IEIE Transactions on Smart Processing and Computing, vol. 5, no. 4, August 2016
http://dx.doi.org/10.5573/IEIESPC.2016.5.4.294

CPU-GPU^2 Trigeneous Computing for Iterative Reconstruction in Computed Tomography

Chanyoung Oh and Youngmin Yi

School of Electrical and Computer Engineering, University of Seoul / Seoul, South Korea {alspace11, ymyi}@uos.ac.kr

***** Corresponding Author: Youngmin Yi

Received June 25, 2016; Revised August 3, 2016; Accepted August 5, 2016; Published August 30, 2016

** Regular Paper*

***Abstract*:** In this paper, we present methods to efficiently parallelize iterative 3D image reconstruction by exploiting trigeneous devices (three different types of device) at the same time: a CPU, an integrated GPU, and a discrete GPU. We first present a technique that exploits single instruction multiple data (SIMD) architectures in GPUs. Then, we propose a performance estimation model, based on which we can easily find the optimal data partitioning on trigeneous devices. We found that the performance significantly varies by up to 6.23 times, depending on how SIMD units in GPUs are accessed. Then, by using trigeneous devices and the proposed estimation models, we achieve optimal partitioning and throughput, which corresponds to a 9.4% further improvement, compared to discrete GPU-only execution.

***Keywords*:** Heterogeneous computing, Data partitioning, Image reconstruction, Computed tomography

1. Introduction

Due to severe power constraints, heterogeneous computing has become mainstream, and specialized devices such as graphics processing units (GPUs) and field-programmable gate arrays (FPGAs) are integrated into a processor to achieve maximum performance with a limited power budget. In this context, CPU-GPU heterogeneous systems have become prevalent across all computing platforms, from embedded systems to servers. However, it is crucial to obtain optimal partitioning of an application in order to fully utilize heterogeneous systems. A lot of research has been conducted to find optimal partitioning and mapping in such systems [1, 2]. Lee et al. proposed a performance estimation model from which the optimal workload distribution of a feature extraction application can be obtained for CPU-GPU heterogeneous platforms [3]. Our previous work presents efficient mapping of a face detection application that fully exploits both the CPU and the GPU in the NVIDIA Tegra K1 SoC [4].

These days, we can easily find not just CPU-GPU heterogeneous systems but also trigeneous systems in which the three different types of device exist. The term *trigeneous computing* was first introduced by Rethinagiri et al. [5], where energy-efficient platforms composed of a CPU, a GPU, and an FPGA were proposed for embedded platforms as well as server platforms. The three devices were used for a single application, although all three devices were not utilized at the same time.

In this paper, we extend the performance model of Lee et al. [3] so as to find the optimal distribution for the trigeneous platform, and we confirm the viability of the proposed model with a trigeneous platform (consisting of a multicore CPU, an integrated GPU, and a discrete GPU) for an iterative image-reconstruction application.

On the other hand, image reconstruction is used in computed tomography (CT), which aims to reconstruct a 3D volume image inside an object (*e.g.*, the human body) with a number of X-ray images. Since modern X-ray systems use cone-beam geometry, which produces a 2D image in a single acquisition [6], a 3D volume image composed of 3D slice images is reconstructed from a number of 2D images along the angles shown in Fig. 1. Therefore, it requires huge computational power.

Moreover, recent methods such as the simultaneous algebraic reconstruction technique (SART) and maximum likelihood-expectation maximization (ML-EM) adopt iterative models [7, 8]. These iterative algorithms repeat backprojection and forward projection, updating the projected images based on the measured images with every iteration. Even though these iterative algorithms require

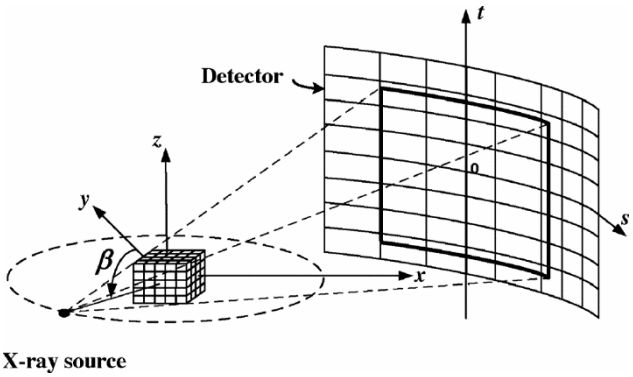

Fig. 1. The scheme of 3D cone-beam CT [9].

much more computation than the analytic algorithms [6], their importance and the demand for them in the fields are dramatically increasing since the iterative algorithms can reconstruct a high-quality 3D image with the same, or a smaller, number of X-ray images. This implies that the X-ray dose for a patient can be reduced, and scanning time can also be shortened [9].

To meet the huge computational demands of iterative image reconstruction, many researchers have tried to take advantage of hardware accelerators for image reconstruction: FPGAs were used [9, 10], and GPUs [11-13].

On the other hand, it is necessary to utilize not only discrete GPU but also other units in the system to get the most out of the underlying platforms. Most modern processors already include an integrated GPU on the same chip, which is also programmable, like the discrete GPU. Compared to the conventional CPU-GPU heterogeneous system, a trigeneous system that consists of an integrated GPU, a discrete GPU, and a CPU can improve overall performance without any additional cost.

In this paper, we first present how much performance improvement could be achieved by exploiting the characteristics of the single instruction multiple data (SIMD)-based architecture in GPUs for iterative image reconstruction. Then, we propose a performance estimation model that finds optimal partitioning for this trigeneous platform (what we call the CPU-GPU^2 trigeneous platform) to achieve the maximum throughput, with which iterative image reconstruction can be done in a timely fashion.

The rest of the paper is organized as follows. In Section 2, the iterative image reconstruction algorithm called SART will be explained, and the acceleration techniques for image reconstruction on GPUs will be explained in Section 3. Then, the proposed performance model for trigeneous platforms, as well as efficient mapping of SART on the platform, will be explained in Section 4. Section 5 shows the experimental results, which is followed by a conclusion in Section 6.

```
Input: measured projection (X-ray) images P
Output: reconstructed volume image V

1:  Initialize V
2:  Until convergence of E do
3:     E = Forward projection of V
4:     E' = Comparison of E based on P
5:     V' = Backprojection of E'
6:     V = Correction of previous result V
    using V'
7:  End
```

Fig. 2. Iterative image reconstruction.

2. Simultaneous algebraic reconstruction technique

Fig. 2 describes an overview of general iterative image reconstruction. First, forward projection is performed to generate estimated projection images with the initial 3D volume image. The output of forward projection is compared with the measured input X-ray images, building the error map. Then, the backprojection operator reconstructs a temporary volume image using the error map. Finally, we correct the output volume image based on the reconstructed error map. This procedure is repeated until the errors between the estimated projection images and the input X-ray images fall below a specific threshold.

Most iterative algorithms have the same structure described above, and share the equivalent forward projection and backprojection modules. The difference lies in how they update the volume at each iteration. For SART, the value of every voxel i, v_i, is updated at the k^{th} iteration using the following equation:

$$v_i^{(k+1)} = v_i^k + \lambda \frac{\sum_j \left[a_{ij} \frac{p_j - \sum_{i=1}^{N} a_{ij} v_i^k}{\sum_{i=1}^{N} a_{ij}} \right]}{\sum_j a_{ij}} \quad (1)$$

where p_j is the value of pixel j, a_{ij} represents the geometric information of the X-ray system, and λ is a relaxation factor introduced for noise control. Eq. (1) describes the overall procedure in Fig. 2. First, the term $\sum_{i=1}^{N} a_{ij} v_i^k$ corresponds to forward projection, which calculates line integrals from the X-ray source to the detector. Then, a comparison is performed by subtracting the calculated line integrals from the measured input X-ray images denoted as p_j. The denominator $\sum_{i=1}^{N} a_{ij}$ in the numerator is for normalization. The differences are summed for all j, normalized, and scaled by λ before being distributed to voxels. These correspond to backprojection and correction.

3. Acceleration on SIMD Devices

We will now discuss the method to exploit GPUs, which takes advantage of the massive number of SIMD units.

3.1 GPU and OpenCL

GPUs were initially designed to efficiently process graphics applications. Since those applications typically require the processing of a tremendous amount of polygons, GPUs evolved to have highly parallel structures. To provide GPUs for general-purpose computing, NVIDIA introduced a framework called the Compute Unified Device Architecture (CUDA). In this paper, we adopt Open Computing Language (OpenCL), which is a similar parallel computing framework, not only for specific GPUs but also for a wide range of heterogeneous devices. These days, even FPGAs, as well as most GPUs and CPUs, support OpenCL. It employs a hierarchical programming model to leverage the highly parallel SIMD architecture of devices. The smallest execution unit is called a *work-item*, which can be perceived as a thread of conventional multicore processing. The *work-group* represents a group of threads, and every work-item in a work-group has its own program counter running concurrently as single instruction multiple threads (SIMTs), which is an extension of SIMD. The state-of-the-art GPUs have approximately 3000 SIMD units on a single chip. Thus, a GPU kernel can execute thousands of threads at once. Although GPUs provide high memory bandwidth, because the number of SIMD units is massive, memory contention often occurs, becoming a major performance bottleneck.

3.2 Acceleration of SART

Since the image reconstruction algorithm deals with a massive number of pixels and voxels, it is well-matched to the GPU architecture with thousands of SIMD units. By simply mapping each pixel or voxel into a work-item, one can achieve performance improvement a dozen times over the sequential CPU implementation. However, we should consider more steps to maximize performance. Generally, a GPU is more suitable for compute-intensive tasks, since each core has a tiny-sized cache, and frequent global memory access would degrade performance. Thus, we have to map the pixels or voxels more carefully to mitigate this problem.

The SART algorithm can be separated into four parts: forward projection, comparison, backprojection, and correction. Comparison and correction are computed on the CPU, since these are memory-intensive, rather than compute-intensive. Backprojection and forward projection conduct almost the same job, which consists of geometry computation and interpolation. In backprojection, the straightforward implementation suffices, and no further optimization is required. In contrast, forward projection requires memory access optimization to better utilize underlying hardware architectures. First of all, forward projection cannot be computed in a single kernel since it must perform two interpolations with different dimensions,

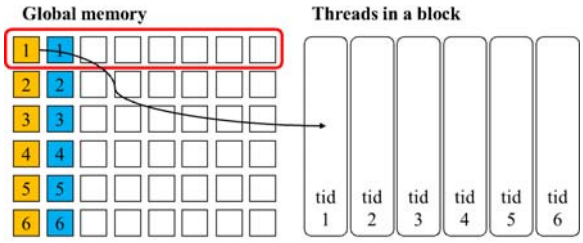

Fig. 3. Memory access pattern which is not coalesced.

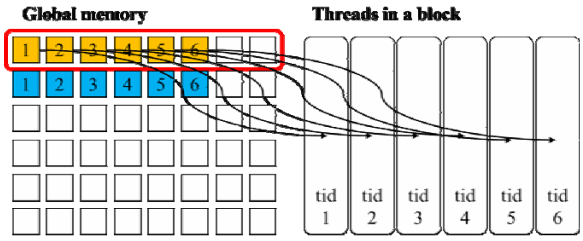

Fig. 4. Memory access pattern which is coalesced.

and OpenCL does not allow a thread to access the data written by another thread that resides outside its work-group. Although OpenCL supports data sharing among work-items through local memory, it is only allowed access to the work-items in the same work-group. Since the first interpolation produces totally new volume data, the second interpolation cannot access its entire input unless the kernel utilizes only a single work-group, which would cause a significant performance decrease. Therefore, we divide forward projection into two separate kernels: one for the *rotation* using β as defined in Fig. 1, and the other for the *projection*. Then, the threads in the projection kernel can access the output of the rotation kernel, if the permutation of x-axis and z-axis in the volume is carried out between two kernels. The permutation can be done implicitly in one of two ways:

A. writing the result of the rotation kernel in the permuted order
B. reading the input as the permuted order in the projection kernel

In the former, threads write the result to global memory along the z-axis. However, GPU memory and threads are aligned along the x-axis. Thus, the consecutive voxels along the z-axis, such as (x, y, z) and (x, y, z+1), are not consecutive in memory but are located far from each other. If the memory locations to be accessed are consecutive, or within a small stride, they are grouped into a single transaction in a GPU in order to amortize the high cost of global memory access. Because this is not the case with approach A, it suffers from low memory bandwidth. In contrast, approach B can achieve much higher memory bandwidth, since the threads in the rotation kernel write the result to global memory along the x-axis. Although the threads in the projection kernel now have to read global memory along the z-axis, as the range is limited in this interpolation, many threads read the same global memory locations, resulting in higher bandwidth.

Figs. 3 and 4 illustrate how memory access patterns can affect performance. Suppose that there are six threads in a

block, each of which accesses global memory to read two data denoted with their respective thread id (*tid*). The red round rectangle indicates that, in this example, up to eight accesses can be grouped into a single transaction by the SIMD architecture if they are consecutive along the x-axis. In Fig. 3, 12 transactions are required to read the total of 12 values, since they are stored along the y-axis. Note that the yellow box and blue box cannot be accessed in the same transaction, since the accesses are separate instructions from the same thread: the blue one is accessed after the yellow one. In Fig. 4, however, the threads in a block access consecutive memory locations along the x-axis, allowing access from each thread to be grouped in one transaction. It requires only two transactions to read the same amount of data, which is only one-sixth of the amount in approach A.

4. SART on Trigeneous Platforms

We will now discuss the method to implement iterative image reconstruction with trigeneous devices and how to distribute the workload efficiently. First, we extend the performance estimation model from Lee et al. [3] to a system that has three different types of device. Then, we present the mapping algorithm for iterative image reconstruction using the estimated parameters.

4.1 Performance Estimation Model for Trigeneous Devices

Lee et al. [3] determined the optimal workload distribution using linear interpolation based on a pre-experiment test. In this paper, we extend the model to the CPU-GPU^2 trigeneous computing system. Since the workload for each task in image reconstruction is determined by the number of assigned projection images and increases linearly, we also adopt linear regression to estimate the execution time of the tasks. In contrast to Lee et al.'s model [3], we refine the regression by analyzing the characteristics of devices.

In iterative image reconstruction, total workload W_T is the total number of projection images, which is a fixed value from the given X-ray system, and can be represented as follows:

$$W_T = W_{DGPU} + W_{CPU} + W_{IGPU} \tag{2}$$

where W_{CPU}, W_{IGPU}, and W_{DGPU} are the workloads for the CPU, for the integrated GPU, and for the discrete GPU, respectively.

It is well known that each processing element (*i.e.*, device) in the system should have the same amount of execution time to maximize the overall performance when we parallelize a single task in a data-parallel fashion. Therefore, the execution time of a single task for all devices with optimal workload distribution ratio OW_{device} can be described as shown in Eq. (3).

$$\alpha OW_{DGPU} + \beta = \gamma OW_{CPU} + \delta = \varepsilon OW_{IGPU} + \zeta \tag{3}$$

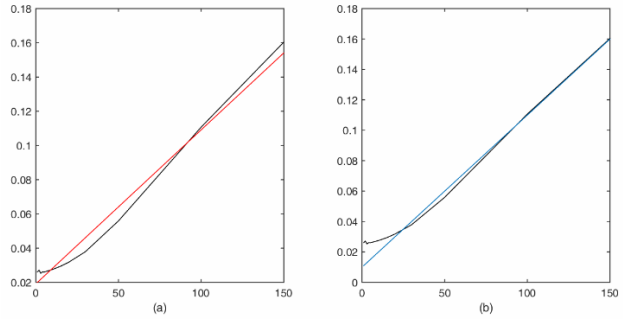

Fig. 5. (a) Improper regression, (b) proper regression considering device characteristics, when the x-axis is the workload and the y-axis is execution time (in seconds).

The execution times for each device are represented in Eqs. (4)-(6) in linear form, since they vary proportionally to the amount of the given workload, in general. Eqs. (4) and (6) are for a discrete GPU and an integrated GPU, respectively, and we will use the terms DGPU and IGPU from now on for convenience.

$$T_{DGPU}\left(W_{DGPU}\right) = \alpha W_{DGPU} + \beta \tag{4}$$

$$T_{CPU}\left(W_{CPU}\right) = \gamma W_{CPU} + \delta \tag{5}$$

$$T_{IGPU}\left(W_{IGPU}\right) = \varepsilon W_{IGPU} + \zeta \tag{6}$$

The parameters α to ζ are derived from the data obtained in advance: we should measure the execution time for each device several times with the various workload settings. In contrast to Lee et al. [3], who used linear interpolation to obtain the parameters, we use linear regression, since simply interpolating two points may cause overfitting.

When linear regression is used, first, take into account whether or not the measured data are actually linear. Although the measured points are generally in a linear shape, some intervals can show different slopes, forming a nonlinear shape, which can occur due to the characteristics of the device. For example, Fig. 5 shows the measured execution time and the estimated time as the number of assigned images increases on a DGPU. It shows an almost linear shape, but the slope at the beginning is nearly horizontal, which implies that the increase in workload does not lead to the increase in the execution time until it reaches a certain point; a DGPU usually has a huge number of processing cores and requires enough workload to fully utilize the cores. Until it is fully utilized, even if the workload increases, the tasks are executed simultaneously without increasing execution time. If this is not considered, the linear regression would result in a line with a larger error, as shown in Fig. 5(a). Then, the model could be refined so as to have multiple slopes, but it would require nonlinear regression. Thus, we chose to exclude that interval, since the points in the interval are rarely chosen as the optimal partitioning for the DGPU. Fig. 5(b) results in a line with less of an error under this consideration.

Once the parameters are set by linear regression, we

```
Input: parameter table P
Output: The optimal workload for K devices
1:   Calculate W(K) with given P according to
     Eqs. (7) - (9)
2:   Floor W(K)
3:   While there exists workload to distribute
     do
4:      Find device k which has the minimal
     execution time with W(k)+1 workload
5:      Increment W(k)
6:   End while
7:   If estimated time with W(K) exceeds
     single-threaded execution time then
8:      Return Execute-on-Host
9:   Else
10:     Return W(K)
```

Fig. 6. Workload partitioning for each task.

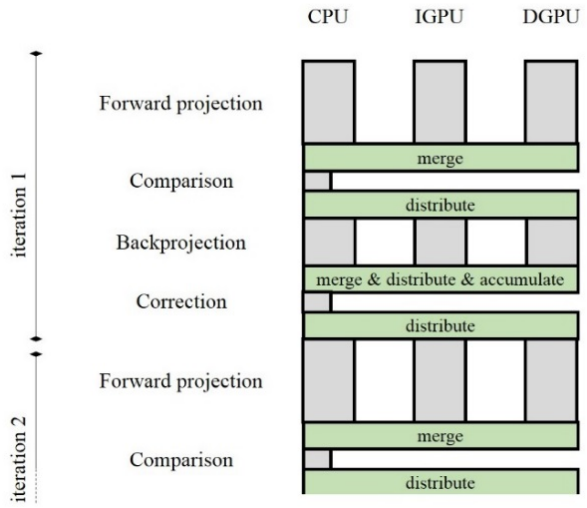

Fig. 7. An example of trigeneous computing for iterative image reconstruction.

can decide the optimal workload distribution ratio by simultaneously solving Eqs. (2) and (3) as follows:

$$OW_{CPU} = \frac{\alpha\varepsilon W_T - \alpha\delta + \alpha\zeta + \beta\varepsilon - \delta\varepsilon}{\alpha\gamma + \alpha\varepsilon + \gamma\varepsilon} \quad (7)$$

$$OW_{IGPU} = \frac{\alpha\gamma W_T + \alpha\delta - \alpha\zeta + \beta\gamma - \gamma\zeta}{\alpha\gamma + \alpha\varepsilon + \gamma\varepsilon} \quad (8)$$

$$OW_{DGPU} = \frac{\gamma\varepsilon W_T - \beta\gamma - \beta\varepsilon + \gamma\zeta + \delta\varepsilon}{\alpha\gamma + \alpha\varepsilon + \gamma\varepsilon} \quad (9)$$

4.2 Task Mapping and Implementation for Trigeneous Devices

In Section 4.1, we derived the optimal workload distribution for trigeneous devices. However, there are some issues in implementing iterative image reconstruction. First of all, the results obtained with Eqs. (7)-(9) are floating point values. However, the workload for each device in image reconstruction is an integer value, since the workload is the projection image. The naïve way to determine an integer value is rounding off the obtained floating point value, which could, however, result in non-optimal performance. Therefore, we must determine the integer values carefully by taking into account the resultant execution time of each device, as shown in Fig. 6. The procedure first obtains the floor of the float value calculated using the equations (lines 1-2). Then, it distributes the remaining workload in a greedy fashion, finding a device with the minimal execution time when one more image is assigned to it (lines 3-6).

Fig. 6 also shows the method to determine whether to parallelize a task or not (lines 7-10). Some tasks, such as comparison and correction, show better performance with single-threaded execution on a CPU, compared to parallelized execution over the three devices, since these tasks are highly memory-intensive with little computation for the processing cores, as mentioned in Section 3.2. Since the computational complexity of the iterative image reconstruction application is decided by the given X-ray system and does not vary at runtime, the parallelization can be determined with pre-experiment data. Fig. 7 illustrates an example of trigeneous execution for an iterative image reconstruction application based on Fig. 6.

Table 1. Platform specification.

Types	Devices	Specs
CPU	Intel Core i7-4770	3.4 GHz, 8 MB Cache 8 Compute Units (4 Cores)
IGPU	Intel HD 4600	350 MHz up to 1.2GHz 20 Compute Units
DGPU	NVIDIA GTX 960	1,127 MHz, 2 GB Memory 8 Compute Units (1,024 Cores)

Although the IGPU and the DGPU are idle while the CPU executes comparison and correction, the mapping and partitioning depicted in Fig. 7 is optimal. It is not only SART that has loop-carried dependency, but each task in SART also requires merged results from the previous task. Due to this, synchronizations between tasks are inevitable and make it impossible to execute tasks in parallel in a pipelined fashion. Thus, the devices can exploit only the data-parallelism by executing a task simultaneously with the optimal data partitioning for each device. With this inherent constraint, comparison and correction should be executed either in a data-parallel fashion or sequentially. And, it is more efficient for those tasks to run on a CPU, since the overhead for communication and synchronization is larger than the reduced execution time via data-parallel execution.

5. Experiments

In the experiments, we reconstruct a 128x128x128 volume image from 210 images at 256x200 using synthetic projection images [14]. The specifications of the three different devices that we used for iterative image

Table 2. Average execution time for a single iteration of the proposed image reconstruction on a GTX960.

Kernel Function	Approach A (ms)	Approach B (ms)
Backprojection	168.70	175.84
Projection_rotation	899.65	144.49
Projection_projection	295.68	297.07
Total	**1364.03**	**617.40**

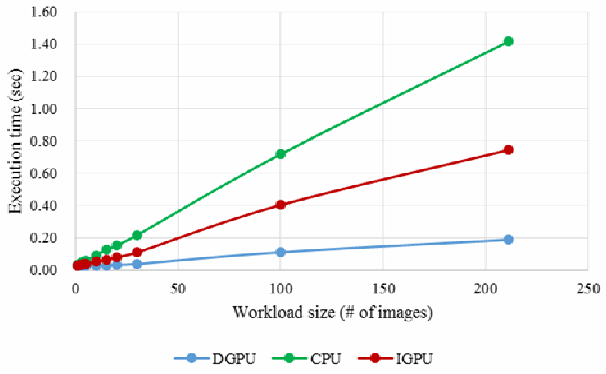

Fig. 8. Execution time of forward projection over varying workload size for each device.

reconstruction are described in Table 1.

Table 2 shows the execution time of our SART implementations. Approach A and B, explained in Section 3.2, correspond to those in Table 2. Note that only the GPU kernel and memory copy time are shown, since the execution time of the functions that run on a CPU (such as update) are negligible. The rotation kernel in Approach B is 6.23 times faster than that of Approach A thanks to the efficient memory accesses, which result in 2.21 times speedup in total execution time on the GPU. The following experiments were conducted with Approach B.

We profiled each task to obtain pre-experiment data, which are used for the proposed performance estimation model and to distribute the workload. Fig. 8 shows the profiling information for forward projection. Note that we plotted only a part of the data points out of the entire dataset. Fig. 8 shows that the DGPU outperforms the CPU and IGPU by up to 9.10 times and 6.00 times respectively. The severe imbalance in the computational power of the three devices makes it hard to improve overall performance; additional gain with the IGPU and the CPU, compared to DGPU-only execution, can be small if the DGPU is too powerful. However, the recent trends show that the performance of an IGPU is becoming higher. Moreover, the proposed model would be used for other trigeneous platforms.

Table 3 shows the parameters that are derived from the proposed performance estimation model with the profiling data, and Fig. 9 shows the estimated execution times for tasks using the model. As mentioned in Section 4, only forward projection and backprojection are parallelized, since the other tasks are too small to be distributed efficiently across the three devices. With the parameters in Table 3 and from Fig. 6, optimal partitioning was

Table 3. Parameters of the performance estimation model for trigeneous computing in iterative image reconstruction.

Forward projection					
α	β	γ	δ	ε	ζ
0.0020	0.0061	0.0180	0.0209	0.0118	0.0428
Backprojection					
α	β	γ	δ	ε	ζ
0.0007	0.0533	0.0063	0.0279	0.0032	0.0178

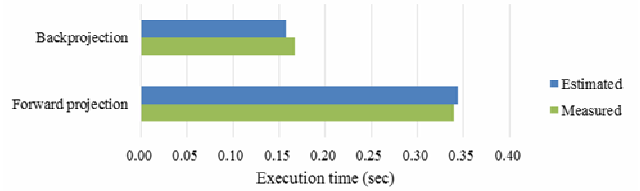

Fig. 9. Performance comparison between estimated and measured results for backprojection and forward projection.

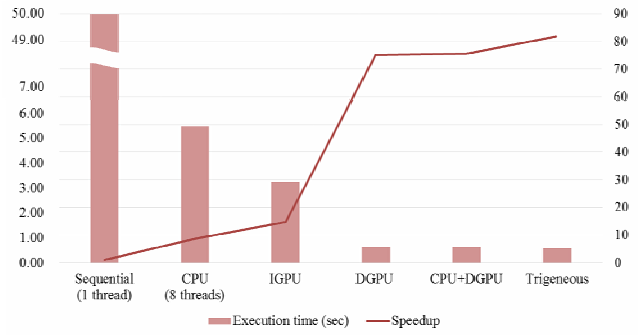

Fig. 10. Comparison of execution time and corresponding speedup for each device and the proposed trigeneous computing.

determined to be (169, 17, 25) for forward projection, and (149, 20, 42) for backprojection, where x, y, and z in notation (x, y, z) represent the number of assigned projection images (*i.e.*, workload) for the DGPU, CPU, and IGPU, respectively. As shown in Fig. 9, the proposed model estimates execution time accurately with an acceptable level of error: 1.19% for forward projection and 5.96% for backprojection.

This optimal partitioning of forward projection and backprojection achieved performance enhancement of 18.70% and 11.12%, respectively, compared to DGPU-only execution. Note that performance enhancement would vary depending on the specific devices in the system.

Fig. 10 shows the end-to-end execution time of the iterative image reconstruction application with different devices. The proposed trigeneous execution with optimal partitioning improves performance by 9.49%, compared to DGPU-only execution, which corresponds to 81.91 times speedup, compared to the sequential application. Note that CPU+DGPU hybrid execution does not improve much,

compared to the process of Lee et al. [3]. This is because iterative image reconstruction is a compute-bound application with relatively negligible data copy time, which is well-suited to the DGPU and tends to assign almost all workload to the DGPU. This results in little improvement, even if a CPU is utilized at the same time.

6. Conclusion

As heterogeneous computing has become prevalent these days, to efficiently exploit heterogeneous devices simultaneously has been of keen interest to many researchers. In this paper, we presented an efficient SART implementation using trigeneous devices: a CPU, a discrete GPU, and an integrated GPU. We first presented how SART can be accelerated efficiently using GPUs by considering SIMD architectures. Then, we proposed a performance estimation model and a mapping algorithm for optimal data partitioning on trigeneous devices to further increase the throughput of SART. By considering the memory access pattern, the execution time of SART on a GPU was reduced by 6.23 times, compared to a naïve GPU implementation. By utilizing a trigeneous device with the proposed estimation model, we further improved throughput by 9.4%. To the best of our knowledge, this is the first paper that deals with CPU-GPU^2 trigeneous devices.

Acknowledgement

This work was partly supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. R0190-16-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development) and (No. R0101-15-0054, WiseKB: Big data based self-evolving knowledge base and reasoning platform)

References

[1] A. K. Singh, et al: "Mapping on Multi/Many-core Systems: Survey of Current and Emerging Trends," *ACM DAC*, No. 1, May. 2013. Article (CrossRef Link)

[2] S. L. Shee, and S. Parameswaran: "Design Methodology for Pipelined Heterogeneous Multiprocessor System," *ACM DAC*, pp. 811-816, Jun. 2007. Article (CrossRef Link)

[3] S. Lee, et al.: "CPU-GPU hybrid computing for feature extraction from video stream," *IEICE Electron. Express*, Vol. 11, No. 22, Nov. 2014. Article (CrossRef Link)

[4] C. Oh, et al.: "Real-time face detection in Full HD images exploiting both embedded CPU and GPU," *IEEE ICME*, Jul. 2015. Article (CrossRef Link)

[5] S. K. Rethinagiri, et al.: "Trigeneous Platforms for Energy Efficient Computing of HPC Applications," *IEEE HiPC*, pp. 264-274, Dec. 2015. Article (CrossRef Link)

[6] L. A. Feldkamp, et al.: "Practical cone-beam algorithm," *J. Opt. Soc. Amer. A*, Vol. 1, No. 6. pp. 612-619, Jun. 1984. Article (CrossRef Link)

[7] A. H. Anderson, and A. C. Kak: "Simultaneous Algebraic Reconstruction Technique (SART): A Superior Implementation of the ART Algorithm," *Ultrasonic Imaging*, Vol 6, No. 1, pp. 81-94, Jan. 1984. Article (CrossRef Link)

[8] L. Shepp, and Y. Vardi: "Maximum Likelihood Reconstruction for Emission Tomography," *IEEE Trans. Med. Imaging*, Vol. 1, No. 2, pp. 113-122, Oct. 1982. Article (CrossRef Link)

[9] J. K. Kim, et al.: "Fast Iterative Image Reconstruction in X-Ray CT," *IEEE Trans. Signal Processing*, Vol 60, No. 10, pp. 5508-5518, Oct. 2012. Article (CrossRef Link)

[10] N. Sorokin: "Parallel Backprojector for cone-beam Computer Tomography," *IEEE ReConFig*, pp. 175-180, Dec. 2008. Article (CrossRef Link)

[11] Y. Lu, et al.: "Accelerating Algebraic Reconstruction Using CUDA-Enabled GPU," *IEEE CGIV*, pp. 480-485, Aug. 2009. Article (CrossRef Link)

[12] J. K. Kim, et al: "Hardware Acceleration of Iterative Image Reconstruction for X-Ray Computed Tomography," *IEEE ICASSP*, pp. 1697-1700, May. 2011. Article (CrossRef Link)

[13] K. Mueller, and R. Yagel: "Rapid 3-D cone-beam reconstruction with the simultaneous algebraic reconstruction technique (SART) using 2-D texture mapping hardware," *IEEE Trans. Med. Imaging*, Vol. 19, No. 12, pp. 1227-1237, Dec. 2000. Article (CrossRef Link)

[14] K. Kim: 3D Cone beam CT (CBCT) projection backprojection FDK, iterative reconstruction Matlab examples, Mar. 2012. Article (CrossRef Link)

Chanyoung Oh received his B.S. degree in electrical & computer engineering from University of Seoul in 2015. He is currently a M.S.-Ph.D. student in University of Seoul. His research interest includes parallel software design, heterogeneous computing, embedded GPU platforms, computer vision, pattern recognition and medical imaging. He is working at task mapping design methodology for CPU+GPU heterogeneous manycore platforms.

Youngmin Yi received the B.S. degree in computer engineering and the M.S. and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, Korea, in 2000, 2002, and 2007, respectively. He was a Postdoctoral Researcher at the University of California, Berkeley from 2007 and 2009, and a senior researcher at Samsung Advanced Institute of Technology from 2009 to 2010, before he joined the faculty of the School of Electrical and Computer Engineering, University of Seoul, where he is currently an Associate Professor. His research interest includes algorithm/architecture co-design for heterogeneous manycore platforms, GPU computing, high-performance distributed framework using manycore accelerators, and computer vision applications design for multiprocessor system-on-chip.