Domain-Specific Processor with 3D Integration for Medical Image Processing

Jason Cong, Karthik Guruaj, Muhuan Huang, Sen Li, Bingjun Xiao, Yi Zou
Computer Science Department
University of California, Los Angeles

***Abstract*— The growth of 3D technology had led to opportunities for stacked multiprocessor-accelerator computing platforms with high-bandwidth and low-latency TSV connections between them, resulting in high computing performance and better energy efficiency. This work evaluates the performance and energy benefits of such an advanced architecture and addresses associated design problems. To better utilize the reconfigurable hardware resource and to explore the opportunity of kernel sharing across applications, we propose to use a dedicated domain-specific computing platform. In particular, we have chosen medical image processing as the domain in this work to accelerate due to its growing for real-time processing demand yet inadequete performance on conventional computing architectures. A design flow is proposed in this work for the 3D multiprocessor-accelerator platform and a number of methods are applied to optimize the average performance of all the applications in the targeted domain under area and bandwidth constraints. Experiments show that the applications in this domain can gain a 7.4x speed-up and 18.8x energy savings on average running on our platform using CMP cores and domain-specific accelerators as compared to their counterparts coded in CPU only.**

I. Introduction

It was shown in [1, 2] that general-purpose processors (GPPs) can be several orders of magnitude less efficient in performance and energy compared to ASIC or field programmable gate array (FPGA) implementations. The inefficiency of GPPs comes from several sources: 1) instruction fetch and decoding, 2) data movement between instructions, 3) possible inefficiency in data reuse (due to cache replacement policy), and 4) speculation. The ASIC architecture outperforms GPPs in both power and performance because it can take advantages of application knowledge to design the hardware. However, ASICs can only support narrow workloads and are time-consuming in system design. A promising solution is to use GPPs and accelerators which are implemented with programmable fabrics. By offloading critical tasks from GPPs to accelerators, we can achieve high performance and power-efficient designs while supporting a wide variety of tasks.

In this case, data communication bandwidth between GPPs and accelerators is very important. If the bandwidth cannot sustain the rate at which memory requests are generated, data communication would become the bottleneck of the design; then, little or even no speedup could be achieved by using accelerators. This is especially the case when the tasks to be executed on accelerators are relatively fine-grained (e.g., the tasks are part of a loop body) and require data communication every time they are performed. Thus, high bandwidth is critical for a desirable system design.

However, beginning with the last decade, communication (interconnect) delay has been directly impacting system performance [3]. Moreover, as multi-core and eventually many-core processors emerge as a means for improving processor performance, the demand for bandwidth will continuously increase as the number of cores increases [3]. To address this ever-larger demand for bandwidth, the industry is searching actively for new interconnect technology. 3D interconnect technology, which allows vertical stacking of layers of active electronic components, has emerged as a promising technology to boost the device bandwidth [4, 5]. By deploying 3D interconnect technology, different dies are stacked together, forming 3D integrated circuits (ICs). It allows large numbers of vertical vias between the layers and a drastic increase in communication bandwidth between functional blocks in different layers.

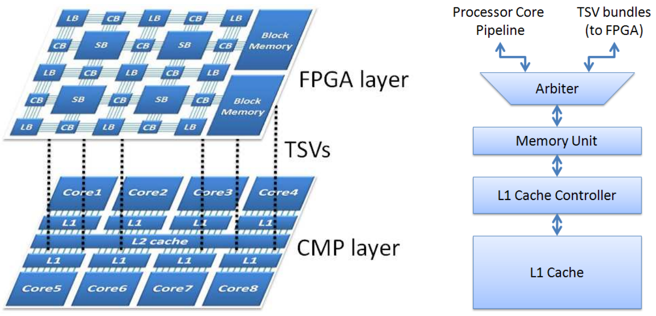
Previous work on 3D ICs mainly targeted GPPs and programmable fabrics separately. In the context of GPPs, interface between the L2 cache and main memory is architected using 3D interconnect technology, and performance is compared to a 2D design for memory-intensive applications [6–10]. In the context of FPGAs, a number of recent studies have shown that 3D FPGAs have better performance than existing 2D designs [11–13].

However, despite these studies, to our knowledge, there is no prior work that discusses connecting FPGAs with processors directly using 3D interconnect technology. We envision that future MPSoCs will be accelerator-rich, and it is natural to interconnect GPPs and accelerators using 3D technology. In this study we evaluate the performance of such an advanced processor-accelerator architecture and we address several design problems associated with the architecture.

The remainder of this paper is organized as follows: Section II describes our CMP-FPGA architecture. Section III overviews the domain-specific applications, and Section IV details our design flow and methodologies to accelerate the applications in our domain. Section V shows experiment results and we conclude our paper in Section VI.

II. Heterogeneous 3D Multiprocessor-Accelerator Architecture

Our proposed 3D multiprocessor-accelerator architecture is constructed by stacking a programmable fabric layer above the CMP layer, and the required communication between these two layers is provided by TSVs. The overall architecture is shown in Fig. 1a.



(a) The overview of the 3D processor.

(b) Interface.

Fig. 1: The architecture of the 3D CMP-FPGA computing platform.

Our system interfaces accelerators with the L1 cache of the core which invokes the accelerators. An accelerator issues a series of memory access requests – each request involves a base address and the size of data to be fetched/written. In our system the address specified by the accelerator is the virtual address of the memory location at which data needs to be accessed. Each memory access request is then inserted into the memory unit pipeline of the core which invokes the accelerator (see Figure 1b). The memory unit performs address translation using the TLB (identical to a normal software load/store instruction) and passes the request to the L1 cache controller. The L1 cache controller services the request as it would service any normal software memory access operation. If it is an L1 cache hit, the request is completed and the data is transferred to the FPGA (where it is copied to a local memory). If it is a miss, the cache controller will issue a fill request to the L2 cache. Additionally, since the L1 cache controller sees no difference between an accelerator memory access and a core-issued access, the coherence protocol implemented by the system is used for all accesses in a uniform fashion.

The amount of data that is requested by the accelerator could be much larger than the amount of data that an L1 cache can provide in one request. Typical L1 caches in modern systems have 128-bit wide read/write ports [14], meaning each request to the L1 cache can return/accept at most 128 bits of data. Hence, we need an additional counter at the FPGA to split large requests into 128-bit chunks and issue these chunks to the cache. The width of the L1 cache port also decides how many TSVs we should use to connect the FPGA and CMP system; Using more TSVs than the port width is pointless. We assume the TSV pitch to be 10um [3]. Under this assumption, the area of a 128-bit TSV bundle is 10% the area of a 32-KB L1 cache (as reported by McPAT [15]). The resistence and capacitance of TSVs are modeled as $R = 0.17\Omega$ and capacitance $C = 8 \times 10^{-16}$F according to the characteristic parameters in [16].

III. Domain-Specific Application

Domain-specific accelerators try to make use of domain knowledge to be able to accelerate a set of representative applications inside a domain. A domain-specific accelerator is different form an individual application accelerator because it concerns a domain of applications rather than a single application. One important distinction of the domain-specific approach is that it must explore many accelerator-sharing opportunities among diffenent applications. Currently, we use FPGA fabric to realize the domain-specific accelerators, but our methodology is also applicable for accelerator implementations using ASICs.

In this work we choose medical imaging as the application domain to exercise the idea of domain-specific acceleration. Medical imaging is one of the critical computational techniques that have become a routine tool in today's diagnosis and treatment of many medical problems. However, novel image processing algorithms are often infeasible for real-time clinical use. FPGA-based acceleration can help improve the performance of these applications to address the real-time challenge, while still maintaining a low-power envelope for the computing system. We build a medical image processing pipeline consisting of algorithms for 1) image denoising, 2) image deblurring, 3) image registration using fluid registrations (reorientation of a study image to a reference image), and 4) Level-set segmentation (delineation of regions of interest). Detailed descriptions of these examples are available at [1, 17]. This image pipeline serves as the testbench to validate our 3D hybrid heterogenous architecture.

Analysis shows that these applications may share some common computing kernels as shown in Fig. 2. For example, both image deblurring and image registrations employ Gaussian smoothing as the critical kernels. We strive to select and implement a set of "base accelerators" that can accelerate many, if not all, applications in the domain. Moreover, it is important to consider these sharing opportunities to "place" various accelerators onto the reconfigurable fabric under the area constraint.

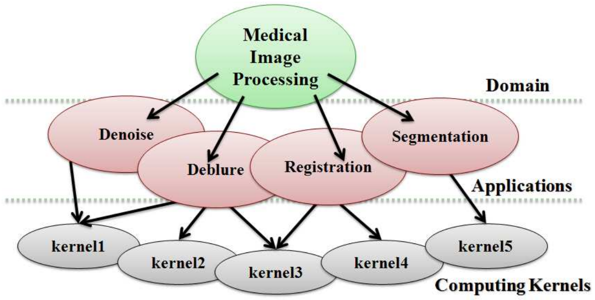


Fig. 2: Kernel-sharing in medical imaging domain.

IV. Design Methodology for the Proposed Architecture

Given a set of applications within one domain and the resource constraint, our goal is to implement these applications on our proposed architecture to optimize the average computing performance and energy efficiency of all applications. The overall design flow consists of the following three steps.

1) Identify the kernels to be implemented on the FPGA layer based on the characteristics of all applications.
2) Implement those kernels on the FPGA layer and perform several code transformation techniques to improve their performance and reduce energy consumption.
3) Select a set of implementations of these kernels under resource constraints such that the average computing performance and energy efficiency of all applications is optimized.

Each step will be discussed briefly as follows.

First we need to identify the kernels of each application that are to be implemented on the FPGA layer. We select the kernels based on the following insights. 1) Certain kernels are shared across some applications. For example, the "denoise" kernel is used in both the "denoise" and "deblur" applications, and the "Gaussianblur" kernel is used in both the "deblur" and "registration" applications. FPGA on-chip resources could be saved by reusing these kernels that are shared across applications. 2) The kernels selected to be implemented on FPGA can be computation-intensive and time-consuming, i.e., the bottleneck of computation on CMP. Therefore we profile each application using *gprof* [18] to obtain the percentage of execution time for every function within one application. We pick the functions that occupy most of the total execution time of the application (typically more than 90% of the total execution time) to be the kernel candidates that will be implemented on FPGA. Our profiling result shows that the kernels are typically in the format of nested loops. This provides us with a lot of optimization opportunities to improve performance by code transformation when implementing the kernels on the FPGA layer.

Once we have all the kernels selected, the next step is to optimize them and provide different implementations of each kernel under different resource constraints. We therefore perform several code transformation techniques including *data reuse* [19], *computation reuse* and *loop tiling* [20], *utilization of wide bus* and *module duplication*, and *memory partition* [21]. These techniques provide us with a number of benefits that include reducing bandwidth pressure, eliminating redundant computations, fully utilizing bus bandwidth to improve performance, achieving maximum throughput, and greatly saving on-chip resources with little performance degradation.

The final step is optimal selection of implementations under area and bandwidth constraints. For each kernel, we can get multiple implementations. Typically the implementations with more area/bandwidth consumption provide better performance for a kernel. Since the resource on a computing platform is limited, we have to make trade-offs between the performance and resource usage. Though the trade-off can be performed by duplication of implementations, the properties of different implementation versions still have to be considered. We discovered that an implementation version with large size can offer a high resource usage efficiency, but one with small size can offer a fine-grained trade-off result. Also in some cases, two implementations versions can achieve similar performance but at cost of very different ratios of area/bandwidth usage. The selection opportunities of these kinds of implementation cases make our processor more adaptive to both area-bound and bandwidth-bound cases. Note that there are kernels shared across applications in our domain, and our objective is to accelerate the overall computation within this domain. To achieve the best overall performance, the acceleration strategy for each kernel, such as whether it needs acceleration or not, and how much area/bandwidth should be allocated, and which implementation versions should be selected for the kernel acceleration, is yet to be optimized from a global view. It can be formulated as an optimization problem, and we can solve it by dynamic programming. Details about this problem are omitted here due to page limitation.

V. Experimental Results

A. Settings

Latency and energy are the two key properties to perform on the medical-imaging domain on CMP and its accelerated counterpart running on FPGA. The latency of CMP is evaluated via the PAPI library [22], and the power of CMP is evaluated via the McPAT tool [15]. The implementation of kernels on the FPGA is carried out by the high-level synthesis tool, *Autopilot* [23]. Then we use the *Xilinx ISE* design suite to do the mapping and P&R on FPGA and obtain a summary of latency and resource usage. The power consumption of FPGA implementations is given by *Xilinx xPower*. In our study, we use a 2GHz Intel Nehalem processor and *Virtex6* XC6VLX240T to build up the 3D heterogeneous platform. As reported in [24], the Nehalem processor has one 256-bit write port and one 256-bit read port to the L2 cache. Based on these settings, the bandwidth of the connection between CMP and FPGA can be calculated as 2.56×10^{11}bit/s taking the simulated cache missing rate into account. The acceleration is targeted for four applications in the medical image processing domain, including “denoise,” “deblure,” “registration” and “segmentation,” all of which are frequently called functions in this domain. They can be found in [17].

B. Implementations of Kernels

In the first step of our experiments, we pick four kernel candidates to be implemented (see in Table I). These kernels are selected according to the principle of kernel sharing, suitability for FPGA calculation, and portion of overall computation process. Then we implement these kernels on an FPGA into different versions using the implementation technologies mentioned in the previous section. Table II on the top of next page is an abstract of representative implementations of each kernel. Here the implementation IDs refer to the implementation versions developed by us to accelerate the corresponding kernels.

TABLE I: The kernels found in the four applications in our experiments.

application	kernel
denoise	rician
deblur	Gaussianblur, rician
registration	Gaussianblur, calDisplacement
segmentation	eval3DtwoPhase

C. Optimal Solution of Implementation Selection

Table III shows the optimal solution of implementation selection in our experiments. Here the duplication factor refers to how many copies of one implementation are duplicated to accelerate together. If an implementation is not listed in Table III, then it is not used in the optimal solution. Note that none of the implmentations of “calDisplacement” are selected to accelerate it, mainly due to their lower resource efficiency. Table IV shows the acceleration result of each kernel. Table V shows the acceleration result of all the applications in the domain. We also have a study concerning the impact of bandwidth on the performance of the 3D CMP-FPGA computing platform. We scan the bandwidth from 1.6GB/s to 32GB/s with a step of 1.6GB/s (1GB/s = 16Gbit/s), and the result is shown in Fig. 3. We see that the 3D integration technology brings a significant gain in performance compared to the conventional communication technologies such as PCI express (PCIe), HyperTransport (HT) and Front-Side Bus (FSB).

TABLE III: The optimal solution of implementation selection.

kernel	impl. ID	dupl. fac	area	bandwidth (bit/s)
rician	impl11	1	51.43%	3.84e10
Gaussianblur	impl21	1	12.00%	1.02e11
	impl22	2	13.07%	1.02e11
calDisplacement	-	-	-	-
eval3DtwoPhase	impl43	1	20.46%	4.80e9
total	-	-	96.95%	2.48e11

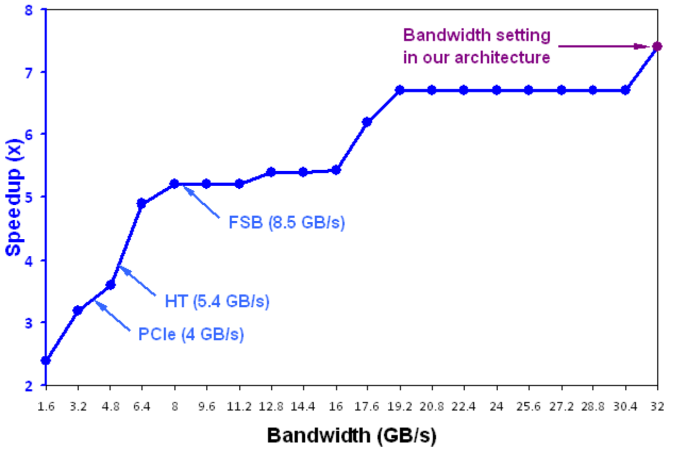


Fig. 3: The impact of bandwidth on the performance of the 3D CMP-FPGA computing platform.

TABLE II: An abstract of representative implementations of each kernel.

kernel name	implementation ID	area	bandwidth (bit/s)	CPU latency (s)	FPGA latency (s)	CPU energy (J)	FPGA energy (J)
rician	impl11	51.43%	3.84e10	1.33e-2	5.54e-4 (21.9x)	1.24e-1	1.23e-3 (101.3x)
Gaussianblur	impl21	12.00%	1.02e11	1.36e-2	3.42e-3 (4.0x)	1.33e-1	7.41e-3 (18.0x)
calDisplacement	impl31	30.00%	3.84e10	9.35e-1	6.37e-1 (1.5x)	8.74	1.72 (5.1x)
eval3DtwoPhase	impl41	59.44%	1.92e10	5.06e-2	1.60e-3 (31.6x)	5.07e-1	3.53e-3 (143.5x)

TABLE IV: The acceleration result of each kernel.

kernel	CPU latency (s)	FPGA latency (s)	CPU energy (J)	FPGA energy (J)
rician	1.33e-2	6.04e-4 (21.7x)	1.24e-1	1.23e-3 (101.3x)
Gaussianblur	1.36e-2	1.56e-3 (8.7x)	1.33e-1	3.40e-3 (39.2x)
calDisplacement	9.35e-1	9.35e-1 (1.0x)	8.74	8.74 (1.0x)
eval3DtwoPhase	5.06e-2	5.49e-3 (9.2x)	5.07e-1	1.21e-2 (41.8x)
average	-	10.2x	-	45.8x

TABLE V: The acceleration result of all the application in the domain.

application	CPU latency (s)	FPGA latency (s)	CPU energy (J)	FPGA energy (J)	EDP improvement
denoise	5.25e-2	2.42e-3 (21.7x)	4.97e-1	5.35e-3 (92.8x)	2041.6x
deblur	4.39e-2	3.72e-3 (10.8x)	4.02e-1	8.08e-3 (49.7x)	589.2x
registration	4.71	1.91 (2.5x)	4.65e1	1.61e1 (2.9x)	7.1x
segmentation	8.92	1.7 (5.2x)	8.87e1	8.89 (9.3x)	48.8x
geometric average	5.46e-1	7.36e-2 (7.4x)	5.35	3.90e-1 (18.8x)	143.0x

VI. Conclusions

This paper presents a novel domain-specific processor with 3D integration for medical image processing. It is based on a platform consisting of stacked microprocessors and accelerators. A design flow is proposed to accelerate a set of applications in the domain. We select a set of kernels from the applications based on profiling and domain knowledge. Then we implement these kernels using FPGA accelerators using a variety of implementation settings and arrive at multiple versions for each kernel with different performances and area usages. We alos introduce a number of methods to obtain the optimal average speed-up and energy-savings throughout the targeted domain under area and bandwidth usage constraints. On a platform of Xeon CPU as microprocessor, Xilinx FPGA as accelerator, TSVs as 3D integration technology and I/O blocks available in the current product, we can achieve a 7.4x speed-up and an 18.8x energy savings. A similar 3D architecture and design methodology is also used to accelerator the vision and navigation application domain [25].

VII. Acknowledgements

This work is partially supported by Center for Domain-Specific Computing (NSF Expedition in Computing Award CCF-0926127), and grants from Altera Corp. and Mentor Graphics Corp. under UC Discovery program.

References

[1] J. Cong, V. Sarkar, G. Reinman, and A. Bui, "Customizable domain-specific computing," *IEEE Design and Test of Computers*, vol. 28, pp. 6–15, 2011.

[2] P. Schaumont and I. Verbauwhede, "Domain-specific codesign for embedded security," *Computer*, vol. 36, no. 4, pp. 68–74, 2003.

[3] "International technology roadmap for semiconductors," 2009. [Online]. Available: http://www.itrs.net/Links/2009ITRS/Home2009.htm

[4] K. Takahashi and M. Sekiguchi, "Through silicon via and 3-D wafer/chip stacking technology," in *Symposium on VLSI Circuits*, 2006, pp. 89–92.

[5] E. Beyne, "3D System Integration Technologies," in *VLSI-TSA*, 2006, pp. 1–9.

[6] Y. Liu *et al.*, "Fine grain 3D integration for microarchitecture design through cube packing exploration," in *ICCD*, 2007, pp. 259 –266.

[7] J. Bautista, "Tera-scale computing and interconnect challenges," in *DAC*, 2008, pp. 665–667.

[8] D. H. Woo, N. H. Seong, D. Lewis, and H.-H. Lee, "An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth," in *HPCA*, 2010, pp. 1–12.

[9] P. Franzon *et al.*, "Design and CAD for 3D integrated circuits," in *DAC*, 2008, pp. 668–673.

[10] U. Kang *et al.*, "8 Gb 3-D DDR3 DRAM Using Through-Silicon-Via Technology," *JSSC*, vol. 45, no. 1, pp. 111–119, 2010.

[11] R. Le, S. Reda, and R. I. Bahar, "High-performance, cost-effective heterogeneous 3D FPGA architectures," in *GLVLSI*, 2009, pp. 251–256.

[12] M. Lin and A. El Gamal, "A routing fabric for monolithically stacked 3D-FPGA," in *FPGA*, 2007, pp. 3–12.

[13] K. Siozios, D. Soudris, and G. Economakos, "Three dimensional FPGA architectures: A shift paradigm for energy-performance efficient DSP implementations," in *DSP*, 2009, pp. 1–6.

[14] D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems," in *MICRO*, 2009, pp. 413–422.

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

[16] R. Weerasekera, M. Grange, D. Pamunuwa, and H. Tenhunen, "On signalling over Through-Silicon Via (TSV) interconnects in 3-D Integrated Circuits," in *DATE*, 2010, pp. 1325–1328.

[17] "Medical Image Benchmark." [Online]. Available: http://www.cdsc.ucla.edu/resources/software-release

[18] "GNU profiler," http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html.

[19] J. Cong, H. Huang, C. Liu, and Y. Zou, "A reuse-aware prefetching algorithm for scratchpad memory," in *DAC*, 2011, pp. 960–965.

[20] J. Cong, M. Huang, and Y. Zou, "Accelerating fluid registration algorithm on multi-fpga platforms," in *FPL*, 2011.

[21] J. Cong, W. Jiang, B. Liu, and Y. Zou, "Automatic memory partitioning and scheduling for throughput and power optimization," in *ICCAD*, 2009, pp. 697–704.

[22] "PAPI library," http://icl.cs.utk.edu/papi/.

[23] J. Cong *et al.*, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *TCAD*, vol. 30, pp. 473–491, 2011.

[24] "Intel 64 and IA-32 Architectures Optimization Reference Manual," http://developer.intel.com/products/processor/manuals/.

[25] J. Cong, B. Grigorian, G. Reinman, and M. Vitanza, "Acceleratin Vision and Navigation Applications on a Customizable Platform," in *ASAP*, 2011.