HMCTherm: A Cycle-accurate HMC Simulator Integrated with detailed Power and Thermal Simulation

Zhiyuan Yang
Synopsys Inc., Sunnyvale
zhiyuan.yang@synopsys.com

Michael Zuzak
University of Maryland, College Park
mzuzak@umd.edu

Ankur Srivastava
University of Maryland, College Park
Ankurs@umd.edu

ABSTRACT

The hybrid memory cube (HMC) is a type of 3D stacked memory which is promising to overcome the "memory wall" problem of the conventional DRAM. However, the operating temperature of the HMC can be very high thus degrading the performance and even causing malfunction of the HMC. Therefore, fine-grained thermal-aware management (TM) techniques for HMCs are gaining increasing research interests. In order to design an efficient TM technique, a myriad of simulations should be performed at the early stage. However, current TM simulations for HMCs depend on complex frameworks which involve different tools. These conventional frameworks are very inefficient due to the heavy workload of generating and transferring intermediate data. In order to address this problem, we propose *HMCTherm*, a cycle-accurate simulator for the HMC with the integration of detailed power and thermal estimation. Compared to the conventional TM simulation framework, *HMCTherm* can significantly improve the simulation efficiency by avoiding redundant interface process.

CCS CONCEPTS

• **Computing methodologies** → **Simulation tools**; • **Hardware** → *Temperature simulation and estimation*; • **Software and its engineering** → *Main memory*;

KEYWORDS

HMC, Thermal, Simulation

1 INTRODUCTION

Dynamic random access memory (DRAM) is one of the widely employed types of main memory in today's system. After several years development, the DRAM technology has hit the "memory wall" due to the performance bottlenetck and excessive power consumption [2, 19]. One promising solution to this problem is to vertically stack DRAM chips and use through silicon vias for inter-layer connection. For several years, this 3D stacked DRAM has been actively studied [2, 3, 10, 12]. A prime example that currently exists as an industry standard is the Hybrid Memory Cube (HMC) specification [10]. The HMC is implemented by stacking multiple DRAM dies over a logic die. The logic die is used to control the DRAM banks and connect the HMC with the host CPU/GPU or other HMCs through high speed serial links (*i.e.* SerDes interface). Compared to 2D DRAMs, the HMC can significantly increase the main memory bandwidth and reduce the power per useful data transmission.

The benefits of HMCs come with severe thermal problems due to the increased power density and several layers of dielectrics that block the heat dissipation path [2]. According to [16], the operation temperature in HMCs can easily exceed $90°C$. In future, if we stack HMCs on one or multiple layers of processor (*i.e.* the stacked-HMC-on-processor structure [21]), the temperature will be even higher. High temperature will cause malfunction of the HMC, degrade the memory performance and reliability *etc.* . Under the high temperature, conventional device-level management techniques may cause significant waste in energy and reduction in memory bandwidth [6]. As a result, fine-grained thermal-aware management methods for HMCs [6, 7, 16, 26] have attracted a lot research interests. For example, authors of [6, 16] proposed to tune the refresh rate at DRAM-row-level. In order to design an efficient TM method, a myriad of simulations should be performed during the early stage to test and improve the technique. As a result, it is necessary to develop an accurate and efficient HMC simulator for fine-grained TM techniques.

For several years, there is no free and publicly accessible HMC simulators. Therefore, the academia depends on simulators designed for 2D DRAMs (*e.g.*, *DRAMSim*[20]) [7] to study the architecture and function of 3D DRAMs including HMCs. However, these simulators cannot accurately simulate the function of HMCs since HMCs have a special communication protocol with the host processor. Later, a simulator that is designed specialized for HMCs (known as *HMC-Sim*) [13] was proposed. However, the function and power model of this simulator is very coarse, therefore it is not a good choice for simulating the fine-grained dynamic thermal management for HMCs. Recently, Jeon *et al.* developed a new HMC simulator called *CasHMC* [11]. Compared to its previous counterparts, *CasHMC* can provide cycle-accurate simulation. However, this simulator still does not contain a detailed power or thermal model.

Despite the HMC simulator, current TM simulation frameworks for DRAMs are very inefficient since they are usually comprised of different tools/simulators [7, 16, 26]. Each tool performs a typical type of simulation (*e.g.*, *CasHMC* for function simulation, *DRAMPower* [4] for power simulation, *HotSpot* [9] for temperature simulation *etc.*). Therefore, a significant extra effort should be paid to process the intermediate files. This type of simulation frameworks may be sufficient for coarse-level simulations due to the small amount of intermediate data. For example, in 2D DRAMs, the information of device-level power/temperature is enough since both

MEMSYS '18, October 1–4, 2018, Old Town Alexandra, VA, USA

ACM ISBN 978-1-4503-6475-1/18/10…$15.00
https://doi.org/10.1145/3240302.3240319

the peak temperature and the thermal variance in 2D DRAMs are not large. In HMCs, however, if we still use the conventional framework to perform simulations of the fine-grained TM techniques [6, 7, 16, 26], we will encounter great interface cost (since extremely large amount of intermediate data will be generated, reformatted, and transferred due to the large spatial/temporal sampling size).

In this paper, we develop ***HMCTherm***[1], an HMC simulator based on *CasHMC* [11] with the integration of the detailed power and thermal estimation. The main contributions of this paper are as follows:

(1) We propose a cycle-accurate and DRAM-cell-accurate power model for the HMC.
(2) We propose to integrate the power model and thermal model into a cycle-accurate HMC simulator (*i.e. CasHMC* [11]) such that the simulator can provide cycle-accurate power and temperature estimation without redundant process of the intermediate data.
(3) We propose a simulation framework based on our proposed *HMCTherm* to efficiently perform the simulation for a TM technique for HMCs.

The rest of this paper is organized as follows. Section 2 introduces the background and motivation of this paper. The architecture of the proposed *HMCTherm* simulator is introduced in Section 3. In Section 4, we build a TM simulation framework based on *HMCTherm* and use it to simulate a thermal-aware refresh management technique [16]. The results will be discussed in Section 5.

2 BACKGROUND AND MOTIVATION

In this section, we first introduce the HMC architecture. Then we review the existing function and power simulators for HMCs and traditional DRAMs. The motivation of this work will also be discussed.

2.1 The HMC Architecture

An HMC is constructed by stacking multiple DRAM layers on a logic layer (as illustrated by Figure 1) [10]. Each layer is composed of 16 or 32 partitions where vertically adjacent partitions constitute a *vault*. A vault is comprised of up to 16 DRAM banks, resulting in a maximum of 512 banks per HMC device. Each vault has its own *vault controller* implemented on the logic layer which is analogous to the DIMM controller in 2D DRAMs. Vault controllers and DRAM banks are connected with through-silicon-vias (TSVs) which can achieve high-bandwidth and energy-efficient communication inside the HMC. An HMC device connects host processors or other HMCs through high-speed serial links which is composed of either 8 or 16 lanes (*e.g.*, implemented using TSVs in the stacked-HMC-on-processor structure). Each lane of the link enjoys large signaling rate (up to 15 Gb/s) [10].

In order to access data from the HMC, the host processor sends *packets* through the links. The packet is then routed from the link transceivers to the corresponding vault controllers (whose address is determined in the packet) through a *crossbar network*. The vault controller will then parse the packet and issue low-level DRAM commands to the corresponding DRAM banks. Compared to 2D DRAMs, HMCs can achieve high bandwidth and support process-in-memory.

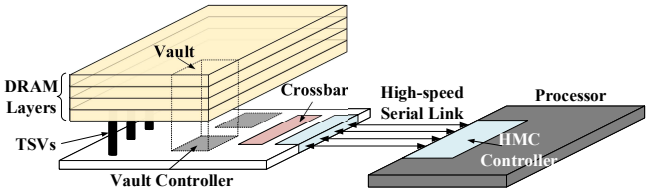

Figure 1: Illustration of the hybrid memory cube (HMC)

Despite the significant improvement in bandwidth, the HMC suffers from severe thermal problems due to the vertically stacked structure and the increased power density. According to [16], the operation temperature of an HMC can reach over $90°C$. The temperature will be even higher if we stack the HMC on a processor [21]. Such high temperature will degrade the performance and even cause malfunction of the HMC. Although we can mitigate the thermal problems during the design time, this may cause overdesign and significant waste of energy. Therefore, thermal-aware management methods for HMCs are gaining more interests (*e.g.*, dynamic refreshment management [6, 7, 16] and thread migration [26] *etc.*). In order to design an efficient TM technique, a myriad of simulations should be performed during the early stage to test and improve the technique. As a result, it is necessary to develop an accurate and efficient TM simulator for HMCs.

2.2 Related Work and Motivation

In order to study the functionality of DRAM, several DRAM simulators have been proposed. *DRAMSim* [20] is one of the most famous simulators for DDR2/DDR3 memory systems. It can provide cycle-accurate simulation results. Since the detailed architecture of HMC is not disclosed, *DRAMSim* is also used in the simulation of 3D DRAMs [7]. However, this simulator is not sufficiently accurate to simulate the function of HMCs due to the unique structure and memory-processor interface of HMCs. *HMC-Sim* [13] is a kind of simulator that targets at HMCs. However, the embedded model of this simulator is very coarse, thus it cannot provide accurate cycle-based simulation results. Recently, Jeon *et al.* proposed a cycle-accurate HMC simulator called *CasHMC* [11]. This simulator can provide cycle-by-cycle simulation of every module in the HMC and generate analysis results including the number of access, data bandwidth, and latency *etc.* . Compared to *HMC-Sim*, *CasHMC* has a much finer model for analyzing the function of the HMC.

The main problem of all existing DRAM/HMC simulators is that they do not contain detailed power or temperature models. Therefore, external tools are required to estimate the power and temperature. For example, *DRAMPower* [4] is often used to estimate the power and energy for a wide range of DRAM configurations (*i.e.* DDR2, DDR3, and LPDDR *etc.*). However, existing DRAM power simulators do not contain the power model for HMCs. Moreover, most of such tools do not estimate the detailed physical distribution of the DRAM power (*e.g.*, the power of each DRAM cell). Therefore, they cannot meet the requirement of the simulation for fine-grained TM techniques for HMCs [6, 16]. As for the temperature estimation, there are various thermal simulators for 3D integrated circuits. *HotSpot* [9] is one of the most widely used simulators. However, this

[1]HMCTherm is available at https://github.com/zyyang1111/HMCTherm.

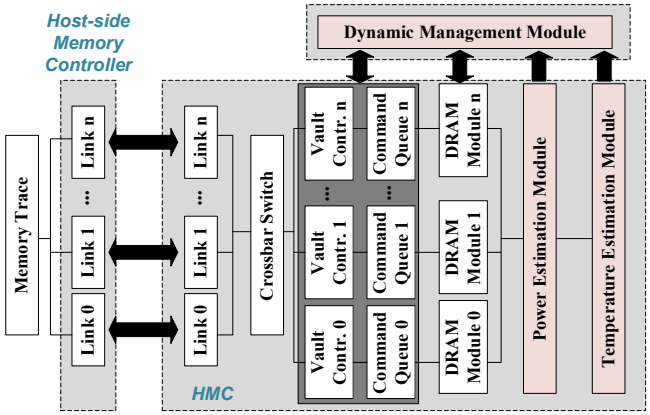

Figure 2: Illustration of the architecture of *HMCTherm* which is built based on *CasHMC* [11]. The pink blocks indicate new modules of *HMCTherm*.

thermal simulator is not designed specified to the HMC. Complex process of the intermediate data is required in order to use *HotSpot*. Obviously, this will lead to heavy interface workload if we want to perform cycle-accurate simulations for the HMC.

In this work, we integrate a detailed power and thermal model to the HMC simulator and propose *HMCTherm*. With the proposed simulator, we can perform cycle-accurate function, power, and temperature simulation without redundant process of the intermediate data.

3 HMCTHERM

Figure 2 shows the architecture of *HMCTherm*. Our simulator is built based on *CasHMC* [11]. The architecture of *CasHMC* is developed to mimic the data/control path of the HMC specification. The input to *CasHMC* is a trace of memory requests (*i.e.* memory address and memory command type) from the host processor. *CasHMC* first parses the request into the packetized request and delivers it through the serial link module which simulates the SerDes host-HMC interface. Following the link module, *CasHMC* routes the memory requests to the vault controllers through the crossbar switch. The vault controller module decodes the packetized memory request into the DRAM command which is sent to the corresponding DRAM module for process. More details of the *CasHMC* architecture can be found in [11].

In *HMCTherm*, we implement power and thermal estimation modules to the original *CasHMC* architecture as illustrated by the pink blocks in Figure 2. Section 3.1 will introduce the power model used in *HMCTherm* and the thermal model will be described in Section 3.2. Besides, *HMCTherm* provides an interface for user-defined thermal-aware management modules. Section 4 describes a TM module integrated to *HMCTherm* which performs the thermal-aware refreshment management.

3.1 Power Estimation Module

HMCTherm contains a power model that can estimate the power of each DRAM cell. Therefore it is able to support the simulation of fine-grained power and temperature management methods for

operation	I_{DD}	I_{DDQ}	t_{OP}
ACT	$I_{DD0} - I_{DD3N}$	0	t_{RAS}
PRE	$I_{DD0} - I_{DD2N}$	0	$t_{RC} - t_{RAS}$
RD	$I_{DD4R} - I_{DD3N}$	I_{DD4RQ}	t_{RD}
WR	$I_{DD4W} - I_{DD3N}$	I_{DD4WQ}	t_{WR}
REF	$I_{DD5} - I_{DD3N}$	0	t_{RFC}
BACK_{pre}	I_{DD2N}	0	t_{bpre}
BACK_{act}	I_{DD3N}	0	t_{bact}

Table 1: Datasheet Information of Memory Operations

HMCs. The power of HMCs is modeled in two parts: (1) basic memory operation power (Section 3.1.1) and (2) logic layer power (Section 3.1.2). In Section 3.1.3, we will introduce the power estimation scheme in the *HMCTherm*.

3.1.1 Basic Memory Operation Power. Despite the stacked structure and the serial link interface, the internal memory access of HMCs basically follows the DDR3 scheme. Therefore, we can use the same power modeling method for DDR3 to model the basic memory operation power (*e.g.*, the power of DRAM write/read, activation *etc.*) in the HMC. Currently, the most viable method to estimate the total power consumption for each DRAM operation is to use the JEDEC-specified current, voltage and time values from memory datasheets which are acquired based on real hardware measurements [3]. The datasheet specifies the supply voltage of the memory. It also gives the execution time of each memory operation and the average current the memory consumes during this period of time. Table 1 lists the datasheet information for seven memory operations: Activate (ACT), Precharge (PRE), Read (RD), Write (WR), Auto-Refresh (RF), Precharged-mode Background (BACK_{pre}), and Active-mode Background (BACK_{act})[2]. In this table, we use t_{OP} to indicate the period that a memory operation is used. t_{RAS}, t_{RC} and t_{RFC} are given by the datasheet. t_{RD} and t_{WR} is determined by the burst-length (BL), and is usually defined as $t_{RD} = t_{WR} = BL/2$ due to double rate access. t_{bpre} and t_{bact} are the total time period the HMC spent in the precharged and active modes, respectively. Given the above information, the average power consumed by the HMC during a period of time (t_{TOT}) can be calculated using the following equation:

$$P = \frac{1}{t_{TOT}} \times Energy = \frac{1}{t_{TOT}} \sum_{i=1}^{N_{op}} (I^i_{DD} + I^i_{DDQ}) \times V_{DD} \times t^i_{OP} \quad (1)$$

In this equation, N_{OP} is the total number of memory operations within t_{TOT}. It should be noted that the accuracy of this model is determined by the accuracy of values provided by the datasheet. Since we do not have the detailed specification of HMCs, we can use the datasheet for DDR3 to calculate the power [4].

Specify the power for each DRAM *mat*: As described in Section 2.2, in 3D stacked memory systems, merely knowing the total memory power is not sufficient to simulate the fine-grained TM techniques [6, 16]. Therefore, in *HMCTherm*, we further evaluate the power in a much finer granularity. We assume that each layer of the HMC works similarly to 2D DRAMs, such that it is organized as an array of DRAM tiles (also known as "mat" in many

[2] BACK_{pre} and BACK_{act} are classified as two memory operations just for the simplicity of illustration. They actually indicate two states of the background power.

literature). *Mat* is the basic organization of modern DRAMs [18, 23]. Each mat has its own supporting circuits (such as decorders, row buffers, sense amplifiers *etc.*) located at the peripheral of the mat. In this work, we will specify the power of each mat by spreading the power as evaluated above to the corresponding mats. In order to achieve this, we first spread the power to specific DRAM cells and then calculate the power of mats. The power of different memory operations is spread differently:

- **RD, WR.** For each Read/Write operation, a set of data is loaded from/stored to a set of continuous DRAM cells. This set of DRAM cells is determined by the requested memory address (*i.e.* {vault, bank, row, column}) and the number of cells is determined by the burst length (*BL*). The power of each Read/Write operation is spread among the corresponding cells.
- **ACT, PRE, REF.** Activate, Precharge, and Auto-Refresh operations are performed for a row of DRAM cells [3]. The power of such operations will be spread across the cells in the corresponding row.
- **BACK$_{pre}$, BACK$_{act}$.** The background power of a DRAM bank is uniformly distributed among all cells within the bank.

With the above-mentioned method, we can specify the power for each DRAM cell. Following this, with a user-defined DRAM-cell-to-mat mapping function (or profile), we can calculate the power for each mat by summing up the power of DRAM cells within the mat. Currently, a simple mapping function is built into the power model and we allow the user to specify the size of mat from the command line (by specifying the value of N_x and N_y, which indicates dividing a DRAM bank into $N_x \times N_y$ mats, respectively). Users are also allowed to input their own mapping file.

3.1.2 Logic Layer Power. Compared to the basic DRAM operation power, the power of the logic layer in the HMC is much harder to model. This logic layer serves a number of functions such as the communication between the HMC and the host CPU/GPU through serial links, the vault controller, and the cross-bar switch between the links and vault controllers *etc.* . Since we do not have either the detailed circuit-level technologies or the measured power values for the logic layer, it is difficult for us to accurately model the power of this layer. According to [10], the average power of the logic layer is around 1.83 times of the power of DRAM layers (*i.e.* the total basic memory operation power). Therefore, in this work, after getting the total power of the DRAM layers, we multiply this power by 1.83 to estimate the total power of the logic layer. And we assume this power is distributed uniformly in the logic layer.

3.1.3 Power Estimation Diagram. The previous two sections introduced the power modeling methodology in *HMCTherm*. In this section, we will describe how this power model is implemented in *HMCTherm*. Figure 3 illustrates the power estimation diagram in our simulator. In the proposed simulator, the power and temperature are estimated every *power sampling epoch* which is defined by the user in the command line. The HMC power estimation process is introduced as follows:

[3]Suppose the distributed refresh scheme (which is the most common scheme in modern DRAMs [16]) is used

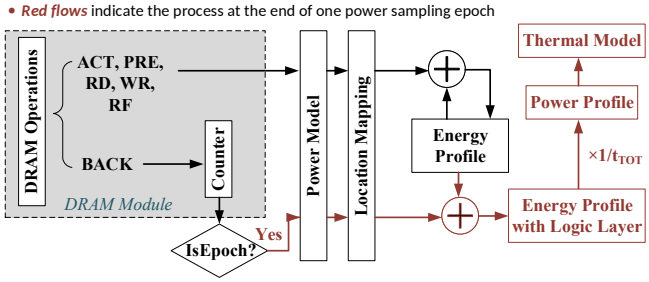

Figure 3: Diagram of the power estimation in *HMCTherm* (*isEpoch* indicates whether one power sampling epoch is reached).

When a DRAM operation (*i.e.* ACT, PRE, RD, WR, RF) is performed, the energy of this operation is calculated using the power model (Equation 1). Then the energy, the memory address and the operation type are sent to the *location mapping module* to spread the energy to the corresponding DRAM cells (or subarrays). As for the background energy (*i.e.* BACK$_{pre}$ or BACK$_{act}$), we use a counter to count the number of cycles that each bank of the HMC falls in each memory mode (*i.e.* active or precharged). At the end of a power sampling epoch, we use this statistic to calculate the background energy during this period of time and spread this energy for each bank. The spread background energy is then added to the energy profile. At this time, the total energy of DRAM layers is acquired and the energy profile of the logic layer is thus calculated (Section 3.1.2). Following this, we divide the energy of each DRAM cell (or subarray of DRAM cells) by the length of the epoch (t_{TOT}) to get the power profile. This power profile is then fed into the thermal model (Section 3.2) to get the temperature estimation.

Besides the HMC power, our simulator also provides an interface such that the user can input the CPU/GPU power. In this way, our simulator can achieve integrated thermal simulation for the stacked-HMC-on-processor structure.

3.2 Thermal Estimation Module

3.2.1 Transient Model. One of the most popular thermal models is the thermal resistance and conductance network (RC-network) which is analogous to the electrical network [9, 22]. The whole 3D stack is divided into thermal grids. In the current version of *HMCTherm*, the thermal grid is the same as the DRAM subarray defined by the user (as introduced in Section 3.1). The heat flux (which is the analog of the electrical current source) of each grid is calculated based on the power distribution of the HMC. The average temperature of a thermal grid is analogous to the voltage. The temperature difference between two adjacent thermal grids is caused by the heat flow between the two grids. The heat conduction between two thermal grids is represented by a thermal resistor which is calculated based on the geometrical dimension and material properties of the related thermal grids. Analogous to the electrical capacitance, each thermal grid has a thermal capacitor which represents the ability of the grid to store the thermal energy. A Thermal RC-network is illustrated by Figure 4. In the network, each thermal grid is represented by a node point.

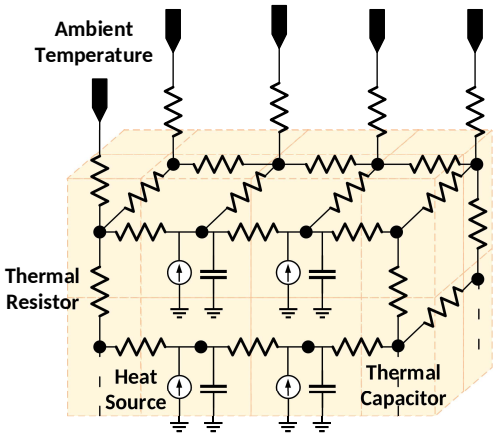

Figure 4: Illustration of the thermal RC network.

The relationship between the temperature and the power is expressed as follows:

$$GT = P + C\frac{dT}{dt} \tag{2}$$

In Equation 2, t represents the time. $P \in \mathbb{R}^N$ and $T \in \mathbb{R}^N$ are vectors containing the power and temperature of each thermal grid, respectively. $G \in \mathbb{R}^{N\times N}$ is the conductance matrix and $C \in \mathbb{R}^{N\times N}$ is a diagonal matrix with each element in the diagonal representing the thermal capacitance of the corresponding thermal grid. G and C are calculated with thermal parameters determined by the user, and we assume they are constant during the simulation. Normally, G and C are sparse matrices.

The transient temperature profile is calculated every power sampling epoch (as introduced in Section 3.1.3). At the end of each epoch, we estimate the average power profile (*i.e.* P) during this epoch using the method introduced in Section 3.1. The current temperature is calculated by solving the Equation 2 using explicit method [27] with the temperature at the end of the previous epoch as the initial state. Note that, since G and C are constant, they are only calculated once at the beginning of the simulation and are reused afterwards.

3.2.2 Steady State Model. Sometimes, it is useful to know the steady state temperature profile after the simulation. Therefore, we also implement a steady state thermal model which is used to calculate the temperature at the end of the simulation. The steady state thermal model only contains the thermal resistors, thereby Equation 2 is reduced to:

$$GT = P. \tag{3}$$

In this equation, P is the profile of the average power during the period of simulation. In order to solve the linear equation (*i.e.* $T = G^{-1}P$), we use SuperLU [5], which is a library to solve large sparse linear equations.

3.3 Thermal Model Validation

The proposed thermal model is evaluated against the FEM simulation results performed with ANSYS Workbench. We use both methods to estimate the temperature of a one-layer multi-core processor. The power profile of this processor is generated from

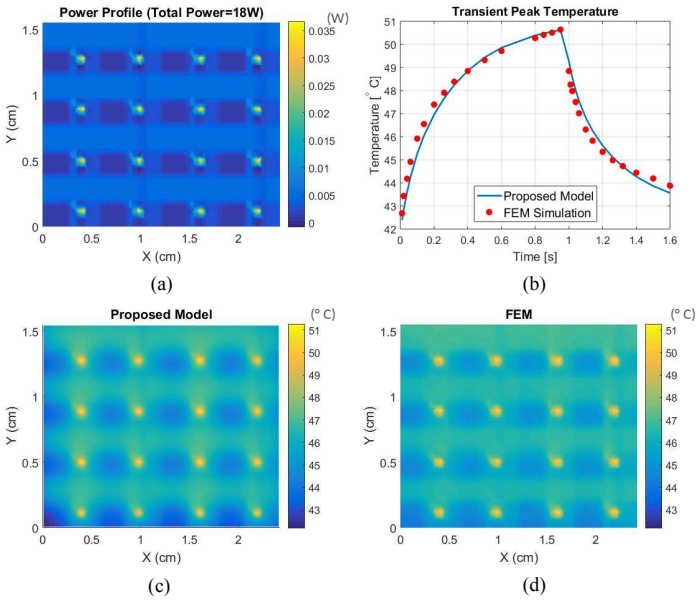

Figure 5: (a) The original power profile, (b) the transient result for for the peak temperature, (c) the temperature profile at 1s calculated using our thermal model and (d) the temperature profile at 1s calculated using the FEM method

[17]. The simulation is taken for 1.6s. During the first one second, the processor power stays constant and the total power equals to 18W. The power profile during this period of time is illustrated in Figure 5(a). After 1s, the processor is switched to a low-power mode and the total power is reduced to 4.5W. The transient peak temperature is shown in Figure 5(b) where the red dots indicating the FEM simulation results while the blue line indicating the result of the proposed model. Figure 5(c) and (d) illustrate the temperature profile at 1s estimated using the proposed model and the FEM method, respectively. As illustrated by the figure, our proposed model matches the FEM results accurately in both temporal and spatial domain. Moreover, our method can estimate the temperature in a much efficient way. Also note that although we use the processor power to validate the proposed model, this model is applicable to the memory temperature estimation.

4 CASE STUDY: THERMAL-AWARE REFRESH MANAGEMENT

HMCTherm supports cycle-accurate function, power and thermal simulation, thereby it provides a much more efficient method to test and develop fine-grained thermal-aware dynamic management techniques for the HMC. In this section, we will use *HMCTherm* to simulate a thermal-aware refreshment management technique similar to the one introduced in [16].

4.1 Background

Since DRAM cells lose data due to the leak of capacitor charge, *refresh* is required to preserve the data integrity. Refresh is a process that periodically performs readout and rewrite to DRAM cells thus causing energy waste and performance degradation [16]. The overhead of refresh increases as the DRAM capacity increases. In a

4GB DRAM, around 30% DRAM energy will be consumed by the refreshment [16].

The refresh period is determined by the *retention time* of DRAM cells. The retention time of a DRAM cell is determined as the time that a DRAM cell loses data. Since the retention time varies among DRAM cells, the refreshment period is taken as the shortest retention time among all DRAM cells. The temperature has significant impacts on the retention time of DRAM cells. When the temperature increases, the retention time decreases exponentially [8, 15]. In 2D DRAMs, since temperature is usually not very high, the temperature's impacts on the refresh period is addressed in a coarse way: during the normal temperature operation (*i.e.* below $85°C$), a DRAM cell is refreshed every 64ms, while the refresh period reduces to 32ms when the temperature exceeds $85°C$ but below $95°C$.

Due to the stacked structure, the operation temperature of HMCs can easily exceed $90°C$. If considering the stacked-HMC-on-processor structure, this temperature will be even higher. The high temperature will significantly exacerbate the overhead of DRAM refresh (*e.g.*, generating more energy). In order to solve this problem, several people have proposed run-time management techniques to tune the refresh rate according to the temperature [7, 16]. These authors build complicated simulation frameworks to evaluate their proposed techniques. In this section, we will show how to use *HMCTherm* to perform such simulations. In order to demonstrate this, we build a framework based on *HMCTherm* to simulate the thermal-aware refresh management technique introduced in [16]. This refresh management technique is briefly introduced below and our framework will be described in Section 4.2.

Liu *et al.* [16] proposed a row-level refresh management technique based on *distributed refresh* scheme. The distributed refresh requires the memory controller to periodically issue an auto-refresh command to the DRAM which then chooses specific rows to refresh according to an internal counter. The number of rows to refresh during one auto-refresh command is determined by the memory capacity. The time between two auto-refresh commands is indicated by t_{REFI}. Conventionally, all DRAM cells have the same t_{REFI} which is determined by the shortest retention time among all DRAM cells. However, authors of [16] discovered that a significantly large amount of DRAM cells do not need that frequent refreshment. Therefore, they proposed to allow each row of DRAM cells to have its own t_{REFI} which is determined by the shortest retention time of the cells in the row. They adopted three levels of refresh period: {64ms, 128ms, 256ms}. If the retention time of a row (*i.e.* the minimum retention time of DRAM cells in the row) falls in the range of [128ms, 256ms), this row is refreshed every 128ms. Each row has a counter that determines if the row is ready to refresh. The temperature's impact on the retention time is also considered. They used the model presented in [8] to scale the retention time of each row according to the temperature. However, the authors depend on external tools to estimate the temperature, which is very inefficient. In the next section, we will introduce a framework to simulate the above-mentioned dynamic management technique.

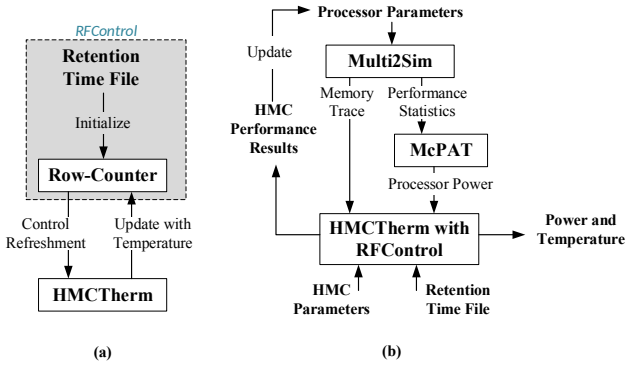

Figure 6: Illustration of the (a) RFControl module and (b) the whole simulation framework.

4.2 Simulation Framework

In this section, we will introduce a framework that is built based on the proposed *HMCTherm* to implement the thermal-aware refresh management technique as introduced in [16] on a stacked-HMC-on-processor structure. We stack a HMC on a multi-core processor. Our framework includes the architectural-level function and power simulators of multi-core processors, *HMCTherm* and the scripts processing interface data. The whole framework is illustrated in Figure 6 and is introduced in detail as follows.

4.2.1 Refresh Control Module. The kernel of this framework is our proposed *HMCTherm*. By default, the refresh in *HMCTherm* follows the distributed refresh scheme. In order to support the refresh management technique, we add a refresh rate control module (called *RFControl*) to the original *HMCTherm*. This module has the following three functions (as shown in Figure 6(a)):

(1) **Initialize the Retention Time.** RFControl module reads the data from a text file which specifies the initial retention time of each DRAM cell. This file is generated by the user. In this work, the retention time of each DRAM cell is generated according to the probability distribution reported in [16].
(2) **Row-level Refreshment Counter.** RFControl implements a counter (*Row-Counter*) for each row of the HMC as introduced in [16] to achieve the row-level management of the refresh rate. The Row-Counter is initialized with the initial retention time for each DRAM cell (as introduced above).
(3) **Thermal-aware Management.** RFControl adjusts the retention time of each row of the HMC (by adjusting the Row-Counter) according to the temperature profile calculated by *HMCTherm* (as introduced in Section 3).

Note that the implementation of RFControl is not difficult. However, the main benefits of using *HMCTherm* is that we can get the power and thermal information of HMCs at fine-grained spatial/temporal granularity without transferring data between the HMC simulator, power estimator and thermal simulator.

4.2.2 Other Modules in the Framework. The whole framework is illustrated in Figure 6(b). We use *Multi2Sim* [24] to perform the architectural function simulation of a multi-core processor. The architectural parameters of the processors are defined by the user. The main memory latency is initialized with a typical value and

then updated iteratively using the value from the *HMCTherm* simulation result. The output of *Multi2Sim* will be a memory trace, the performance statics (*e.g.*, the number of cycles, the branch hit rate *etc.*). The performance statics are fed into *McPAT* [14] to calculate the power of each module in the processor, which is then used to generate the power profile of the processor according to the processor's floorplan specified by the user. We provide a MATLAB script to help this conversion. The power profile of the processor layer, the memory trace, and the initial DRAM cell retention time are then fed into the *HMCTherm* simulator to perform memory simulation. *HMCTherm* simulation will give you the HMC performance, power and temperature. The HMC latency is used to update the corresponding architectural parameter for the CPU simulation. This is processed in a loop which will stop when the change in HMC latency is negligible.

5 RESULTS AND ANALYSIS

In this section, we perform simulations with *HMCTherm* and compare the results with/without using the refresh management technique introduced in [16]. The baseline case (*i.e.* **Without Management**) is simulated simply with *HMCTherm* while the "**With Management**" case is simulated using the framework described in Section 4.2. For the baseline case, the refresh period for all DRAM cells is 32ms due to the high operation temperature in the HMC. For the "With Management" case, four levels of refresh period ({32ms, 64ms, 128ms, 256ms}) are applied. The processor layer is implemented with 16 cores. The operating frequency of the processor is 2GHz. Other parameters of the processor are the same as introduced in [21] except for that we will update the main memory latency according to the *HMCTherm* simulation results. The operating frequency of the HMC is 1.25GHz and the power sampling epoch used in this experiment is 200000 HMC cycles. We run simulations with three benchmarks from the PARSEC [1] and SPLASH-II [25] benchmark suits. In Section 5.1 we will introduce the output files of *HMCTherm*. The simulation results will be discussed in Section 5.2.

5.1 Output files

When the simulation is over, a setting log file and six output files containing the information of the performance (*i.e.* latency, bandwidth *etc.*), power, and temperature of the HMC are saved to the user-defined result folder. The setting log file (**_setting.log*) and the performance output file (**_result.log*) are the same as introduced in [11]. The *HMCTherm* has five CSV files storing the power and temperature information of the HMC.

- *power_trace.csv* file stores the power of each power grid for each power sampling epoch. The format of each line of data in this CSV file is "epoch ID, layer, x-coordinate, y-coordinate, power".
- *temperature_trace.csv* file stores the temperature of each thermal grid[4] for each power sampling epoch. The format of each line of data in this CSV file is "epoch ID, layer, x-coordinate, y-coordinate, temperature".
- *Average_Power_Profile.csv* file stores the average power of each power grid during the time period of the simulation. The format of each line of data in this CSV file is "layer, x-coordinate, y-coordinate, power".
- *static_temperature.csv* file stores the static temperature of each thermal grid at the end of the simulation (Section 3.2.2). The format of each line of data in this CSV file is "layer, x-coordinate, y-coordinate, temperature".
- *power_statics_trace.csv* file stores the power breakdown for each type of DRAM operation for each power sampling epoch. The format of each line of data in this CSV file is "epoch ID, total power, RD power, WR power, ACT power, RF power, PRE power".

[4]In the current version, the thermal grid is the same as the power grid.

5.2 Results for Refresh Management

Figure 7-9 illustrate the simulation results for the LU, FLUIDANIMATE, and STREAMCLUSTER benchmark, respectively. In each figure, (a)-(d) shows the trace of total HMC power, refresh power, DRAM access power (*i.e.* RD, WR, ACT and PRE power), and the HMC transaction latency, respectively. Red lines in the figures illustrate the traces of the conventional refreshment technique while the blue lines show the traces of the case when using the thermal-aware refresh as introduced in [16]. As demonstrated by the sub-figures (a) and (b) in Figure 7-9, if the conventional refresh technique is used, more than half of the HMC power is consumed by the refresh. Frequent fresh will slow down the memory access rate (as illustrated by the sub-figure (d) in each figure), thus leading to a relatively small DRAM access power (as shown by the sub-figure (c) in each figure). According to [16], most of the refreshment is unnecessary. With the management technique proposed by [16], the HMC is refreshed less frequently, thereby the total power and the transaction latency are reduced.

6 CONCLUSION

HMC is a kind of 3D stacked DRAM which can achieve high memory access bandwidth. However, the thermal problem of HMCs and the future HMC-on-processor structure necessitates fine-grained thermal-aware management for HMCs. In this paper, we propose *HMCTherm*, a cycle-accurate HMC simulator that is integrated with detailed power and thermal model. *HMCTherm* can support simulations of fine-grained thermal-aware management techniques with higher efficiency compared to traditional simulation frameworks due to the significant reduction in the workload of generating and transferring the intermediate data.

ACKNOWLEDGMENTS

The authors would like to acknowledge that this work has been funded by NSF grant 1642424.

REFERENCES

[1] C. Bienia, et al. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.
[2] B. Black, et al. Die stacking (3D) microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–479. IEEE Computer Society, 2006.
[3] K. Chandrasekar, et al. System and circuit level power modeling of energy-efficient 3D-stacked wide I/O DRAMs. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 236–241. IEEE, 2013.
[4] K. Chandrasekar, et al. DRAMPower: Open-source DRAM power & energy estimation tool. *URL: http://www. drampower. info*, 22, 2012.

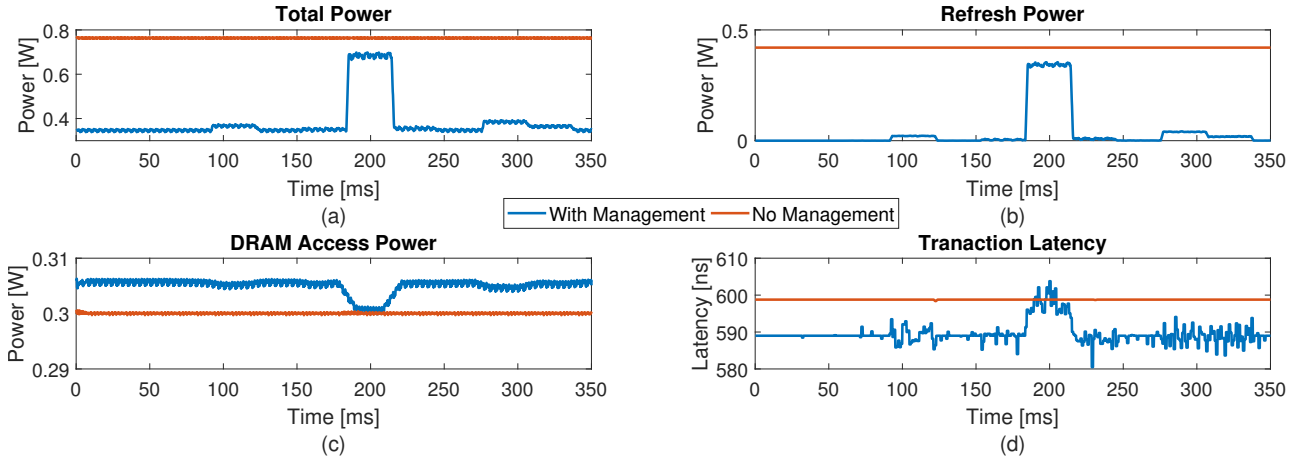

Figure 7: For LU benchmark: the trace of (a) Total power, (b) refresh power, and (c) DRAM access power of the HMC with and without the refresh management [16].

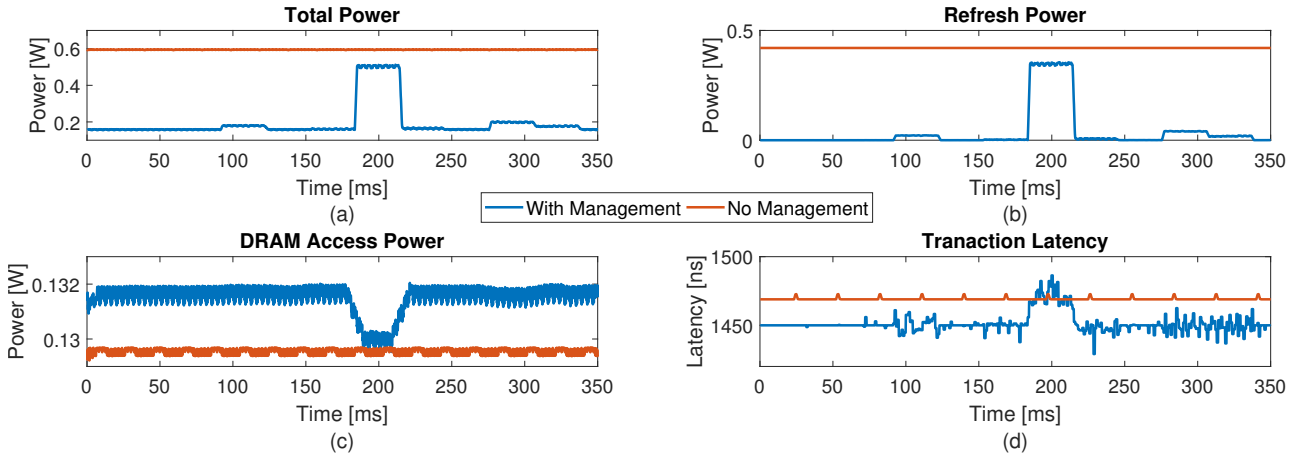

Figure 8: For FLUIDANIMATE benchmark: the trace of (a) Total power, (b) refresh power, and (c) DRAM access power of the HMC with and without the refresh management [16].

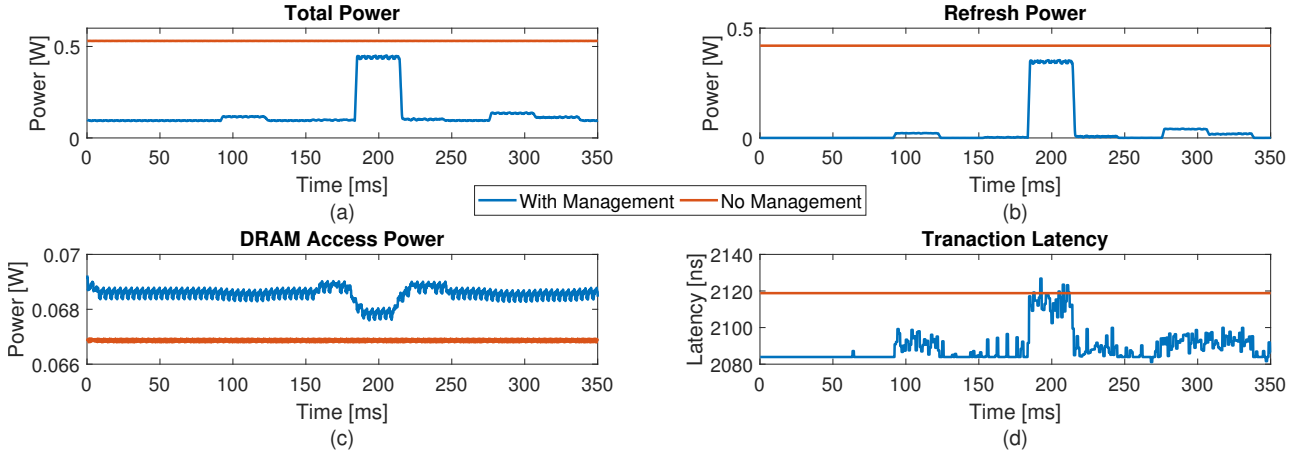

Figure 9: For STREAMCLUSTER benchmark: the trace of (a) Total power, (b) refresh power, and (c) DRAM access power of the HMC with and without the refresh management [16].

[5] J. W. Demmel, et al. An asynchronous parallel supernodal algorithm for sparse gaussian elimination. *SIAM Journal on Matrix Analysis and Applications*, 20(4):915–952, 1999.

[6] M. Ghosh and H.-H. S. Lee. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 134–145. IEEE Computer Society, 2007.

[7] M. Guan and L. Wang. Temperature aware refresh for DRAM performance improvement in 3D ICs. In *Quality Electronic Design (ISQED), 2015 16th International Symposium on*, pages 207–211. IEEE, 2015.

[8] T. Hamamoto, et al. On the retention time distribution of dynamic random access memory (DRAM). *IEEE Transactions on Electron devices*, 45(6):1300–1309, 1998.

[9] W. Huang, et al. HotSpot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, 2006.

[10] J. Jeddeloh and B. Keeth. Hybrid memory cube new DRAM architecture increases density and performance. In *VLSI Technology (VLSIT), 2012 Symposium on*, pages 87–88. IEEE, 2012.

[11] D.-I. Jeon and K.-S. Chung. Cashmc: A cycle-accurate simulator for hybrid memory cube. *IEEE Computer Architecture Letters*, 16(1):10–13, 2017.

[12] U. Kang, et al. 8Gb 3D DDR3 DRAM using through-silicon-via technology. In *Solid-State Circuits Conference-Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pages 130–131. IEEE, 2009.

[13] J. D. Leidel and Y. Chen. Hmc-sim: A simulation framework for hybrid memory cube devices. *Parallel Processing Letters*, 24(04):1442002, 2014.

[14] S. Li, et al. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480. IEEE, 2009.

[15] J. Liu, et al. An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 60–71. ACM, 2013.

[16] J. Liu, et al. RAIDR: Retention-aware intelligent DRAM refresh. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 1–12. IEEE Computer Society, 2012.

[17] T. Lu, et al. TSV-Based 3-D ICs: Design Methods and Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(10):1593–1619, 2017.

[18] M. O'Connor, et al. Fine-grained DRAM: energy-efficient DRAM for extreme bandwidth systems. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 41–54. ACM, 2017.

[19] P. Rosenfeld. *Performance exploration of the hybrid memory cube*. PhD thesis, 2014.

[20] P. Rosenfeld, et al. DRAMSim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, 2011.

[21] C. Serafy, et al. Thermoelectric codesign of 3-D CPUs and embedded microfluidic pin-fin heatsinks. *IEEE Design & Test*, 33(2):40–48, 2016.

[22] B. Shi, et al. Non-uniform micro-channel design for stacked 3D-ICs. In *Proceedings of the 48th Design Automation Conference*, pages 658–663. ACM, 2011.

[23] Y. H. Son, et al. Reducing memory access latency with asymmetric DRAM bank organizations. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 380–391. ACM, 2013.

[24] R. Ubal, et al. Multi2Sim: a simulation framework for CPU-GPU computing. In *Parallel Architectures and Compilation Techniques (PACT), 2012 21st International Conference on*, pages 335–344. IEEE, 2012.

[25] S. C. Woo, et al. The SPLASH-2 programs: Characterization and methodological considerations. In *ACM SIGARCH computer architecture news*, volume 23, pages 24–36. ACM, 1995.

[26] D. Zhao, et al. Temperature aware thread migration in 3D architecture with stacked DRAM. In *Quality Electronic Design (ISQED), 2013 14th International Symposium on*, pages 80–87. IEEE, 2013.

[27] O. C. Zienkiewicz and R. L. Taylor. *The finite element method for solid and structural mechanics*. Elsevier, 2005.