3D-MMC: A Modular 3D Multi-Core Architecture with Efficient Resource Pooling

Tiansheng Zhang*, Alessandro Cevrero†, Giulia Beanato†, Panagiotis Athanasopoulos†,
Ayse K. Coskun* and Yusuf Leblebici†
*ECE Department, Boston University, Boston, MA, USA *{tszhang, acoskun}@bu.edu,
†LSM, EPFL, Lausanne, Switzerland
†{alessandro.cevrero, giulia.beanato, panagiotis.athanasopoulos, yusuf.leblebici}@epfl.ch

***Abstract*—This paper demonstrates a fully functional *hardware and software* design for a 3D stacked multi-core system for the first time. Our 3D system is a low-power *3D Modular Multi-Core* (3D-MMC) architecture built by vertically stacking identical layers. Each layer consists of cores, private and shared memory units, and communication infrastructures. The system uses shared memory communication and *Through-Silicon-Vias* (TSVs) to transfer data across layers. A serialization scheme is employed for inter-layer communication to minimize the overall number of TSVs. The proposed architecture has been implemented in HDL and verified on a test chip targeting an operating frequency of 400MHz with a vertical bandwidth of 3.2Gbps. The paper first evaluates the performance, power and temperature characteristics of the architecture using a set of software applications we have designed. We demonstrate quantitatively that the proposed modular 3D design improves upon the cost and performance bottlenecks of traditional 2D multi-core design. In addition, a novel *resource pooling* approach is introduced to efficiently manage the shared memory of the 3D stacked system. Our approach reduces the application execution time significantly compared to 2D and 3D systems with conventional memory sharing.**

I. Introduction

The scaling to nanometer technologies has led to a transition from single-core to multi-core processors, and the trend is now moving towards many-core architectures [1]. While hundreds of millions of transistors can now be placed on a single chip, they cannot be fully exploited due to interconnect/memory latency, power consumption, and yield related issues. 3D integration is an emerging technology aiming to overcome the limitations faced by traditional (2D) design. In 3D circuits, multiple dies can be stacked on top of each other and interconnected by TSVs. In this way, it is possible to improve yield (owing to smaller chip area) and also tackle the latency, bandwidth, and power challenges of interconnects. In chip multiprocessors (CMPs), whose performance is typically limited by the memory access bottleneck, increased communication and memory access bandwidths are distinguishing advantages of 3D stacking. Nevertheless, the benefits offered by 3D integration may vanish without an efficient software infrastructure to fully exploit the additional available resources. This paper introduces both the architectural design and software development for 3D-MMC, a low-power, modular 3D-CMP system.

Previous architecture research on *3D-CMPs* focuses either on stacking memory layers on top of core logic to boost memory bandwidth, or on augmenting the capabilities of planar CMPs including additional logic layers. In both cases, each tier has a different layout. To the best of our knowledge our system is the first fabricated CMP that combines multiple identical tiers in a modular fashion to increase system performance. A similar approach has already been adopted for 3D-DRAM, where identical memory chips are stacked to increase the overall memory capacity [2]. The 3D-MMC platform has several advantages owing to the modular design approach. First, it dramatically simplifies the chip design process and reduces *Non Recurring Engineering* (NRE) costs by creating a portfolio of architectures using the same mask set. Second, homogeneity of the system allows using the same testing protocol for each die within the stack, leading to pre-bond testability without any additional effort for test engineers.

978-3-9815370-0-0/DATE13/©2013 EDAA

In addition to introducing the specifics of the 3D-MMC architecture, this paper evaluates the performance, power, and thermal behavior of our 3D system. Following our analysis, we implement a novel resource management approach to manage memory accesses and increase performance. Our specific contributions are as follows:

- We introduce a novel 3D-CMP architecture based on the integration of identical layers to augment the system performance with minimal design cost compared to conventional planar IC design. The system uses shared memory communication and TSVs to transfer data among the layers. A serialization scheme is employed to minimize the TSV overhead. The system is implemented to run at 400MHz and 3.2Gbps inter-layer data transfer bandwidth.
- We study performance, power, and thermal characteristics of the 3D-MMC architecture. Our results demonstrate the low power consumption and reliable thermal profiles.
- We propose a *resource pooling* technique to optimize memory access latency in the 3D-CMP. Resource pooling allows cores to leverage available memory resources on remote layers to minimize memory contention. While resource pooling concept has been introduced in recent work [3], our work demonstrates the first specific, fine-grained implementation on a real-life 3D-CMP design.

The rest of the paper starts with discussing related work. Section III describes the 3D-MMC design. Section IV introduces our resource pooling method. Section V evaluates the performance and Section VI concludes the paper.

II. Related work

3D integration has received considerable attention from academia and industry recently [4] [5] [6]. 3D systems can be broadly classified into two categories: memory + logic systems and logic + logic systems. The former generally refers to cache or main memory stacked over logic. Stacking SRAM or DRAM on logic achieves shorter memory access latency compared to the baseline performance of multi-core systems [7]. Two examples in this category, utilizing GlobalFoundries' 130 nm process and Tezzaron's FaStack technology, respectively, are 3D-MAPS [8], where the logic die consists of 64 cores operating at 277MHz and the stacked memory die contains 256KB SRAM, and Centip3De [9], a configurable near-threshold 3D stacked system with 64 ARM Cortex-M3 cores.

The logic-on-logic case involves splitting a planar design's logic area into two or more layers, such as the 3D version of an Intel Pentium 4 family processor in Garrou et al.'s work [7]. A processor where a baseline microarchitecture is augmented by vertically stacking additional blocks (e.g., more caches, reservation station, etc.) to target different market segments is proposed by Loh [10]. Exploring vertically stacked microprocessors, Homayoun et al. [3] envision resource pooling as an efficient method to achieve mocroarchitectural resource sharing at a fine granularity.

Our work introduces both the hardware architecture and the software implementation for a novel 3D-CMP architecture. We focus on exploiting shared memory resource pooling and provide a practical implementation. Our system uses homogeneous stacking, which results in lower wafer and 3D bonding costs compared to heterogeneous partitioning, as demonstrated by Zhao et al. [11]. Thus, 3D-MMC offers a desirable tradeoff among complexity, cost and performance.

III. Optimized 3D Modular Design

This section provides the details of the proposed modular 3D architecture, 3D-MMC. We then describe our performance evaluation setup. Finally, considering the significance of thermal challenges in 3D stacking, we demonstrate the thermal feasibility of our system.

A. Architecture Description

Figure 1 provides a diagram of the 3D-MMC system composed of two layers interconnected by TSVs. The number of layers in the stack determines the total core count as well as the memory size and the level of memory resource pooling. A single layer can function either as a stand-alone 2D-CMP and as part of a 3D-system when integrated with the 3D communication infrastructure. We next describe the planar architecture and the 3D communication infrastructure.

1) Planar (single-layer) architecture: A single layer is composed of four *Processing Elements* (PE) that exchange data through a shared memory, which is placed in the *Peripheral Subsystem* (PS) unit. A system of semaphores arbitrates the access of PEs to the shared memory. The routing between each PE and the shared memory occurs through a Network-on-Chip (NoC). Several 3D-NoC topologies have been proposed recently, and their superior performance over 2D-NoC have been demonstrated by Pavlidis et al. [12]. In 3D-MMC, a specific source-routed NoC is implemented to manage the signals routing to and from 6 directions (North, South, East, West, Up, Down). In the 3D stack, NoC on different layers are interconnected to enable the management of the signals in both horizontal and vertical directions.

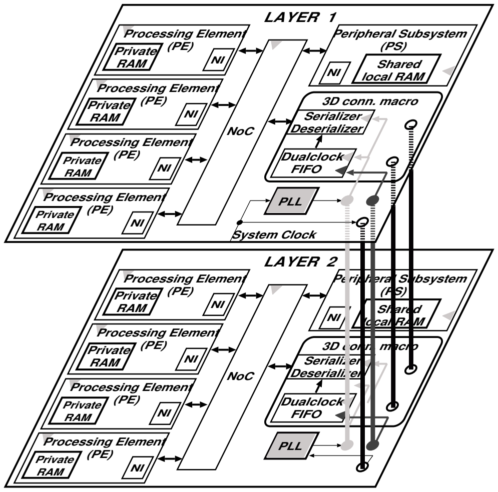


Fig. 1. Overview of the 3D-MMC built with stacking identical layers. The figure does not show the data TSVs for clarity.

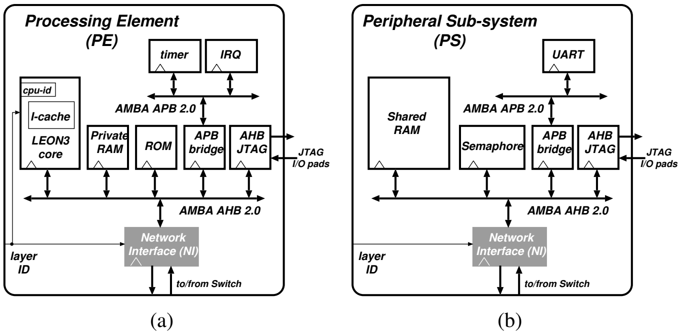
Figures 2(a) and 2(b) illustrate the internal architecture of a PE and a PS, respectively. Each PE is built out of a 32-bit RISC processor, the open-source LEON3 unit from Aeroflex Gaisler, which is connected to the slave modules through an AMBA bus. An AHB JTAG master module is included for debugging purposes. Slave devices for each PE are a privately addressable memory space including a ROM for booting and a private RAM to host the program code. The Network-Interface (NI) block is a master located within both PE and PS. It interfaces the AMBA bus to the NoC, and is responsible of transferring data packets to/from the shared memory, which has an address space visible to each core. Similar to PEs, the PS contains NI and AHB JTAG acting as masters, whereas all the remaining units (semaphore, shared RAM) act as slaves.

Each PE in an N-layer system has access to N+1 different memory modules that can be accessed in parallel: a private-RAM contained in its own PE, a shared local-RAM located in the PS of its layer, and N-1 shared remote-RAMs situated in the PS of the other stacked layers. Similar to the architecture designed by Benini et al. [13], the proposed memory hierarchy that uses shared data memory for inter-processor communication simplifies the hardware complexity and avoids memory coherency overhead. The multi-core synchronization is handled at the software level.

2) 3D communication and control infrastructure: Inter-layer communication is achieved through a 3D communication unit, *3D-macro*, which leverages an array of TSVs as a vertical data bus. To limit the TSV area overhead, we use a serializer-deserializer (serdes) module to minimize the total number of TSVs. This serdes module is explained in detail in our prior work [14]. Data signals are serialized before the transmission through TSVs, and de-serialized at the receiving layer. The bandwidth loss due to serialization can be compensated by increasing the serdes clock frequency. Serialization is more cost-efficient than parallel buses in 3D-ICs [15].

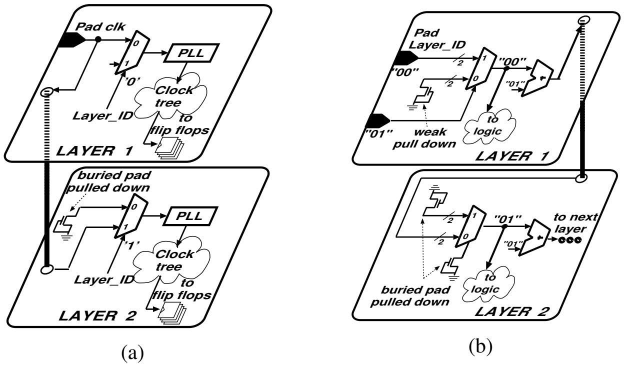
A challenging issue for 3D-ICs is the reliable distribution of the clock signal [16]. In 3D-MMC, each stacked layer has an independent clock domain. The clock is injected onto a pad of the top layer, passes through a PLL module, and is both distributed in the circuit and sent to the next layer. The bottom layer receives the clock from power TSVs, and forwards it to a PLL module to be re-generated for maintaining its integrity. Hence, all layers operate at the same frequency, but are asynchronous from each other due to potential phase shifts among the clocks. In the multi-clock domain approach, signals are transmitted among the layers together with their clock and then they are re-synchronized to the clock domain of the receiving layer using a Dual Clock FIFO. Clock propagation is demonstrated in Figure 3(a).

Once the identical layers are stacked, they need to operate as a complete system without further modification. For this purpose, a dedicated control signal, namely the *layer identification number* (LayerID) is implemented for enabling auto-configuration of the layers depending on their positions in the 3D-system. As shown in Figure 3(b) for a sample case of two layers, the sequence "00", is injected through the pads and selected by a multiplexer as the layerID of the top layer. The value is also forwarded to a half adder that computes the ID for the next layer below, "01". The pads of the bottom tier are designed to be pulled-down when no signal is applied to them. As a result, the multiplexer on the second layer selects the LayerID transmitted by the TSVs.



(a) (b)

Fig. 2. (a) PE internal architecture; (b) PS internal architecture.



(a) (b)

Fig. 3. (a) Clock distribution and propagation between two layers using three redundant TSVs; (b) LayerID generation and propagation between two layers using three redundant TSVs.

B. Evaluation platform

The proposed architecture has been implemented in HDL and verified on a test chip targeting an operating frequency of 400MHz and a vertical bandwidth of 3.2Gbps. The 2D-CMPs have been fabricated using a standard UMC 90nm CMOS technology. Single dies have been tested and verified, and then processed for in-house TSV fabrication and stacking.

The performance study in this paper is conducted through cycle-accurate post-layout simulations in ModelSim. We have designed a set of software benchmarks to evaluate performance. The benchmarks are compiled using a SPARC compiler, loaded into the ROM, and executed following the boot script.

C. Thermal Evaluation

A significant challenge in 3D stacking is the power density increase per footprint, which may cause temperature to increase beyond reliable thresholds. This section demonstrates the thermal feasibility of 3D-MMC.

The power consumption of each component in a layer of the 3D stack is estimated via statistical power analysis using Encounter Power System by Cadence. We assume a switching rate of 50% for each flip flop and each input port, and we use a 100% toggling rate for the clock. The tool automatically estimates the power consumption based on the average toggling rate of each gate. Table I provides the power consumption of all the components at 400MHz including leakage power. Core power in the table includes the logic, I-Cache, ROM, and all other sub-blocks of the core except for the local RAM. Table I highlights the low power consumption of 3D-MMC, where each layer consumes 267mW.

We use HotSpot version 5.02 [17] for thermal simulations. The package and die parameters used in the simulation are provided in Table I. The floorplan of each layer is identical and is shown in Figures 1 and 4. To take the impact of TSVs into account during thermal evaluation, we use a modified version of HotSpot that enables modeling heterogeneity within a layer [18]. We compute joint thermal resistivities for each TSV block based on the ratio of TSV (Cu) area to overall TSV array area (including all the spacing between the TSVs). We simulate the system without a heat sink by using a very small number for the heat sink thickness in HotSpot. Layers are stacked using a glue (interface material) layer of Parylene-C.

Figure 4 provides the steady state peak temperatures for a single layer chip and 3D systems including 2 layers, 4 layers,

TABLE I. POWER CONSUMPTION AND THERMAL PROPERTIES OF 3D-MMC

Power Consumption Characteristics	
Components	**Power (mW)**
Core	37.98
Local RAM for each core	17.13
Router	10.07
Data TSV arrays	1.6 (smaller array) to 8.11 (larger array)
Shared memory	22.16
PLL	5
Package and Die Thermal Characteristics	
Die area	3.5mm x 3.5mm
Die thickness (bottom layer)	280μm
Die thickness (other layers)	50μm
Die (Si) resistivity	0.01mK/W (meter-Kelvin per Watt)
Glue conductivity	0.082W/mK at 25°C
Glue thickness	2μm

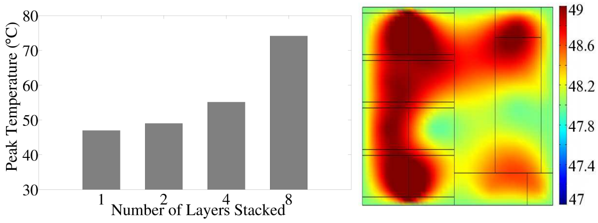


Fig. 4. The figure demonstrates the peak temperatures at steady state for a single layer as well as 2, 4, and 8-layered stack. On the right, we show the thermal map of the top layer for the 2-layered stacks. Thermal variations are similarly low (limited to a few degrees only) for 4 and 8-layered stacks.

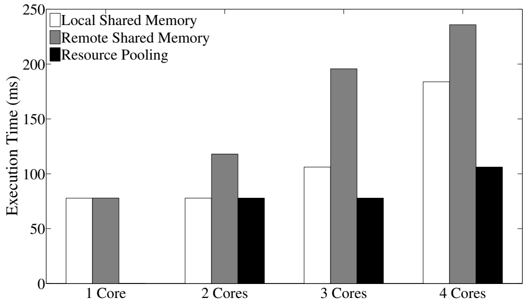


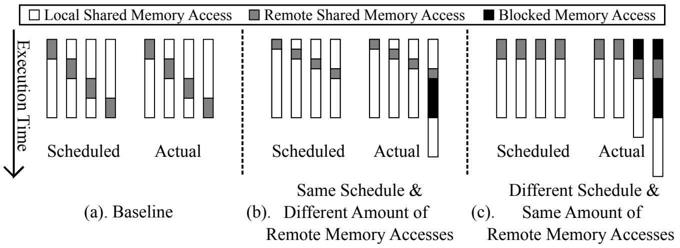
Fig. 5. Comparison of execution time of *Memory Stress* benchmark when all cores access local memory, all cores access remote memory, and when memory resource pooling is applied.

and 8 layers when all cores are active. In the figure, we also provide a thermal map for the top layer of the 2-layered system. For 2 and 4-layered systems, even though cores overlap on top of each other and all cores are active, we do not observe high temperatures. For the 8-layered stack, peak temperature reaches 74°C, which is still below the typical 85° thermal thresholds used in most processor chips. As we focus on a 2-layered stack in this paper, we do not apply thermal management strategies.

IV. Resource Pooling For Efficient Memory Access

In traditional 2D design, as the number of cores increases, the bandwidth of the shared memory becomes a performance bottleneck. In 3D-MMC the possibility to utilize all layers' resources as a whole allows to address the memory bottleneck problem using resource pooling. Resource pooling in 3D systems refers to the sharing of resources between vertical stacked layers [3]. This section first demonstrates the memory bottleneck problem on a 2-layer 3D-MMC, and then proposes a methodology to find the optimal way to apply resource pooling.

3D-MMC's memory subsystem has only one write port and one read port. When the memory access rate exceeds a certain level, access blocking occurs. Aiming to evaluate memory bottlenecks and resource pooling, we create a memory-intensive benchmark, *Memory Stress*, which performs 1000 writes of integer-length values into the shared memory. We perform experiments that write on local (same layer's) shared memory and remote (different layer's) shared memory respectively, with 1-core, 2-core, 3-core, and 4-core cases. In this group of experiments, all active cores are on the same layer. The resulting execution times for both local and remote shared memory cases are shown in Figure 5. This figure shows that as more cores attempt to access shared memory, performance penalty increases. When there are either more than two cores accessing the remote shared memory or more than three cores accessing the local shared memory, the extra cores are blocked for a while. Blocking of cores happens because of the memory bottleneck and the communication limitation between layers.



(a). Baseline

(b). Same Schedule & Different Amount of Remote Memory Accesses

(c). Different Schedule & Same Amount of Remote Memory Accesses

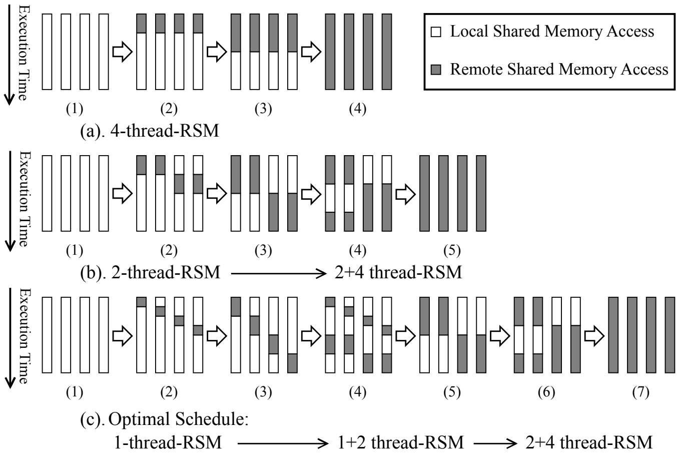
Fig. 6. Performance under different workload allocation and scheduling combinations. (a) serves as the baseline, (b) has the same schedule with (a) but has fewer remote memory accesses, while (c) has the same number of remote memory accesses but has a different schedule.

To overcome the local shared memory bottleneck, we propose utilizing the remote shared memory to mitigate the access competition. We call this scenario *Memory Resource Pooling*. For the experiment in Figure 5, for the multi-core cases we assign one core to access the remote shared memory while the others still access the local shared memory. In the 3-core case, one core is assigned to access remote shared memory while the other two write to the local shared memory. For 3-core and 4-core cases, memory resource pooling brings 26.6% and 42.3% reduction in execution time, respectively.

The above memory resource pooling strategy schedules memory accesses at the core granularity; thus, we call it Core Level Resource Pooling (CLRP). Task Level Resource Pooling (TLRP) includes adjustable *Workload Allocation* and *Workload Scheduling* within each core. With TLRP, the workload of each core is divided into two parts: local memory accesses and remote memory accesses. For each core in the system, workload allocation determines the ratio of local and remote memory accesses, while workload scheduling defines the execution sequence of memory accesses. We allocate equal amount of workload to all the cores for a fair comparison.

In Figure 6, each group of 4 bars represents the workload execution of four cores on the same layer. White and gray blocks stand for local and remote memory accesses, respectively, and black ones represent the memory stalls. In the same figure, *Scheduled* refers to the sequence of workload execution planed for each core, while *Actual* shows the real execution. We consider (a) as a baseline case, where simultaneous local memory accesses from 4 cores are avoided. In this case, all cores behave as scheduled and no core is blocked because of contention. Case (b) shows the situation where the system applies the same workload schedule with (a) but with fewer remote memory accesses. As local shared memory allows for at most 3 cores to simultaneously access to it, there is a noticeable performance loss once all four cores access the local shared memory. Case (c) demonstrates the system's behavior when the cores have the same amount of remote memory accesses as (a) but they are scheduled to access local and remote shared memory at the same time, which causes considerable performance loss compared to (a). Thus, combination of workload allocation and scheduling in TLRP has significant effects on performance.

To optimize performance via memory resource pooling, we should avoid the memory bottleneck as much as possible. Next,



(a). 4-thread-RSM

(b). 2-thread-RSM ⟶ 2+4 thread-RSM

(c). Optimal Schedule:
1-thread-RSM ⟶ 1+2 thread-RSM ⟶ 2+4 thread-RSM

Fig. 7. Workload schedules for task level resource pooling. (a). 4 threads accessing remote shared memory at the same time–*4-thread-RSM*; (b). (1)-(3): 2 threads accessing remote shared memory at the same time–*2-thread-RSM*; (c). (1)-(3): 1 thread accessing remote shared memory–*1-thread-RSM*.

we propose an approach for computing the relationship between performance and the number of remote memory accesses. We introduce three workload schedules as shown in Figure 7, where each schedule is applicable to any workload allocation. In Figure 7, remote memory accesses increase gradually from left side to right side. Schedule (a) always makes all four cores access the remote shared memory (*4-thread-RSM*, where RSM stands for remote shared memory). Schedule (b) issues two cores to access remote shared memory at a time (*2-thread-RSM*) from (1) to (3). In (4-7), there are too many remote memory accesses to be scheduled using *2-thread-RSM*, thus we apply mixed *2+4 thread-RSM* until the ratio of remote memory accesses increases to 100%. *1-thread-RSM* has only one thread accessing remote shared memory at a time to minimize simultaneous local memory access, as shown in case (c) from (1) to (3). As the remote memory accesses increase, schedule (c) uses *1-thread-RSM*, *1+2 thread-RSM*, and *2+4 thread-RSM* successively, which minimizes simultaneous local memory accesses.

To compare the performance of these 3 schedules, we observe the execution time of the whole application, i.e., the longest execution time among all four cores. The execution time on each core is the product of *Instruction Count*, *Cycles per Instruction* and *Cycle Time*. Since most memory-intensive applications contain a large number of memory accesses, in this case we can substitute instructions with memory accesses, which means execution time equals the product of *# of Memory Accesses*, *Cycles per Memory Access* and also *Cycle Time*. To apply TLRP, we need to split memory accesses into local memory accesses and remote memory accesses. In the following equation, T_{exec} stands for the execution time, N_{MemAcc} is the total number of shared memory accesses in the application, W_L and W_R represent the weight (i.e., ratio) of local and remote memory accesses, and C_{Li} and C_{Ri} refer to the number of cycles when i cores are accessing local shared memory and remote shared memory, respectively. C_{Li} and C_{Ri} can be obtained from the shared memory access test. After replacing the variables with their values we can compute the function of T_{exec} and W_R, and thus we can calculate the execution time according to the ratio of remote memory accesses.

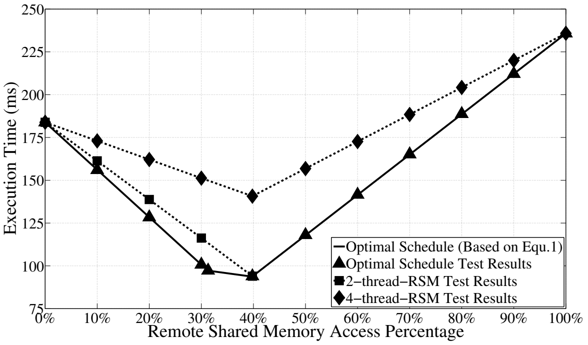


Fig. 8. Test results of different memory resource pooling schedules and the optimal schedule's curve based on Eqn. (1).

$$T_{exec} = (\sum(W_{Li} \times N_{MemAcc} \times C_{Li}) + \sum(W_{Ri} \times N_{MemAcc} \times C_{Ri})) \times T_{Cycle} \quad (1)$$

$$W_L + W_R = \sum W_{Li} + \sum W_{Ri} = 1 \quad (2)$$

Figure 8 shows the test results and fitted curves of the workload schedules in Figure 7. For the optimal schedule, we also draw the theoretical curve based on Eqn. (1). The experimental results fit very well with the theoretical curve. Although all of the three schedules can take advantage of resource pooling, the optimal one improves the performance by up to 48.9%, which coincides with the curve for 2-thread-RSM. The optimal schedule shown in the figure demonstrates the potential benefits of memory resource pooling and TLRP workload scheduling.

In Eqn. (1), N_{MemAcc} is related to the application, T_{cycle} depends on the architecture, and C_{Li} and C_{Ri} are both application and architecture related. Thus, for most of the shared-memory systems and applications, the proposed approach is applicable for quantifying the potential performance improvement of memory resource pooling.

V. Performance evaluation

This section demonstrates the performance benefits of 3D-MMC. To evaluate the performance, we design four benchmarks: **1D DCT**, a single dimension 8×8 matrix discrete cosine transformation; **1D FFT**, 8×8 matrix 12 butterfly Fast Fourier Transformation; **1D Median Filter** with a window size of 3 and an input array with 64 integers; **Matrix Multiplication**, 8×8 multiplication implemented using divide and conquer algorithm.

The test results are shown in Figure 9. Ideal performance improvement refers to the improvement we can get from the multi-core system if the benchmarks can be fully parallelized. For 1-core to 4-core cases, the cores are all on one layer and thus the system can be viewed as a 2D layer only. As for 8-core case, we have two 2D layers with four cores on each, which is the 3D-MMC architecture described above. It can be observed from this figure that the performance improves significantly (61% on average) from 4-core (2D) to 8-core (3D). The difference of improvement among benchmarks is because

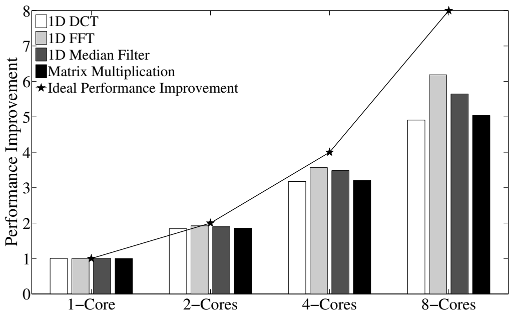


Fig. 9. Performance improvement compared to single core.

TABLE II. EXECUTION TIME OF DIFFERENT SHARED MEMORY ACCESS SCENARIOS WHEN ALL 8 CORES ARE ACTIVE.

Benchmark	**All cores access a single shared memory (ns)**	**All cores access their local shared memory (ns)**
1D DCT	16716	16495
1D FFT	26260	26063
Median Filter	8674	7420
Matrix Multiplication	52838	51935

the benchmarks vary in their scalability. For example, both DCT and FFT have the same input matrix and large amounts of computation, but FFT is more computation-bound compared to DCT. Reading the input can be viewed as the serial part of a benchmark and the computation is the parallel part since it can be done locally in each PE. Thus, the scalability of DCT is lower compared to FFT, and this difference in turn results in different performance improvements. This figure demonstrates the performance benefits of using 3D stacking. Assuming a negligible area overhead is imposed by stacking, performance per area is 61% better than a 2D chip on average.

For the 8-core case there are two ways of accessing shared memory for the cores: accessing a single shared memory in the system or each core accessing their local shared memory only. Table II shows the execution times of these two scenarios. For DCT and FFT, execution times differ by 1.3% and 0.8% only. Matrix Multiplication has 1.7% difference between these two situations because of its slightly higher memory access rate. There is a much larger difference (16.9%) for Median Filter because it is the most memory-intensive benchmark.

Finally, we apply memory resource pooling to the Median Filter benchmark. As this benchmark also needs a lot of private memory accesses, four cores running this benchmark do not stress the shared memory sufficiently to reach the memory bottleneck, limiting the performance improvement to 5%. The available benefit from memory resource pooling is proportional to the memory access rate. The memory access rate becomes higher with a larger number of cores on 2D layer and/or by running more memory-intensive applications. When applying resource pooling to more than 2 layers in a 3D system, the benefits are expected to increase as the cores can utilize a larger number of shared memory blocks across different layers.

VI. CONCLUSION

In this paper, we have demonstrated the potential of 3D stacking technology to build a low power multi-core system based on homogeneous stacking. A 3D-MMC prototype has been fabricated, delivering a vertical data bandwidth of 3.2 Gbps. The system uses data serialization during inter-layer communication to reduce the overall number of TSVs. We have also described an optimal way of exploiting the additional resources offered by the 3D stacked system through *memory resource pooling*. In addistion, the paper has shown an analytical approach to evaluate the benefits of resource pooling. For a set of software applications we have designed, 2-layered 3D-MMC achieves up to 61% performance improvement compared to a single layered 4-core chip.

ACKNOWLEDGMENT

This work has been partially funded by the NSF CAREER grant #1149703.

REFERENCES

[1] J. D. Owens *et al.*, "Research challenges for on-chip interconnection networks," *IEEE Micro*, vol. 27, no. 5, pp. 96–108, Sept. 2007.

[2] U. Kang *et al.*, "8 Gb 3-D DDR3 DRAM using through-silicon-via technology," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 1, pp. 111–119, Jan. 2010.

[3] H. Homayoun *et al.*, "Dynamically heterogeneous cores through 3D resource pooling," in *18th International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2012, pp. 1–12.

[4] G. H. Loh and Y. Xie, "3D stacked microprocessor: Are we there yet?" *IEEE Micro*, vol. 30, no. 3, pp. 60–64, May 2010.

[5] T. Thorolfsson, K. Gonsalves, and P. D. Franzon, "Design automation for a 3DIC FFT processor for synthetic aperture radar: a case study," in *46th Annual Design Automation Conference (DAC)*, 2009, pp. 51–56.

[6] G. H. Loh, "3D-stacked memory architectures for multi-core processors," *SIGARCH Comput. Archit. News*, vol. 36, no. 3, pp. 453–464, Jun. 2008.

[7] P. Garrou, C. Bower, and P. Ramm, *Handbook of 3D Integration: Technology and Applications of 3D Integrated Circuits*. John Wiley & Sons, 2008, no. v. 2.

[8] D. H. Kim *et al.*, "3D-maps: 3D massively parallel processor with stacked memory," in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2012, pp. 188–190.

[9] D. Fick *et al.*, "Centip3De: A 3930DMIPS/W configurable near-threshold 3D stacked system with 64 arm cortex-M3 cores," in *ISSCC*, 2012, pp. 190–192.

[10] G. Loh, "A modular 3D processor for flexible product design and technology migration," in *Proceedings of the 5th conference on Computing frontiers*, 2008, pp. 159–170.

[11] J. Zhao, X. Dong, and Y. Xie, "Cost-aware three-dimensional (3D) many-core multiprocessor design," in *DAC*, June 2010, pp. 126–131.

[12] V. Pavlidis and E. Friedman, "3-D topologies for Networks-on-Chip," *IEEE Transactions on VLSI Systems*, vol. 15, no. 10, pp. 1081–1090, Oct. 2007.

[13] Benini, L. et al., "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *Design, Automation Test in Europe Conference (DATE)*, March 2012, pp. 983–987.

[14] G. Beanato *et al.*, "Design and testing strategies for modular 3-d-multiprocessor systems using die-level through silicon via technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 2, pp. 295 –306, June 2012.

[15] Sun, F. et al., "Design and feasibility of multi-gb/s quasi-serial vertical interconnects based on TSVs for 3D ICs," in *18th IEEE/IFIP VLSI System on Chip Conference (VLSI-SoC)*, 2010.

[16] V. Pavlidis, I. Savidis, and E. Friedman, "Clock distribution networks for 3-d integrated circuits," in *Custom Integrated Circuits Conference (CICC)*, Sept. 2008, pp. 651 –654.

[17] K. Skadron *et al.*, "Temperature-aware microarchitecture," in *30th Annual International Symposium on Computer Architecture (ISCA)*, June 2003, pp. 2–13.

[18] J. Meng, K. Kawakami, and A. Coskun, "Optimizing energy efficiency of 3-D multicore systems with stacked DRAM under power and thermal constraints," in *DAC*, June 2012, pp. 648–655.