Dynamic Row Activation Mechanism for Multi-Core Systems

Tareq A. Alawneh
Department of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
t.alawneh@herts.ac.uk

Raimund Kirner
Department of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
r.kirner@herts.ac.uk

Catherine Menon
Department of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
c.menon@herts.ac.uk

ABSTRACT

The power that stems from modern *DRAM* devices represents a significant portion of the overall system power in modern computing systems. In multi-core systems, the competing cores share the same memory banks. The memory contention between these cores may lead to activate a large *DRAM* row only to access a small portion of data. This row over-fetching problem wastes a significant *DRAM* activation power with a slight performance gain.

In this paper, we propose a dynamic row activation mechanism, in which the optimal size of *DRAM* rows is detected at run-time based on monitoring the behavioural changes of the memory requests in accessing sub-rows. The proposed mechanism aims at providing significant memory power savings, reducing the average memory access latency, and maintaining the full *DRAM* bandwidth. Our experimental results using four-core multi-programming workloads revealed that the proposed mechanism in this study can achieve both significant memory power reduction and average *DRAM* memory access latency improvement with negligible area overhead.

CCS CONCEPTS

• **Hardware** → **Dynamic memory**.

KEYWORDS

DRAM, Main Memory, Energy-Efficiency, Over-Fetching

ACM Reference Format:
Tareq A. Alawneh, Raimund Kirner, and Catherine Menon. 2021. Dynamic Row Activation Mechanism for Multi-Core Systems. In *CF '21: Computing Frontiers Conference, May 11–13, 2021, Catania, Sicily, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Dynamic Random Access Memory (*DRAM*) is the primary memory technology used in modern computing systems. With the increasing appetite for more *DRAM* capacity and bandwidth as a natural consequence of integrating more processing cores on a chip, the power and performance of the main memory have become a growing concern, especially when running memory-intensive applications on these cores. Several studies have revealed that the main memory power has become a significant portion of the system's total power, exceeding 50% in some computer systems [15].

In multi-core systems, the processing cores share the same banks of the main memory [11]. A memory contention, also referred to as a memory interference, arises when two or more cores attempt to simultaneously access the shared banks. When these accesses are mapped to different *DRAM* rows in the same bank, a row-buffer conflict occurs. In this case, the entire content of row is written back to *DRAM* arrays (bank precharge), a new row is transferred into the corresponding bank's row-buffer (row activation), and finally the column access (read or write operation) is performed. Activating a large *DRAM* row size (1*KB* or 2*KB*) to read or write only a small portion of data (usually 64*B*) is known as the row over-fetching problem [10, 15]. Row activation (*ACT*) and bank precharge (*PRE*) are expensive memory commands in terms of power and delay [1, 3, 23]. The portion of *DRAM* power consumed during these commands is known as activation power. This power category contributes a high fraction of total *DRAM* power. When *ACT* and *PRE* commands are performed on a fewer number of bitlines (i.e. small *DRAM* row size), significant activation power savings are obtained [23]. On the other hand, the increased interference between memory accesses issued from different cores can potentially degrade the available row-buffer locality as more cores are integrated on a chip [9]. This, in turn, increases the number of row-buffer conflicts. Due to the row over-fetching problem, this has a serious negative impact on memory power and performance [12].

Several Fine-Grained Activation (*FGA*) mechanisms [4, 7, 9, 13, 15, 18, 20, 22, 23] have been previously suggested to reduce the negative impact of the *DRAM* row over-fetching problem on the *DRAM* power consumption and performance. However, some of these mechanisms [4, 20] lead to a significant bandwidth loss and, in turn, performance degradation. Additionally, these methods do not take into account the dynamic nature of applications executing concurrently by making use of *DRAM* rows with various sizes determined at run-time. As a result, they do not achieve optimum performance and power efficiency.

In this paper, we address this deficiency by proposing a dynamic row (wordline) activation mechanism, in which the optimal size of *DRAM* row being accessed during a row activation is detected at run-time. Our proposed dynamic wordline activation technique gives full consideration to the access behaviour of the memory requests across the *DRAM* sub-rows (wordline segments) at run-time. In this way, it delivers significant activation power and performance improvements.

The contributions of this work can be summarized as follows:

- The mechanism proposed in this paper ensures less power wastage due to the *DRAM* row over-fetching problem while

CF '21, May 11–13, 2021, Catania, Sicily, Italy
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

simultaneously providing significant performance improvements. This is accomplished by employing a dynamic *DRAM* row activation mechanism, in which the optimal number of the wordline segments being accessed during a row activation is detected at run-time based on monitoring the access behaviour of the memory requests on recently targeted sub-rows.

- Once the optimal size for *DRAM* row access is determined further relaxation of the Four Activation Window (T_{FAW}) timing constraint can be achieved, which in turn leads to a significant performance improvement. Unlike prior studies, our proposed mechanism facilitates the efficient identification of the optimal number of wordline segments to be activated at run-time based on the behavioural changes of applications executing concurrently. Additional performance benefits are also obtained through the adoption of an early precharge technique that works in unison with our proposed mechanism. This early precharge is useful in converting the potential row-buffer conflicts into row-buffer empties (i.e. mapping to a specific bank with no active row in the corresponding row-buffer).
- The dynamic row activation proposed in this study sustains the full bandwidth that is available in conventional *DRAM* memory systems.

2 DRAM BACKGROUND

Modern *DRAM* devices are characterized by a 3*D* structure; columns, rows, and banks. The memory bank contains thousands of memory rows (pages). Each memory row consists of multiple memory columns. The memory column is built from a group of memory locations (storage cells). Each *DRAM* storage cell consists of one transistor and one capacitor. Modern memory banks are physically split into sub-arrays. Each sub-array is a horizontal row of smaller components known as *MATs*. Each *MAT* is a 2*D* cell array, typically of dimensions 512×512 *DRAM* cells. Each horizontal row of cells within *MAT* is connected by a local wordline. All *MATs* within a sub-array have the same set of global wordlines [1, 3]. Subsequently, these *MATs* operate in lockstep when accessing a specific *DRAM* row within a sub-array [1, 3]. Every signal driven by a global wordline must be strengthened before being transmitted to the local wordline [1, 3]. Therefore, each horizontal row of *DRAM* cells inside *MAT* is linked to a wordline driver [1, 3]. Similarly, each vertical column of *DRAM* cells within *MAT* shares a wire called the local bitline [1, 3]. A single sense amplifier is attached to each local bitline to detect and amplify the small charge held by at most one *DRAM* cell [1, 3]. The sense amplifiers of the row of consecutive *MATs* together form the local row-buffer of a *DRAM* sub-array. The entire content of the local row-buffer of at most one sub-array is transferred to the global row-buffer through shared wires, known as global bitlines [1, 3].

3 RELATED WORK

Several recent fine-grained row activation approaches [4, 7, 9, 13, 15, 18, 20, 22, 23] have been suggested to address the *DRAM* row over-fetching problem. In these approaches, target memory rows are partially transferred into the corresponding row-buffer during activation for a new row. Thus, a significant activation power consumption reduction is achieved. However, the design improvements suggested in some mechanisms [4, 20] do not sustain the full bandwidth of conventional *DRAM* memory systems. This leads to significant performance degradation (a detailed description of this problem is provided in the next section). Several studies [10, 23] also propose *Half–DRAM* row activation architectures, in which half of the target row is accessed during a row activation. In another work [15], the authors suggested using the partial row activation policy only in the case of write memory operations.

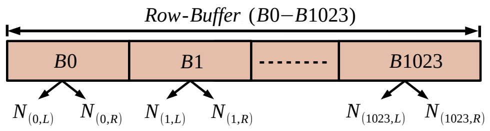
None of these techniques introduces a full solution to the *DRAM* over-fetching problem in multi-core systems. Rather, these are static policies that do not take into account the dynamic nature of applications executing concurrently, or the consequent potential for using *DRAM* rows with various sizes determined at run-time. As a result, the desired overall system performance improvements and power savings may not be sufficiently realised. In contrast, the mechanism introduced in this paper is a dynamic wordline activation mechanism taking into account the full consideration of the access behaviour of the memory requests across the *DRAM* sub-rows at run-time.

4 DATA MAPPING WITHIN DRAM MEMORY SUB-ARRAYS

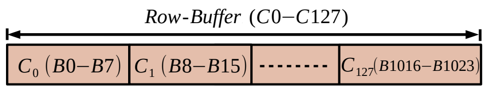
DRAM activation power is proportional to the number of accessed wordline segments (sub-rows) during a row activation operation. Besides the substantial power reduction, the activation of fewer wordline segments of the target row is an attractive mechanism to relax *DRAM* power timing constraints, which in turn improves *DRAM* performance. However, if any design suggested for a fine-grained row activation does not consider the way data is accommodated and fetched in a *DRAM* device, this may lead to a significant degradation in the overall system performance. In this section, we expand on this to provide the necessary background on how data is fetched and accommodated in modern *DRAM* memory systems.

DRAM memory bank is characterized by its two-dimensional structure; rows and columns. Figure 1a depicts an example of a 1*KB* memory row; 1024 bytes numbered from 0 to 1023 ($B0-B1023$). Each byte consists of two nibbles; right and left. For instance, $N_{(0,L)}$ refers to the left or the most significant 4-bit of the byte number 0 ($B0$), while $N_{(0,R)}$ denotes the right or the least significant 4-bit of the byte number 0 ($B0$). The smallest unit that can be addressable in the modern *DRAM*-based main memory systems is called a column. If the column access granularity is eight bytes, then the total number of columns in 1*KB* memory page is 128 numbered from 0 to 127 ($C0-C127$) (see Figure 1b). For example, the first column ($C0$) in a row includes the contiguous bytes from 0 to 7 ($B0$-$B7$), the bytes from $B8$ to $B15$ represent the second column ($C1$), and the last column ($C127$) contains the bytes from $B1016$ to $B1023$.

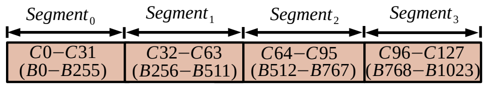
The transferred data between the external memory and Last Level Cache (*LLC*) is in the form of contiguous bytes called a cache line (block). In modern *DRAM* devices, the cache line is transferred through multiple data bursts. As mentioned before, every *DRAM* sub-array consists of a group of *MATs*. Each data burst is therefore mapped into different *MATs* within a specific sub-array. These *MATs*



(a) A 1*KB* memory row in terms of bytes.



(b) A 1*KB* memory row in terms of columns.



(c) A 1*KB* memory row divided into 4 equal-sized segments.

Figure 1: A 1*KB* memory row (page).

operate in lockstep to fetch or write a data of one burst; each *MAT* contributes with a 4-bit. As shown in Figure 2a, we assume a Double Data Rate (*DDR*) memory system with 16 *MATs* for each sub-array and a 64-bit memory data bus width. We also assume that the dimension of each *MAT* is 512 × 512 cells, hence the row-buffer size is 1*KB* (= $\frac{16 \times 512}{8}$). A read memory operation is issued by the memory controller to fetch the column number 0 ($C0$) that residing in an activated row. This leads to fetching a 64*B* cache line through 8 data burst (beat) transactions. In every burst transaction, an 8-byte word is transferred. Each *MAT* contributes with a nibble (4-bit). For instance, the first data burst represents the bytes numbered from 0 to 7 ($B0$–$B7$). The left nibble of $B0$ ($N_{(0,L)}$) and the right nibble of $B0$ ($N_{(0,R)}$) are transmitted from *MATs* 0 and 1 respectively, whereas the left nibble of $B7$ ($N_{(7,L)}$) and the right nibble of $B7$ ($N_{(7,R)}$) are delivered from *MATs* number 15 and 16 respectively. As a general rule, every cache line in the *DRAM* row is divided into words. The word (usually 8*B*) is partitioned into multiple bytes, the right and left nibbles of each byte are linearly accommodated to two adjacent *MATs* within a specific sub-array.

In *DDR* memory systems, the data is transmitted at the rising and falling edge of memory clock cycles. Thus, the eight 8*B* data bursts take 4 memory cycles to transfer them via the memory data bus; transferring two bursts in one cycle. However, some previous fine-grained row activation studies [4, 20] suggested enabling only a single *MAT* or a group of adjacent *MATs* to reduce the *DRAM* row activation granularity. Unfortunately, such techniques reduce the number of bits that can be read or written simultaneously from a sub-array (i.e. wasting/dropping bandwidth). This, in turn, increases the number of bursts needed to move the entire cache line to/from a sub-array. Figure 2b illustrates how the entire 64*B* cache line is transferred through the data memory bus when employing the Fine-Grained Activation (*FGA*) mechanism that activates only one-eighth of the target wordline by enabling 2 out of 16 *MATs*. This mechanism will be termed as *FGA_*(1/8*Bank*) in this work. Instead of delivering 8*B* per burst (beat), each group of two *MATs* conveys 8 bits (1*B*) at a time to/from the target sub-array. Therefore, 7/8 of bandwidth is wasted. The entire cache line is accommodated within two contiguous *MATs*, hence sixty-four 1*B* bursts are required to transfer the entire 64*B* cache line. This takes 32 memory cycles to transfer them through the *DDR* memory data bus, which leads to a significant degradation in the overall system performance. Increasing the Burst Length (*BL*) to deliver 64*B* data chunk during memory access means that the time spent by data on the bus is also increased. This, in turn, leads to wasting a significant amount of power consumption.

Figure 1c represents 1*KB DRAM* row divided into four equal-sized segments. The data of the first segment (*Segment*$_0$), for instance, consists of columns (data bursts) numbered from 0 to 31 ($C0$–$C31$). As discussed above, the bytes of these columns/bursts logically appear as one contiguous block, however these bytes are physically distributed throughout different *MATs* in a sub-array.

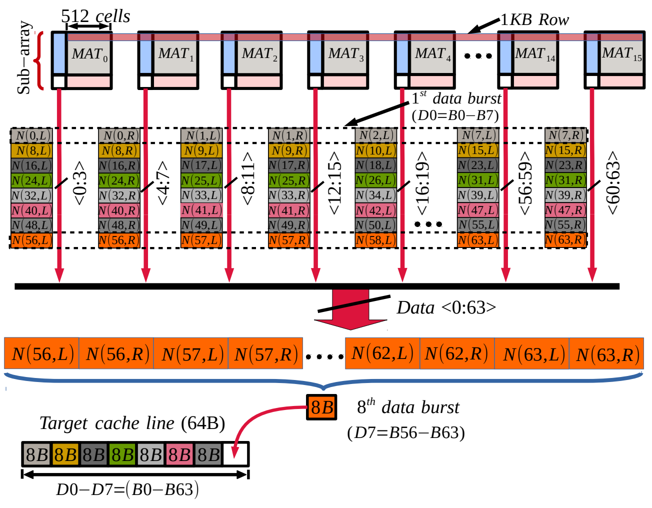
This deep understanding of the ways in which wordline segment data can be accommodated (mapped) within *MATs* of a specific sub-array is foundational to this work. This allows us to precisely determine the required *MAT* structure modifications that allow the dynamic row activation mechanism in providing a significant reduction in activation power without sacrificing the performance and full memory bandwidth.

5 PROPOSED ARCHITECTURE

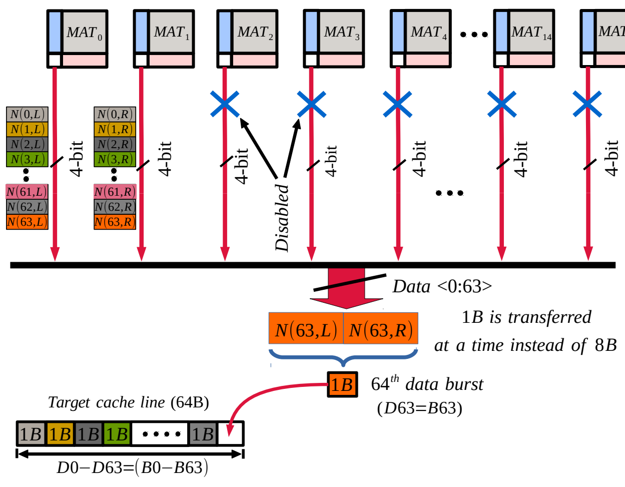
In this paper, we propose a new dynamic row activation mechanism, in which the optimal number of bitlines (*DRAM* row size) driven by a single wordline is detected at run-time. This mechanism aims to achieve a significant reduction in *DRAM* activation power consumption without sacrificing *DRAM* memory bandwidth or performance. This represents a novel contribution in the form of permitting a full consideration of run-time detection of the optimal number of activated wordline segments based on monitoring the access behaviour of the memory requests on recently targeted sub-rows. Specifically, we maintain history information at run-time to calculate the value of a new metric, termed as Permutation Rate between Wordline Segments (*PRWS*). This metric is useful in selecting the optimal number of segments of the wordline being activated during memory access. It represents the rate of permutation of the memory requests at run-time between different segments of the recently activated wordline. With increasing the number of successive memory requests that are mapped to different wordline segments, the value of *PRWS* increases.

The history information that is used in calculating *PRWS* can be maintained at three different granularities; per-page, per-bank, and per-rank (global). Maintaining the history information at the *DRAM* page-level provides the highest accuracy in terms of identifying the optimal number of wordline segments being accessed during a row activation. Contrarily, coarse-grained (rank) granularity is the lowest accuracy. However, tracking data at a fine-scale would incur a significant hardware overhead. In order to strike balance between prediction accuracy and hardware overhead, we therefore maintain the history information in this study at the *DRAM* bank-level. We also suggest further optimizations that work in unison with the dynamic row activation mechanism proposed in this work. These optimizations, which will be discussed later in Section 5.1, aim at achieving further improvements in prediction accuracy and performance.

The value of *PRWS* at the bank-level is calculated by using Equation 1. All parameters used in this equation are described in Table 1.



(a) Conventional *DRAM memory system.*


4-bit
Disabled
Data <0:63>
1B is transferred at a time instead of 8B
64th data burst (D63=B63)
Target cache line (64B)
D0−D63=(B0−B63)

(b) *DRAM* memory system that employs $FGA\_(1/8Bank)$ mechanism.

Figure 2: (a) An example shows the data fetching and accommodating in conventional *DRAM* memory systems. (b) An example shows the bandwidth drop that arises when fetching data in the *DRAM* memory system that employs *FGA_(*1/8*Bank)* mechanism. *Dn* and *Bm* denote data burst number *n* and byte number *m* respectively.

Table 1: Brief description of the parameters used in calculating *PRWS_bank*.

Parameter	*Description*
$PRWS\_bank_{(i,r)}$	The rate of permutation between wordline segments of the memory requests that go to the bank *i* in rank *r*.
$TNMR\_bank_{(i,r)}$	The total number of memory requests that access the bank *i* within rank *r*.
$TNP\_bank_{(i,r)}$	The total number of transitions between wordline segments of the memory requests targeting the bank *i* within rank *r*.

If *PRWS* value for a specific bank is less than or equal to a certain threshold, then Partial Wordline Activation (*PWA*) is the preferred policy to follow. The exact number of how many wordline segments being activated is proportional to the *PRWS* value. On the other hand, whenever the value of *PRWS* is above the given threshold, Full Wordline Activation (*FWA*) is the optimal choice to enforce. This is a good indication that subsequent memory requests are likely to access all wordline segments of the targeted *DRAM* rows. Using the *PWA* policy, in this case, would fail to take advantage of the available row-buffer locality, which in turn would lead to *DRAM* performance degradation. Assuming that *DRAM* rows are divided equally into *8* segments, Table 2 shows the exact number of wordline segments that must be accessed when activating a new row based on the value of *PRWS*, which is calculated at the bank-level. It can be seen that the higher the ratio, the more wordline segments of the target *DRAM* row are activated. If the *PRWS* ratio is less than or equal to 75% (given threshold), then the *PWA* policy is applied. More specifically, one out of the eight wordline segments (i.e. 1/8*th*) is activated, if the *PRWS* ratio is less than or equal to 25%. When the *PRWS* ratio is between 26%–50% a quarter of the target row (i.e. 2/8*th*) is accessed. When it is greater than 50% and less than 76%, half of the original row (4 wordlines segments) is accessed. The entire content of the target row is activated, only if the *PRWS* ratio is greater than 75%.

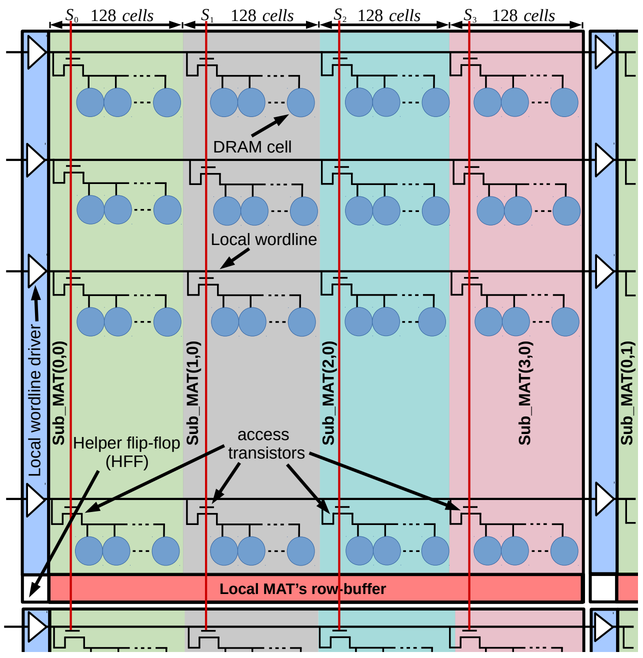
$$PRWS\_bank_{(i,r)} = \frac{TNP\_bank_{(i,r)}}{TNMR\_bank_{(i,r)} - 1} \tag{1}$$

Table 2: Number of wordline segments being accessed during row activation based on the value of *PRWS*. Every *DRAM* wordline/row is equally divided into *8* segments.

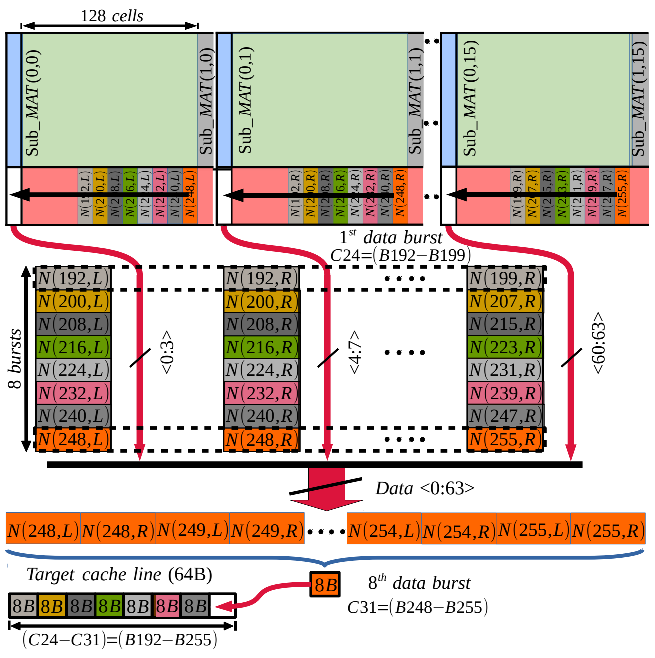
PRWS value		*Number of activated*
From	*To*	wordline segments (ratio)
0%	25%	1 (*one-eighth*)
26%	50%	2 (*quarter*)
51%	75%	4 (*half*)
76%	100%	8 (*full*)

5.1 Detailed Description of the Proposed Architecture

Figure 3a shows the changes to the *MAT* structure required in our proposed mechanism when each memory row is divided into four equally-sized sub-rows (wordline segments). Physically, each *MAT* is split into four *sub-MATs*. Every *sub-MAT* contains a linear portion of the contiguous data nibbles for one of the sub-rows. In each *sub-MAT*, the contiguous *DRAM* cells in each local wordline are grouped together and their gates are linked to a single access transistor. The source of each access transistor is attached to the respective local wordline. The gates of all access transistors attached to *sub-MATs* containing a portion of data belonging to the same sub-row are controlled by one bit selection signal. Each of these bits enables the *sub-MATs* that belong to the same sub-row to work in lockstep when activating it in the respective portion of the row-buffer. Therefore,



(a) Enhanced *MAT* structure.



(b) Fetching data from sub-array in our proposed mechanism.

Figure 3: (a) Overview of the enhanced *MAT* structure in this study when dividing each wordline/row into 4 segments. (b) An example demonstrates how data fetching from *sub-MATs* of a *DRAM* sub-array is implemented in our proposed work. *Sub_MAT(n,m)* indicates the *MAT* area that contains a portion of data belonging to sub-row number *n* in the *MAT* number *m* of a specific sub-array.

we need 4-bit selection signals (S_0–S_3) to control the activation of the data of the four sub-rows separately. In such manner, we sustain the full bandwidth that is available in the conventional *DRAM* memory by maintaining the contribution of each *MAT* by a 4-bit to deliver 8*B* chunk data per burst as shown in Figure 3b. Furthermore, this *MAT* modification enables the access to rows of different sizes at run-time, ranging from a quarter of the original row size to a full row size. The optimal row size is determined at run-time based on adaptation to the dynamic nature of the memory access characteristics of the applications executing concurrently.

As mentioned before, the predicated number of wordline segments to be activated by the proposed Dynamic Row Activation (*DRA*) mechanism at the bank-level might not be always the optimal choice. Maintaining the history information at per-page level provides the highest accuracy, but with the consequent disadvantage of a significant hardware overhead. In this study, several optimizations have been added working in line with the proposed architecture to increase the accuracy of predicting the optimal wordline segments being accessed at run-time.

The identification made by our suggested *DRA* mechanism for selecting and deselecting wordline segments during a row activation is partially based on the information gathered by monitoring the access behaviour of the memory requests on *DRAM* sub-rows. However, it also takes into account other wordline segments that will need to be activated for memory requests pending in the transaction queue at the time of issuing an *ACT* command to the *DRAM* devices. To this end, we employ an 8-bit register for each *DRAM* bank; 1-bit assigned for each wordline segment. When a new memory request is mapped to an empty row-buffer or inactive segment of the recently accessed wordline/row, this memory request is given a high priority by the memory scheduler, i.e. it is placed at the head of the transaction queue. Meanwhile, the assigned bit corresponding to the target row segment in that bank's register is set to 1. In each *ACT* issued to the *DRAM* device, the register assigned for the target bank is looked up. If there are bits set to 1 corresponding to wordline/row segments which will be targeted by memory requests which are pending at this moment in the transaction queue and different from that proposed by the *DRA*, then all bits assigned for these row segments in the S_0–S_7 signals are set to 1. Then they are sent to the *DRAM* device through the address inputs in the clock cycle that follows the issuing of the *ACT* command. In this way, we avoid adding new *DRAM* pins dedicated only to the wordline segment selection signals (S_0–S_7). On the other hand, one additional cycle is relatively insignificant compared to either the hundreds of cycles incurred in each memory access or those that could be avoided as a result of the relaxation of power timing constraints [20].

To prevent over-frequent issuing of multiple *ACT* commands for the same partially activated *DRAM* row, we utilise a technique that can quickly respond to the requirement to activate the full content of the recently partially accessed row. Each bank is associated with a 2-bit saturating counter to keep track of the state of activation mode for the row that has already been activated in the bank's row-buffer. If the bank is precharged (empty row-buffer), the row activation mode state is *IDLE* (see Figure 4). When a memory access goes to an unopened row, an *ACT* command is issued to the *DRAM* device to open this row fully or partially. The state becomes Fully Activated (*FA*) when the decision is made to activate the entire content of the target row (*Open_FA*). Similarly, the state changes to Strong Partially Activated (*S_PA*) if the decision is made to partially activate the target row (*Open_PA*). The state of the target row remains in the *S_PA* state as long as the consecutive memory accesses hit in the portion of the row that has been recently activated. Otherwise, the state changes to Weak Partially Activated (*W_PA*) if any memory request goes to an unopened segment of the target row (*Unopened_sub_row*). Similarly, it remains in the state of *W_PA* as long as the mapped memory requests hit in the portion of the row that has been recently activated. Otherwise, if the state is *W_PA* and a memory request mapped to an unopened segment is received, then the decision is made to issue an activation for the remaining unopened sub-rows, and accordingly the state changes to *FA*.

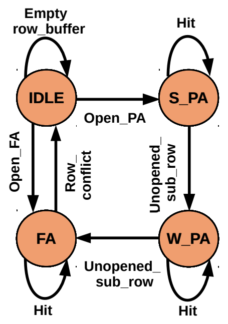


Figure 4: The finite state of activation mode for the recently accessed row.

On the other hand, several static and dynamic row-buffer management policies, also known as page policies, have been studied previously [8, 11]. The optimal page policy plays an important role in achieving a higher *DRAM* performance when efficiently exploiting the embedded localities of the applications that run concurrently. Due to the dynamic nature of the applications in accessing *DRAM* pages at run-time, we suggest a dynamic page policy that works in line with the proposed approach in this study. In the conventional execution time of the running applications, we utilise the Open Page (*OP*) policy, in which the recently activated row is left open. However, early precharge can be performed proactively when a row-buffer conflict is expected to occur in the next memory request. In case of activation to a new row, the dynamic mechanism introduced in this work can predict that the optimal row size to be activated is one-eighth of the original row size. This activated row size might be not re-accessed for a specific period of time (T_{RC}) [11]. In anticipating that the next memory request going to the same bank will likely target a row different from the recently activated one (i.e. row-buffer conflict), an early precharge for this row is proactively issued. This early precharge is useful in converting the potential row-buffer conflicts into row-buffer empties. The row-buffer empty incurs a shorter delay than the row-buffer conflict. Because the time needed to precharge the bank for the next memory request is saved. Therefore, further performance improvements can be obtained through the use of this early precharge technique in our proposed mechanism.

6 EVALUATION METHODOLOGY AND RESULTS

In this section, we present an evaluation and analysis of the results obtained whilst running the evaluated multi-programming workloads on our proposed approach.

6.1 DRAM Power and Energy Consumption Estimation

The power consumption of modern *DRAM* devices is categorized into three main components, namely activation power, read and write power, and background power [16, 23]. The activation power represents the power that is used during the row activation and bank precharge operations, whereas the read and write power is consumed by the read and write operations, the output driver and *RD/WR* termination. The background power includes refresh power and static power. More specifically, the background power comes from the power that is consumed during active standby (*ACT_Stdby*), precharged standby (*PRE_Stdby*), active power-down, (*ACT_PD*), precharged power-down (*PRE_PD*), and refresh (*REF*). That is, the background power represents the power that is consumed all the time with or without *DRAM* access activities.

Table 3: Key *DRAM* currents, voltage, and power parameters that are used in this study.

Description	*Symbol*	*Value*
One bank Activate-to-Precharge	*IDD*0	73*mA*
Burst read operating current	*IDD*4*R*	252*mA*
Burst write operating current	*IDD*4*W*	190*mA*
Precharged standby current	*IDD*2*N*	35*mA*
Active standby current	*IDD*3*N*	49*mA*
Active power-down current	*IDD*3*P*	41*mA*
Precharged power-down current	*IDD*2*P*	37*mA*
Refresh current	*IDD*5*B*	242*mA*
Self refresh current	*IDD*6	20*mA*
Supply voltage	V_{DD}	1.5*V*
Output driver power	P_{io}	5.3*mW*
ODT power	P_{odt}	13.2*mW*

In this study, the *DRAM* power consumption is estimated based on Micron's *DRAM* power model [16]. The drawn *DRAM* currents, supply voltage, output driver power, and On-Die Termination (*ODT*) power of the selected baseline system are listed in Table 3. These values are from the Micron Technology *MT*41*K*512*M*8*DA*-107 datasheet [17]. *IDD*3*N* is the consumption of standby current when at least one memory bank is active (i.e. during *ACT_Stdby*), whereas the standby current drawn when all memory banks are precharged (i.e. during *PRE_Stdby*) is termed as *IDD*2*N*. On the other hand, the consumption of current associated with the row activation (*ACT*) and bank precharge (*PRE*) commands during a row cycle (T_{RC}) is termed as *IDD*0. The current consumed only for *ACT* and *PRE* commands without the standby currents (i.e. *IDD*3*N* and *IDD*2*N*) is termed the pure activation current (I_{ACT}) and is obtained by subtracting the standby currents from *IDD*0 as shown in Equation 2. Then, the pure activation power consumption (P_{ACT}) that stems from performing *ACT* and *PRE* commands is calculated

Table 4: The values of *IDD*0, I_{ACT}, P_{ACT}, and E_{ACT} for various row activation granularities. These values are obtained as discussed in Section 6.1.

% of activated wordline/row	$1/8th$	$2/8th$	$3/8th$	$4/8th$	$5/8th$	$6/8th$	$7/8th$	*Full*
*IDD*0 Value (mA)	52	55	58	61	64	67	70	73
I_{ACT} (mA)	7.07	10.07	13.06	16.06	19.13	22.06	25.06	28.07
P_{ACT} (mW)	10.6	15.1	19.6	24.1	28.7	33.1	37.6	42.1
E_{ACT} (pJ)	507.7	723.3	938.9	1154.5	1370.1	1585.7	1801.3	2016.9

as defined in Equation 3. Energy consumption in any state is calculated by multiplying the power drawn during the state by the time spent in that state. Table 4 shows the value of *IDD*0 at $1/8th$, $2/8th$, $3/8th$, $4/8th$, $5/8th$, $6/8th$, $7/8th$, and *full* row activation. It also shows the values of pure activation current (I_{ACT}), pure activation power (P_{ACT}), and pure activation energy (E_{ACT}) when *DRAM* rows are activated with sizes ranging from $1/8th$ to full row size.

$$I_{\mathrm{ACT}} = IDD0 - \left[\frac{IDD3N\times T_{\mathrm{RAS}}+IDD2N\times(T_{\mathrm{RC}}-T_{\mathrm{RAS}})}{T_{\mathrm{RC}}}\right] \qquad (2)$$

$$P_{\mathrm{ACT}} = I_{\mathrm{ACT}} \times V_{\mathrm{DD}} \qquad (3)$$

6.2 Experimental Setup

To examine the performance and energy efficiency of the proposed mechanism, we developed *VHDL* models for traditional and enhanced *DRAM* memory systems. The enhanced *DRAM* memory system in this study splits the row into *8* wordline segments. For comparison, a *VHDL* model was also developed for a memory system employing the *Half–DRAM* row activation architecture. The *DRAM* power and energy consumption in this study is estimated based on what mentioned before in Section 6.1.

Table 5: Main parameters for the evaluated baseline system.

Component	*Configuration*
Multi-core system size	4 cores
CPU Frequency	3.2*GHz*
*L*1 Cache	32*KB*, 4-way, 64*B* line
*L*2 Cache	4*MB*, 8-way, 64*B* line
Main memory	4*GB*, 933*MHz*, *DDR*3-1866 13-13-13, 14.9*GB/s*, 64-entry transaction queue, single channel, 1 rank/channel, 8 banks/rank, burst of 8, 2*KB* memory page size, *OP* policy, page interleaving address mapping scheme, *FR-FCFS* scheduler

The key *DRAM* timing constraints of the selected baseline system are excerpted from the Micron Technology $MT41K512M8DA$-107 datasheet [17]. The values of the main parameters are shown in Table 5. Experiments were conducted by running *RTL* simulations using Vivado 2019.2 [21] for our design, the half fine-grained row activation architecture, and the baseline. We used workloads from *MediaBench* [14] and *PARSEC* [5] benchmark suites to build four-core multi-programming workloads. The workloads selected from the *PARSEC* [5] suite have different application domains. Table 6 lists the four-core workload combinations that are used to verify the performance of the proposed mechanism in this study. These workload sets are simulated using *GEM5* [6] and their memory traces are captured to replay them during our simulations (similar to [2]).

In this work, more than four rows of different sizes can be activated within a rolling window time (T_{FAW}), but without exceeding the peak power that is permissible to be drawn for four full *DRAM* row activation within T_{FAW}. In such manner, we meet T_{FAW} timing constraint in our proposed mechanism.

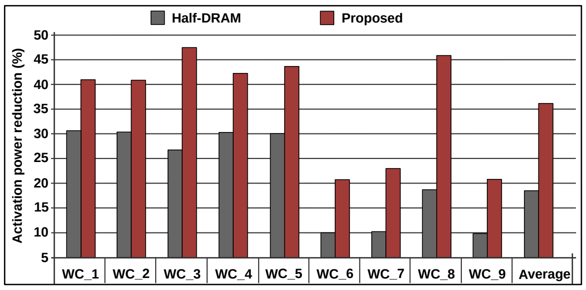
Table 6: The workload mixes that are used in this work. *WC_n*, *E*, and *D* denote workload combination number *n*, encoder, and decoder respectively.

Workload combination	*Workload mixes*
WC_1	*CJPEG, H263_E, JPEG2000_D, MPEG4_E*
WC_2	*DJPEG, MPEG4_E, H263_E, JPEG2000_D*
WC_3	*H263_D, H264_D, MPEG4_E, MPEG2_D*
WC_4	*MPEG4_D, JPEG2000_E, H264_D, H263_D*
WC_5	*MPEG4_E, DJPEG, H264_D, JPEG2000_D*
WC_6	*Fluidanimate, MPEG2_E, JPEG2000_E, Canneal*
WC_7	*Freqmine, Canneal, Blackscholes, Fluidanimate*
WC_8	*JPEG2000_E, H264_E, CJPEG, Bodytrack*
WC_9	*Canneal, Bodytrack, MPEG2_E, JPEG2000_E*

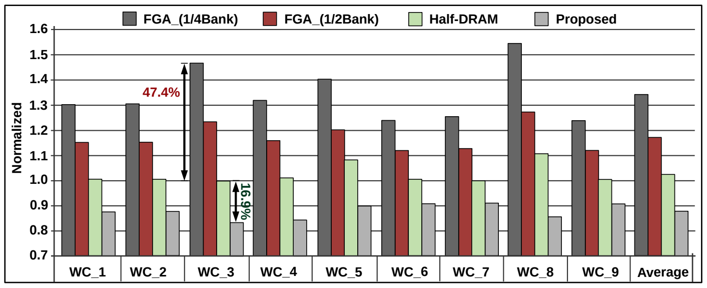
6.3 Evaluation Results and Discussion

Row activation (*ACT*) and bank precharge (*PRE*) are expensive memory operations in terms of power and delay [1, 3, 23]. When these memory commands are performed on a fewer number of bitlines, a significant activation power reduction is achieved. This is attributed to the reduction in the drawn *IDD*0 current during a row cycle (T_{RC}). It can be seen clearly in Table 4 that the activation power and energy consumption is reduced significantly as the size of the activated *DRAM* row decreases. When *DRAM* rows are activated with sizes ranging from $7/8th$ to $1/8th$ of the original row size, the obtained reductions in activation power and energy consumption compared to full row activation are 10.7%, 21.4%, 32.1%, 42.8%, 53.4%, 64.1%, and 74.8% respectively.

Figure 5a shows the reduction in average activation power achieved by our proposed mechanism and the *Half–DRAM* approach compared to baseline. It is clear that all the evaluated multi-programming workloads benefit from a significant reduction in the activation power, up to 47.6% in the case of our proposed approach in this study. On average, our proposed mechanism and the *Half–DRAM* approach provide a 36.2% and 18.5% reduction in average activation power compared to baseline respectively. The workloads running concurrently are affected highly by the memory contention problem. The conventional *DRAM* memory system activates the entire content of large *DRAM* rows only to fetch or write back a 64*B* chunk data. This leads to a massive waste of activation power with no



(a) Average activation power consumption.



(b) Average *DRAM* memory access latency.

Figure 5: (a) The reduction of average activation power consumption for all evaluated four-core multi-programming workloads achieved by *Half–DRAM* and proposed mechanism compared to baseline (b) The average *DRAM* memory access latency for all evaluated four-core multi-programming workloads achieved by the *FGA*_(1/4*Bank*), *FGA*_(1/2*Bank*), *Half–DRAM*, and the proposed mechanisms normalized to the baseline system (the percentage in red color represents a slowdown, whereas percentage in green indicates an improvement).

performance gains. On the other hand, the static half row activation architecture (*Half–DRAM*) delivers a reduction in activation power up to 42.8% compared to full row activation. This may waste the opportunity to achieve a 74.8% power reduction when activating only $1/8th$ of the original row size. The significantly better performance of our proposed mechanism is due to the fact that it selects the optimal number of bitlines being accessed at run-time based on the behavioural changes of applications executing concurrently. It accesses a fewer number of bitlines when a low row hit is shown, whereas a high row hit leads to an increase in the number of accessed bitlines. These factors mean that our proposed approach demonstrates a significant reduction in activation power compared to the baseline and the *Half–DRAM* approach.

Figure 5b shows the average *DRAM* memory access latency obtained by *FGA*_(1/4*Bank*), *FGA*_(1/2*Bank*), *Half–DRAM*, and the proposed mechanism normalized to that of the baseline system. *FGA*_(1/4*Bank*) and *FGA*_(1/2*Bank*) are the fine-grained activation mechanisms that enable respectively one-fourth and half of the total *MATs* in a specific sub-array during a row activation. In order to compensate for the bandwidth loss that appears in these mechanisms, the burst length is set to 32 and 16 respectively. Therefore, these two mechanisms significantly slowdown the workload combinations used in this paper, by an average of 34.6% and 17.3% respectively. The *Half–DRAM* approach degrades the average memory access latency by 2.4% on average and up to 10.5%. On the other hand, our proposed mechanism provides, on average, 12.3% and 14.2% reduction in the average *DRAM* memory access latency compared to baseline and *Half–DRAM* approach respectively. Due to *DRAM* power and energy issues, the number of row activations in the baseline system is limited to four within a rolling window referred to as T_{FAW}. The power and energy savings provided by our proposed mechanism can contribute greatly to improving the row activation rate. This means that the allowed number of row activation commands that can be issued within a T_{FAW} window can be increased to more than four. For instance, the peak energy that can be used within T_{FAW} when activating the entire content of 4 rows is $8067.6pJ$ (4×2016.9). For instance, using $3/8th$ row activation allows us to increase the number of issued *ACT* commands consuming the same amount of energy within T_{FAW} to roughly 9 ($\approx \frac{8067.6}{938.9}$). Thus, significant performance gains are obtained in our proposed approach due to this further relaxation of the T_{FAW} timing constraint. Furthermore, the early precharge technique adopted in this study works in tandem with our proposed dynamic row activation mechanism to convert the potential row-buffer conflicts into row-buffer empties. This, in turn, contributes to a further improvement in the average *DRAM* memory access latency.

Figure 6 shows the *DRAM* energy breakdown of our proposed mechanism and the *Half–DRAM* approach normalized to the baseline system. The achieved average activation power reduction in our proposed approach in this work translates to a significant reduction in the activation energy, by an average of 39% and 19.2% compared to the baseline and the *Half–DRAM* approach respectively. The performance overhead of the workload mixes when using the *Half–DRAM* approach leads to an increase in the background energy by 3.2% over the baseline. On the other hand, the average *DRAM* memory access latency improvement of our dynamic *FGA* mechanism reduces the background energy by 14.9% compared to the baseline. *IDLE DRAM* devices go into low-power mode, thus saving further energy. The reductions in activation and background energy consumption translate to an improvement of 12.8% and 8.3% of total *DRAM* energy efficiency over the baseline and the *Half–DRAM* approach respectively.

In order to enable independent access for the portion of the data that belongs to a specific *DRAM* sub-row, 8 transistors per each *MAT* wordline segment are required in our proposed mechanism. The space existing in each *MAT* wordline segment is used to place the extra 8 transistors. Even without exploiting the existing space, adding 8 transistors for every 512 *DRAM* cells incurs a minimal area overhead, if we take into account that every *DRAM* cell consists of one transistor and one capacitor. On the other hand, the storage required in the memory controller is less than $0.5Kb$. Using *CACTI* [19] and a $32nm$ process node, this additional storage requires $0.0030323mm^2$. Therefore, the area overhead incurred in our proposed mechanism is negligible.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a dynamic fine-grained row activation mechanism, in which an efficient solution addressing the negative impact of the row over-fetching problem on both memory energy

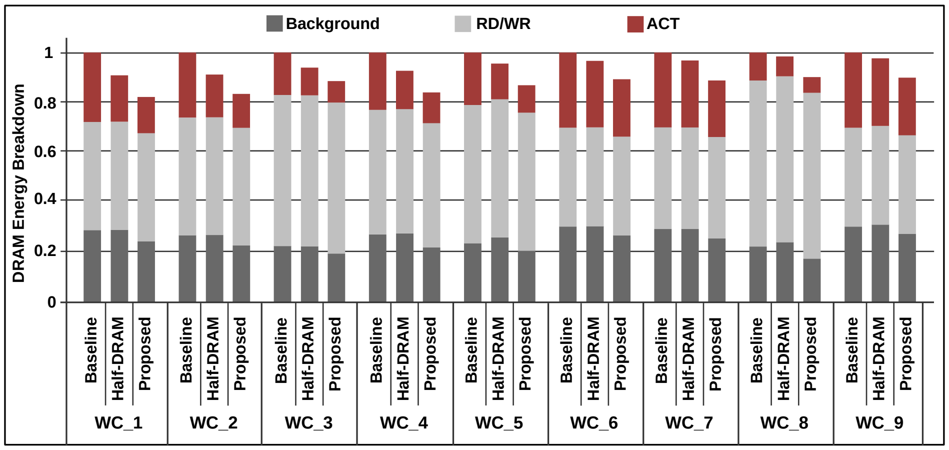


Figure 6: *DRAM* energy breakdown for all evaluated four-core multi-programming workloads when using the proposed mechanism and the *Half–DRAM* approach normalized to the baseline system.

and performance is introduced. On average, a 12.8% and 8.3% reduction of total *DRAM* energy for the evaluated four-core multi-programming workloads is achieved by our proposed approach compared to baseline and *Half–DRAM* approach respectively. Furthermore, our proposed mechanism delivers, on average, a 12.3% and 14.2% improvement in average *DRAM* memory access latency over baseline and *Half–DRAM* approach respectively.

As future work, we will investigate the possibility of integrating our proposed approach with the sub-array level parallelism approach introduced in [12]. This will allow wordline segments from different memory pages to be activated simultaneously in the same bank's row-buffer. Therefore, this could deliver further improvements in memory energy and performance.

REFERENCES

[1] T. Alawneh. 2019. A Dynamic Row-Buffer Management Policy for Multimedia Applications. In *Proceedings of the 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP.* 148–157.

[2] T. Alawneh and A. Elhossini. 2016. A Data Access Prediction Unit for Multimedia Applications. In *Proceedings of the 28th IEEE International Conference on Microelectronics, ICM.* 125–128.

[3] T. Alawneh and A. Elhossini. 2018. A Prefetch-Aware Memory System for Data Access Patterns in Multimedia Applications. In *Proceedings of the 15th ACM International Conference on Computing Frontiers, CF.* 78–87.

[4] E. C. Balis and B. Jacob. 2010. Fine-Grained Activation for Power Reduction in DRAM. *IEEE Micro* 30 (April 2010), 34–47.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT.* 72–81.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. 2011. The Gem5 Simulator. *ACM SIGARCH Computer Architecture News* 39 (May 2011), 1–7.

[7] N. Chatterjee, M. O'Connor, D. Lee, D. R. Johnson, S. W. Keckler, M. Rhu, and W. J. Dally. 2017. Architecting an Energy-Efficient DRAM System for GPUs. In *Proceedings of the 2017 IEEE International Symposium on High Performance Computer Architecture, HPCA.* 73–84.

[8] M. Ghasempour, A. Jaleel, J. D. Garside, and M. Luján. 2016. HAPPY: Hybrid Address-based Page Policy in DRAMs. In *Proceedings of the 2nd International Symposium on Memory Systems, MEMSYS.* 311–321.

[9] N. Gulur, R. Manikantan, M. Mehendale, and R. Govindarajan. 2012. Multiple Sub-Row Buffers in DRAM: Unlocking Performance and Energy Improvement Opportunities. In *Proceedings of the 26th ACM International Conference on Supercomputing, ICS.* 257–266.

[10] H. Ha, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz. 2016. Improving Energy Efficiency of DRAM by Exploiting Half Page Row Access. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO.* 1–12.

[11] D. Kaseridis, J. Stuecheli, and L. K. John. 2011. Minimalist open-page: a DRAM page-mode scheduling policy for the many-core era. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO.* 24–35.

[12] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu. 2012. A case for exploiting subarray-level parallelism (SALP) in DRAM. In *Proceedings of the 39th Annual International Symposium on Computer Architecture, ISCA.* 368–379.

[13] K. Koo, S. Ok, Y. Kang, S. Kim, C. Song, H. Lee, H. Kim, Y. Kim, J. Lee, S. Oak, Y. Lee, J. Lee, J. Lee, H. Lee, J. Jang, J. Jung, B. Choi, Y. Kim, Y. Hur, Y. Kim, B. Chung, and Y. Kim. 2012. A 1.2V 38nm 2.4Gb/s/pin 2Gb DDR4 SDRAM with bank group and ×4 half-page architecture. In *Proceedings of the IEEE International Conferenc Solid-State Circuits, ISSCC.* 40–41.

[14] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. 1997. Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the 30th Annual International Symposium on Microarchitecture, MICRO.* 330–335.

[15] Y. Lee, H. Kim, S. Hong, and S. Kim. 2017. Partial Row Activation for Low-Power DRAM System. In *Proceedings of the 2017 IEEE International Symposium on High Performance Computer Architecture, HPCA.* 217–228.

[16] Micron Technology Inc. [n.d.]. *TN-41-01: Calculating Memory System Power For DDR3.*

[17] Micron Technology Inc., MT41K512M8DA-107 Data sheet. [n.d.]. Micron DDR3 SDRAM Part, 4Gb: x4, x8, x16 DDR3L SDRAM Description. http://www.micron.com/products/dram/ddr3-sdram.

[18] Y. H. Son, S. O, H. Yang, D. Jung, J. H. Ahn, J. Kim, J. Kim, and J. W. Lee. 2014. Microbank: Architecting Through-Silicon Interposer-Based Main Memory Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC.* 1059–1070.

[19] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. 2008. CACTI 5.1. In *Technical Report HPL-2008-20, Hewlett Packard Labs.*

[20] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. 2010. Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores. In *Proceedings of the 37th Annual International Symposium on Computer Architectureg, ISCA.* 175–186.

[21] Xilinx Inc. 2019. Vivado Design Suite. https://www.xilinx.com/support/download.html.

[22] C. Zhang and X. Guo. 2017. Enabling Efficient Fine-Grained DRAM Activations with Interleaved I/O. In *Proceedings of the 2017 IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED.* 1–6.

[23] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie. 2014. Half-DRAM: a High-bandwidth and Low-power DRAM Architecture from the Rethinking of Fine-grained Activation. In *Proceedings of the 2014 ACM/IEEE 41st International Symposium on Computer Architecture, ISCA.* 349–360.