory channels, 54.6% of the pins are used for power delivery. Out of the 628 power pins, 125 of these can be replaced with switchable pins for a single bus. To maintain the same ratio of VDD to GND pins, we allocate 30 of the 125 switchable pins as VDD pins and the remaining 95 as GND pins. After a sensitivity study in Section 4.1, in our experiment we will allocate at most three additional buses via pin switching because adding more leads to a considerable drop in performance.

3.2 Off-Chip Bus Connection

Designing a memory interface which could take the advantage of the switchable pins to dynamically increase off-chip bandwidth is non-trivial. In this section, we propose an off-chip bus connection and instructions to configure the switchable pins for power delivery or for signal transmission.

The two modes of the off-chip bus connection could be described as the *multi-bus mode* and the *single-bus mode*, as shown in Figure 3. In multi-bus mode, several buses (assuming N) are connected to private DRAM interfaces via the individual buses. On the other hand, single-bus mode can only access DRAM by a single bus. Two signal-to-power switches and a signal switch for each signal wire

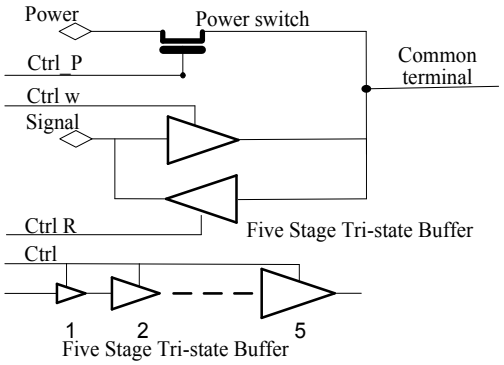


Figure 2 (a). The circuit of a signal-to-power switch

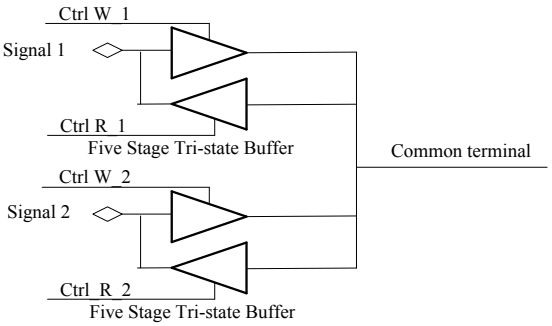


Figure 2(b). The circuit of a signal switch

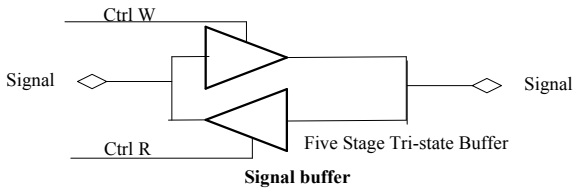


Figure 2(c). The circuit of a signal buffer

of N-1 buses are needed. These signal-to-power switches configure the switchable pins for signal transmission where the signal switches connect the bus to DRAM devices in the multi-bus mode, otherwise the switchable pin is configured for power delivery where the DRAM devices are connected to the shared bus.

In order to implement the mechanism, we control the signal-to-power switch detailed in Figure 2(a) and the signal switch detailed in Figure 2(b) to route signal and power in the two modes. The signal to the DRAM interface could be divided into two groups: command signals and data signals. The command signals running in one direction could be routed via the two switches which only need one direction buffer instead of a pair. On the other hand, the data signals (DQ) are bi-directional and the switches shown in Figure 3 could receive and send signals in both directions.

For the placements of the switches on the printed circuit board (PCB), one signal-to-power switch for each signal

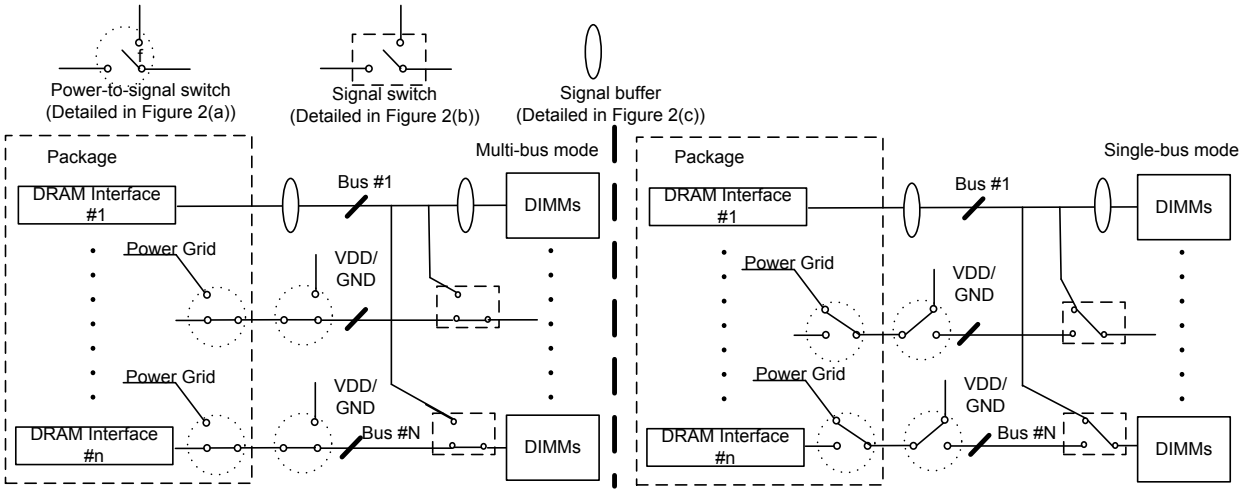


Figure 3. The overview of the hardware design of off-chip bus connection for switching between the Multi-bus mode and the Single-bus mode

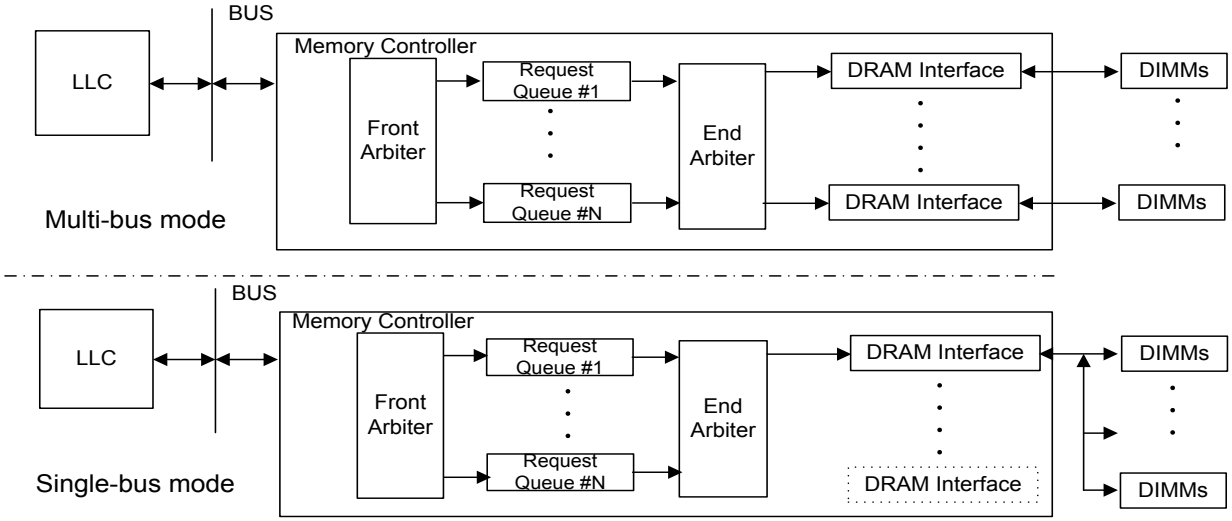


Figure 4. The Overview of the hardware design of memory controller for switching between the Multi-bus mode and the Single-bus mode

line should be placed close to the processor package in order to shorten the signal wire which has to bear high current for power delivery. To avoid signal reflections caused by an impedance mismatch, we keep the width of the signal wires and conduct an experiment to test the feasibility of high current via these signal wires. Based on a specification from the PCB manufacturer [6] and the DDR3 PCB layout guidelines [5], our simulation with COMSOL shows the MTTF of the 6mil signal wire could be more than 2.5×10^5 hours with a 1A current. On the other hand, the signal switch should be placed near the corresponding DRAM device to reduce signal reflections.

3.3 Memory Controller

The data availability of the memory controller is our primary concern. All the available memory buses in the multi-bus mode must be fully utilized to achieve maximum bandwidth while still allowing all the data in single-bus mode to be accessed. Due to the complicated synchronization of memory requests between memory controllers, the switch between the two bus modes is only implemented inside the memory controller. Within a memory controller, a memory interface is designed for each bus to fully exploit the benefit of the multi-bus mode without the interference of traffic from other buses compared to the design of multiple buses sharing a single memory interface.

The memory controller in our design includes dedicated request queues which buffer the incoming requests to the buses shown in Figure 4. Queues individually receive the requests from the front arbiter which employs its address mapping policy when dispatching requests. Once the requests are residing in the queues, they are fetched by the back arbiter. While in multi-bus mode, the requests are fed into their corresponding buses via the corresponding DRAM interfaces. Because memory interfaces can operate

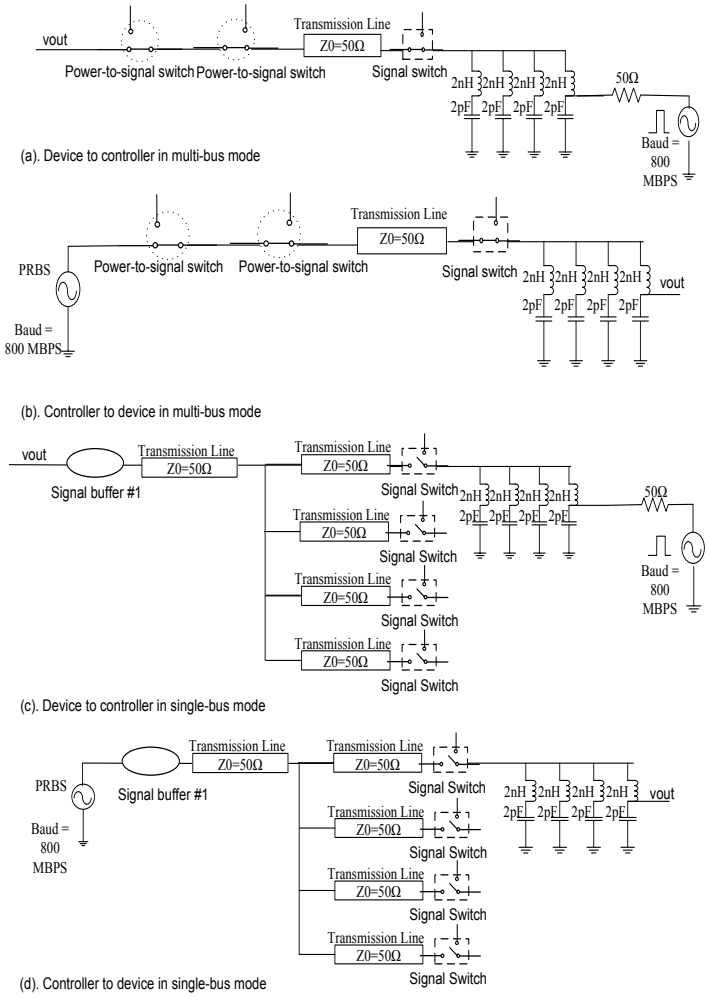


Figure 5. Spice models for signal integrity simulation

independently and in parallel, the memory bandwidth can be amplified by a factor of the number of memory buses. In the single-bus mode, the memory controller works similar to a conventional processor and communicates with the attached DIMMs as appended ranks.

3.4 Area Overhead

The circuit overhead of our design consists of the front arbiter, the end arbiter, and extra DRAM interfaces. As a result of both arbiters, the cost of dispatching requests without buffering them should be negligible. Furthermore, the cost of the additional DRAM interface is inexpensive. The estimated net area of a typical DRAM interface from Opencore [1] is 5,134 μm^2 in 45 nm technology. This estimation is conducted by the Encounter RTL Compiler [2] with the NanGate Open Cell Library [3]. No more than three additional buses in total are used in our experiment thus creating a maximum hardware overhead less than 0.00015 cm^2 which is significantly less than the typical 1 cm^2 die area.

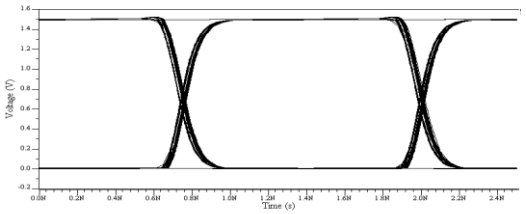


Figure 6(a). DQ Read in multi-bus mode (Device to Controller)

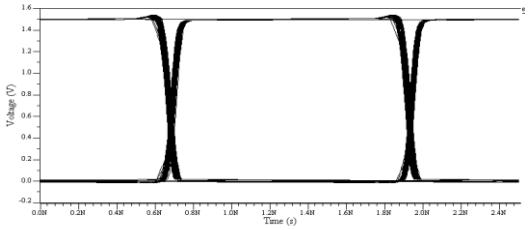


Figure 6(b). DQ Write in multi-bus mode (Controller to Device)

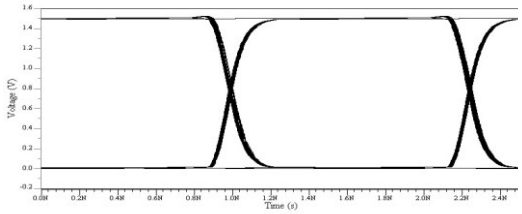


Figure 6(c). DQ Read in single-bus mode (Device to Controller)

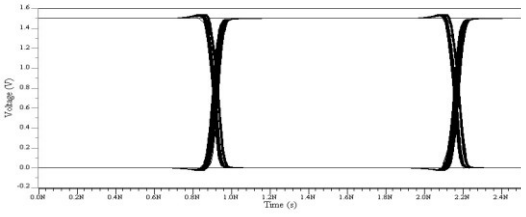


Figure 6(d). DQ write in single bus mode (Controller to Device)

3.5 Address Mapping

Data accesses interleave at the page level via different buses exploiting the benefit of memory-level parallelism while maintaining a high row buffer hit ratio. Interleaving at the block level considerably decreases the row buffer hit ratio resulting in longer off-chip latency per request and extended queue delay. To reduce row-buffer conflicts, we employ XOR banking indexing which could effectively reduce bank conflicts resulting from resource-contention-induced traffic and write-backs. This permutation distributes the blocks stored in the last level cache into different banks as

Table 2. Power network model parameters

Resistance	Value	Inductance	Value
R_{PCB}	0.015 mΩ	L_{PCB}	0.1 nH
$R_{PKG,\,C}$	0.2 mΩ	$L_{PKG,C}$	1 pH
$R_{LOAD,C}$	0.4 mΩ	$L_{LOAD,C}$	1 fH
R_{GRID}	0.01 mΩ	L_{GRID}	0.8 fH
$R_{C4,\,SINGLE}$	40 mΩ	$L_{C4,\,SINGLE}$	72 pH
$R_{SWITCH,ON}$	1.8 mΩ		
Capacitance			
$C_{PKG,C}$	250 µF	$C_{LOAD,C}$	500 nF

Table 3. Processor power and frequency parameters for different number of buses

BUS	1	2	3	4
Current (A)	125	104	80	56
Voltage (V)	1	0.88	0.76	0.64
Power (W)	125	92	61	36
Frequency (GHz)	4	3.2	2.4	1.2

opposed to possibly including tags of physical addresses containing the same bank index.

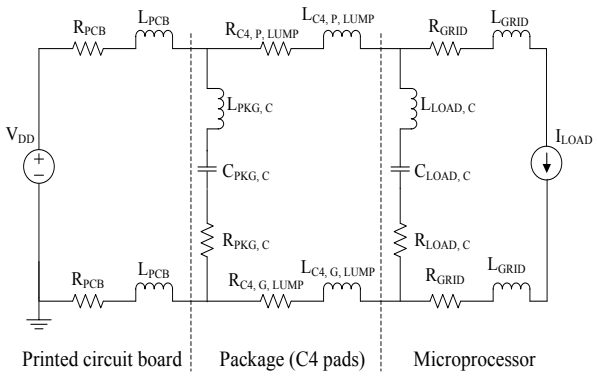
3.6 Signal Integrity

Signal integrity is analyzed to demonstrate feasibility in the single-bus and the multi-bus modes. We simulate SPICE models of our accessory circuit as well as PCB transmission lines, bond wire inductance, and driver capacitance associated with the device package in the AMS packages of Mentor Graphic as shown in Figure 5. The parameters are derived from previous works [21][23]. Signal integrity challenges are alleviated since the DDR3 command signal is unidirectional and its speed is no more than that of the data signals [21]. In this study, we only analyze the effect of our accessory circuit on the data signals which could be viewed as the worst case for all the signals.

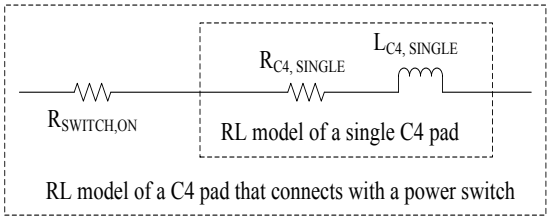
In Figure 6(a-d), the eye patterns of writing data (controller to device) and reading data (device to controller) in the two modes are derived from the corresponding SPICE models in Figure 5(a-d) respectively. They have clear eyes since the signal-to-power switch alleviates the effect of the parasitic capacitance of the power switches. Furthermore, the signal switches as well as signal buffers alleviate the signal reflections caused by discontinuities. Thus, the results indicate our accessory circuit could maintain the signal quality in the two modes.

3.7 Power Delivery Simulation

In this section, we assess the repercussions experienced by the power delivery network (PDN) when the switchable pins are shifted from single-bus mode to multi-bus mode. The PDN is depicted in Figure 7(a). The power delivery path is modeled with RL components (i.e. resistors and inductors) connected in series across the PCB, the package, and the silicon die. Decoupling capacitors are introduced between each individual PDN to control any voltage fluctuations. The on-chip power grids and processor circuits on



(a) Power delivery network



(b) RL model of a C4 pad

Figure 7. RLC power delivery model

the silicon die are modeled separately as RL components with an ideal current source. Figure 7(b) illustrates the RL model of the Controlled Collapse Chip Connection (C4) pads [12] in which the resistance of the on-state power switches is taken into consideration. Table 2 lists the parameter values obtained from prior work [16].

PDN simulations are performed in PSPICE to evaluate the impact of Pin Switching. Due to resistance along the power delivery path, an IR drop exists between the supply voltage and load voltage as current flows through the PDN. We assume a normalized IR drop should be upper-bounded by 5% as prior work dictates [18][19]. This implies that the maximum currents are 125A, 104A, 80A, and 56A for the baseline and then for Pin Switching mechanisms with one, two, and four borrowed buses respectively. In other words, the three Pin Switching diagrams switch 125, 250, and 375 power pins to signal pins providing 16.8%, 36.0%, and 55.2% less current with 19.9%, 39.8% and 59.7% less power pins respectively. The percentage of current decrease is less than that of proportional power pin quantity decrease because the IR drop depends on the resistance in the PCB and power grids.

We assume the processor employs a dynamic voltage and frequency scaling (DVFS) mechanism supporting 4 voltage and frequency operating points. The frequency can be scaled down from 4.0GHz to 1.2GHz. Correspondingly, the voltage will be decreased from 1.0V to 0.64V. According to McPAT [19], the baseline design can work at a frequency of 4.0GHz given the power delivery information. However, the processor frequency must be decreased individually to 3.2GHz, 2.4GHz, and 1.2GHz when the power pins for one, two, and three sets of memory channel pins are

borrowed as I/O pins respectively. The results shown in Table 3 are used in the following evaluation.

3.8 Runtime Switch Conditions

Designing a predictor to choose the most beneficial mode for the next interval is non-trivial for multi-program workloads. Simply switching based on the amount of consumed off-chip bandwidth is not sophisticated enough to improve the overall performance of a system in which only some of the programs that suffer from long off-chip access latency are likely to benefit from multi-bus mode. To identify intervals that will benefit from Pin Switching it is necessary to estimate both the performance change of each program and the overall benefit of switching for the following interval based on the current performance before a switching occurs. We introduce a metric called the switching benefit $B_{ij}(T_c)$ to help identify the most beneficial mode for each 1 millisecond interval, where $B_{ij}(T_c)$ represents the estimated reward for running the interval following time T_c in mode j instead of mode i. Based on the history of the switching benefit, we predict $\tilde{B}_{ij}(T_c)$ as the switching benefit for the following interval using $\tilde{B}_{ij}(T_c) = \sum_{k=1}^{N} B_{ij}(T_c - k * T_{interval})$, where $B_{ij}(T_c - k * T_{interval})$ represents the switching benefits detailed in equation (1) and can be measured from the N intervals ago and N is the length of the history to consider which were carefully chosen to be 2 for our experiment. If the predicted switching benefit is negative, the system will stay in mode i, otherwise, it will switch to mode j.

The switching benefit is calculated using the following equation:

$$B_{ij}(T_c) = \sum_{k=1}^{p}(WS_{j,k}(T_c) - WS_{i,k}(T_c)) \quad (1)$$

where $WS_{i,k}(T_c)$ and $WS_{j,k}(T_c)$ stand for the estimated weighted speedups for program k at time T_c in mode i and mode j respectively, while p represents the number of simultaneously executing programs which is equal to 4 in our experiment. The weighted speedup of each program in mode i during the interval can be estimated based on the information derived from hardware counters and off-line profiling, since the system is running in mode i during the current interval. The weighted speedup is calculated as follows:

$$WS_{i,k}(T_c) = T_{alone,i,k}(T_c)/T_{shared,i,k}(T_c)$$

$$T_{alone,i,k}(T_c) = Committed\ Inst_{alone,k}(T_c)/(average\ IPS_{alone,k})$$

where $T_{alone,i,k}(T_c)$ stands for the execution time of the same instructions running without interfernce from co-runners and $T_{shared,i,k}(T_c)$ denotes the execution time of a fraction of program k running with others during the current interval which is equal to the length of an interval (1 milisecond). Furthermore, $Committed\ Inst_{alone,i,k}(T_c)$ stands for the number of committed instructions during the interval following T_c of program k, directly derived from a hardware counter since it should be identical to the number when program k shares the main memory system with others. Averge IPS obtained from off-line profiling denotes the average number of executed Instructions Per Second (IPS) when program k running alone. These values are used to approximate $T_{alone,i,k}(T_c)$ based on the assumption that the IPS of each program is relatively steady when it runs alone, since an accurate estimation of $T_{alone,i,k}(T_c)$ is challenging [24].

The estimation of the weighted speedup of each program in currently unused mode j is more difficult compared to that in current mode i, since we can only estimate the performance of mode j according to the information collected in mode i. The weighted speedup is calculated as follows:

$$WS_{j,k}(T_c) = T_{alone,j,k}(T_c)/T_{shared,j,k}(T_c)$$

$$T_{shared,j,k}(T_c) = T_{on-core,j,k}(T_c) + T_{off-core,j,k}(T_c)$$

where $T_{alone,j,k}(T_c)$ is identical to $T_{alone,i,k}(T_c)$ and $T_{shared,j,k}(T_c)$ represents the execution time of program k running with others in mode j. It can be divided into two parts based on whether the execution times vary with core frequency: $T_{on-core,j,k}(T_c)$ denotes the portion of the execution time spent inside the core which is inversely proportional to core frequency, while $T_{off-core,j,k}(T_c)$ expresses the portion of execution time incurred by activity outside the core. We estimate $T_{on-core,j,k}(T_c)$ based on the corresponding time $T_{on-core,i,k}(T_c)$ in mode i using:

$$T_{on-core,j,k}(T_c) = T_{on-core,i,k}(T_c) * {freq_{i,k}}/{freq_{j,k}}$$

where $freq_{i,k}$ and $freq_{j,k}$ are the freqencies in mode i and mode j respectively. We estimate $T_{on-core,i,k}(T_c)$ with the same breakdown using

$$T_{on-core,i,k}(T_c) = T_{interval} - T_{off-core,i,k}(T_c)$$

$$T_{off-core,i,k}(T_c) = T_{LLC,i,k}(T_c) + T_{DRAM,i,k}(T_c)$$

where $T_{LLC,i,k}(T_c)$ is the execution time incurred in the shared last level cache (LLC) in mode i, which is estimated using the number of the accesses to LLC, and $T_{DRAM,i,k}(T_c)$ denotes the execution time incurred by activity in the DRAM controller in mode i. $T_{DRAM,i,k}(T_c)$ is the cumulative time spent when there is at least one in-flight read requests in the DRAM controller, since it can avoid the overestimation due to the overlap of multiple in-flight read requests for single thread [26].

On the other hand, $T_{off-core,j,k}(T_c)$ is mainly affected by the number of buses between different modes since the queue delay inside the DRAM controller is typically decreased as more off-chip buses are added. We cacluate the time using:

$$T_{off-core,j,k}(T_c) = T_{off-core,i,k}(T_c) + T_{queue\ delay,j,k}(T_c) - T_{queue\ delay,i,k}(T_c)$$

$$T_{queue\ delay,j,k}(T_c) = T_{queue\ delay,i,k}(T_c) * {N_{request,j,k}(T_c)}/{N_{request,i,k}(T_c)}$$

Table 4. The configuration of the simulated system

Processor	4 X86 OoO cores with issue width 4
L1 I cache	Private 32KB, 8 way, 64B cache line, 2 cycles
L1 D cache	Private 32KB, 8 way, 64B cache line, 2 cycles
L2 Cache	Shared 8MB, 8 way, 64B cache line, 20 cycles
Memory controller	FR-FCFS scheduling, open row policy
Channel	1
Bus per channel	2 /3/4 (additional buses 1/2/3)
Rank per bus	2
Bank per rank	8
Bank	8*8 DDR3-1600 chips Parameters of DDR3-1600 from Micron datasheet[23]

Table 6. The memory statistics of benchmarks

Benchmark	IPC	LLC MPKI	Row buffer hit ratio	Bandwidth(MByte/s)
libquantum	0.30	58.14	96%	4441.57
milc	0.16	41.86	81%	3641.48
leslie3d	0.62	20.72	85%	3311.84
soplex	0.31	31.34	80%	2501.53
lbm	0.36	23.12	87%	2151.90
mcf	0.15	57.54	19%	2138.81
astar	0.25	29.12	51%	1871.53
omnetpp	1.38	0.49	83%	172.09
gromacs	1.34	0.38	82%	129.60
h264	1.13	0.13	32%	38.03
bzip2	1.13	0.12	94%	35.54
hmmer	1.95	0.00	38%	0.28

where $T_{queue\ delay,i,k}(T_c)$ and $T_{queue\ delay,j,k}(T_c)$ denote the execution time incurred inside the queue of the DRAM controller in modes i and j respectively, while $N_{request,i,k}(T_c)$ and $N_{request,j,k}(T_c)$ stand for the average number of waiting requests per incoming read requests which have to wait until they have been completed in modes i and j. $T_{queue\ delay,i,k}(T_c)$ can be estimated by the time when there is at least one read request in the queue of DRAM controller. $T_{queue\ delay,j,k}(T_c)$ can be estimated by sampling the number of waiting requests in different modes

3.9 Switching Overhead

Any runtime overhead incurred by switching comes from the DVFS and IR drop fluctuations caused by the pin switch. The overhead for DVFS is 20µs [20] and the time for the IR drop to re-stabilize is also bounded by 20µs according to our power delivery simulation. Because both of these delays overlap each other, the estimated total overhead is 20µs and is taken into consideration. Therefore, the penalty is 40µs when a phase is incorrectly identified. However, the overall switching overhead is still negligible since the average length of the identified phases shown is much longer than the overhead in our workloads. Since most programs only switch a few times during execution, nearly all the program phase transitions have been identified by the predictor.

Table 5. The selected memory-intensive and compute-intensive workloads

workload				
Memory-intensive programs				
M1	lbm	milc	soplex	libquantum
M2	lbm	milc	leslie3d	libquantum
M3	lbm	milc	soplex	leslie3d
M4	lbm	soplex	libquantum	leslie3d
M5	milc	soplex	libquantum	leslie3d
M6	mcf	mcf	mcf	mcf
M7	mcf	mcf	astar	astar
M8	astar	astar	astar	astar
Mixed programs				
MIX1	lbm	milc	bzip2	bzip2
MIX2	lbm	milc	omnetpp	omnetpp
MIX3	lbm	soplex	omnetpp	omnetpp
MIX4	milc	soplex	omnetpp	omnetpp
MIX5	lbm	milc	omnetpp	bzip2
MIX6	milc	soplex	omnetpp	bzip2
Compute-intensive programs				
C1	bzip2	bzip2	bzip2	bzip2
C2	hmmer	hmmer	hmmer	hmmer
C3	gromacs	bzip2	omnetpp	h264ref
C4	gromacs	bzip2	sjeng	h264ref
C5	gromacs	omnetpp	sjeng	h264ref
C6	bzip2	omnetpp	sjeng	h264ref

4. Experimental Setup

To evaluate the benefit of our design, we simulate the x86 system documented in Table 4 using the Gem5 simulator [10]. We modify the DRAM model integrated in Gem5 to accurately simulate the proposed method. Throughout the experiments, multi-bus mode will utilize all available buses with the corresponding core frequency shown in Table 3. The buses are partially unutilized with a high core frequency between multi-bus and single-bus modes. We employ off-chip DVFS to maintain the same frequency on all 4 cores at any given time.

4.1 Performance and Energy Efficiency Metrics

We use weighted speedup [29] lists as follows to represent the throughput of our system shown in the following equation.

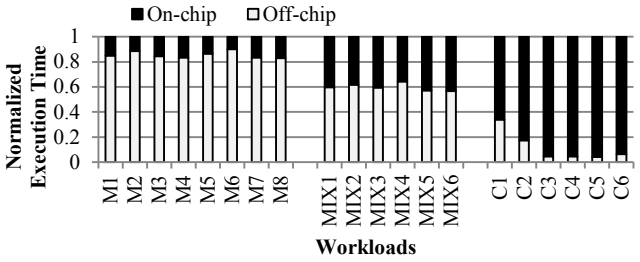


Figure 8. The normalized off-chip latencies and on-chip latencies of workloads against the total execution time.

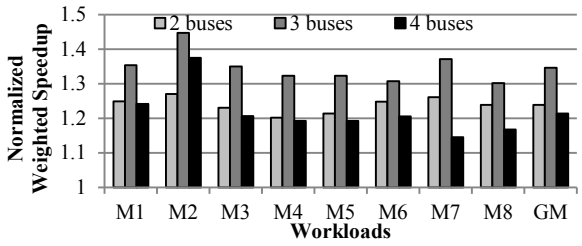


Figure 9. The normalized weighted speedup of memory-intensive workloads with 2, 3, and 4 buses against the each baseline

$$Weighted\ Speedup = \sum_{i=0}^{N-1} \frac{1/T_i^{Shared}}{1/T_i^{Alone}}$$

Here, T_i^{Shared} and T_i^{Alone} denote the execution time of a single program running alone and the execution time running with other programs respectively. Because the IPC is distorted by the frequency change from the employed DVFS, the execution time is used in place of it.

We utilize Energy per Instruction (EPI) for the evaluation of energy efficiency. This metric can be obtained from dividing consumed energy by the number total number of instructions committed.

4.2 Workloads

Various multi-program workloads consisting of SPEC 2006 benchmarks [28] are used for our evaluation. As listed in Table 5, the benchmarks are categorized into two separate groups based on their relative memory intensities: memory-intensive programs and compute-intensive programs. Each workload consists of four programs from one of these groups to represent a memory-intensive workload or compute-intensive workload accordingly. Memory-intensive workloads are used to demonstrate the benefit of multi-bus mode while the compute-intensive workloads demonstrate that there are negligible side-effects. Without losing generality, we also include six MIX workloads consisting of mixed memory- and compute-intensive programs which demonstrate moderate performance benefits.

We select a simulated region of 200 million instructions for each benchmark based on their memory characteristics collected from Pin [7]. The regions are independently executed to gather instructions per cycle (IPC), last-level-cache misses per 1,000 instructions (LLC MPKI), row buffer hit ratio, and the bandwidth displayed in Table 6. The bandwidth and LLC MPKI numerically portray the memory access intensity, making them indicators of our design's potential benefit. Row buffer hit ratio reveals the memory access locality and latency. Programs with low row buffer hit ratios suffer from longer bank access latency due to the row buffer miss penalty. Longer memory accesses increase the queue delay which impedes the following incoming requests in the buffer.

The simulation for a workload does not end until the slowest program finishes its 200 million instructions. Faster programs continue running after committing the first 200 million instructions. Execution time of each program is collected after the program finishes its instructions. The compute-intensive programs of MIX workloads run 600 million instructions, which prolong the execution time comparable to that of memory-intensive benchmarks in the same group.

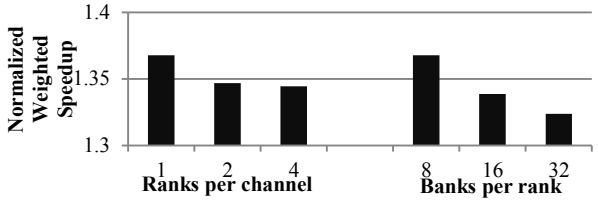


Figure 10. The average normalized weighted speedup of memory workloads in geometric mean with multi-bus mode. Each normalize to the same configuration with single bus mode

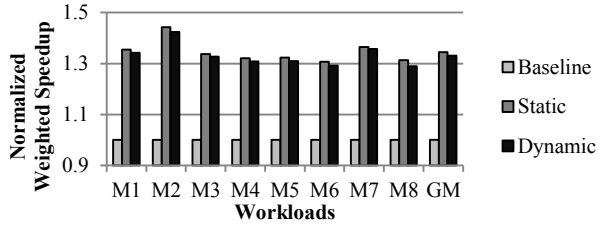


Figure 11. The normalized weighted speedup of memory intensive workloads boosted by Static Switching and Dynamic Switching with 3 buses against the baseline

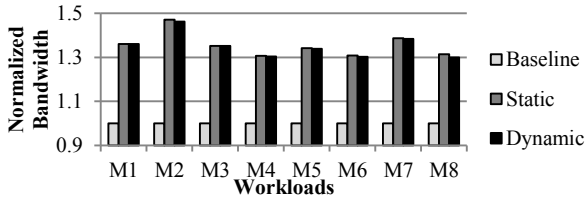


Figure 12. The increased bandwidth due to pin switching. The normalized bandwidth of baseline, static pin switching, and dynamic pin switching.

5. Results

The execution latency of a program is composed of the on-chip and off-chip latency. The percentage of latency in the total execution time reveals which factor tends to be more influential to the overall performance of a workload. In

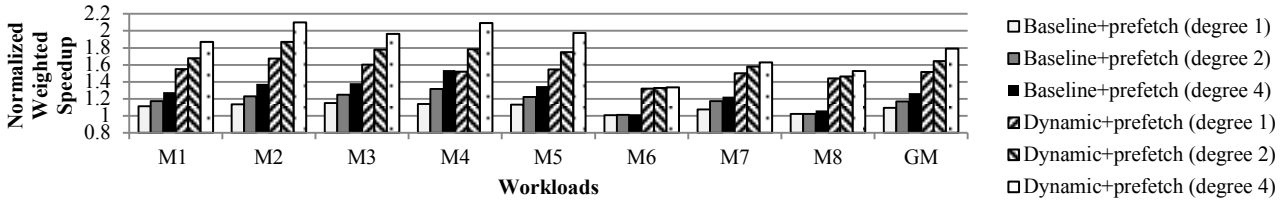


Figure 13. The improved throughput of Dynamic Switching boosted by a stride prefetchers (degree = 1, 2, 4) for memory-Intensive workloads.

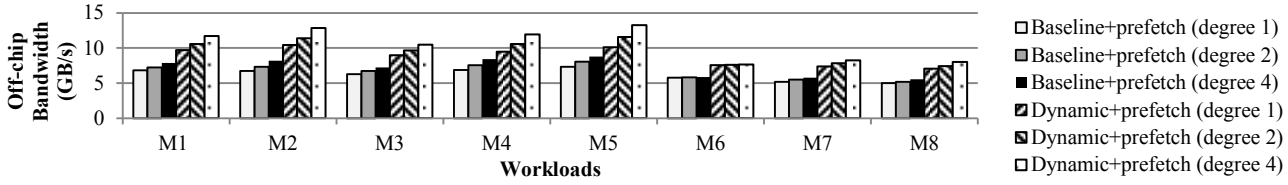


Figure 14. The off-chip bandwidth of Dynamic Switching improved by a stride prefetcher (degree = 1, 2, 4) for memory-Intensive workloads.

Figure 8 we demonstrate the off-chip and on-chip latency for the selected workloads in the three categories. Specifically, more than 80% of the latency of memory-intensive workloads comes from off-chip latency, while more than 60% of the latency of compute-intensive workloads are from on-chip latency. This implies that the memory-intensive and mixed workloads could be sped up by our Pin Switching.

5.1 Memory-Intensive Workloads

Figure 9 shows the performance improvements of memory-intensive workloads enhanced by 2, 3, and 4 buses. The weighted speedup of each case is normalized against its own baseline. The baseline is the simulated system fixed in the single-bus mode with the corresponding number of buses and DRAM devices when the processor runs at 4.0GHz. Remarkably, the improvements experienced with 3 buses consistently surpass 2 and 4 buses in all workloads. These results stem from the balance between core performance and off-chip bandwidth that the 3 buses experience to maximize the throughput of the simulated system. Based on our specific hardware configuration and selected workloads, the multi-bus mode with 3 buses is the optimal choice and therefore referred to as the default configuration for the discussion of Static and Dynamic Switching that will be presented in later sections. Figure 10 illustrates the performance improvement for multi-bus mode tested using various DRAM configurations. The weighted speedup for each configuration is normalized against the same configuration in single-bus mode. As can be seen from the figure, all banks and ranks have weighted speedups greater than 32%. As the number of ranks per channel or the number of banks per rank increases, improvement is slightly diminished due to the resulting lower row buffer hit ratio causing shorter bank access latency.

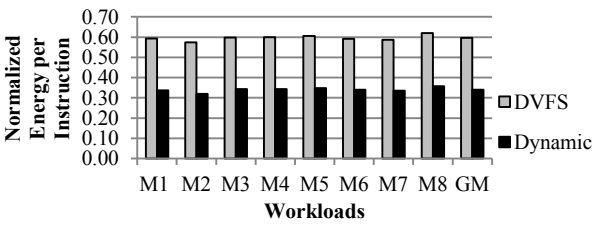


Figure 15. The normalized EPI of Dynamic Switching for memory intensive workloads with 3 buses, and the EPI from DVFS (running on 2.4GHz with the single bus)

Figure 11 presents the benefits of Static Switching and Dynamic Switching with 3 buses versus the baseline of a simulated system that does not use the pin switch mechanism on memory-intensive workloads. Both schemes are able to speed up the execution of all workloads by more than 1.3 times, while an approximately 42% performance improvement is observed for M2. The geometric means of Static Switching and Dynamic Switching are respectively 1.34 and 1.33 due to more than 99% of the running time being identified as typical memory-intensive phases by Dynamic Switching.

The benefit of the multi-bus mode is mainly attributed to the increase of consumed bandwidth as shown in Figure 12. The increase is similar to this of the weighted speedup in Figure 11. For example, M2 and M7 gain 47% and 39% off-chip bandwidth when switching from the single-bus mode to the multi-bus mode for static switching, while their performances are improved by 44% and 36% respectively. This similarity results from the fact that their execution latencies are largely dominated by off-chip latency. On the other hand, Dynamic Switching achieves a slightly smaller increase in bandwidth, which results in its performance being close to that of Static Switching.

The throughput improvement of Dynamic Switching could be strengthened by using prefetchers which can utilize extra bandwidth brought by additional buses in our design. In our experiment, we use a stride prefetcher in the last level cache to demonstrate the benefit. More sophisticated prefetchers could be employed to further improve the

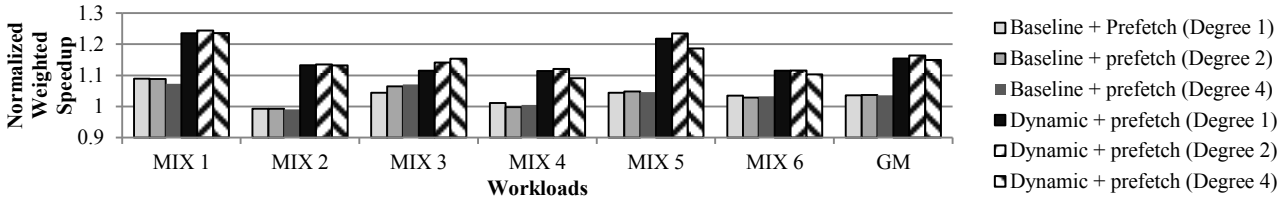


Figure 17. The improved throughput of Dynamic Switching boosted by a stride prefetchers (degree = 1, 2, 4) for mixed workloads.

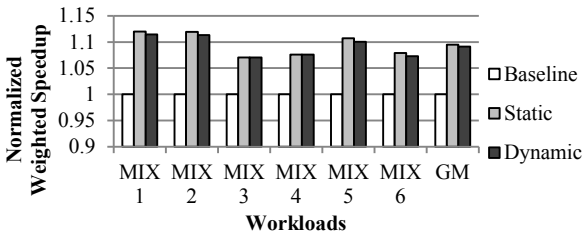


Figure 16. The normalized weighted speedup of mixed workloads boosted by Static Switching and Dynamic Switching.

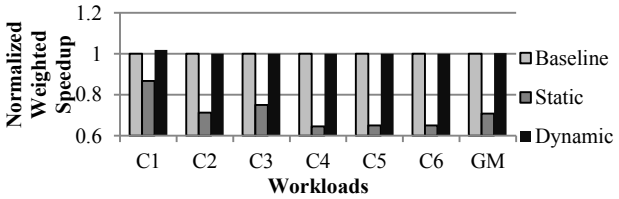


Figure 18. The normalized weighted speedup of Compute-Intensive workloads with Static Switching and Dynamic Switching.

system performance. The stride prefetcher used here has a prefetch degree of 1, 2, or 4, which denotes the number of prefetches issued on every memory reference. As illustrated in Figure 13, the geometric mean of the performance improvements of Dynamic Switching for all memory-intensive workloads with a prefetch degree of 1, 2, and 4 are 1.51, 1.64, and 1.79 respectively, compared with those of the baseline which are 1.10, 1.17, and 1.27. The gap of the improvements between Dynamic Switching and the baseline increases as the prefetch degree increases, which imply an aggressive stride prefetch could benefit more from Dynamic Switching. This observation could be demonstrated in all workloads except M6 which only gains a slight performance improvement from increasing the prefetch degree, since the stride prefetcher has a low coverage on *mcf* [14]. This performance improvement could be verified by the higher consumed off-chip bandwidth of Dynamic Switching shown in Figure 14. It implies that Dynamic Switching could boost the performance of the prefetch by providing more off-chip bandwidth.

The energy efficiency of the system could be also improved by Dynamic Switching. Figure 15 details the energy efficiency improvement of the simulated system. In theory, the energy savings come from two sources: (1) low voltage and frequency scaling; and (2) the execution reduction time stemming from multiple buses brought by pin switching. We quantify the first part by setting the core frequency of the simulated system to 2.4 GHz (relating to the frequency of our multi-bus mode scheme) with the corresponding voltage for single bus. The results depicted as gray bars in Figure 15 demonstrate 40% improvement in the geometric mean of the EPI for all the workloads over the baseline. Note that the overall execution time of this setting is only slightly longer than that of the baseline system because all workloads are memory-intensive. Furthermore, the multi-bus mode offers an average of 66% improvement in the geometric mean of the EPI for all the workloads over the baseline resulting from execution time reduction.

5.2 Mixed Workloads

Figure 16 shows the system performance improvement of mixed compute-intensive and memory-intensive workloads using Pin Switching. The highest benefit is achieved using 2 buses and per-core DVFS [30], which is the configuration used in this experiment after we explored the configuration space for these workloads. The geometric means of the normalized weighted speedup from using Static Switching and Dynamic Switching are 1.10 and 1.09 respectively, implying that Dynamic Switching captures the most benefit of Pin Switching for these mixed workloads. Figure 17 shows the co-improvement of Pin Switching and stride prefetching with varying degrees (1, 2, 4) compared with the improvement of the prefetching alone. The geometric means of the normalized weighted speedup of Dynamic Switching with prefetching degree (1, 2, 4) are 1.15, 1.16, 1,15 respectively, while the means with prefetching alone are all 1.04. The co-optimization for all workloads saturates, or even slightly drops as the degree increases, which implies aggressive prefetching wastes off-chip bandwidth rather than exploiting the benefit of MLP for workloads. This can be confirmed by observing the performance of the baseline using prefetching alone as the degree increases.

5.3 Compute-Intensive Workloads

Figure 18 depicts the Dynamic Switching efficiency of compute-intensive workloads in comparison to Static Switching at the cost of lower core frequency and the baseline. The geometric mean of performance degradation for compute-intensive workloads introduced by the Static Switching scheme is 29%. The worst case results in a 35% slowdown of C5. In contrast, Dynamic Switching retains

the same performance as the baseline during compute-intensive workloads because our metric successfully identifies non-memory-intensive phases when the rewards of the multi-bus mode are limited. Furthermore, Dynamic Switching surpasses the baseline for the C1 workload by identifying compute-intensive and memory-intensive phases. Overall, Dynamic Switching exhibits no performance penalty on compute-intensive workloads, in contrast to Static Switching.

The energy consumption of the Dynamic Switching mechanism is almost the same as the baseline since the processor runs at single-bus mode most of the time for compute-intensive programs. Therefore, we do not illustrate the EPI comparison figure here.

6. Conclusion

The limited off-chip memory bandwidth has been widely acknowledged as a major constraint to prevent us from obtaining commensurate performance benefit from the faster processor cores. This is especially challenging in the current multi-core era due to a high volume of memory requests coming from an increasing number of processor cores. To alleviate the shortage of off-chip bandwidth, we propose an innovative pin switching technique which dynamically allocates pins for power delivery or signal transmission with minimal changes to the circuit. By accurately identifying memory-intensive phases at runtime, the proposed strategy converts a portion of the pins used for power delivery to signal transmission mode, providing additional off-chip bandwidth and improving the overall performance. As shown by the evaluation results, along with other techniques including Dynamic Switching and stride prefetching, our scheme is capable of significantly accelerating the program execution.

References

[1] http://opencores.org

[2] http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx

[3] https://www.si2.org/openeda.si2.org/projects/nangatelib

[4] http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/4th-gen-core-family-desktop-vol-1-datasheet.pdf

[5] http://www.freescale.com/files/32bit/doc/app_note/AN3940.pdf

[6] http://www.protoexpress.com/content/stcapability.jsp

[7] http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

[8] Jung Ho Ahn, Norman P. Jouppi, Christos Kozyrakis, Jacob Leverich, and Robert S. Schreiber. 2009. Future scaling of processor-memory interfaces. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09).

[9] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In MICRO, 2008.

[10] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. SIGARCH Comput. Archit. News 39, 2 (August 2011), 1-7.

[11] M. Chen, X. Wang, and X. Li. Coordinating Processor and Main Memory for Efficient Server Power Control. In ICS, 2011.

[12] K. DeHaven, J. Dietz "Controlled collapse chip connection (C4)-an enabling technology," Electronic Components and Technology Conference, 1994. Proceedings., 44th , vol., no., pp.1,6, 1-4 May 1994.

[13] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. 2012. CoScale: Coordinating CPU and Memory System DVFS in Server Systems. In MICRO '12.

[14] Sorin Iacobovici, Lawrence Spracklen, Sudarshan Kadambi, Yuan Chou, and Santosh G. Abraham. 2004. Effective stream-based and execution-based data prefetching. In Proceedings of ICS '04.

[15] E. Ipek, O. Mutlu, J. Martinez, and R. Caruana. Self Optimizing Memory Controllers: A Reinforcement Learning Approach. In Proceedings of ISCA, 2008.

[16] R. Jakushokas, M. Popovich, A.V. Mezhiba, S. Kose, and E.G. Friedman. Power Distribution Networks with On-Chip Decoupling Capacitors. 2011.

[17] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. 2012. A case for exploiting subarray-level parallelism (SALP) in DRAM. SIGARCH Comput. Archit. News 40, 3 (June 2012).

[18] K. L. Kishore and V.S.V. Prabhakar. VLSI Design, I K International Publishing House, 2009.

[19] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen and N. P. Jouppi. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In MICRO, Dec. 2009.

[20] Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. 2011. Scalable power control for many-core architectures running multi-threaded applications. In Proceedings of ISCA 2011.

[21] Krishna T. Malladi, Benjamin C. Lee, Frank A. Nothaft, Christos Kozyrakis, Karthika Periyathambi, and Mark Horowitz. 2012. Towards energy-proportional datacenter memory with mobile DRAM. In Proceedings of ISCA '12.

[22] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis. Power Management of Datacenter Workloads Using Per-Core Power Gating. In Computer Architecture Letter 2009.

[23] Micron Corp. Micron 2 Gb x 4, x8,x16, DDR3 SDRAM: MT41J512M4, MT41J256M4, and MT41J128M16, 2011.

[24] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In Proceedings of MICRO, 2007.

[25] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems. In Proceedings of ISCA, 2008.

[26] Moinuddin K. Qureshi, Daniel N. Lynch, Onur Mutlu, and Yale N. Patt. 2006. A Case for MLP-Aware Cache Replacement. In ISCA '06.

[27] Brian Rogersy, Anil Krishnaz, Gordon Bellz, Ken Vuz, Xiaowei Jiangy, Yan Solihin. Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling. In Proceedings of ISCA, 2009.

[28] Standard Performance Evaluation Corporation. SPEC CPU 2006.

[29] A. Snavely and D. M. Tullsen. Symbiotic job scheduling for a simultaneous multithreading processor. In ASPLOS-9, 2000.

[30] Jose Tierno, Alexander Rylyakov, Daniel Friedman, Ann Chen, Anthony Ciesla, Timothy Diemoz, George English, David Hui, Keith Jenkins, Paul Muench, Gaurav Rao, George Smith III, Michael Sperling, Kevin Stawiasz. A DPLL-based per core variable frequency clock generator for an eight-core POWER7 x2122 microprocessor. In VLSIC. 2010.

[31] Young Hoon Son, O. Seongil, Yuhwan Ro, Jae W. Lee, and Jung Ho Ahn. 2013. Reducing memory access latency with asymmetric DRAM bank organizations. In Proceedings of ISCA '13.

[32] Aniruddha N. Udipi, Naveen Muralimanohar, Niladrish Chatterjee, Rajeev Balasubramonian, Al Davis, and Norman P. Jouppi. 2010. Rethinking DRAM design and organization for energy-constrained multi-cores. In Proceedings of ISCA '10.

[33] Doe Hyun Yoon, Jichuan Chang, Naveen Muralimanohar, and Parthasarathy Ranganathan. 2012. BOOM: enabling mobile memory based low-power server DIMMs. SIGARCH Comput. Archit. News 40, 3 (June 2012).

[34] Hongzhong Zheng, Jiang Lin, Zhao Zhang, and Zhichun Zhu. 2009. Decoupled DIMM: building high-bandwidth memory system using low-speed DRAM devices. In Proceedings of ISCA '09.

[35] Hongzhong Zheng, Jiang Lin, Zhao Zhang, Eugene Gorbatov, Howard David, and Zhichun Zhu. 2008. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In MICRO 41