Asymmetric DRAM Synthesis for Heterogeneous Chip Multiprocessors in 3D-Stacked Architecture

Minje Jun, Myoung-Jin Kim, and Eui-Young Chung
School of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea.
{jjuninho, mjkim86, eychung}@yonsei.ac.kr

***Abstract*—Various computational requirements of real-world applications have leveraged moving to heterogeneous chip multiprocessors (CMPs) from homogeneous ones. In the meantime, three-dimensional integration of DRAMs and processors using Through Silicon Vias (TSVs) has emerged as the most viable solution for breaking the memory wall in CMP environment by bringing much higher memory bandwidth compared to current PCB level processor-DRAM integration. However, most researches on 3D-stacked DRAM have focused on increasing the memory bandwidth to improve the overall throughput of a system, even though the memory access requirements of real-world applications are various just as the computational requirements. To tackle this problem, we propose an *asymmetric 3D-stacked DRAM architecture* where the DRAM die is divided into multiple segments and the segments are optimized for different memory requirements. Also, since the optimal architecture of the DRAM can be different for different heterogeneous CMPs, we propose an automatic synthesis method for the asymmetric 3D-stacked DRAM architecture. The experimental results show that the area-power-product is reduced by 65.1% on average compared to the conventional architectures for the four realistic benchmarks and many of their derivatives.**

I. Introduction

Chip multiprocessor (CMP) has emerged as the most viable solution to meet the increasing demand for performance within a tight power budget. In fact, CMPs are already prevalent nowadays in various market entries, from servers to smartphones. In CMP, it is expected that the throughput scales up as more cores are integrated provided that tasks are properly assigned to the cores. However, the throughput scaling of CMP is upper-bounded by the theoretical limitation of applications' parallelism. Moreover, there are many practical issues which further limit the performance scaling of CMP, such as memory bottleneck also known as *memory wall*.

In order to break the memory wall, three-dimensional integration of DRAMs and processors (so called, DRAM-stacked processor) has received enormous attention recently [1]–[7]. Most of those works focused on utilizing the abundant memory bandwidth brought by a large number of I/Os implemented with dense Through Silicon Vias (TSVs) to increase the system throughput. This direction is not irrelevant to the tendency that most researches on CMP focused on the overall throughput of a system rather than the latency of a single application. However, real-world workloads have different requirements for their throughput and latency, and many workloads are oriented more to the latency than the throughput, e.g. compiler and compressor [8]. Even among throughput-oriented workloads, the required throughput can vary.

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA

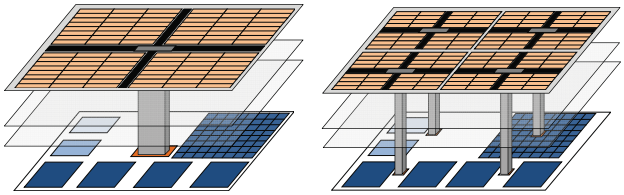
For this reason, *heterogeneous CMP* has been eagerly studied and adopted by many products these days, e.g. Intel's Sandy Bridge, AMD's Fusion, and most of the recent application processors for smartphones. In heterogeneous CMPs, the cores may have different functionalities and instruction set architectures (ISAs), and each core executes workloads suitable for it; for example, a complex out-of-order CPU executes latency-oriented workloads or those having high instruction-level parallelism (ILP), while a GPU (or a cluster of simple in-order cores) executes throughput-oriented or massively parallel general-purpose workloads as well as graphics-related ones.

Unlike the heterogenous CMP is gaining more popularity, recent researches for 3D-stacked DRAM (hereafter, **3D-DRAM** for short) still focused on maximizing memory bandwidth [1], [3]–[7]. However, real-world workloads impose various requirements on DRAMs just as they do on the cores. For example, latency-oriented workloads (or the cores executing them) are very likely to require short access latency from memory, while throughput-oriented workloads are likely to require high memory bandwidth. It is an impractical (or impossible) solution to achieve both short latency (which needs finer-grained array partitioning) and high bandwidth (which needs wider I/O width, more banks and ports) with the current DRAM architecture since it inevitably degrades both the density and the energy efficiency of DRAM significantly.

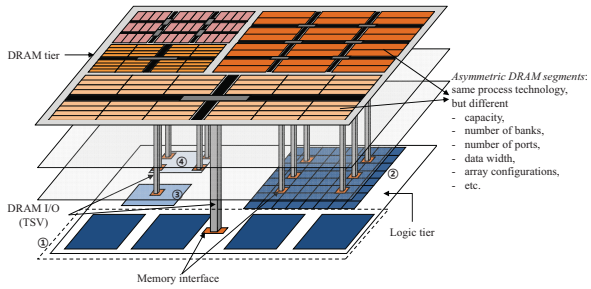
Meanwhile, DRAM is spotlighted as a major power consumer in modern systems. In servers, the operating costs induced by the power consumption of DRAM chips easily exceed their purchase cost [11]. Also, it was reported that the power consumption of DRAM may exceed that of CPU depending on running applications in smartphones [12]. This trend makes the traditional cost-per-bit minimization approach of DRAM questionable, and motivates to put more weights on the energy efficiency even though the cost-per-bit is somewhat compromised.

By tackling those issues, we propose a novel *asymmetric 3D-DRAM* architecture for heterogeneous CMPs. In the proposed architecture, the entire DRAM die is divided into multiple independent *segments*[1] where the segments have different configurations in order to meet the different memory requirements of heterogeneous cores. The proposed 3D-DRAM architecture is shown in Fig. 1, in comparison with the conventional architectures (details will be discussed in Section IV). Note that different heterogeneous CMPs may have completely different memory requirements and there exist infinitely many combinations of cores for composing a heterogeneous CMP. Therefore, we also propose a synthesis method which finds the best asymmetric 3D-DRAM architecture for the given heterogeneous CMP and the memory requirements of the cores in it. To the best of our knowledge, this is the first work which exploits the various implementation choices of DRAM arrays and their asymmetric in-

[1]The terms 'DRAM segment' and 'segment' will be used interchangeably throughout the paper depending on the context.



(a) Traditional 3D-DRAM [9] (b) Symmetrically segmented 3D-DRAM [6], [7], [10]



(c) Proposed asymmetric 3D-DRAM

Type	Latency	Req.BW	MLP	Capacity
①	moderate	moderate	small	very large
②	loose	high	very large	large
③	very tight	low	small	small
④	tight	high	large	small

(d) Memory access characteristics of the cores

Fig. 1: The concepts of the asymmetric 3D-DRAM architecture and the conventional ones.

tegration, especially for DRAM-stacked heterogeneous CMPs. Our experimental results show that the proposed architecture can reduce the area-power-product of 3D-DRAM enormously compared to the conventional 3D-DRAM architectures.

The rest of the paper is organized as follows. In Section II, the related works will be summarized. In Section III, the background for this work will be briefly presented. In Section IV, motivational examples will be given. In Section V, the automatic synthesis method for the asymmetric 3D-DRAM will be presented. In Section VI, the evaluation results for the proposed architecture and the synthesis method will be demonstrated and discussed. In Section VII, the concluding remarks and future works will be given.

II. Related Works

Many researchers have proposed 3D-DRAM architectures for DRAM-stacked homogeneous CMPs. Loh proposed *true 3D* architecture in [1] which aggressively partitions the DRAM arrays and uses very large number of TSVs to maximize the bandwidth benefit of 3D-stacked environment. The author evaluated his method in general-purpose quad-core environment for multi-programmed workloads focusing on the overall throughput. Woo *et al.* proposed *SMART-3D* architecture to tackle the performance of single-threaded workloads as well as the overall throughput in [2]. Woo *et al.* also proposed heterogeneous 3D-DRAM architecture in [3] which tightly integrates SRAM row caches with DRAM arrays to improve performance and energy efficiency. However, all of these works did not consider various requirements of heterogeneous CMPs and did not consider the various array partitioning of DRAM.

In [4], Tsai *et al.* presented design space exploration for 3D-stacked SRAM cache with the consideration of various implementation choices for SRAM arrays. They showed that different array partitioning can have significant effect on delay and energy. However, besides the fact that they focused on SRAM not DRAM, they did not consider the asymmetric architecture of memory dies unlike our work.

There have been several works for exploring the design space of memory system. The authors of [13], [14] presented extensive optimization methods covering software-level memory optimizations and memory system organizations. In their methods, the number of ports and the word size of each memory can be explored as well as the number of memories. In [15], Pasricha *et al.* proposed a co-design method of on-chip communication architecture and memory system. Their work reduces the number of on-chip memories by merging the data blocks considering their access patterns. They also tries to select the proper type of memories among DRAM, SRAM, eDRAM, and EEPROM for the memory blocks. In [5], Weis *et al.* demonstrated design space exploration results for 3D-DRAM. They evaluated four different array partitioning of 128Mbit bank for several technology nodes. This work was extended by Gomony *et al.* in [16] which proposed a selection method for the type and array configuration of DRAM. Specifically, the authors in [16] proposed a framework which selects the appropriate DRAM among LPDDR, LPDDR2, and 3D-DRAMs of several configurations for complex memory requirements of heterogeneous CMPs. However, all of these works considered only a limited set of memory configurations and, above all, they considered only the *symmetric* architecture for the memory.

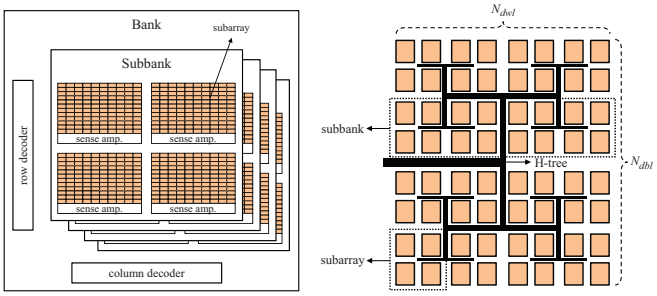
Compared to the previous works, the proposed work has the following contributions:

- We propose an asymmetric 3D-DRAM architecture in which the DRAM die is divided into multiple segments. Each segment is optimized for different requirements of heterogeneous cores. Since the differentiation of the segments are done in the layout level, the proposed architecture does not need any change in current 3D-DRAM fabrication process.
- We propose an automatic synthesis method for the asymmetric 3D-DRAM architecture. It finds the configurations of all the DRAM segments, such as numbers of banks and ports, I/O width, and array partitioning, which minimize the power consumption and area of the 3D-DRAM while satisfying the various memory requirements of the heterogeneous cores.

Meanwhile, a drawback of the proposed architecture is that there can be white space on the die due to the asymmetric segmentation (as shown in Fig. 4b). However, this white space can be used for *through-DRAM TSVs*. As tackled in [17], there necessarily exist a bunch of through-DRAM TSVs in DRAM-stacked processors since the DRAM tiers are located between the processor tier and the package substrate. Many power and signal I/Os should be delivered from the package to the processor tier through the DRAM tiers. By using the white space for through-DRAM TSVs, the drawback of the proposed architecture can be mitigated.

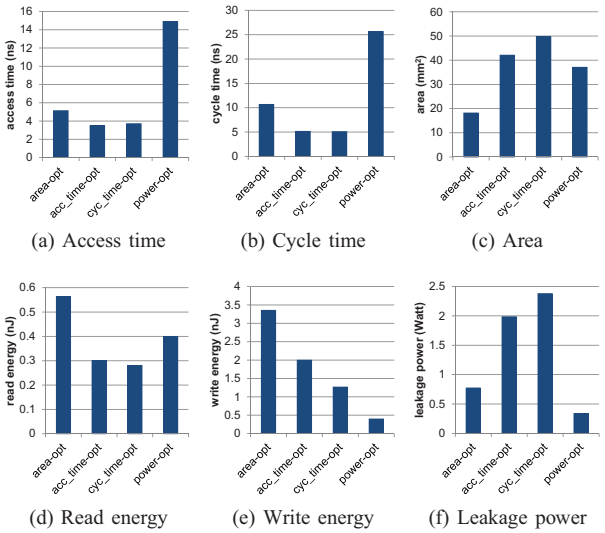
III. Background

In this section, the information needed to understand this work will be given. Throughout the paper, we adopt the basic architecture and terms for DRAM used in [18]. Fig. 2 shows the basic architecture of DRAM. In modern DRAMs, a bank is the basic unit of independent DRAM operation. A bank consists of one or more subbanks, as shown in Fig. 2a, and one subbank is read or written during each memory access to the bank. A subbank is divided into multiple subarrays and each subarray returns or stores a portion of the requested data. A subarray consists of arrays of the memory cells and the supporting peripherals such as sense amplifiers and column multiplexors. Fig. 2b



(a) Logical view of a DRAM bank (b) Layout view of a DRAM bank

Fig. 2: The architecture of a DRAM bank.



(a) Access time (b) Cycle time (c) Area

(d) Read energy (e) Write energy (f) Leakage power

Cap.	Bank	Port	Page size	Burst len.	Prefetch	Tech. node
128MB	1	8 x 1	8KB	8	8	32nm

(g) Given configurations for the DRAM

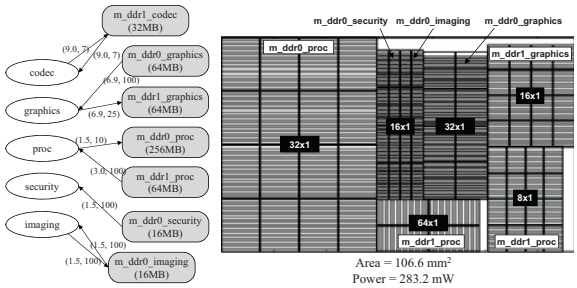
Fig. 3: Various implementations of a DRAM and their characteristics.

shows layout view of the DRAM in Fig. 2a. The entire DRAM array is partitioned into plenty of identical smaller arrays in order to reduce the latency and the arrays are connected via H-tree. Typically but not generally, the latency decreases and the area increases as dividing the entire array into more pieces.

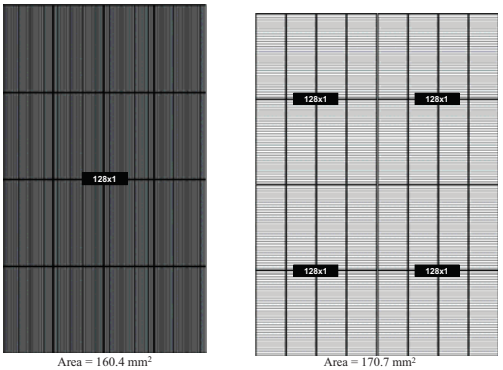
Fig. 3 shows how widely the characteristics of a DRAM can vary depending on the array partitioning. The values shown in Fig. 3a to 3f are obtained by *Cacti 6.5* [19] for the DRAM described in Fig. 3g, with the different objectives represented by the x-axis. The results show that, for instance, the DRAM optimized for the power consumption has 2.0 times larger area than the one optimized for the area. On the other hand, the DRAM optimized for the area consumes 8.5 times more energy for a write operation and 2.3 times more leakage power than the one optimized for the power consumption. Likewise, there exist plenty of trade-off points for a DRAM, but most commercial DRAMs have been optimized mainly for the cost-per-bit (i.e. area).

IV. Motivational Examples

Fig. 1 compares the proposed asymmetric 3D-DRAM architecture with the conventional ones, where the four types of cores impose conflicting requirements on the DRAM as described in Fig. 1d.



(a) Core-DRAM communication graph (b) Proposed



(c) Traditional (d) Mimic of JEDEC Wide I/O

Fig. 4: Comparison of 3D-DRAM architectures obtained for `mobile_ap×30`.

Fig. 1a represents a traditional 3D-DRAM in which the design of the DRAM die is almost unchanged from the 2D DRAM. In this architecture, the DRAM is connected to the processor via the central memory interface. This architecture may have the smallest area but is not adequate for the core types ② and ④ which have high bandwidth requirements and large memory level parallelism (MLP), since all the memory accesses should compete for the central memory interface. The *symmetrically segmented* 3D-DRAM architecture, shown in Fig. 1b, may improve the performance of the memory accesses with large MLP. Recently released JEDEC Wide I/O SDRAM [20] basically falls into this category while it constrains some logical and physical configurations.[2] This architecture is suited rather for homogeneous CMP environment where the cores have similar or arbitrary memory requirements than for heterogeneous CMP environment. For example, if the entire DRAM arrays are optimized to high bandwidth, the tight latency requirements of the core types ③ and ④ may not be satisfied. Many throughput-oriented 3D-DRAM architectures in [1]–[3] may also have the same problem due to their symmetric architectures. Unlike the 3D-DRAM in Fig. 1b, the DRAM is divided into *asymmetric segments* optimized for the serving cores in the proposed architecture, as shown in Fig. 1c. For example, for the core types ③ and ④ which run latency-oriented workloads, the DRAM arrays are partitioned in finer-grained compared to the other segments. For the core type ② which requires relatively loose latency, coarser-grained array partitioning is used, but multiple ports are used to support the high bandwidth requirements and the large MLP.

[2]It specifies the number of segments to four, the data width to 128b per channel, and the location of the interfaces to the center of the chip.

TABLE I: Parameters for a DRAM segment.

High level parameters		Array parameters	
Symbol	Description	Symbol	Description
C	memory capacity (e.g. in MB)	N_{dwl}	number of divisions of a wordline in a bank
N_{bank}	number of banks	N_{dbl}	number of divisions of a bit-line in a bank
N_{port}	number of ports	N_{dcm}	degree of muxing at bitline
W_{data}	width of data I/O	N_{dsam1}	degree of level-1 muxing at sense amp.
W_{page}	page size	N_{dsam2}	degree of level-2 muxing at sense amp.
W_{pref}	prefetch width		
BL	burst length		

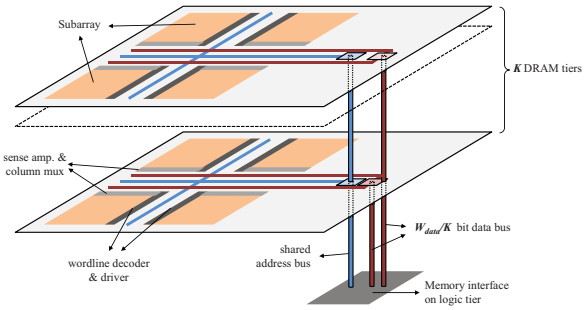


Fig. 5: The architecture of a subbank in the proposed 3D-DRAM when multiple DRAM tiers are stacked.

Fig. 4 shows the results for one of the benchmarks used in our experiments. Fig. 4a shows the memory requirements of the cores where the ovals represent the cores and the gray rectangles (with rounded corners) the memory regions (see Section V-B for detail). An edge is annotated with the required bandwidth in GB/s and the latency constraint in *ns*, in the first and the second places in the parenthesis, respectively. Fig. 4c, 4d, and 4b are the obtained DRAM layouts based on the architectures in Fig. 1a, 1b, and 1c, respectively. Fig. 4d is a mimic of JEDEC Wide I/O SDRAM where the number of segments (or channels), the number and the width of the ports are fixed to those specified in the standard (see Section VI-A for detail). Compared to the proposed architecture, the ones in Fig. 4c and 4d suffer from large area and power consumption since either the entire DRAM chip or all the segments should support the tightest latency constraint (7*ns*) and the highest bandwidth requirement (9GB/s) (a quarter of 9GB/s for each segment in Fig. 4d). On the other hand, in the proposed architecture, the segments are optimized for different memory requirements of the heterogeneous cores. As a result, both the area and the power consumption can be significantly reduced, as shown in Fig. 4b.

V. Asymmetric 3D-DRAM Synthesis

A. Preliminaries and Assumptions

A segment can have independent configuration parameters listed in Table I. In this work, we consider only the bi-directional read-write port since it is observed during the experiments that considering exclusive read/write port hardly improves the solution while increasing the synthesis time considerably.

We also assume that there is no direct connection between the different segments on the DRAM die and, therefore, a core which tries to access a segment must first access the memory interface connected to the segment via the on-chip network, similarly to [7], [10]. We also assume that the software-level memory optimizations to reduce the memory sizes and accesses are already done prior to our synthesis process by, for example, the methods in [13], [14].

As for the case where more than one DRAM tiers are stacked, we adopt the *coarse-grained 3D array partitioning* in [6] with an exception; in [6], a TSV bus interface is located at the center of each 3D subbank which consists of multiple stacks of 2x2 subarrays, so that the H-tree delay can be minimized. However, this may result in too many TSVs especially when the array is partitioned in very fine-grained. Therefore, in our architecture, this constraint is relaxed and one TSV bus interface is located at the center of each DRAM segment so that it can serve multiple subbanks in the segment. Note that, in the multiple tier case, a segment consists of multiple *segment slices* across all the tiers. The subbank architecture for the multiple tier case is shown in Fig. 5. Each tier takes an evenly distributed portion of the requested data, i.e. W_{data}/K bits, and the data is read from or written to the identical planar location of each tier. Therefore, the same address is broadcast to all the tiers while the separate data TSVs should be provided for each tier.

Meanwhile, we use the product of area and power consumption (area-power-product) as the cost function of the synthesis. The reason is that, as mentioned in Section I, the cost-per-bit cannot be the only criteria of DRAM anymore in modern systems, but the power consumption is as important as the cost-per-bit. We also assume *closed page* policy for DRAM since it has been reported in [11] that the row buffer hit rate decreases as the number of cores increases.

B. Problem Definition

Our synthesis process takes the followings as inputs:

- A *core-DRAM communication graph (CDCG)* is a directed graph $CDCG = G(V, E)$. A vertex $v \in V$ denotes either a core or an *abstract memory region (AMR)*. We will denote the set of vertices of the first type as V_{core} and that of the second type as V_{AMR}. An edge $e(v_i, v_j) \in E$ denotes the communication from v_i to v_j where v_i and v_j are of different types.
- An AMR has an attribute indicating its capacity, denoted as $Cap(v)$. An edge has attributes $BW_{wr}(e)$ and $Lat_{wr}(e)$ which are respectively the required bandwidth (e.g. in MB/s) and the latency constraint (e.g. in *ns*) for write operation. Similarly, $BW_{rd}(e)$ and $Lat_{rd}(e)$ exist for read operation.
- The number of DRAM tiers K.
- The DRAM technology-related information, such as technology node (e.g. 32*nm*) and cell type (e.g. high performance or low standby power).

Our synthesis problem is defined as follow; **Given** the input information listed above, **find** the best mapping of n AMRs to m DRAM segments, where $m \leq n$, and the best configuration parameters for each segment, **such that** the area-power-product of the 3D-DRAM is minimized while satisfying all the requirements in the CDCG.

C. Overview of the Synthesis Process

Our synthesis process consists of two phases; in the first phase (let say Phase-I), it assigns a dedicated segment to each AMR and finds the best configuration for each segment (left half of Fig. 6). The resulting DRAM from Phase-I can be badly fragmented due to a large number of segments, with large amount of white space and poor area efficiency. In order to improve area efficiency, the second phase (let say Phase-II) iteratively merges two AMRs into a new AMR, and finds the best segment again for the new AMR, until the given iteration count is reached (right half of Fig. 6). Both Phase-I and Phase-II employ the procedure `GA-AMRtoSEG` as their core

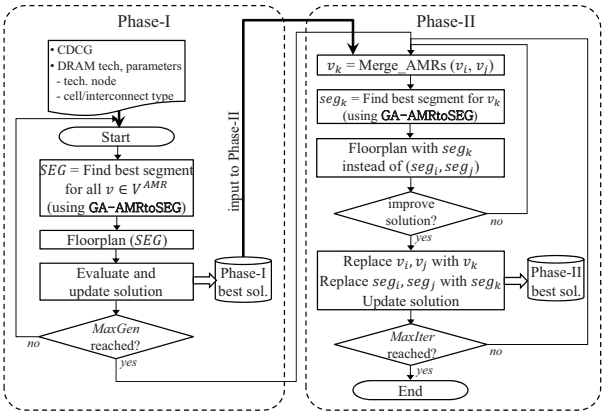
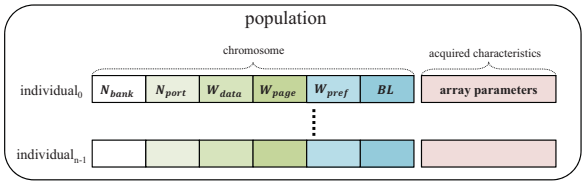
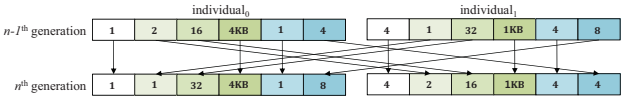


Fig. 6: The overview of the synthesis process.



(a) Encoding



(b) Crossover

Fig. 7: The proposed GA encoding and crossover strategies.

which realizes an AMR to a DRAM segment based on the genetic algorithm (GA).

D. Finding Best Physical DRAM Segment for an AMR

Before explaining Phase-I and Phase-II, we first explain the procedure `GA-AMRtoSEG` which plays core role in our synthesis process; it realizes an AMR into a physical DRAM segment by obtaining the best segment configuration for the AMR using the GA. Fig. 7 shows our encoding and crossover strategies for the GA. A DRAM segment is represented by an *individual*; an individual is identified by its *chromosome* and *acquired characteristics*, as shown in Fig. 7a. The chromosome contains relatively high level information for a DRAM segment while the acquired characteristics contain the parameters related to the array partitioning. At a certain generation, the individuals in the population represent various segment configurations for the corresponding AMR. They are evaluated for their *fitness* (area-power-product in this work), and the individual having the best fitness survives to the next generation, while the rest individuals are replaced with new individuals produced by *crossover* (Fig. 7b). As generations pass by, only the chromosomes are inherited generation to generation while the acquired characteristics are indeed *acquired* depending on the weighting coefficients for delay, area, and energy, which vary adaptively over generations.

The pseudo-code of `GA-AMRtoSEG` is shown in Algorithm 1. The procedure takes an AMR v and the violation ratios of access time and cycle time for the AMR in the previous generation (r^v_{acc} and r^v_{cyc}, respectively) as its inputs, and outputs the best segment configuration parameters. r^v_{acc} (r^v_{cyc}) is defined as the number of individuals in the population which cannot meet the latency (bandwidth) constraint imposed on v, divided by the number of individuals in the population.

Algorithm 1 GA-AMRtoSEG

Input: AMR v, r^v_{acc}, r^v_{cyc}
Output: best segment $best\_seg$
1: produce_population POP;
2: W_v = renew_weights(r^v_{acc}, r^v_{cyc});
3: **for** $ind \in POP$ **do**
4: seg = find_best_array(ind, W^v, $Cap(v)/K$);
5: evaluate_segment(seg);
6: **if** seg is the best segment so far **then**
7: $best\_seg = seg$;
8: **end if**
9: **end for**

It starts by producing $MaxPop$ individuals into the population POP (line 1), where $MaxPop$ is a user-defined parameter indicating the number of individuals in a population. After that, the weighting coefficients for the access time (w^v_{acc}), cycle time (w^v_{cyc}), area (w^v_{area}), and energy (w^v_{energy}) are renewed based on r^v_{acc} and r^v_{cyc}. The set of these weighting coefficients is denoted as W^v. Specifically, w^v_{acc} and w^v_{cyc} are increased (decreased) by a user-defined parameter δ if the corresponding violation ratios are greater (smaller) than another user-defined parameter α.

From lines 3 to 9, the best segment for every individual is obtained and evaluated, and the best segment for the AMR v is finally chosen for the output. At line 4, the best array configuration of the individual (i.e. the 'acquired characteristics' of the individual) is obtained by using *Cacti 6.5*, which is a widely used optimization tool for SRAM and DRAM. The capacity of the AMR v, the chromosome of the individual, and the weighting coefficients are used as the inputs to Cacti, and then Cacti outputs the best array parameters and the physical characteristics of the obtained array, e.g. height, width, access time, cycle time, read/write energy, etc. Since we assume the coarse-grained partitioning shown in Fig. 5, all the segment slices in K tiers have an identical shape. Therefore, the array parameters for only one segment slice is obtained by inputting $\lceil Cap(v)/K \rceil$ and $\lceil W_{data}/K \rceil$ to Cacti and the remaining segment slices are just duplicates of the obtained segment slice.

At line 5, the obtained segment is evaluated for its feasibility (i.e. whether all the memory requirements are met) and cost (i.e. area-power-product). The following inequality should hold in order for the segment to satisfy the latency constraint.

$$\forall e \text{ incident to } v,$$
$$t_{acc} \leq min(Lat_{rd}(e), Lat_{wr}(e)) \quad (1)$$

where t_{acc} is the random access time of the segment.

Checking bandwidth constraint is more complicated since it is influenced by many parameters of DRAM. For example, longer burst length may reduce the effective random cycle time by reducing the frequency of opening new rows. Also, larger number of banks may reduce the effective cycle time by increasing the chance of bank interleaving. Assuming that the chance of bank interleaving is proportional to the number of banks, we calculate the effective cycle time, t_{eff_cyc}, as follows;

$$t_{eff\_io\_clk} = \frac{t_{arr\_io}}{W_{pref}} \quad (2)$$

$$t_{eff\_cyc\_no\_il} = \frac{t_{cyc} + t_{eff\_io\_clk} \times (BL-1)}{BL} \quad (3)$$

$$t_{eff\_cyc} = \frac{1}{N_{bank}} \times t_{exp\_cyc\_no\_il} + (1 - \frac{1}{N_{bank}}) \times t_{il\_cyc} \quad (4)$$

Algorithm 2 Phase-I

Input: $CDCG$
Output: $best\_SEG\_p1$
1: **for** $v \in V_{AMR}$ **do**
2: initialize W^v, r^v_{acc}, r^v_{cyc};
3: **end for**
4: **for** $g = 1 \rightarrow MaxGen$ **do**
5: $SEG = \emptyset$;
6: **for** $v \in V_{AMR}$ **do**
7: $best\_seg$ = GA-AMRtoSEG(v, r^v_{acc}, r^v_{cyc});
8: $SEG = SEG \cup \{best\_seg\}$;
9: **end for**
10: floorplan(SEG);
11: evaluate_curr_solution(SEG);
12: **if** SEG is the best solution so far **then**
13: $best\_SEG\_p1 = SEG$;
14: **end if**
15: $\{r^v_{acc}, r^v_{cyc}\}$ = update_violation_ratio;
16: **end for**

Algorithm 3 Phase-II

Input: $best\_SEG\_p1$
Output: $best\_SEG\_p2$
1: $best\_SEG\_p2 = best\_SEG\_p1$;
2: **while** $iter \leq MaxIter$ **do**
3: $\{v_i, v_j\}$ = choose_two_AMRs;
4: v_k = merge_AMRs(v_i, v_j);
5: seg_k = GA-AMRtoSEG(v_k);
6: $SEG = \{best\_SEG\_p2 \backslash \{seg_i, seg_j\}\} \cup \{seg_k\}$
7: floorplan(SEG);
8: **if** current merge improves solution **then**
9: $best\_SEG\_p2 = SEG$;
10: **end if**
11: $iter++$;
12: **end while**

$t_{eff\_io\_clk}$ is the effective clock period at the I/O side and t_{arr_io} is the delay between data I/O and the row buffer of the DRAM array. Since I/O clock period longer than t_{acc_io} will result in underutilized DRAM performance in burst transfer, we assume that t_{acc_io} determines the I/O clock period in single data rate (SDR) transfer. When double data rate (DDR) transfer with $W_{pref} > 1$ is used, the effective clock period at the I/O is reduced by W_{pref}. $t_{eff\_cyc\_no\_il}$ is the effective random cycle time when burst transfer is considered but bank interleaving is not, which is derived from the random cycle time of the segment (t_{cyc}) and $t_{eff\_io\_clk}$. In turn, the effective cycle time of the segment t_{eff_cyc} is derived from $t_{eff\_cyc\_no\_il}$ and the interleave cycle time t_{il_cyc}. Finally, we calculate the effective bandwidth of the segment BW_{eff} as follow.

$$BW_{eff} = \frac{W_{data} \times N_{port}}{t_{exp\_cyc}} \tag{5}$$

Then, the following inequality should hold to meet the bandwidth constraint.

$$BW_{eff} \geq \sum_{e \text{ incident to } v} (BW_{rd}(e) + BW_{wr}(e)) \tag{6}$$

E. Phase-I and Phase-II

The detailed algorithms of Phase-I and Phase-II are explained in this subsection. First, the pseudo-code of Phase-I is shown in Algorithm 2. The algorithm starts by initializing the weighting coefficients to the same positive value (0.25 in this work) and the violation ratios to zero for all AMRs (lines 1 to 3). From line 5 to 9, the best segments for all the AMRs are obtained by using `GA-AMRtoSEG` and added to the set of the best segments (SEG). After all the AMRs are realized into the physical DRAM segments, next step is to locate the segments in the planar DRAM die such that the die area is minimized, namely, to *floorplan* the segments (line 10). We use *DeFer* [21] to floorplan the segments. With the final area obtained by the floorplanning, the area-power-product of the current generation is calculated and the best solution is updated through lines 11 to 14. The violation ratios are updated at line 15, and the procedure from line 5 to 15 is repeated until the generation count reaches $MaxGen$.

The result of Phase-I is the input of Phase-II. The purpose of Phase-II is to alleviate the possible area inefficiency of Phase-I due to the segmentation by iteratively merging the AMRs. The pseudo-code of Phase-II is shown in Algorithm 3. At the beginning of the loop body (line 3), two AMRs to be merged are selected. The AMRs are basically selected randomly but a record of the tried merges are used to avoid selecting the same pair of AMRs repeatedly. The selected AMRs, let say v_i and v_j, are merged to a new AMR v_k, and the best segment configuration for v_k, let say seg_k, is obtained by using `GA-AMRtoSEG` (lines 4 and 5). After that, floorplanning is performed with the set of segments where the segments corresponding to v_i and v_j are replaced by seg_k (lines 6 and 7). If the current merge improves the area-power-product, the best set of segments are updated (lines 8 to 10), and the above procedure is repeated until the iteration count reaches a user-defined parameter $MaxIter$.

TABLE II: Benchmarks description.

Name	$\lvert V_{core}\rvert$	$\lvert V_{AMR}\rvert$	Tot.Cap. (MB)	Tot.BW (MB/s)	Description
mpeg4decoder	7	7	512	3473	MPEG-4 decoder [22]
mobile_ap	5	7	512	1360	mobile application processor [23]
mobile_mmp	4	8	512	1106	mobile multimedia player SoC [23]
game_soc	18	13	1024	9190	game SoC [23]

VI. Experiment

A. Experiment Setup

We evaluated the proposed architecture and the synthesis method by applying them to the benchmarks listed in Table II. The second and the third columns of Table II are the numbers of cores and AMRs, respectively, and the forth and the fifth columns are the total capacity and the sum of bandwidth requirements, respectively. We obtained the CDCGs of the benchmarks basically by extracting core- and memory-related parts from the communication graphs (CGs) given in [22] and [23]. More specifically, from the original CGs, the processing elements such as CPUs, GPUs, VLIW processors, and some of ASIC blocks are mapped to $v \in V_{core}$ of the corresponding CDCGs. Similarly, the memory elements such as DRAMs, SRAMs, and eDRAMs in the original CGs are mapped to $v \in V_{AMR}$ of the CDCGs. When a memory is accessed by multiple cores in the original CG, we divided the memory into multiple AMRs for some cases so that each part of the memory can be optimized to its serving cores. The CPUs in the benchmarks represent the latency-oriented cores which require relatively tight latency, moderate bandwidth, and large memory size, thus correspond to the type ① in Fig. 1d. GPUs and VLIW processors represent the cores of the type ②. The cores which typically execute real-time tasks, e.g. `codec` in `mobile_ap` and `mobile_mmp`, are assumed to have very tight latency constraint and high bandwidth requirement, corresponding to the type ④. Some ASIC blocks are assumed to have the characteristics of the type ③.

Since the sizes of the memories are unavailable in the original CGs, we assigned the sizes of the AMRs on our own discretion

TABLE III: Parameter settings used in the experiments.

Cacti parameters					
Tech. node	Cell type		Peri. type		Temp.
32nm	commodity dram		high performance		350K
Synthesis parameters					
Num.tiers (K)	$MaxGen$	$MaxPop$	$MaxIter$	α	δ
1	15	15	30	0.3	0.1

by the following rules: 1) the AMRs communicating mainly with CPUs are considered as main memories, thus relatively large sizes (128 to 256MB) are assigned. 2) The AMRs communicating mainly with GPUs or VLIW processors are assigned sizes between 64 to 128MB, and those communicating with other cores are assigned small sizes around 8 to 16MB. Total memory size is assumed as 512MB for `mpeg4decoder`, `mobile_ap`, and `mobile_mmp`, and 1GB for `game_soc`. Also, we assigned the latency constraint values for `mpeg4decoder` by ourselves since they are unavailable in [22]. We also modified the latency values for the rest three benchmarks to make the problems more challenging.[3] In addition, since the bandwidth requirements of the original benchmarks are actually not challenging when considering the bandwidth provided in modern DRAMs,[4] we multiply the bandwidth requirements of the original benchmarks by *bandwidth scale factor*, denoted as *BWx*. The derived benchmark by multiplying *BWx* will be denoted with the suffix $\times$*BWx* next to the original benchmark name.

We compare the proposed architecture to the following ones:

- **Central** represents the traditional 3D-DRAM architecture in Fig. 1a.
- **Dist-*n*** represents the symmetrically segmented 3D-DRAM architectures in Fig. 1b where each segment is accessed via its dedicated memory interface, and *n* is the number of segments.
- **Dist-JEDEC** is a variant of **Dist-4** in which the number of ports and the data width are forced to those specified in JEDEC Wide I/O standard to mimic it.

Note that all the results of these compared methods are also obtained by our synthesis method but with their architectural limitations considered. For **Dist-*n*** and **Dist-JEDEC**, we assume that the requests are evenly distributed over the segments. We conducted our experiment on a virtual machine with 2 cores and 1GB main memory, of which the host machine is a 2.8GHz Core i5-based PC. The values of the given parameters used in the experiments are summarized in Table III.

B. Experimental Results

Fig. 8 shows the normalized area-power-products obtained for the proposed architecture and the compared ones, while varying *BWx* from 1 to 30. The values are normalized to those of **Central**. At the first glance, **Dist-JEDEC** shows very poor results for all the benchmarks when *BWx* is small. The reason is that, when the bandwidth requirement is low, the 128b-wide I/O width for all the four segments can be very large overhead for both area and power consumption. Note that the large area incurred by the wide I/O width accompanies increased delay due to longer wire length, and it in turn results in increased power consumption due to more aggressive repeater insertion and larger drivers to meet the latency constraint. For this reason, the overhead of **Dist-JEDEC** is alleviated as the required bandwidth increases (i.e. as *BWx* increases).

On the contrary, **Central** shows the best results for all the benchmarks when *BWx*=1 but becomes worse as the required bandwidth increases. Since the only memory interface should cope with all of the memory requests, it experiences the highest pressure for the bandwidth increase among the compared architectures. As a result, the wider I/O width and more banks are required for **Central** to meet the bandwidth requirements than the other architectures, as the required bandwidth increases.

The proposed architecture shows significant reductions in area-power-product for all the benchmarks, especially when the required bandwidth is high. Compared to **Central**, the largest reduction is 94.4% achieved for `game_soc`$\times$`25`, and the average reduction over all the benchmarks and *BWx*'s is 48.2%. Compared to the symmetric architectures, i.e. **Dist-*n*** and **Dist-JEDEC**, the reduction in area-power-product is up to 91.7% (achieved for `game_soc`$\times$`30`) and 69.4% on average.

We also conducted experiments while increasing the capacity of the DRAMs by double ($\times$2) and quadruple ($\times$4) for the benchmarks with *BWx*=30.[5] Note that this experiment indirectly shows the effect of more tightened latency constraint since the delay of DRAM typically increases as the capacity increases. The results are shown in Fig. 9 where the values are normalized to those of $\times$1 of **Central**. First of all, the results show that **Central** fails to find the valid solutions for `mpeg4decode`$\times$`30` with quadrupled capacity and `game_soc`$\times$`30` with doubled capacity. This infers the limitation of the traditional 3D-DRAM architecture on the latency-oriented tasks, showing that the just increasing the capacity of DRAM can make their constraint violated. The symmetric 3D-DRAM architectures, i.e. **Dist-*n*** and **Dist-JEDEC**, manage to find valid configurations but the area-power-products dramatically increase as the capacity increases. The proposed asymmetric 3D-DRAM architecture again shows significant area-power-product reductions with the increased capacity. When the capacity is doubled, the average reductions (for successful cases) are 63.0% and 75.5% compared to **Central** and the symmetric architectures, respectively.

The synthesis time is dependent mainly on the number of invocations of Cacti, thus almost linearly proportional to $MaxPop \times MaxGen \times |V_{AMR}|$. In our experiments, the longest synthesis time is about 27 minutes (for `game_soc`$\times$`30`).

VII. Conclusions and Future Works

Based on the observation that the desired properties for the cores vary widely among applications, heterogeneous CMP is gaining its popularity. In this work, we propose the asymmetric 3D-DRAM architecture for heterogeneous CMPs and the automatic synthesis method for it, by tackling the inadequateness of the previous 3D-DRAM architectures in heterogeneous CMP environment. The experimental results support our idea by showing enormous reductions in area-power-product, where the reductions are more than 60% (conservatively) for most of the test cases. At this time, our work has limitations on capturing and exploiting the properties of memory accesses; the MLP is not considered when evaluating the adequateness of the DRAM segments. Also, the shared memory

[3]More specifically, very tight latency constraint (7*ns*) is assigned to the communications with the AMRs originated from SRAMs. The communications between CPUs and DRAMs are assigned tight latency constraint (10*ns*). The communications between other cores and DRAMs are assigned relatively loose latency constraints from 25 to 40*ns*.

[4]For example, Intel's contemporary PC platform (LGA2011) already supports the peak DRAM bandwidth of 42.7GB/s (assuming DDR3-1333, 1.333GT/s$\times$8Bytes/channel$\times$4-channels) while the highest total required memory bandwidth among the benchmarks is 9.2GB/s in `game_soc`.

[5]The experiments for `game_soc` with quadrupled capacity could not be done due to the limitation of Cacti 6.5 that it cannot support 4GB of memory.

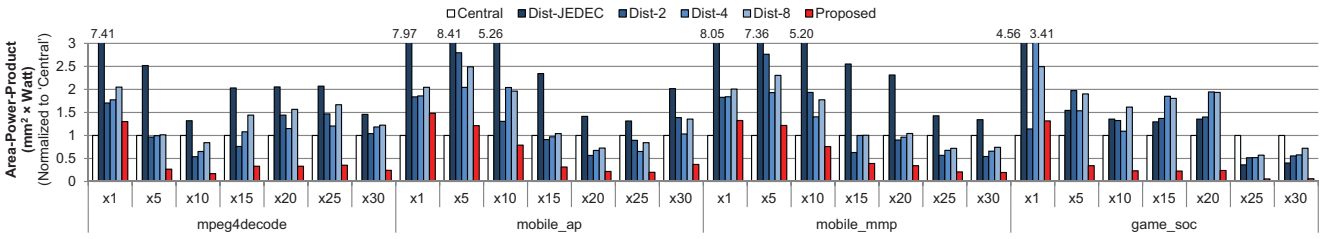


Fig. 8: Comparison of the area-power-products for various bandwidth requirements. The values are normalized to those of **Central**.

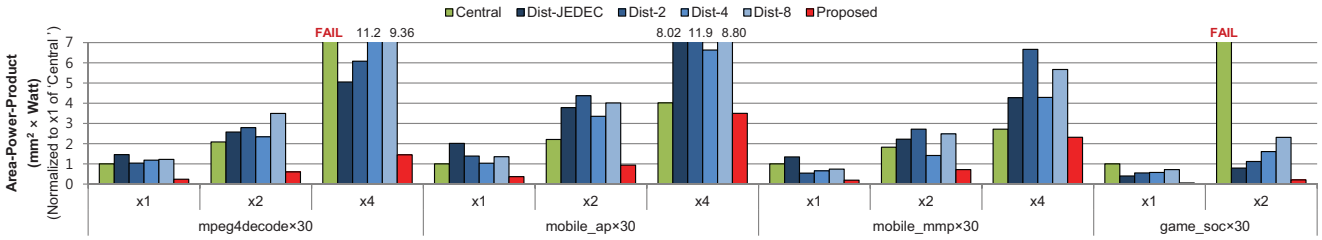


Fig. 9: Comparison of the area-power-products for various capacities. The values are normalized to those of $\times 1$ of **Central**. This indirectly shows the effect of more tightened latency constraints.

regions for data exchange among cores need to be considered more sophisticatedly. We plan to solve those problems to improve the work.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0026454 and No. 2011-0027625).

References

[1] G. Loh, "3d-stacked memory architectures for multi-core processors," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 453–464.

[2] D. Woo, N. Seong, D. Lewis, and H. Lee, "An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010, pp. 1–12.

[3] D. Woo, N. Seong, and H. Lee, "Pragmatic integration of an sram row cache in heterogeneous 3-d dram architecture using tsv," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, no. 99, pp. 1–13.

[4] Y. Tsai, F. Wang, Y. Xie, N. Vijaykrishnan, and M. Irwin, "Design space exploration for 3-d cache," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 4, pp. 444–455, 2008.

[5] C. Weis, N. Wehn, L. Igor, and L. Benini, "Design space exploration for 3d-stacked drams," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[6] H. Sun, J. Liu, R. Anigundi, N. Zheng, J. Lu, K. Rose, and T. Zhang, "3d dram design and application to 3d multicore systems," *Design & Test of Computers, IEEE*, vol. 26, no. 5, pp. 36–47, 2009.

[7] I. Loi and L. Benini, "An efficient distributed memory interface for many-core platform with 3d stacked dram," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 99–104.

[8] J. Meng and K. Skadron, "A reconfigurable simulator for large-scale heterogeneous multicore architectures," in *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 119–120.

[9] C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, "Bridging the processor-memory performance gap with 3d ic technology," *Design & Test of Computers, IEEE*, vol. 22, no. 6, pp. 556–564, 2005.

[10] A. Marongiu, M. Ruggiero, and L. Benini, "Efficient openmp data mapping for multicore platforms with vertically stacked memory," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 105–110.

[11] A. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. Jouppi, "Rethinking dram design and organization for energy-constrained multi-cores," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 175–186.

[12] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*. USENIX Association, 2010, pp. 21–21.

[13] P. Slock, S. Wuytack, F. Catthoor, and G. De Jong, "Fast and extensive system-level memory exploration for atm applications," in *System Synthesis, 1997. Proceedings., Tenth International Symposium on*. IEEE, 1997, pp. 74–81.

[14] A. Vandecappelle, M. Miranda, E. Brockmeyer, F. Catthoor, and D. Verkest, "Global multimedia system design exploration using accurate memory organization feedback," in *Design Automation Conference, 1999. Proceedings. 36th*. IEEE, 1999, pp. 327–332.

[15] S. Pasricha and N. Dutt, "Cosmeca: application specific co-synthesis of memory and communication architectures for mpsoc," in *Proceedings of the conference on Design, automation and test in Europe: Proceedings*. European Design and Automation Association, 2006, pp. 700–705.

[16] M. D. Gomony, C. Weis, B. Akesson, N. Wehn, and K. Goossens, "Dram selection and configuration for real-time mobile systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*. IEEE, 2012, pp. 1–6.

[17] Q. Wu, K. Rose, J. Lu, and T. Zhang, "Impacts of though-dram vias in 3d processor-dram integrated systems," in *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*. IEEE, 2009, pp. 1–6.

[18] S. Thoziyoor, "A comprehensive memory modeling tool for design and analysis of future memory hierarchies," Ph.D. dissertation, University of Notre Dame, 2008.

[19] Cacti. [Online]. Available: http://www.hpl.hp.com/research/cacti

[20] *Wide I/O Single Data Rate*, JEDEC Std. JESD229, 2011.

[21] J. Yan and C. Chu, "Defer: deferred decision making enabled fixed-outline floorplanning algorithm," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 3, pp. 367–381, 2010.

[22] E. Van Der Tol and E. Jaspers, "Mapping of mpeg-4 decoding on a flexible architecture platform," *Media Processors 2002*, vol. 4674, 2002.

[23] J. Yoo, S. Yoo, and K. Choi, "Topology/floorplan/pipeline co-design of cascaded crossbar bus," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 8, pp. 1034–1047, 2009.