3D-Stacked Memory-Side Acceleration: Accelerator and System Design

Qi Guo, Nikolaos Alachiotis, Berkin Akin, Fazle Sadi, Guanglin Xu, Tze Meng Low,
Larry Pileggi, James C. Hoe, and Franz Franchetti
Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA
Email: {qguo1, nalachio, bakin, fsadi, guanglix, lowt, pileggi, jhoe, franzf}@andrew.cmu.edu

Abstract—**Specialized hardware acceleration is an effective technique to mitigate the dark silicon problems. A challenge in designing on-chip hardware accelerators for data-intensive applications is how to efficiently transfer data between the memory hierarchy and the accelerators. Although the Processing-in-Memory (PIM) technique has the potential to reduce the overhead of data transfers, it is limited by the traditional process technology. Recent process technology advancements such as 3D-die stacking enable efficient PIM architectures by integrating accelerators to the logic layer of 3D DRAM, thus leading to the concept of the 3D-stacked Memory-Side Accelerator (MSA). In this paper, we initially present the overall architecture of the 3D-stacked MSA, which relies on a configurable array of domain-specific accelerators. Thereafter, we describe a full-system prototype that is built upon a novel software stack and a hybrid evaluation methodology. Experimental results demonstrate that the 3D-stacked MSA achieves up to 179x and 96x better energy-efficiency than the Intel Haswell processor for the FFT and matrix transposition algorithms, respectively.**

I. Introduction

As transistor density continues to grow exponentially, the limited power budget allows only a small fraction of active transistors, which is referred to as dark silicon [1]. Dark silicon forces us to trade silicon area for energy. Specialized hardware acceleration has emerged as an effective technique to mitigate the dark silicon, as it delivers up to several orders of magnitude better energy efficiency than general-purpose processors. Heading towards the big data era, a key challenge in the design of hardware accelerators is how to efficiently transfer data between the memory hierarchy and the accelerators, mainly when targeting emerging data-intensive applications (e.g., key-value store, graph database, etc.).

Processing-in-memory (PIM) [2], [3] can reduce the overhead on performance and energy consumption incurred by the data movement between memory and computational components. By integrating the computational logic into memory, PIM allows computation to occur where the data reside, leading to improved performance and reduced energy consumption. However, an obstacle to make PIM viable is that the computation implemented by the memory process is typically much slower than that manufactured by logic process [4].

Fortunately, the emergence of the 3D die-stacking technique, e.g., Micron's Hybrid Memory Cube (HMC) [5], is promising to make PIM more practical. The 3D die-stacking technique allows to integrate multiple process technologies (e.g., logic process and memory process) within a single stack by using through-silicon vias (TSV). Therefore, incorporating the hardware accelerator implemented by the logic process technology to logic dies of the memory stack leads to the concept of the 3D-stacked Memory-Side Accelerator (MSA).

This work was sponsored by DARPA under agreement HR0011-13-2-0007. The content, views and conclusions presented in this document do not necessarily reflect the position or the policy of DARPA.

To design and implement the 3D-stacked MSA system, there are several challenges to address. From the hardware perspective, a key challenge is how to design hardware accelerators that are adaptable to various applications, as the relatively specialized accelerators typically trade generality for efficiency. Another challenge is how to design efficient hardware accelerators that can saturate the excessively high memory bandwidth of 3D-stacked DRAM, given the limited area budget of the logic layer. From the software perspective, the entire software stack, including the operating system, the device driver, the library, and the user interface, should provide sufficient support for the programmer to easily and efficiently use the 3D-stacked MSA in order to achieve increased energy-efficiency.

Contribution. In this paper we present a full-system prototype to address the above challenges. More specifically, our work makes the following contributions:

1) *Overall architecture.* We present the overall architecture of 3D-stacked MSA systems to achieve energy-efficiency for data-intensive applications.
2) *Configurable accelerator array.* We design a hardware architecture to configure the array of multiple accelerators to adapt to various applications.
3) *Software stack.* We implement a novel software stack to make the 3D-stacked MSA easily accessible for the programmers.
4) *Full-system prototype.* We build a full-system prototype on a commodity machine with the help of a novel hybrid (in simulation and on real machine) evaluation methodology to accurately and efficiently evaluate the 3D-stacked MSA architecture.

Related work. Many specialized hardware accelerators have been proposed to accelerate various applications, e.g., [6], [7], [8], [9], [10]. Our solution can integrate and configure such accelerators in the 3D-stacked memory, and ultimately provide an easy-to-use interface for programmers. Recently, there are also several proposals on incorporating computation to the logic die of 3D-stacked DRAM. Pugsley et al. [11] integrated energy-efficient processor cores to the logic die of 3D-stacked DRAM for MapReduce workloads, and they

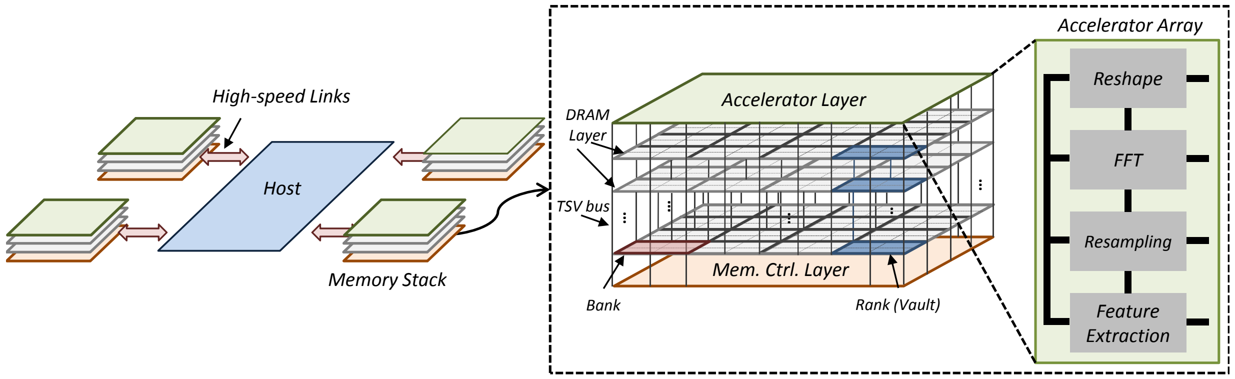

Fig. 1: The overall architecture of the proposed 3D-stacked MSA system.

further compared the benefits of different near-data computing approaches for in-memory MapReduce [12]. Zhang et al. [13] proposed to integrate programmable GPU to 3D-stacked DRAM to offer high throughput. However, to the best of the author's knowledge, our work is the first to build and evaluate a full-system 3D-stacked MSA architecture on commodity machines.

II. Overview of 3D-stacked MSA

Overview. 3D-stacking technology enables the integration of specialized accelerators and 3D DRAM. As shown in Figure 1, the entire 3D-stacked Memory-Side Accelerator (MSA) system consists of a central host processor and multiple memory stacks, and they are connected with high-speed links that are also used in the HMC system. Each memory stack consists of multiple dies, while each die could be a DRAM die or a logic die. The logic die can contain the memory controller, the accelerators, or both. Each layer communicates with others in the same memory stack via high-bandwidth, low-latency, and low-energy TSVs. As the off-stack data traffic is drastically reduced, an expected use of this architecture is to efficiently process emerging data-intensive applications that typically exhibit huge memory footprints.

Accelerator layer. The accelerator layer mainly consists of a configurable array of multiple domain-specific accelerators. These accelerators target various application domains such as signal processing, graph computing, machine learning, and others. Targeting signal processing applications for instance, the potential accelerators could include *Reshape* for data layout transformations (e.g. linear to blocked, row-major to column-major), *FFT (fast Fourier transform)* as a basic computation kernel [14], *Resampling* for data interpolation, geometric transformations and image alignment, and *Feature Extraction* for edge and shape detection of images. Such primitive accelerators can be further configured to accomplish complicated tasks. For instance, a Synthetic Aperture Radar (SAR) image formation algorithm typically requires Reshape, FFT, and Resampling accelerators [15]. To offer such reconfigurability to the users, we have carefully designed the hardware architecture of an array of accelerators as well as the required configuration interface.

III. The Configurable Accelerator Array

The overall design of the configurable array of accelerators is shown in Figure 2. In addition to the series of domain-specific accelerators, the MSA architecture contains all the necessary configuration logic to implement every possible datapath described by an accelerator descriptor.

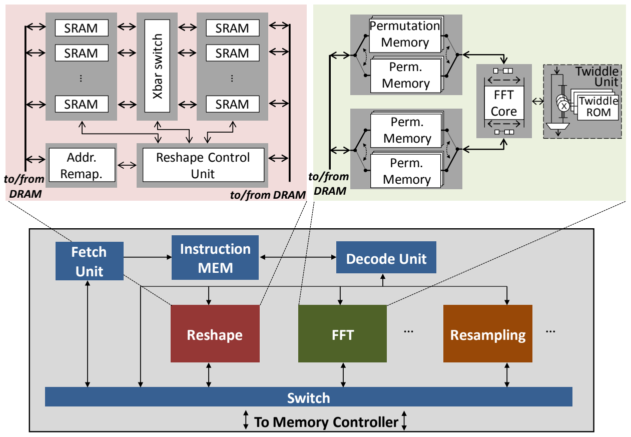

Fig. 2: The architecture of the configurable array of accelerators.

Accelerator descriptor. The accelerator descriptor typically consists of one or more passes. Each pass contains a series of processing instructions that all together describe a datapath. Each datapath is a complete processing operation that starts by retrieving input data from the main memory, implements a pipeline of the accelerators included in the pass, and stores the output data back to main memory. Additionally, the accelerator descriptor may contain flow instructions to allow the convenient implementation of iterative operations that contain one or more passes.

Architecture. When the host processor is ready to deploy the accelerator, it stores the accelerator descriptor in a pre-allocated memory space. When this happens, the *FETCH* unit transfers the entire descriptor to the local Instruction Memory (*IMEM*) and activates the *DECODE* unit. The DECODE unit parses the descriptor sequentially until the end of a pass is detected. For every instruction in the pass, the DECODE unit activates the corresponding accelerator and appropriately configures the switch logic at the input and output ports. When all accelerators in a pass are activated, the DECODE unit enables an accelerator initialization process during which each accelerator retrieves domain-specific configuration data from the main memory. When all accelerators in the current pass are individually configured, the DECODE unit initiates the processing phase. During processing, the very first accelerator in the pass fetches input data from main memory while the last accelerator stores the output data back to main memory. It is notable that the data stays on the accelerator die during processing. The DECODE unit monitors the status of the last accelerator in the pass to detect when the pass processing is over in order to proceed with the next pass in the descriptor.

IV. Software Design of 3D-stacked MSA

To make the 3D-stacked MSA easily accessible for programmers, we revisit the entire software stack at various levels.

OS kernel level. In a typical HMC system that contains multiple memory stacks, each stack may contain several MSAs. For a given MSA, the memory stack that it is integrated can be regarded as *Local Memory Stack* (LMS), but the external memory stacks are considered as *Remote Memory Stacks* (RMS). Hence an MSA accesses the entire memory space in an asymmetric way, which is conceptually similar to NUMA (Non-Uniformed Memory Access) systems. To efficiently use the accelerator, the data for processing should reside in the LMS rather than the RMS. Therefore, in the kernel, we reserve the contiguous memory region of the LMS for MSA, so that the upper-level memory management functions (which will be elaborated later) only allocate/free memory in the LMS.

LMS is also used for communication and synchronization between the CPU and the MSA. Therefore, we further divide the entire LMS to *command* and *data* space. The command space primarily contains the command address, the status address, and the configuration parameters of the employed accelerators. The command address is monitored by the hardware, and once special commands are written to that address, the MSA is invoked for processing the task. The status address stores the state of the MSA (e.g., busy or free), which can be retrieved by the CPU. The configuration parameters of used accelerators, e.g., chaining of accelerators, the addresses of the input/output data, the number of tiles, etc., are employed to configure the accelerator array.

Driver level. In the driver level, we implement a device driver, allowing the users to directly manipulate the physical memory space (i.e., command and data space) of LMS. During the installation of the device driver, the command space is allocated from the LMS, and then it is mapped to virtual memory space via *mmap*. Thus, the corresponding MSA can be directly controlled by writing commands to the mapped space. Regarding the data space, it is also allocated/freed through the device driver. More specifically, the device driver provides *ioctl* to process the memory allocation/free requests from upper-level library. After that, *mmap* is implemented in the device driver to map the allocated contiguous physical memory to virtual memory space. Therefore, the host and the MSA share the same virtual space, and the host is in charge of the virtual-to-physical translation to reduce the hardware cost.

Library level. We provide the *native* and the *wrapped* library. The native library provides a standard library for MSA control and LMS management. As stated, CPU controls the MSA through a pre-allocated *command* space. As shown in Figure 3, the users can leverage *msa_acc_plan* to generate the *accelerator descriptor* as introduced in Section III. Such accelerator descriptor is stored in a structure called *acc_plan*. Then, the users can invoke the accelerators with the same configuration many times by using *msa_acc_execute*, where the accelerator descriptor is written to the command space by a lightweight user-level *memcpy*. Finally, the accelerator descriptor can be destroyed by calling *msa_acc_destroy*. Additionally, the native library is also used for LMS management, and it works on the *data* space. It provides memory allocation (*msa_mem_alloc*) and free (*mea_mem_free*) functions, as shown in Figure 4.

```
// generate the accelerator descriptor
acc_plan msa_acc_plan(const char *tdl,
                      long *buf_addr,
                      long *buf_size,
                      long length);

// invoke the accelerators
void msa_acc_execute(acc_plan p);
// destroy the accelerator descriptor
void msa_acc_destroy(acc_plan p);
```

Fig. 3: The native functions working on *command* space.

```
// allocate memory from LMS
void *msa_mem_alloc(long size);
// free memory to LMS
void msa_mem_free(void *addr);
```

Fig. 4: The native functions working on *data* space.

The wrapped library aims to ease the programming burden of the users for specific applications. Assuming FFT for instance, since FFTW [16] is the most widely used interface, we implement a wrapper on the native library to provide the same interface as the FFTW. Therefore, programs written with the FFTW library can directly use MSA with only trivial modifications (e.g., by adding the MSA-related header files).

User level. In the user level, we provide a Task Description Language (TDL) to configure multiple accelerators for various applications. In the example shown in Figure 5, TDL offers 5 basic primitives, i.e., *component*, *pass*, *task*, *parallel*, and *loop*. The *component* block corresponds to a specific accelerator component (e.g., FFT, Reshape, Resampling, etc.). Multiple component blocks constitute the *pass* block, and each pass block has its own input and output data buffers. The *task* block contains multiple pass blocks, and it represents a basic computational task. The *parallel* block contains several computational tasks, and all these tasks can be executed simultaneously. The *loop* block indicates that all included tasks can be executed multiple times. In the example shown in Figure 5, the loop block should be executed 10 times. The user-specified TDL file is the input to the *msa_acc_plan* function to generate the corresponding accelerator descriptor.

V. Evaluation Methodology

One of the major issues during the design of accelerator-based architectures is the evaluation methodology. The traditional way to evaluate a novel architecture is to conduct cycle-accurate full-system simulation on architectural simulators such as Gem5 [17]. However, such cycle-accurate full-system simulation methodology exhibits two major problems. The first one is that the simulation may be inaccurate due to specification and abstraction errors [18], which may deviate

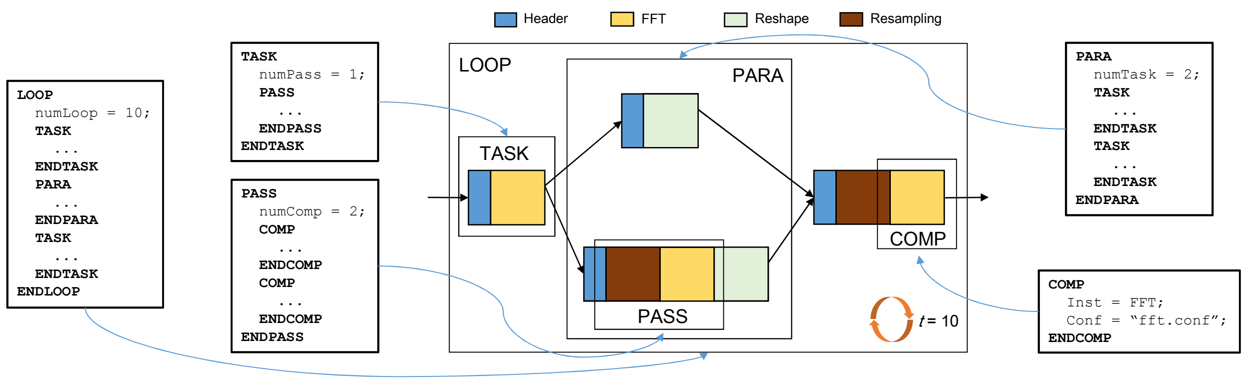

Fig. 5: An example to leverage user-level task description language (TDL) to configure multiple accelerators.

the final design decision. The second one is that the simulation speed is too slow to explore the entire design space. To address these issues, in this paper, we propose a hybrid approach that combines simulation, modeling, and native execution to accurately and efficiently evaluate the 3D-stacked MSA on commodity systems.

A. Overview

Since a modern commodity system (e.g., Intel Haswell machine) typically contains multiple memory channels, and each channel has multiple memory DIMMs, each DIMM could be used to mimic one memory stack of 3D DRAM system. Therefore, we can assume that the accelerators could be attached to any of such DIMMs. Figure 6 shows the overall evaluation system for the 3D-stacked MSA architecture. We assume that the accelerators are attached to DIMM3, and thus the memory region of DIMM3 is treated as LMS (local memory stack). The behavior of DIMM3 is simulated and modeled by several tools, including Synopsis Design Compiler, DesignWare, McPAT [19], 3D DRAM Model, and overall 3D MSA Model [14]. On the other hand, the performance and power of the remaining DIMMs and the central processor are measured by native execution. Finally, the overall system performance and power are estimated by combining the simulated and the measured results.

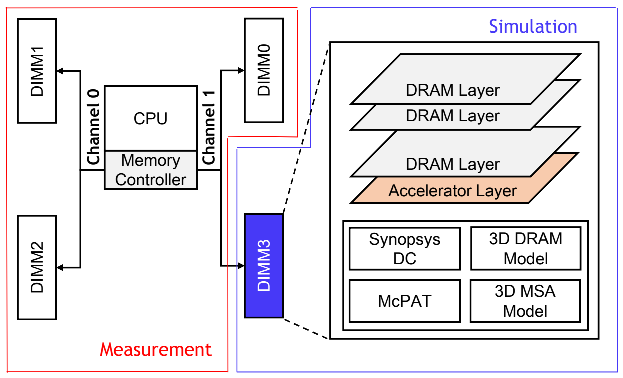

Fig. 6: Overview of the hybrid evaluation methodology.

B. Measurement Methodology

The measurement is conducted on the host processor, an 8-core Intel i7-4770K (Haswell) processor. It has two 2 memory channels, and supports at most 4 memory DIMMs. Since we are interested in the energy efficiency of the entire system, we use PAPI [20] to measure both performance and energy consumption of the host processor, as well as the "common" memory DIMMs (i.e., DIMM0, DIMM1, and DIMM2) through the Running Average Power Limit (RAPL) interface [21]. The measurement stops when entering the simulation process, and resumes when exiting the simulation.

C. Simulation Methodology

Simulation trigger. In a real 3D-stacked MSA system, the accelerator is invoked by writing control commands to a specific address space. To characterize this behavior, in the evaluated system, such command space is monitored by memory page protection. In more detail, once the protected address space is updated with control commands, the entire system will be trapped into a user-defined signal handler, where all the simulation and modeling tools are invoked to estimate the performance and power of 3D-stacked MSA.

Disable channel interleaving. Since one specific DIMM is used to mimic the LMS of the MSA, we need to determine the memory region of this DIMM so that the kernel can reserve contiguous memory for LMS. However, in commodity multi-channel systems, memory channel interleaving is automatically enabled to improve the overall memory bandwidth. In the memory channel-interleaving mode, one physical page is equally distributed across different channels in cache-block granularity. In this case, the physical address is not contiguous within one memory channel. Therefore, for simulation purposes, we need to disable the memory channel interleaving on the evaluated platform.

To disable the memory channel interleaving, we only populate three memory DIMMs (i.e., DIMM0, DIMM1, and DIMM3 in Figure 6) with the same size on the motherboard to convert the channel-interleaving mode to the asymmetric mode. In the asymmetric mode, where the memory capacities of different channels are not equal, the entire memory is divided into symmetric and asymmetric zones. The symmetric zone starts at the lowest address until the asymmetric zone begins, and the symmetric zone is still in channel-interleaving mode, while the asymmetric zone is in single-channel mode. In the example shown in Figure 6, the address spaces of DIMM0 and DIMM1 are still interleaved, while the address space of DIMM3 is separate from others. Therefore, DIMM3 can be used to simulate the behavior of LMS of MSA.

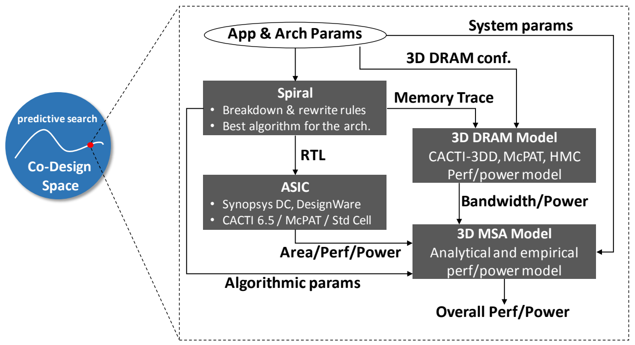

Fig. 7: The design flow to explore the co-design space of 3D-stacked MSA.

Performance/power model. The performance/power model of 3D-stacked MSA contains multiple simulation and modeling tools, including Synopsis Design Compiler, DesignWare, McPAT, 3D DRAM Model, and 3D MSA Model. The Design Compiler, DesignWare and McPAT are used to estimate the performance/power/area of the MSA logic. The 3D DRAM model is built upon CACTI-3DD [22] and a performance/power model offering the bandwidth/power estimation of 3D-stacked DRAM. According to the above information, the accelerator-specific 3D MSA model is customized to estimate the overall performance and power of the 3D-stacked MSA architecture.

One challenge in the design of a 3D-stacked MSA is that there is a huge co-design space, which consists of application parameters, accelerator architectural configurations, and 3D DRAM configurations to explore. To facilitate the exploration of such co-design space of 3D-stacked MSA, we use an automated framework based on the performance/power model. As shown in Figure 7, for each design point consisting of the application parameters (e.g., algorithm type and problem size, etc.), the architectural configurations (e.g., local memory size and data-path width, etc.), and the 3D DRAM configurations (e.g., number of TSVs and number of layers, etc.), the framework first uses Spiral [23], an automated tool to generate and optimize RTL design. The generated RTL design then undergoes the typical ASIC synthesis flow to obtain the estimated area, performance and power of the MSA logic. The 3D DRAM configuration is also sent to the 3D DRAM Model to get the estimation of the bandwidth and power of the 3D DRAM. Finally, the application parameters, the 3D DRAM configuration, the estimated area/performance/power of the MSA logic, and the estimated bandwidth/power of the 3D DRAM are treated as the inputs of the 3D MSA model to produce the overall performance and power of the 3D-stacked MSA. The above process can be automatically iterated for many times to explore the entire design space. Moreover, several advanced design space exploration techniques [24] can be also employed to significantly accelerate the search process.

VI. Experimental Results

We present the experimental results for (i) running an FFTW-compatible program and (ii) matrix transposition on the 3D-stacked MSA system with an FFT accelerator and a reshape accelerator, respectively.

Experimental setup. Table I shows the configuration of the 3D-stacked MSA system with memory-side FFT and reshape accelerators. The FFT accelerator processes radix-2 2D FFT, the problem size is 4k×4k, and each point is a complex single-precision number. For design space exploration, we also consider several design options, i.e., streaming width ($s = 2, 4, 8, 16$), tile width ($t = 8, 16, 32, 64$), and frequency ($f = 0.4, 0.8, 1.2, 1.6, 2$ GHz). The reshape accelerator performs the in-place matrix transposition on various size matrices and the design is adopted from [25].

The 3D-stacked memory has 1 GB capacity, 4 DRAM layers, 8 banks per layer, and 2,048 data TSVs. It also features 8 high-speed SERDES links each with 40 GB/s bandwidth, total 320 GB/s, similar to HMC [5]. The maximal achievable internal bandwidth of this 3D-stacked memory is about 510 GB/s and the external bandwidth can reach up to 320 GB/s. The internal power consumption is 23 W at the peak bandwidth hence the energy per bit is 5.6 pj/bit. When the data is transferred externally the energy per bit reaches 12 pj/bit.

Category	Configuration	Values
FFT	Type	2D, radix-2
	Size	4k × 4k
	Precision	Complex Single
Reshape	Type	Matrix transpose
	Size	1k × 1k → 16k × 16k
	Precision	Single
	Local SRAM	2 × 8 × 32 KB = 512 KB
3D DRAM	N_{stack}	4
	N_{bank}	8
	N_{TSV}	2,048
	Bandwidth	510 GB/s

TABLE I: Configuration of 3D-stacked MSA with the FFT and the Reshape accelerators.

Experimental results. Figure 8 shows the performance (GFLOPS) and power (Watt) of the investigated design space for the FFT acceleration. The energy-efficiency of different designs varies between 6 GFLOPS/W and 46 GFLOPS/W, and the configuration of the optimal design is, $s = 16$, $t = 32$, $f = 2$ GHz. We further compare the energy-efficiency of all design options to that of the Intel Haswell as shown in the right part of Figure 8, where the results are normalized to the Haswell. We observe that the improvement over the Haswell is up to 179x in terms of GFLOPS/W.

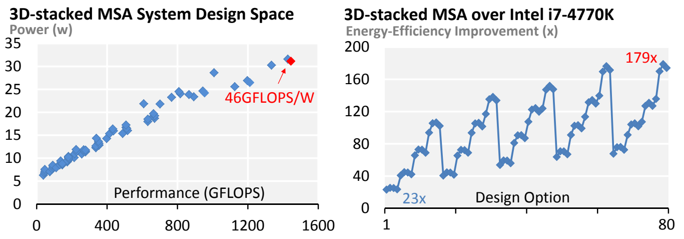

Fig. 8: The design space of 3D-stacked MSA system with FFT accelerator and comparison with Intel Haswell processor in terms of GFLOPS/W.

We also evaluate the matrix transposition using MSA and traditional multi-threaded software implementations in Figure 9 in terms of performance and energy efficiency. We report

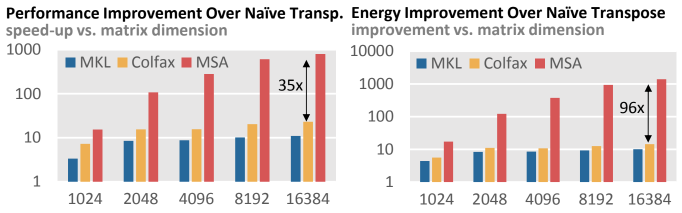

Fig. 9: Matrix transpose using MSA with reshape accelerator, Intel MKL and optimized software from Colfax research. Results are normalized to a naive matrix transpose implementation on Intel Haswell.

performance and energy of an Intel MKL-based implementation [26], an implementation from Colfax Research [27] and a MSA-based reshape accelerator where the results are normalized to a reference naive matrix transpose implementation. The software implementations are based on the Intel Haswell. The reshape accelerator within the 3D-stacked DRAM implements a blocked transposition algorithm to transfer DRAM page size data chunks. It also features multiple SRAM banks and exploits the parallelism of multiple vaults/layers. We observe that MSA provides up to 35x performance and 96x energy improvements over the best software implementation.

Offloading the execution from the host to the MSA incurs significant overhead, which is included in the results presented in Figure 8 and Figure 9. To offer more insights for the offloading overhead, we also demonstrate the runtime and energy spent on the host processor due to communication and synchronization (e.g. invocation of the accelerator, cache flush). Although the offloading overhead comprises 10% and 6% of the overall energy and runtime, respectively, for large matrices, it can span 94% and 96% of the overall runtime and energy, respectively, for small matrices. Therefore, to efficiently use the MSA, it is necessary to further investigate highly efficient communication and synchronization mechanisms between the host processor and the accelerator.

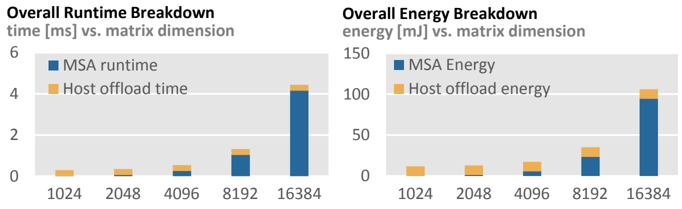

Fig. 10: Runtime and energy breakdown for offloading the execution to the MSA and the actual MSA operation.

VII. Conclusion

As we enter the dark silicon era, a 3D-stacked MSA architecture is promising to improve the energy-efficiency for processing emerging data-intensive applications. In this paper, we build a full-system prototype of the 3D-stacked MSA system. This system is enabled by a configurable accelerator array on the logic layer of 3D-stacked DRAM, and a novel software stack to ease the burden of programming. Experimental results demonstrate that the 3D-stacked MSA system achieves up to 179x and 96x better energy-efficiency than the Intel Haswell processor for the FFT and matrix transposition algorithms, respectively.

References

[1] M. B. Taylor, "Is dark silicon useful? harnessing the four horesemen of the coming dark silicon apocalypse," in *DAC*, 2012, pp. 1131–1136.

[2] M. Gokhale *et al.*, "Processing in memory: The terasys massively parallel pim array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[3] M. Oskin *et al.*, "Active pages: A computation model for intelligent memory," in *ISCA*, 1998, pp. 192–203.

[4] M. L. Chu *et al.*, "High-level programming model abstractions for processing in memory," in *WoNDP*, 2013.

[5] J. Jeddeloh *et al.*, "Hybrid memory cube new dram architecture increases density and performance," in *VLSI Technology (VLSIT), 2012 Symposium on*, June 2012, pp. 87–88.

[6] G. Venkatesh *et al.*, "Conservation cores: Reducing the energy of mature computations," in *ASPLOS*, 2010, pp. 205–218.

[7] H. Esmaeilzadeh *et al.*, "Neural acceleration for general-purpose approximate programs," in *MICRO*, 2012, pp. 449–460.

[8] K. Lim *et al.*, "Thin servers with smart pipes: Designing soc accelerators for memcached," in *ISCA*, 2013, pp. 36–47.

[9] L. Wu *et al.*, "Navigating big data with high-throughput, energy-efficient data partitioning," 2013, pp. 249–260.

[10] O. Kocberber *et al.*, "Meet the walkers: Accelerating index traversals for in-memory databases," in *MICRO*, 2013, pp. 468–479.

[11] S. H. Pugsley *et al.*, "NDC: analyzing the impact of 3d-stacked memory+logic devices on mapreduce workloads," in *ISPASS*, 2014, pp. 190–200.

[12] S. Pugsley *et al.*, "Comparing implementations of near-data computing with in-memory mapreduce workloads," *Micro, IEEE*, vol. 34, no. 4, pp. 44–52, 2014.

[13] D. Zhang *et al.*, "Top-pim: Throughput-oriented programmable processing in memory," in *HPDC*, 2014, pp. 85–98.

[14] B. Akin *et al.*, "Understanding the design space of dram-optimized hardware FFT accelerators," in *ASAP*, 2014, pp. 248–255.

[15] F. Sadi *et al.*, "Algorithm/hardware co-optimized sar image reconstruction with 3d-stacked logic in memory," in *HPEC*, 2014.

[16] M. Frigo *et al.*, "The design and implementation of fftw3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[17] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[18] A. Gutierrez *et al.*, "Sources of error in full-system simulation," in *ISPASS*, 2014, pp. 13–22.

[19] S. Li *et al.*, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009, pp. 469–480.

[20] V. Weaver *et al.*, "Measuring energy and power with papi," in *ICPPW*, 2012, pp. 262–268.

[21] "Intel 64 and ia-32 architectures software developers," October 2014. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

[22] K. Chen *et al.*, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *DATE*, 2012, pp. 33–38.

[23] M. Püschel *et al.*, "SPIRAL: Code generation for DSP transforms," *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, vol. 93, no. 2, pp. 232– 275, 2005.

[24] T. Chen *et al.*, "Archranker: A ranking approach to design space exploration," in *ISCA*, 2014, pp. 85–96.

[25] B. Akin *et al.*, "Hamlet: Hardware accelerated memory layout transform within 3d-stacked dram," in *HPEC*, 2014.

[26] "Intel math kernel library (MKL)," http://software.intel.com/en-us/articles/intel-mkl/.

[27] A. Vladimirov, "Multithreaded transposition of square matrices with common code for intel xeon processors and intel xeon phi coprocessors," http://research.colfaxinternational.com, Aug 2013.