# A 3D-Stacked Memory Manycore Stencil Accelerator System

Jiyuan Zhang, Tze Meng Low, Qi Guo, Franz Franchetti
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA, USA
Email: {jiyuanz, lowt, qguo1, franzf}@andrew.cmu.edu

***Abstract*—Stencil operations are an important class of scientific computational kernels that are pervasive in scientific simulations as well as in image processing. A key characteristic of this class of computation is that they have a low operational intensity, i.e., the ratio of the number of memory accesses to the number of floating point operations it performs is high. As a result, the performance of stencil operations implemented on general purpose computing systems is bounded by the memory bandwidth. Technologies such as 3D stacked memory can provide substantially more bandwidth than conventional memory systems and can enhance the performance of memory intensive computations like stencil kernels. In this paper, we leverage this 3D stacked memory technology to design an accelerator for stencil computations. We show that for the best efficiency one needs to find the balance between computation and memory accesses to keep all components consistently busy. We achieve this by exploring how blocking and caching schemes to control the compute-to-memory ratio. Finally, we identify optimal design points that maximize performance.**

## I. Introduction

Stencil operations are an important class of kernels in the scientific domain. They constitute an integral component of finite differential methods to solve partial differential equations, which are applied in a wide range of scientific areas like heat diffusion, fluid dynamics, and electromagnetics[1], [2]. Stencil kernels are also ubiquitous in image processing in performing image denoising and variational models [3].

Implementing stencil kernels on general purpose computers is a challenging task. Stencil computations perform iterative operations on a large data set and feature a low computational intensity, i.e., the amount of floating point operations (FLOPs) is small compared to the memory access operations [4]. As a result, the performance of stencil kernels on general computing system is severely hampered by the memory bandwidth.

Software and compiler techniques have been developed to improve the performance of stencil computation. The idea is to exploit the data locality characteristics of the stencil computation to enhance the compute-to-memory ratio [5], [6], [7], [8]. This is helpful to increase the number of operations performed per memory access. Yet it can only compensate for the low bandwidth of the general purpose systems to some extent. Stencils in higher dimensional grids which demand more memory bandwidth or stencils in multigrid methods which possess less locality or require fewer iterations are still hard to optimize.

The recently emerging 3D stacked memory promises to deliver higher bandwidth and thus a solution to the memory bottleneck problems. Using this technique, memory chips are stacked vertically and connected using short and fast *Through-Silicon-Vias* (TSVs), resulting in substantially higher bandwidth than conventional memory systems. In addition, 3D stacking enables the integration of logic dies with memory dies. This feature allows the implementation of computation near the memory for memory access latency reduction and higher bandwidth. For data-intensive but logic-simple computation, this near data computing system can provide orders of magnificient improvement on performance and power efficiency [9], [10]. In this work, we leverage the 3D-stacked memory system to accelerate stencil computations. The contributions are as follows:

- **A many-core stencil accelerator.** We propose a many-core parallel stencil accelerator integrated on the logic layer of a 3D-stacked memory system. The accelerator architecture is designed for better parallelism and to fully exploit the internal bandwidth of the stacked memory system. The 3D stacked accelerator system overall is able to reach orders of magnitude improvement on the performance compared to general computing systems.
- **Computation and bandwidth tradeoff.** In order to achieve the peak floating point computational capability we show that a balance between computation and memory access is necessary. We analyze how spatial and temporal blocking schemes affect the compute-to-memory ratio and system performance.
- **Design space exploration.** Based on the above analysis, we perform design space exploration to identify optimal design points that maximize performance under different system configurations.

The paper is organized as follows. Section II provides the background of stencil computation and 3D stacking memory. Section III elaborates on the details of stencil accelerator system. Section IV introduces the model for performance estimation. The experiments are discussed in Section V and we conclude in Section VI.

## II. Background

### A. Stencil Operation

Stencil operations are an important motif in scientific computation. The computation involves iteratively updating every element in a data grid by using its neighboring elements. One round of updates across the entire grid is called one time

Listing 1: Example of 2D-5 point Jacobi stencil code

```
for (t=0; t<Tmax; t++)
   for (i=0; i<Xdim; i++)
      for (j=0; j<Ydim; j++) {
        A[t+1][i][j]=a*A[t][i][j] + b*(
            A[t][i-1][j] + A[t][i][j-1] +
            A[t][i][j-1] + A[t][i][j+1]);
}
```

step. Different stencils may use a different number of elements for updating. Listing 1 shows an example of a Jacobi 2D 5-point stencil. The outer loop is the time loop. Within one time iteration, the code sweeps through the entire 2-dimensional data domain, updating each element using its North, East, West and South nearest neighbors. This code is common in finite differential methods for partial differential equations. There are also other Jacobi patterns, such as 3D 7-point stencil which updates elements on a 3-dimensional grid using neighbors at $x-1$, $x+1$, $y-1$, $y+1$, $z-1$, $z+1$ directions, or 3D 27-point stencil which uses all the elements at the corner, face and edge of the surrounding $3 \times 3 \times 3$ cube.

### B. 3D Stacked Logic-in-Memory system

3D stacked DRAM has become a reality with the commercial availability of the Micron hybrid memory cube (HMC). The 3D-stacked DRAM adopts fine-grained rank-level stacking [11], i.e., banks stacked in 3D fashion to form a 3D rank (vault). This can better exploit bank-level parallelism. The vertically-stacked banks in a rank are connected by shared through-silicon-vias (TSVs). The system is composed of ranks equal to the number of banks in a die. On the bottom of the 3D stacked system is the logic die. Peripheral logic is implemented on the logic die. The peripheral logics include the vault and link controllers, which are in charge of moving data between vaults or from vaults to external IO pins. Each vault controller is associated with a vault and is placed under its corresponding vault.

## III. Accelerator Design

In this section, we start with an overview of the many-core stencil accelerator system. Then we explain how the stencil computation is performed on this accelerator with the example of 2D 5-point stencil.

### A. System Overview

As wire technology keeps scaling to smaller feature sizes, wire delay has become a more critical issue. Utilizing clustered and simple processing units is an effective way to mitigate the wire issue. Our design reflects this idea. The proposed stencil accelerator is composed of many small and simple accelerating cores as shown in Fig. 1. Multiple cores are grouped into a cluster. These clusters form a 2-dimensional array that is of the same size as the vault array. Each cluster is mapped to a vault. FIFO channels are built across neighboring cores as well as neighboring clusters to facilitate data transferring between neighbors. This many-core structure not only has shorter wiring than a large monolithic core, moreover, it also removes complex communication interconnections between

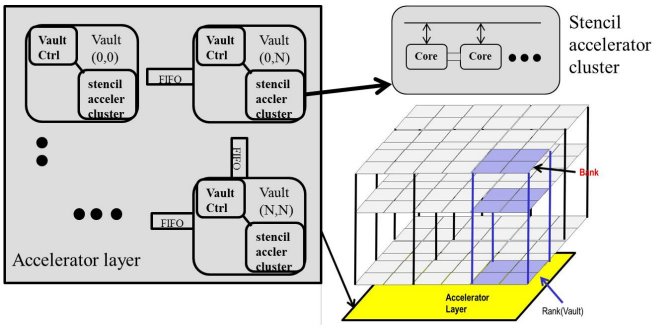


Fig. 1: 3D stacked memory system overview.

the accelerator and vault controllers [12], better exploiting vault-level parallelism.

### B. Data Partition

To perform a stencil computation on this accelerator system, the computation needs to be partitioned and mapped properly to each core.

**Vault partition.** The original data region is partitioned into multiple small regions that are processed separately on each cluster. These smaller regions are called *vault regions*. To facilitate the mapping of vault region to vaults, the partitioning of the original space is organized into the same structure of the vault array: the two dimensions of the data space are partitioned evenly to form an N-by-N array of vault regions.

### C. Blocking

Spatial and temporal blocking (or tiling) are effective techniques to optimize data reuse and reduce memory access in stencils. In this section, we describe how the space and temporal blocking is performed in the vault region.

**Spatial blocking.** Spatial blocking is an effective technique to improve data reuse when the data size is larger than the cache size. In our stencil accelerator, spatial blocking is applied to transform the the vault region to small blocks that can fit the local cache of the cluster. The block is referred as *cluster block*. Unlike the blocking in matrix multiplication, blocking scheme in stencils is different and called partial blocking. The cluster block is blocked on $N-1$ dimensions (assume the vault region is N-dimension) and takes stride 1 at the unblocked dimension. The unblocked dimension is the *streaming dimension*: The partial blocks are streamed into the accelerator cluster along the streaming dimension. The accelerator cluster finishes one cluster block before processing another. Multiple partial blocks are stream buffered on chip, because of data reuse between consecutive blocks.

Fig. 2 shows the partition and blocking for the 2D 5-point stencil problem. The original 2D data space is partitioned into a 2D array of vault regions. In a vault region, the cluster block is formed by partially blocking the $x$ dimension. The resultant cluster block is a one dimension row. A core block is a chunk of the row assigned to each core. The blocks are streamed into the accelerator cluster along the $y$ dimension. Stencil operations are performed on every element of the block. The results are streamed out to memory in blocks. Since the 5-point stencil operation involves elements of three different $y$

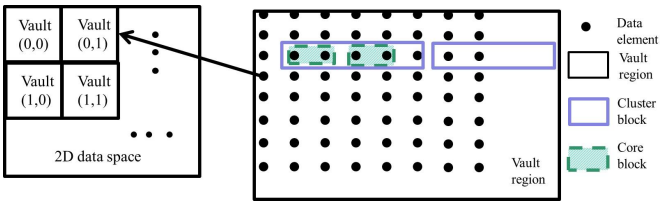


Fig. 2: Data space partition for 2D stencil.

indexes, they are scattered across three consecutive blocks. The computation on the latest streaming block requires the previous two blocks, which are thus buffered on chip. A hardware circular buffer can be adopted to implement the streaming buffer function.

**Temporal blocking.** In addition to spatial blocking, we can further reduce the memory bandwidth requirement by performing time blocking. Compared to spatial blocking which exploits data reuse only at spatial iterations, temporal blocking further reduces memory bandwidth by leveraging data reuse across time iterations. The idea of temporal blocking is as follows. Spatial blocking computes results of one time step based on the data read from memory. Instead of writing the computed results immediately to memory, they are kept in a cache and are reused to compute results of later time steps. Only the data at the last time blocking step is written to memory. Fig. 3 shows the time blocking process with a blocking parameter of four. Temporal blocking requires a larger cache to keep the intermediate blocks. It has to keep in cache the two latest blocks for every time step. The blocks are stream buffered similarly as in the spatial blocking method.

**Communication.** Cores have to communicate with their neighbors to get elements to support computing at the boundary region, so called ghost region. There are two circumstances that need communication. *Inter-core* communication happens when cores process elements at the *core block* boundary. *Inter-vault* communication happens when processing elements at the vault region boundary. *Inter-core* communication happens whenever the core reaches the last few elements of each row. *Inter-vault* communication happens after the accelerator has finished computing the entire vault region for a certain time (based on the time blocking parameter). The *Inter-vault* communication volume depends on the time blocking parameter. The larger the time blocking parameter, the larger the ghost region that needs to be transferred between clusters.

### D. Banking Scheme

Temporal blocking requires caching the values of elements at intermediate time steps. The intermediate values of one element cannot be stored at the same location, since this will result in a new value at a later time step replacing the old value that are still required. To avoid this, the storing of the intermediate values should be staggered. For instance, in a 2D stencil the updated value of element (x, y) (denoted as $(x, y)_{t+1}$) cannot be stored at the same place as the old value $(x, y)_t$, because computing a later element ( e.g., $(x, y+1)_{t+1}$) will need the old value of $(x, y)_t$. But $(x, y)_{t+1}$ can be stored at the location of $(x-2, y-2)_t$ because $(x-2, y-2)_t$ will no longer be used by the later element. Generally a value at a later time step is stored with a staggerred distance equal to the stencil operation range.

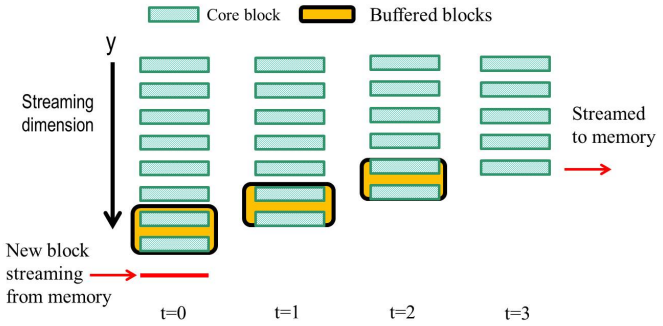


Fig. 3: Temporal blocking of 2D stencil for four time steps. The blocks of intermediate time steps are cached in local SRAM. Each time step stores two latest blocks.

Another problem in the stencil computation is the simultaneous data access. One stencil operation requires multiple elements as inputs and whether they can be accessed simultaneously is critical to the performance. Utilizing multiple memory banks and mapping the data points to disjoint banks is an effective way to eliminate the access conflicts. In the 2D 5-point stencil example, one stencil operation requires five elements as input and outputs one element every iteration (e.g., the stencil operation on element $(x, y)$ requires read from five locations $(x, y)(x-1, y)(x+1, y)(x, y-1)(x, y+1)$ and write to one location $(x-2, y-2)$ ). But $(x-1, y)$ and $(x, y)$ are overlapping inputs with previous operations. Therefore only locations of $(x+1, y)(x, y-1)(x, y+1)$ need to be read. Therefore a 4-banked scratchpad SRAM is adopted here, where consecutive cluster blocks are stored at different banks. In this way, all the reads and writes of one stencil operation can be simultaneously performed.

### E. Accelerator Core Architecture

Fig. 4 shows the core architecture customized for a given stencil computation. The architecture consists of the scratchpad SRAM, a computation unit and a control module. The scratchpad SRAM is composed of multiple banks. The bank number is determined by the specific stencil pattern. It is equal to the operand number of one stencil operation excluding the operands overlapping with previous operation. The computation logic is the floating point computation logic for this stencil pattern. The control of these functional units is driven by the controller. The controller is implemented as state machines whose transitions are based on the stencil context registers in the sequencer unit. The control signals to all these functional units are encoded as microcodes and stored in the micoprogram memory.

## IV. Performance Analysis

Best performance is achieved when the computation's operation-to-memory ratio matches with the system. In this section, we model and analyze the performance of the accelerator system and identify the best design points.

### A. Performance Model

We first explain the relevant parameters for estimation.

**Problem parameters.** The stencil problem can be represented with parameters $D$, Dim, $R$ and $P$. The $D$ is the

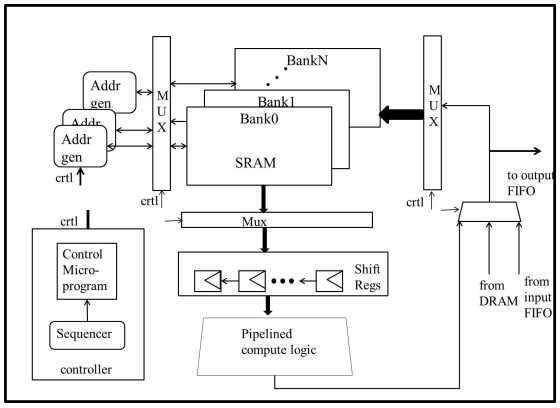


Fig. 4: Stencil accelerator core architecture.

dimension of the stencil. $\mathrm{Dim}$ is the size of each dimension. $R$ is the radius of the stencil operation region (i.e., the distance from the farthest element to the center). $P$ is the total number of elements used to compute a new value.

**Blocking parameters.** $B_{\mathrm{vault}}$ denotes the size of the vault region. The original data space is partitioned into 2 dimensional array of vault regions and each vault region is mapped to a vault. The partition is performed on 2 dimensions out of the $D$ dimensions with the parameter $B_{\mathrm{vault}}$.

$B_{\mathrm{cluster}}$ denotes the cluster block parameter. The vault region is usually larger than the local SRAM size of the accelerator cluster. The vault region is partitioned into smaller cluster blocks that can fit into the scratch SRAM size. The blocking parameter is $B_{\mathrm{cluster}}$. The dimension that is not blocked is referred to as the *streaming dimension*.

$B_{\mathrm{core}}$ is the core block parameter. The cluster block is evenly split into core blocks for every core in the cluster. The splitting is done along 1 dimension with parameter $B_{\mathrm{core}}$. $B_{time}$ denotes the temporal blocking parameter.

### B. Memory-to-Computation Analysis

Blocking parameters have an important impact on the performance by affecting the computation-to-memory balance of the system. In the 2D stencil example, without any optimization, the computation of one element requires five elements. That is 48 bytes for double precision memory accesses compared to 5 floating point operations. The bytes-to-op ratio (memory-to-computation ratio) is 4.6, which is relatively high. After applying spatial blocking, the computation reads in one block of data, performs stencil operations and writes out one data block of a later time step. That is $16B_{\mathrm{cluster}}$ bytes of memory accesses counting both reads and writes compared to $5B_{\mathrm{cluster}}$ operations. With temporal blocking, the memory-to-computation ratio is further reduced by a factor of $B_{\mathrm{time}}$ because a data block read from memory can support computing $B_{\mathrm{time}}$ blocks.

More generally, the memory-to-compute ratio for an arbitrary stencil problem is shown in Table I. For the baseline version, the $p$-point stencil needs $P$ elements to compute one new value. The memory bytes is $8(P+1)$ ($P$ reads and 1 write, double precision). The memory-to-compute ratio is thus $8(P+1)/P$. Spatial blocking can reduce the algorithm's memory-to-compute ratio. In a $D$ dimension stencil, the cluster block is of size $B_{\mathrm{cluster}}^{D-1}$. The memory transfer for the block is $16B_{\mathrm{cluster}}^{N-1}$ counting both reads and writes (double precisions). This block allows to perform $PB_{\mathrm{cluster}}^{N-1}$ floating point operations. With time blocking, the memory-to-compute ratio is further reduced by the time blocking factor.

TABLE I: Computation and memory ratio.

| Blocking scheme | Bytes-to-Ops ratio | SRAM/core (Bytes) |
|---|---|---|
| No Blocking | $8(P+1)/P$ | 0 |
| Space Blocking | $16/P$ | $16RB_{\mathrm{core}}B_{\mathrm{cluster}}^{D-2}$ |
| Time Blocking | $16/(PB_{\mathrm{time}})$ | $16RB_{\mathrm{time}}B_{\mathrm{core}}B_{\mathrm{cluster}}^{D-2}$ |

### C. Design Space Exploration

The increase in the time blocking parameter allows an increase on the operations performed for a certain memory access, reducing the bandwidth requirement of the computation. But larger blocking comes with the price of larger cache size. Table I shows the corresponding SRAM size of different blocking strategies. Spatial blocking requires buffering several cluster blocks for reuse by subsequent clusters: for a stencil operation that touches element in R distances, $2R$ blocks need to be buffered. If the time blocking is counted in, more blocks need to be cached: $2R$ blocks for each time step.

With a fixed area constraint, the blocking degrades the performance because the cache used for blocking takes up the area for computation logic. Therefore, the blocking parameter needs to be carefully chosen to build a balanced computation in order to achieve the optimal performance. We use an example below to exhibit how the balance affects the performance. Table II gives three different implementations on a total accelerator area of 25 $\mathrm{mm}^2$. The memory layers consist of 4 dies partitioned into 16 vaults whose peak bandwidth is 400GB/s. The spatial blocking parameter is 32 elements. The first implementation with a cache size of 512 Bytes each core has performance bounded by memory. The attained performance is far below the peak performance the system can provide. A large number of accelerator cores are idle. The second implementation with a cache size 8KB per core has performance bounded by computation. There is still margin to the peak memory bandwidth but the system has reached the maximum computation capability. The third design is a more balanced design in terms that it enables more computation with the bandwidth. It achieves the highest performance among the three configurations. This table demonstrates how the choices of computation and storage affect the final performance. We see that only when the computation and memory access is balanced, the optimal performance can be achieved.

## V. Experiments

In this section we evaluate the design space of the stencil accelerator architecture on 3D stacked DRAM and identify the optimal design points for different system and problem configurations. CACTI 6.5 [13] is used to evaluate the power and area of the scratchpad SRAM. The 3D DRAM power and the bandwidth is evaluated using CACTI-3DD [14]. The area

TABLE II: Attainable performance for three different implementations on a total accelerator area of 25 $\mathrm{mm}^2$.

| $B_{time}$ | SRAM size | Area /core [$\mathrm{mm}^2$] | Cores /vault | Peak GFLOPS | Attained GFLOPS | Required BW [GB/s] |
|---|---|---|---|---|---|---|
| 1 | 512B | 0.10 | 30 | 2400 | 125 | 400 |
| 16 | 8KB | 0.16 | 10 | 800 | 800 | 160 |
| 8 | 4KB | 0.13 | 12 | 960 | 960 | 380 |

TABLE III: 3D-stacked DRAM configuration examples.

| Conf | $N_{\mathrm{stack}}$-$N_{\mathrm{bank}}$- $N_{\mathrm{io}}$ | $t_{\mathrm{RCD}}$-$t_{\mathrm{CAS}}$-$t_{\mathrm{TSV}}$-$t_{\mathrm{RP}}$ (ns) | BW(GB/s) |
|---|---|---|---|
| Conf1 | 4-8-1024 | 7.9-12.1-0.67-16.8 | 269.2 |
| Conf2 | 4-16-512 | 7.2-10.6-0.67-8.4 | 400.3 |
| Conf3 | 4-32-512 | 7.2-10.6-0.67-8.4 | 608.3 |

and power of computation logics is estimated with Synopsis Design Compiler with 32nm process [15].

### A. Area and Bandwidth Evaluation

**Performance exploration.** We show the maximum performance that can be achieved under different area and bandwidth budgets in Fig. 5. The three bandwidth represents a conservative/moderate/high 3D-DRAM configuration as shown in TABLE III. For a certain bandwidth, the results show that when chip area is small, the performance improves linearly with the chip area. In this case, the computation cannot saturate the memory bandwidth. There is no need to integrate cache in each core. As the area budget increases and more cores can be implemented, the memory bandwidth requirement of the accelerator increases to more than the maximum bandwidth the system can deliver. Then the computation becomes memory-bound. At this point, cache is required. Larger cache size slows down the increase of system performance as shown in the graph. The performance improvement has diminishing returns as the chip area increases. Comparing across different memory bandwidths, we see that when the maximum memory bandwidth is small, the performance becomes memory bound at smaller chip area. In the memory bound region, smaller memory bandwidth has more severe diminishing return as the area increases compared to higher memory bandwidth.

Fig. 6 gives an overview of the power breakdown and power efficiency as area varies. It shows that for large areas, large portions of the power is spent on the local cache. The system power efficiency (measured in GFLOPS/W) also drops rapidly. A conclusion from this is that while a large area may enable higher performance, it might not be desirable from a power efficiency point of view because at larger scale the cache can be more expensive than the 3D memory.

**General stencil computation.** We show the balanced performance points for different stencil problems. Fig. 7 shows performance and power efficiency for stencil problem of different patterns and dimensions. The memory system configuration is based on the Conf3 of Table III. We pick three area numbers: 60 $\mathrm{mm}^2$ which is close to the 3D-stacking logic die area reported by Micron's HMC [16], as well as 90 $\mathrm{mm}^2$ and 120 $\mathrm{mm}^2$, to show the performance sensitivity to the area. Higher dimensional stencils achieve less performance for the same area compared to lower dimension stencils since higher

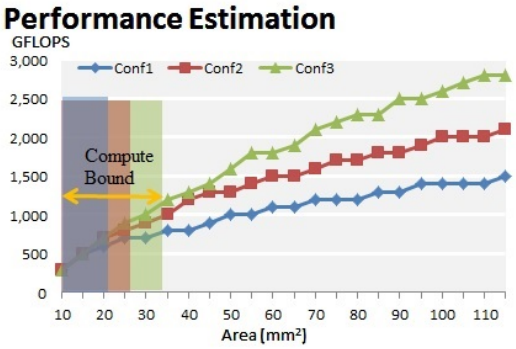


Fig. 5: Performance on different area and bandwidth.

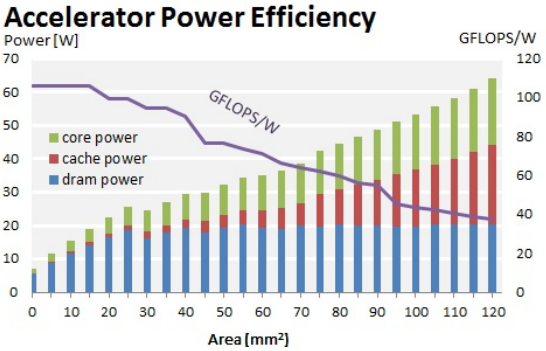


Fig. 6: System power efficiency as area increases.

TABLE IV: Performance results on general CPU and GPU.

| System | Power [W] | Tech [nm] | BW [GB/s] | DP GFlops | Points | Achieved GFLOPS |
|---|---|---|---|---|---|---|
| Xeon E5550 | 95 | 45nm | 51.2 | 85.3 | 7 | 3.36 GStencil/s |
| Xeon E5550 | 95 | 45nm | 51.2 | 85.3 | 27 | 1.19 GStencil/s |
| Core i7 | 47 | 22nm | 30 | 51 | 7 | 180 MFlops |
| GTX 285 | 204 | 55nm | 159 | 93 | 7 | 4.6 GFlops |
| FPGA | 9 ALTERA Stratix V FPGAs | | | | 3D | 236 GFlops |
| Acc | 60 | 32nm | 600 | 2400 | 7 | 2400 GFlops |

dimensional stencil computations are more memory intensive. They require a larger cache for blocking. Stencil computations on more points have better performance than fewer points for the same dimension, since higher-point stencils perform more operations for the same number of memory accesses, i.e., they are more computationally intensive. Stencil computations that are more memory bound have less gain as area budget increases, e.g., the 3D_7pt problem has less improvement when area increases from 60 $\mathrm{mm}^2$ to 120 $\mathrm{mm}^2$ compared to 3D_27pt, because the memory intensive problem will take much more cache to improvement the performance.

**Comparison.** We compare our design against the state-of-art stencil implementations on modern CPUs and GPUs as well as FPGAs. The comparison is shown in Table IV. The performance on the Xeon processor is collected from [17]. The performance result is measured in GStencil/s as the number of points updated per second. The power of the Intel architecture is the TDP power collected from [18]. The performance on GTX and Core i7 is collected from [19]. The power of the GTX card is the maximum dissipating power reported on [20].

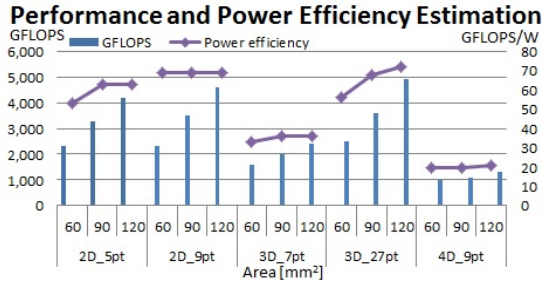


Fig. 7: System power efficiency.

The FPGA implementation uses 9 ALTERA Stratix V FPGAs [21]. The last entry is the results of our accelerator. Coarse-grained as the data is, we can still see that our implementation has an order of magnitude improvement on performance and energy efficiency due to the higher density and more efficient design implementation.

## VI. Conclusion

This paper presents a 3D stacked memory based accelerator design for stencil computation. The accelerator consists of many small processing cores working concurrently on the logic die of the 3D memory system. The cache size of the accelerators need to be designed properly to balance the computation and memory access. Only then can the optimal performance be achieved on the system. Overall the proposed accelerator can achieve an order of magnitude higher performance than conventional computing system.

## Acknowledgment

The work was sponsored by Defense Advanced Research Projects Agency (DARPA) under agreement No. HR0011-13-2-0007. The content, views and conclusions presented in this document do not necessarily reflect the position or the policy of DARPA or the U.S. Government. No official endorsement should be inferred.

## References

[1] T. Porsching, "Numerical solution of partial differential equations: Finite difference methods (GD Smith)," *SIAM Review*, vol. 22, no. 3, pp. 376–376, 1980.

[2] B. Lippmeier and G. Keller, "Efficient parallel stencil convolution in Haskell," *SIGPLAN Not.*, vol. 46, no. 12, pp. 59–70, Sep. 2011.

[3] H. Köstler, "Platform-independent description of imaging algorithms," *PAMM*, vol. 14, no. 1, pp. 945–946, 2014.

[4] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[5] D. Wonnacott, "Achieving scalable locality with time skewing," *Int. J. Parallel Program.*, vol. 30, Jun. 2002.

[6] M. E. Wolf, "Improving locality and parallelism in nested loops," Ph.D. dissertation, Stanford, CA, USA, 1992, uMI Order No. GAX93-02340.

[7] G. Rivera and C.-W. Tseng, "Tiling optimizations for 3D scientific computations," in *proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2000.

[8] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," *ACM SIGPLAN Notices*, vol. 43, no. 6, pp. 101–113, 2008.

[9] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, and F. Franchetti, "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing," in *3D Systems Integration Conference*, 2013, pp. 1–7.

[10] F. Sadi, B. Akin, D. T. Popovici, J. C. Hoe, L. Pileggi, and F. Franchetti, "Algorithm/hardware co-optimized SAR image reconstruction with 3D-stacked logic in memory," 2014.

[11] G. Loh, "3D-stacked memory architectures for multi-core processors," in *Proceedings of the International Symposium on Computer Architecture,*, June 2008, pp. 453–464.

[12] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "A logic-base interconnect for supporting near memory computation in the hybrid memory cube."

[13] "CACTI 6.5, HP labs," http://www.hpl.hp.com/research/cacti/.

[14] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2012, pp. 33–38.

[15] S. Galal and M. Horowitz, "Energy-efficient floating-point unit design," *IEEE Transactions on Computers,*, vol. 60, no. 7, pp. 913–922, July 2011.

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

[17] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," in *proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2008, pp. 1–12.

[18] "Intel architecture," http://ark.intel.com/.

[19] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, "3.5-D blocking optimization for Stencil computations on modern CPUs and GPUs," in *proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2010, pp. 1–13.

[20] "GTX 285," http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-285/specifications.

[21] K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-FPGA accelerator for scalable Stencil computation with constant memory bandwidth," *IEEE Transactions on Parallel and Distributed Systems,*, vol. 25, no. 3, pp. 695–705, March 2014.