3D-stacked memory for shared-memory multithreaded workloads

Sourav Bhattacharya Horacio González–Vélez
Cloud Competency Centre, National College of Ireland
Sourav.Bhattacharya@gmail.com, horacio@ncirl.ie

KEYWORDS

3D-stacked memory; memory latency; computer architecture; parallel computing; benchmarking; HPC

ABSTRACT

This paper aims to address the issue of CPU-memory intercommunication latency with the help of 3D stacked memory. We propose a 3D-stacked memory configuration, where a DRAM module is mounted on top of the CPU to reduce latency. We have used a comprehensive simulation environment to assure both fabrication feasibility and energy efficiency of the proposed 3D stacked memory modules. We have evaluated our proposed architecture by running PARSEC 2.1, a benchmark suite for shared-memory multithreaded workloads. The results demonstrate an average of 40% improvement over conventional DDR3/4 memory architectures.

INTRODUCTION

High Performance Computing (HPC) systems typically use custom-built components such as enterprise-grade GPUs, millions of custom CPUs, and above all super-fast interconnection mechanisms for networking and memory. But, despite using the most cutting-edge materials and components available today, it has become clear that one clear limitation relates to the memory-wall challenge, i.e. the imbalance between memory and core/processor performance and bandwidths which has a cascading effect on the overall performance [12], [13]. In fact, memory latency has been identified as a leading cause of overall performance degradation. The fastest memory available today that is used in high performance systems is Error Correcting Code Double Data Rate Synchronous Dynamic Random-Access Memory, also known as *ECC DDR SDRAM*. DDR4 SDRAM, is an abbreviation for Double Data Rate Fourth-Generation Synchronous Dynamic Random Access Memory. It is a type of memory that has a high bandwidth ("double data rate") interface with a latency under 10 nanoseconds [20]. When this latency is magnified by the number of distant nodes, the overall latency of the supercomputing platform increases. This is the scalability challenge where even the smallest of CPU-RAM latency per node becomes magnified thousands of times and thus results in lower performance [26].

In order to address the latency and scalability challenges in HPC, the solution must be one that works at a granular level. While there are multiple sources through which latency is introduced, we aim to address the most relevant cause of lag: the CPU-memory intercommunication latency, with the help of 3D stacked memory.

Conventional DDR RAM has four primary timing indicators which are used to indicate the overall speed of the DRAM. They are as follows:

1. CAS Latency (tCL/tCAS): number of cycles taken to access columns of data, after getting column address.
2. RAS to CAS Delay (tRCD): It is the time taken between the activation of the cache line (RAS) and the column (CAS) where the data is stored.
3. Row Precharge Time (tRP): number of cycles taken to terminate the access to a row of data and open access to another row.
4. Row Active Time (tRAS): number of cycles taken to access rows of data, after getting row address.

These four parameters are cumulatively known as *memory timings*. The motivation behind our research is to study 3D stacked memory architectures which can have significantly faster memory timings than the conventional DDR-4 RAM used in HPC. The future for high-speed memory seems to favour 3D-stacked configurations, as demonstrated by the patent fillings from a number of memory manufacturers[18], [4].

Since the experimental fabrication of multiple new memory modules based on 3D stacked memory is not economically feasible for low yields [27], we will use a combination of simulators—namely DESTINY [25], [23] and CACTI-3DD [8]—in order to create a feasible model of 3D stacked memory. These simulators will help us design and architect the underlying architecture and identify the best possible configuration needed to achieve the highest bandwidth and lowest of latency while keeping in mind the temperature, power consumption, power leakage, area efficiency and other relevant parameters. After a suitable design and architecture sample, satisfying the energy efficiency criteria and other parameters is obtained, we will use it to simulate a full system benchmark using Gem5, a simulator that is a modular, discrete event driven platform for computer architecture, comprising of system-level architecture as well as processor micro-architecture [7]. The overall objective is to model a 3D-stacked memory subsystem, running on top of a generic X86 CPU, and then run a performance benchmark normally used in supercomputing environments to evaluate the performance gains the 3D stacked memory architecture provides when compared with a traditional DDR-4 memory based architecture. Figure 1 describes the high level the approach we have taken to model the 3D stacked memory architecture.

Communications of the ECMS, Volume 34, Issue 1,
Proceedings, ©ECMS Mike Steglich, Christian Mueller,
Gaby Neumann, Mathias Walther (Editors)
ISBN: 978-3-937436-68-5/978-3-937436-69-2(CD) ISSN 2522-2414

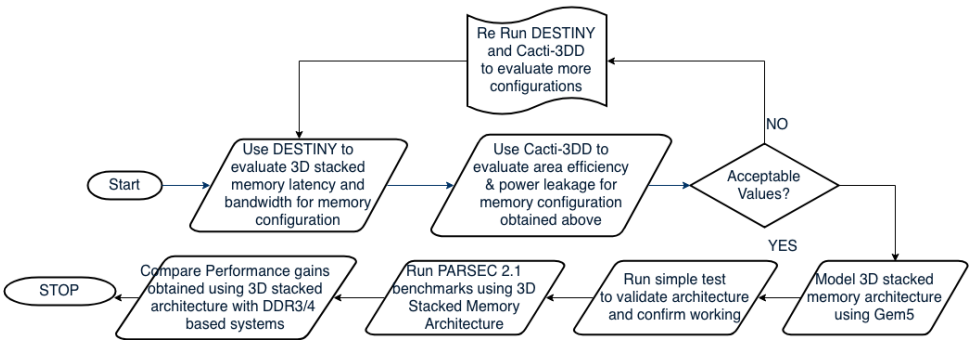


Fig. 1: High Level Approach to Modelling 3D Stacked Memory.

LITERATURE REVIEW

At the turn of this century, researchers focused on making improvements to DRAMs native performance. During any instruction execution, the CPU would have to obtain the next set of instructions from the DRAM. However, as DRAMs locations are off-chip, the apparent attempt was to address delays in accessing data off-chip. By adding components to a chip to increase the memory available in a module, the complexity involved in addressing memory increased significantly [9], [21]. Consequently, the design of a useful interface became more and more challenging. It became evident that communication overhead accounted for some 30% of the DRAM performance, and as a result, moving the DRAM closer to the CPU has become obvious [24].

The next logical step was to try enhancing cache performance, after having failed to solve the latency challenge by improving DRAM performance. The objective was to increase the overall effectiveness of cache memory attached to the processor. Some original ideas in improving cache performance included improving latency tolerance and reduction in cache misses.

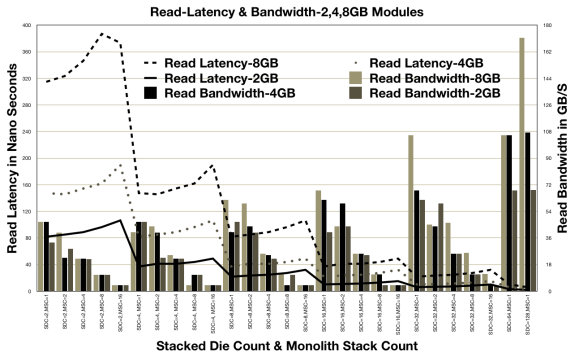
Caching can help improve tolerance by doing aggressive prefetching. However, Simultaneous Multithreading or SMT [11] suggested that the processor could be utilised for other work while it was waiting for the next set of instructions. Other approaches included:

1. Write-buffering: When a write operation to a lower level of cache initiates, the level that generated the write operation no longer has to wait for the process to complete, and can move to the next action. This action can take place assuming there is enough buffer to enable write buffers. This action ensured that a write operation to the DRAM would not result in the latency of DRAM reference.
2. Compression of memory: A compressed memory hierarchy model proposed selectively compressing L2 cache and memory blocks if they could be reduced to half their original size [21]. The caveat with this model was that the overhead for compression and decompression must be less than the amount of time saved by the compression.
3. Critical-word first: Proposed by [17], in the event of a cache miss, the location in the memory that contains the missed word is fetched first. This allows for immediate resolutions to a cache miss. Once the missed word is retrieved and loaded into memory, the rest of the block is fetched and loaded. This allows for faster execution as the critical word is loaded at the beginning.

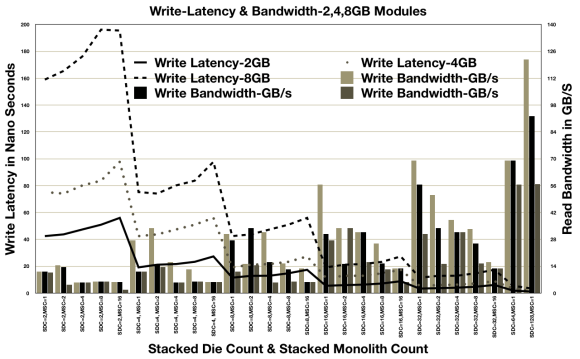
These proposals to improve latency tolerance and improve cache performance worked well when the performance gap between CPU-memory was not extensive. However, some strategies, such as the critical-word first approach did not help much in case the missed word was the first word of the block [1]. In that case, the entire block loading happened the same way it would be loaded typically, which was through contiguous allocation. Similarly, compressing memory did not avoid latency in case of single memory locations, which was the most crucial and significant factor that needed elimination. The only approach that made substantial gains was the write-buffering approach, which was useful if the speed gap between the CPU-memory was huge [10]. The caveat was the size of the buffer, that would keep increasing as the gap in speed between CPU-memory increased.

Similarly, trying to optimise the write performance had minimal use, as most instruction fetch operations are reads. So these approaches also did nothing to reduce the impact of latency. And again, tolerating latency and improving the cache performance becomes extremely difficult when the CPU-DRAM speed differential increases exponentially.After failing to improve the cache performance, researchers looked at reducing the cache misses. A cache miss is when the instruction to be executed is not in the L1 or L2 cache memory and must be retrieved from the main memory [10]. This involves communicating with the main memory, thus invoking the latency issue again. As is evident, by reducing a cache miss, one can bypass the latency tolerance conundrum altogether.

However, improvements in any form of associativity and methods such as memory compression will have much less influence in reducing latency as the cache size keeps growing. Also, managing caches through software will most likely provide the best benefit. Now the biggest challenge in managing cache on a processor using software is the cost involved in losing instructions when using a replacement strategy that is less effective. And as we can see, improving the cache performance is the fastest way to lower the CPU-memory gap in speed. Also, enhancing associativity in cache memory will also provide benefits when leveraged with multithreaded processors, as multithreading and multitasking can hide DRAM latency.



(a) Read-Latency & Bandwidth Comparison.



(b) Write-Latency & Bandwidth Comparison.

Fig. 2: Comparison of StackDie count vs Read-Latency & Bandwidth.

Contribution

To highlight the key elements mentioned above in an approach that will solve the memory wall problem, particularly at an exascale computing level, we are going to detail in the next section our approach to adding 3D stacked memory to a CPU. This architecture will improve the bandwidth, lower the latency while supporting parallelism, associative caching and scalable to the degree that may arguably benefit exascale computing clusters in supercomputers.

METHODOLOGY

In this section, we will describe in detail our approach to create a feasible, energy efficient, working prototype of 3D stacked memory using Destiny, Cacti-3DD, and Gem5.

Using DESTINY to evaluate 3D stacked memory latency and bandwidth

Created as a collaborative design space exploration tool, DESTINY can be used to model 2D and 3D SRAM, eDRAM (enhanced DRAM), STT-RAM (Spin-Transfer Torque Magnetic RAM), and ReRAM (Resistive RAM) [23]. It is a comprehensive modelling tool which can model 22nm to 180nm technology nodes. It can model both conventional and emerging memories, and the results of the modelling are validated against several commercial prototypes. In our case, six parameters have been employed to create a representative 3D-stacked memory architecture: StackedDieCount, PartitionGranularity, LocalTSVProjection, GlobalTSVProjection, TSVRedundancy, and MonolithicStackCount, which are aligned with best practices [22]. We have run a parameter sweep ranging from 128 MB to 8192 MB, and each memory configuration running on varying data bus width, ranging from 32 bits to 1024 bits. For each run, the following output parameters were piped to an output file, with the acceptable values as per the valid range specified in Table I:

TABLE I: Output Parameters and their valid ranges for the memory size

Output Parameters	Valid Range
Timing - Read Latency	<13 nano seconds
Timing - Write Latency	<13 nano seconds
Timing - Refresh Latency	<13 nano seconds
Bandwidth - Read Bandwidth	>40 GB per second
Bandwidth - Write Bandwidth	>30 GB per second

After compiling the results of the test runs and simulations, we observed that by varying the parameters: StackedDieCount and MonolithicStackCount, relevant 3D stacked memory configurations are visible. We then run the results through a bar plot to visualise the results and look for the optimum size memory capacity with low latency a high bandwidth capability, which will display the most appropriate quantity of the 3D stacked memory wafer.

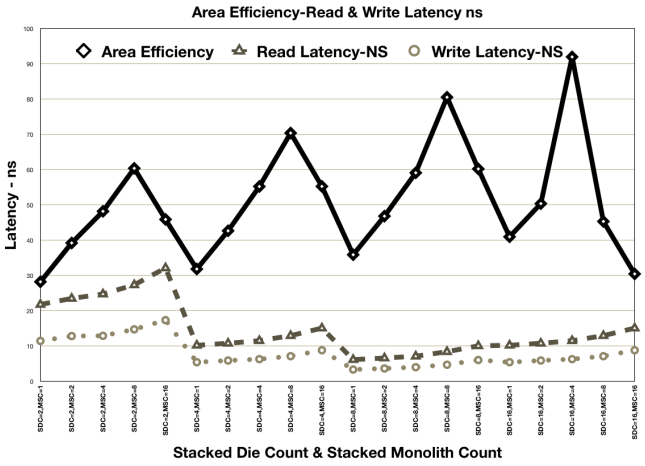
Based on Figures 2a and 2b, the 2GB configuration has been chosen as the prime candidate for the next set of tests with Cacti-3DD, which would help us to model the energy efficiency and power consumption aspects of the selected 3D stacked memory configuration.

Using CACTI-3DD to evaluate Area Efficiency, Power Consumption and Energy Leakage of 3D stacked memory

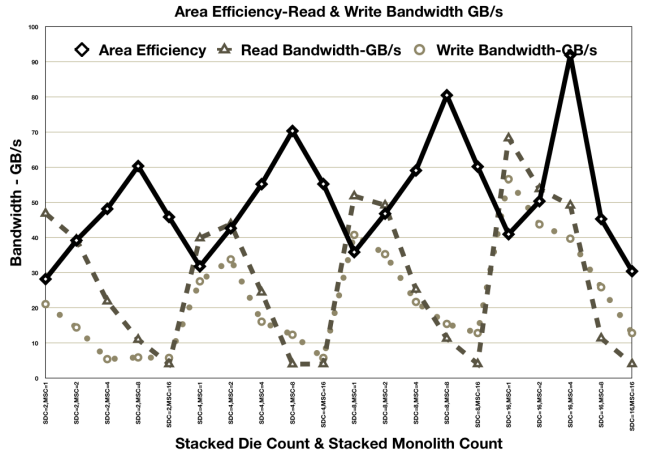
We have then used the 3D-stacked memory configuration obtained from DESTINY to evaluate the area efficiency, power consumption, and thermal efficiency of the 3D stacked memory module. Using a 2GB eDRAM wafer, we had StackedDieCount(SDC)of 1-16 in multiples of 2 and MonolithicStackCount(MSC): 1-16 in multiples of 2. MSC was increased in multiples of 2, from 1,2,4,8,16,32, and for each MSC count, SDC count was kept constant. For each configuration, Cacti-3DD was run to evaluate the area efficiency of the proposed configuration, along with power consumption, power leakage and thermal efficiency. In our case, area efficiency of $< 100\%$ and Power-Leakage < 1 Watt.

For the chosen 2GB memory module, the area efficiency and power leakage parameters have been evaluated in order to find the most energy efficient design, with the lowest power leakage. We see a sharp drop in area efficiency after the SDC & MSC with counts up to 16, as displayed in Figures 3a and 3b respectively. We then overlay the area efficiency plots over the power leakage plots, in order to find the SDC & MSC configuration that has the highest bandwidth and lowest latency.

When we evaluate the power leakage parameter and compare it with the area efficiency parameter, with increasing SDC & MSC, we observe that the area efficiency

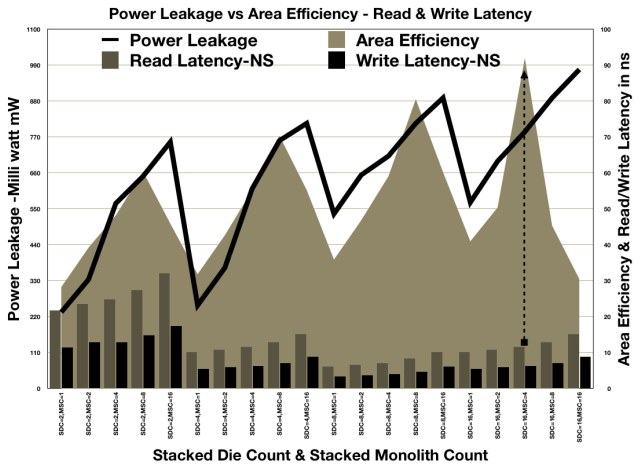


(a) Read/Write-Latency Comparison.

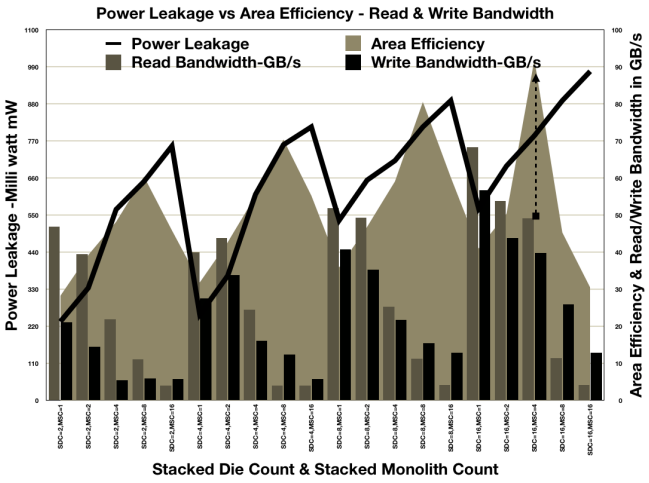


(b) Read/Write-Bandwidth Comparison.

Fig. 3: Efficiency and power leakage evaluation



(a) Power Efficiency-Power Leakage-Latency



(b) Power Efficiency-Power Leakage-Bandwidth

Fig. 4: Power Efficiency-Power Parameterisation for Leakage-Latency and Leakage-Bandwidth

is high at 91% with power leakage under 1 Watt, when the SDC & MSC count is at 16 and 4, as is evident in Figures 4a and 4b. And while this configuration isn't very low in terms of energy leakage, it does showcase the lowest power leakage combined with the lowest latency and highest bandwidth. Hence, we choose this layout for the actual design specification section. In the next section, we will showcase the actual design specification of the 3D stacked memory, based on the configuration parameters detailed above, in the Gem5 simulator. After a successful 3D stacked memory configuration is obtained in Gem5, we will then run a simple hello-world test using Gem 5 to validate the architecture and the simulation of the chip.

Based on the figures 3a, 3b, 4a and 4b, the 2GB candidate with the following specifications using Cacti-3DD and DESTINY was selected for the next stage of design specification using gem5:

```
Bank Organization: 256 x 256 x 8
 - Row Activation    : 1 / 256 x 1
 - Column Activation: 1 / 256 x 1
Mat Organization: 1 x 2
 - Row Activation    : 1 / 1
 - Column Activation: 1 / 2
 - Subarray Size     : 32 Rows x 512 Cols
Mux Level:
 - Senseamp Mux       : 1
 - Output Level-1 Mux: 1
 - Output Level-2 Mux: 2
Local Wire:
 - Wire Type : Local Aggressive
 - Repeater Type: No Repeaters
 - Low Swing : No
Global Wire:
 - Wire Type : Global Aggressive
 - Repeater Type: No Repeaters
 - Low Swing : No
Buffer Design Style: Latency-Optimized
Area:
 - Total Area = 3.93101mm x 39.1354mm
 - Mat Area = 15.3555um x 152.872um
 - Subarray Area = 15.3555um x 73.9001um
 - TSV Area = 1.96um^2
 - Area Efficiency = 91.9679%
Timing:
 - Read Latency = 11.5334ns
 - Write Latency = 6.31317ns
 - Refresh Latency = 3.40672us
 - Read Bandwidth  = 49.2234GB/s
 - Write Bandwidth = 39.741GB/s
Power:
 - Read Dynamic Energy = 294.026pJ
 - Write Dynamic Energy = 293.86pJ
 - Refresh Dynamic Energy = 7.16835uJ
 - Leakage Power = 786.093 mW
```

DESIGN SPECIFICATIONS

A. *Creating a basic CPU with Cache & Memory Controller in Gem5*

When trying to architect the basic 3D-stacked Wide I/O DRAM memories, we are faced with three massive changes to be modelled:

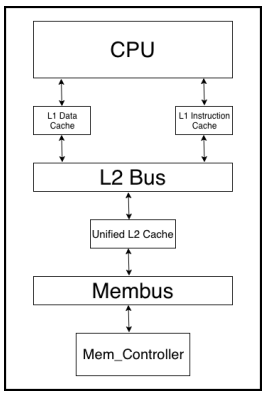


Fig. 5: Simple CPU Layout in Gem5.

1. How to enable 3D stacking of DRAM dies with the help of Through Silicon Via (TSV) interconnects.
2. How to support a minimum of four independent memory channels.
3. How to extend I/O interfaces to x128 bits per channel

Now, when compared to conventional DRAM, a 3D-stacked DRAM architecture offers increased memory bandwidth plus improved energy efficiency, due to the largely increased I/O interface width and significantly reduced I/O power consumption. The reduction in power consumption is achieved by stacking DRAM dies using low capacitance TSVs, compared to the traditional horizontal organisation of DRAM chips on one single plane.

We begin by creating a simple CPU in gem5, with standard CPU core connected to a system-wide memory bus. Please note that setup, installation and configuration of gem5 are not covered in this section. With that, we will connect a simple DDR3 memory channel, also connected to the bus. As gem5 is a modular design simulator, most of its components can be simulated as Sim-Objects such as CPUs, memory controllers, cache memories, buses etc. So we import all the SimObjects from the m5 library, instantiate the system we are going to simulate, set the clock on the system, specify a voltage domain. Next, we need to simulate the memory timings mode for the memory simulation. We also create a simple CPU timing based CPU which executes one single instruction in one clock cycle and then create a memory bus that is connected system-wide. Connecting the cache ports to the CPU, such as the instruction cache port and the data cache port, is the next step. A system cannot function without an interrupt controller, so we create the I/O controller and connect it to the memory bus. So we create a special port in the system to allow the system to read and write memory.

The next step is to create a memory controller and connect it to the memory bus. We use a simple DDR3 controller and connect it to the membus. This creates a system with no caches, temporarily. The next step is to add caches to the CPU model. We create caches separately and import them into the main CPU model file. We create L1 cache, instruction and data, and give it parameters such as associativity, tag_latency, data_latency, response_latency, miss status handling registers etc. Next, we instantiate each cache type, such as L1 data and L1 instruction, and add a size value to each, 16kB for L1 Instruction and 64kB for L1 Data. Similarly, we create another L2 cache with similar parameters and size 256kB. Now, we need to instantiate the caches and connect them to the interconnect. Once this is done, the process that the CPU needs to execute needs to be set up. Gem5 operates in two modes, syscall emulation mode (SE mode) and full system mode (FS mode).

For now, we will run our system in the SE mode by providing it with a pre-compiled executable. We use a simple "hello world" C program, after we create a process and set the process command to the command we want to run, we set the CPU to use the process as its workload and create the functional execution context in the CPU.

The CPU created by the steps mentioned above is detailed in Figure 5. This CPU is our reference CPU which will be later used to run the PARSEC benchmarks, by leveraging the 3D stacked memory architecture. The objective is to create a CPU model Gem5 understands, and then give it a memory subsystem it can use to load and store data, just like a regular CPU uses DRAM to load and store data.

IMPLEMENTATION

With inputs from [3], [2], [19], [16], we set out to create the 3D stacked memory architecture in Gem5. The overall architecture that was created is displayed below in Figures 6 and 7. This memory architecture was created in a separate file, and was imported into the CPU created in the previous section. It uses the following components unique to the 3D stacked memory architecture: Vault Controllers, Serial Links, Internal Memory Crossbars.

The 3D stacked configuration is arranged in layers of DRAM wafers, each layered on top of the other, and connected to each other with the help of TSVs or Through Silicon Vias. A vault is a vertical interconnect across the four layers, each layer containing 128 MB of DRAM, thus creating a vault size of 512 MB. This can be increased or decreased by adding or removing more layers of DRAM stacked on top of each other. The logic layer and the 3D stacked memory crossbar sits under the base layer of DRAM, and provides routing and access to the vaults. The crossbar helps vaults connect to each other. The system designed here contains four DRAM layers and one logic layer die. Within each cube, the memory is organised and stacked vertically. Each vault has a vault controller that manages memory operations such as timing, refresh, command sequencing. Each vault can be configured with a maximum theoretical bandwidth of 10GB/s, thereby giving the 3D stacked architecture with 8GB of memory a total bandwidth of 160GB/s, which is possible using 2GB wafers with 40GB/s bandwidth, as we have explained previously. After a vault is accessed, we have configured a delay which prevents the vault from being accessed again, just like in regular DRAM. Each crossbar is assigned 4 vaults considered to be local. The other 12 vaults are considered remote, and can be accessed with the help of the crossbar switch. The 3D stacked memory must be refreshed periodically, and it is handled internally by the vault and logic controller.

At the bottom of the 3D stacked memory, we have the

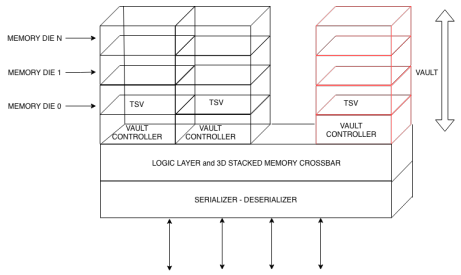


Fig. 6: 3D Stacked Memory Design Architecture

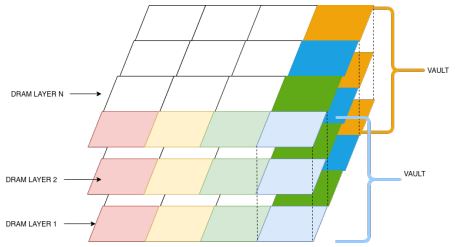


Fig. 7: 3D Stacked Memory - Vaults & Layers

serializer and deserializers, and the high speed links that are comprised of 16 different lanes that can be used to transmit in both transmit and receive directions. Therefore, a theoretical maximum bandwidth of 320 GB/s can be attained with the help of this 3D stacked architecture. Each lane can be configured to run in a full width or half width configuration. From a programmable perspective, the lanes can run in a 16X16, 16X8, 8X16, or 8X8 lane configuration. Figure 8 indicates the relationship between the local vaults, quadrants, crossbars and the remote vaults. The 3D stacked memory architecture created with the parameters and configurations previously described, was compiled and run in Gem5.

EVALUATION

In order to test and evaluate our 3D stacked memory based CPU architecture, we need to run our custom CPU on a clean system that does not have any additional software on it. Table II reports the performance of the custom architecture running PARSEC[5], [6]. We have compared the run times and execution times for the benchmarks using our custom architecture against a standard DDR3/DDR4 memory based system at DDR3/4-

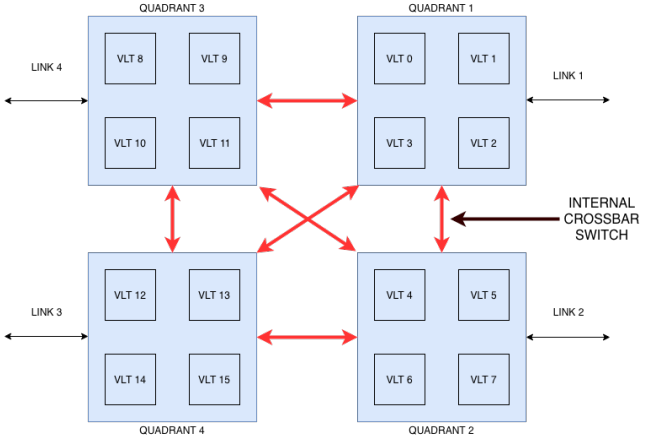


Fig. 8: Vaults, Quadrants and Crossbars

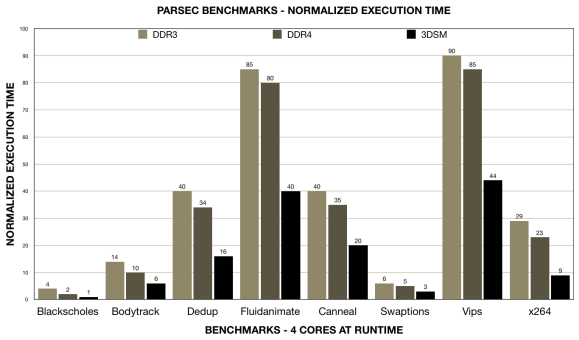


Fig. 9: PARSEC - Multi Core Normalized Execution Times, DDR3/DDR4/3D Stacked Memory.

Fig. 10: PARSEC - Normalized Execution Times, DDR3/DDR4/3D Stacked Memory.

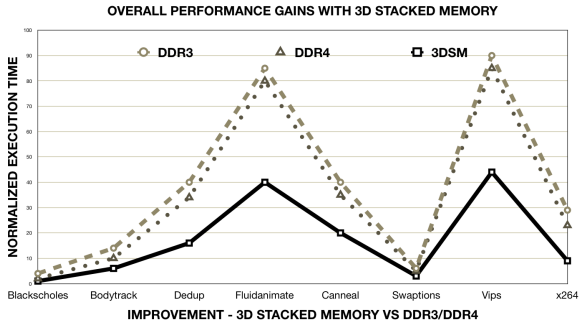


Fig. 11: Performance Gains & Improvement over DDR3/4 memory

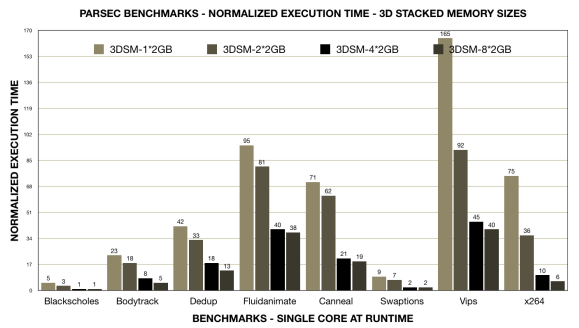


Fig. 12: PARSEC Benchmark Execution times compared by size - 3D Stacked Memory Sizes

TABLE II: Performance Gains - Single and Multi Core PARSEC Tests

Benchmark	Improvement-Single Core	Improvement-Multi
blackscholes	67%	67%
bodytrack	43%	50%
dedup	60%	56%
fluidanimate	55%	51%
canneal	51%	46%
swaptions	63%	45%
vips	50%	49%
x264	63%	65%

3200 with 15-15-15-35 (2T) timings, running the same X86 CPU running at 3.0 GHz.

Figure 10 shows the normalised execution time for 8 of the 13 benchmarks available in PARSEC. The runtimes were compiled for each memory system, using a reference x86 CPU at 3 GHz. Memory capacity was set to 8 GB for each memory type. As is evident from the results, there's not much of a difference between DDR3 and DDR4 in terms of execution speed. However, when we look at our 3D stacked memory architecture, there is an average of 40%-60% reduction in execution time for each benchmark. The benefit is especially evident in memory intensive benchmarks such as Fluidanimate and Vips, both of which are data-parallel models with medium working set data. This was the result of a single core execution, where the reference CPU was launched with just one core running at 3 GHz.

The next set of tests have been executed with multiple cores being launched at runtime. As some of the benchmarks leverage multi threading and multiple cores as well, observing the performance in a multi CPU and multi threaded environment would be extremely relevant. By using the -n switch to specify number of CPUs, we were able to simulate a multi CPU environment. The performance difference, while not hugely different from the single CPU benchmark result, still indicates that some benchmarks are inherently more CPU dependent than memory dependent. In Figure 9, we see the results of using 4 cores assigned to each CPU at run time. As before, all results were normalised according to the execution time of the target architecture with main memory implemented with DDR3. As is evident from the results of two benchmarks displayed in Figure 10 and 9, the 3D stacked memory configuration displays a significant improvement in performance in industrial benchmarks, while delivering improved read & write bandwidth, lower latency than traditional DDR3 and DDR4 memory, while satisfying the area efficiency, power consumption and temperature parameters. By comparing the normalised execution times, we observe the following gains in performance over conventional DDR-3/DDR-4 based memory systems.

For each of the benchmarks evaluated in this paper, performance gains visible, as displayed in Table II and Figure 11, by providing the reference 3.0 GHz CPU with a 3D stacked memory architecture varies from a minimum of 43% in a single-core environment to a maximum of 67% in a multi-core environment. This is a definitive indication that supercomputing and indeed the memory wall challenge will benefit from 3D stacked memory. By observing significant reduction in the execution times of supercomputing standard benchmarks, we can confidently say that the overall performance gains achievable with 3D stacked memory in the configuration detailed in this paper should be a step in the right direction towards achieving exascale levels of computing.

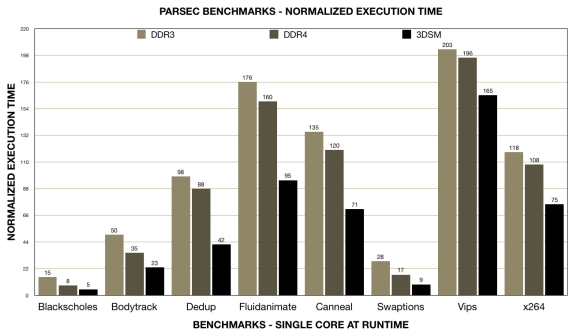


Fig. 13: PARSEC Benchmark Execution times - 2GB - 3D Stacked Memory

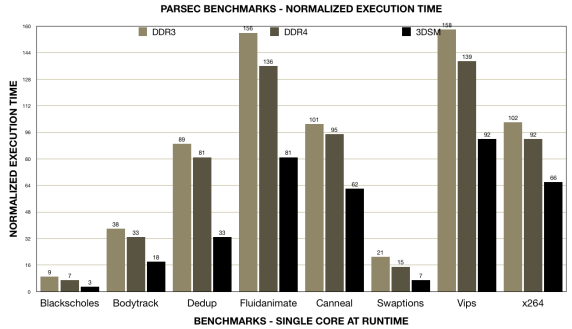


Fig. 14: PARSEC Benchmark Execution times - 4GB - 3D Stacked Memory

We have also run the same set of benchmarks by modifying the 3D stacked memory size. By altering the number of stacked monoliths, we have evaluated the performance of 3D stacked memory for sizes ranging from 2GB, with just one 3D stacked memory wafer, to 16GB, comprising of 8 3D stacked memory wafers, each of size 2GB. The normalised execution times are displayed in Figures 12, 13, 14, and 15. We see conclusive evidence, especially in memory size intensive benchmarks such as vips & x264 encode, where higher the memory size, the lesser is the execution time.

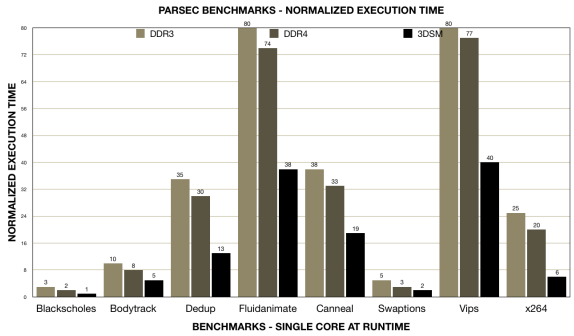


Fig. 15: PARSEC Benchmark Execution times - 16GB - 3D Stacked Memory

CONCLUSIONS AND FUTURE WORK

As we approach exascale levels of computing, we have realised that not just compute, but every single aspect of computing needs to scale massively in order to deliver the expected performance boost. This includes storage, memory, interconnects, space, power consumption and cooling. Not to mention the availability of structured parallel programming frameworks [14], [15] and administration as well. This paper has taken on one of the many challenges we as a race face in achieving and breaching exascale levels of computing. This paper has taken on the challenge of implementing a full scale 3D stacked memory architecture, creating a workable X86 architecture that is compatible with existing CPUs and benchmarks capable of evaluating hardware at supercomputing levels. The results look promising to say the least, with significant improvements visible in memory-intensive benchmarks such as Fluidanimate and Vips, and the system also looks capable of scaling and performing under multi-core environments as well.

This exercise intends to showcase that existing challenges in computing require a different, non-conventional approach such as 3D stacking. The future work on this topic would be to incorporate a machine learning algorithm to evaluate the results of multiple memory sizes, multiple cores, on the PARSEC benchmarks and run the benchmarks at a proper computationally-demanding environment in order to see how much gains are possible in terms of FLOPS. Not just supercomputers, but cloud computing will also benefit from the 3D stacked memory architecture, as many cloud service providers today provide custom instances tailored to running memory intensive workloads. CSPs such as AWS provide X1 instances that are custom built and designed for large-scale and in-memory applications in the cloud, which will benefit tremendously from leveraging a 3D stacked memory architecture.

By providing additional bandwidth, lowered latency, increased and efficient power consumption metrics for systems leveraging 3D stacked memory, cloud service providers will be able to provide high-performance instance capable of running memory intensive workloads such as running in-memory databases such as SAP HANA, big data processing engines like Apache Spark or Presto, and HPC applications. The potential benefits obtainable from such instances will go a long way in providing cheap, high-performance compute platforms to end users.

REFERENCES

[1] K. Ahmed, J. Liu, A. Badawy, and S. Eidenbenz. A brief history of HPC simulation and future challenges. In *2017 WSC*, pages 419–430, Las Vegas, Dec. 2017. IEEE.

[2] J. Ahn, S. Yoo, and K. Choi. Low-power hybrid memory cubes with link power management and two-level prefetching. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(2):453–464, 2016.

[3] E. Azarkhish et al. High performance AXI-4.0 based interconnect for extensible smart memory cubes. In *2015 DATE*, pages 1317–1322, Grenoble, Mar. 2015. IEEE.

[4] Z. Z. Bandic and other. Multilayer 3D memory based on network-on-chip interconnection. US Patent US 10,243,881 B2, Western Digital Technologies, Irvine, CA, Mar. 2019.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT '08*, pages 72–81, Toronto, Oct. 2008. ACM.

[6] C. Bienia and K. Li. PARSEC 2.0: A new benchmark suite for chip-multiprocessors. In *MoBS 2009*, pages 1–9, Austin, June 2009.

[7] N. Binkert et al. The Gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[8] K. Chen et al. CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory. In *2012 DATE*, pages 33–38, Dresden, Mar. 2012. IEEE.

[9] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. High-performance drams in workstation environments. *IEEE Transactions on Computers*, 50(11):1133–1153, 2001.

[10] R. Das. Blurring the lines between memory and computation. *IEEE Micro*, 37(6):13–15, 2017.

[11] S. J. Eggers et al. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, 17(5):12–19, 1997.

[12] A. Geist and D. A. Reed. A survey of high-performance computing scaling challenges. *Int. J. High Perform. Comput. Appl*, 31(1):104–113, 2017.

[13] S. Ghose, T. Li, N. Hajinazar, D. S. Cali, and O. Mutlu. Demystifying complex workload-dram interactions: An experimental study. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(3), Dec. 2019.

[14] M. Goli and H. González-Vélez. Formalised composition and interaction for heterogeneous structured parallelism. *Int. J. Parallel Program.*, 46(1):120–151, 2018.

[15] H. González-Vélez and M. Leyton. A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers. *Softw. Pract. Exp.*, 40(12):1135–1160, 2010.

[16] R. Hadidi et al. Demystifying the characteristics of 3D-stacked memories: A case study for Hybrid Memory Cube. In *2017 IISWC*, pages 66–75, Seattle, Oct. 2017.

[17] J. Handy. *The Cache Memory Book*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[18] J. Hopkins and other. 3D memory. US Patent US 10,170,639 B2, Micron Technology, Boise, ID, Jan. 2019.

[19] G. Kim et al. Memory-centric system interconnect design with hybrid memory cubes. In *PACT '13*, pages 145–156, Edinburgh, 2013. IEEE.

[20] D. Lee et al. Design-induced latency variation in modern DRAM chips: Characterization, analysis, and latency reduction mechanisms. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1):26:1–26:36, June 2017.

[21] W.-F. Lin. Reducing DRAM latencies with an integrated memory hierarchy design. In *HPCA '01*, pages 301–, Monterrey, Jan. 2001. IEEE.

[22] S. Mittal. A survey of architectural techniques for managing process variation. *ACM Comput. Surv.*, 48(4):54:1–29, Feb. 2016.

[23] S. Mittal, R. Wang, and J. Vetter. DESTINY: a comprehensive tool with 3D and multi-level cell memory modeling capability. *J. Low Power Electron. Appl.*, 7(3):23, 2017.

[24] H. A. D. Nguyen et al. A classification of memory-centric computing. *J. Emerg. Technol. Comput. Syst.*, 16(2):13:1–26, Jan. 2020.

[25] M. Poremba et al. DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches. In *2015 DATE*, pages 1543–1546, Grenoble, Mar. 2015. IEEE.

[26] M. Qureshi. With new memories come new challenges. *IEEE Micro*, 39(1):52–53, 2019.

[27] C. Weis, M. Jung, and N. Wehn. 3D stacked DRAM memories. In P. Franzon, E. Marinissen, and M. Bakir, editors, *Handbook of 3D Integration: Design, Test, and Thermal Management*, volume 4, chapter 8, pages 149–185. Wiley, Weinheim, 2019.