High Performance Computing Environment using General Purpose Computations on Graphics Processing Unit

http://dx.doi.org/10.28932/jutisi.v7i2.3715

Riwayat Artikel
Received: 19 Juni 2021 | *Final Revision*: 13 Juli 2021 | *Accepted*: 15 Juli 2021

Andreas Widjaja✉#1, Tjatur Kandaga Gautama#2, Sendy Ferdian Sujadi #3, Steven Rumanto Harnandy #4

Faculty of Information Technology, Universitas Kristen Maranatha

Jl. Prof. drg. Surya Sumantri, M.P.H. No. 65, Bandung - 40164, Jawa Barat, Indonesia

[1]andreas.widjaja@it.maranatha.edu

[2]tjatur.kandaga@it.maranatha.edu

[3]sendy.fs@it.maranatha.edu

[4]sharnandy@gmail.com

Abstract — **Here a report of a development phase of an environment of high performance computing (HPC) using general purpose computations on the graphics processing unit (GPGPU) is presented. The HPC environment accommodates computational tasks which demand massive parallelisms or multi-threaded computations. For this purpose, GPGPU is utilized because such tasks require many computing cores running in parallel. The development phase consists of several stages, followed by testing its capabilities and performance. For starters, the HPC environment will be served for computational projects of students and members of the Faculty of Information Technology, Universitas Kristen Maranatha. The goal of this paper is to show a design of a HPC which is capable of running complex and multi-threaded computations. The test results of the HPC show that the GPGPU numerical computations have superior performance than the CPU, with the same level of precision.**

Keywords— **GPGPU; high performance computing; parallel computing.**

I. INTRODUCTION

Over the past few years, improving the performance of computing systems has been made possible through several technological and architectural advances such as:

1. Increased memory size, cache size, bandwidth, reducing latency all of which contribute to higher performance on each computer / workstation [1].
2. Multicore architecture where one processor consists of more than one core (computing unit).
3. Cluster computing: connecting multiple computers into a cluster.
4. Between compute nodes: increase bandwidth and reduce interconnect latency.
5. Utilizing graphics processing unit (GPU) as a computing unit.

Among above, points 2 through 5, as discussed by [2] and [3], are parallel computing systems that involve more than one computing unit.

Parallel computing systems have recently become increasingly accessible to various users. Programmers can now get high-performance computing devices that are installed even on desktop computers. Most of the new computers sold today have multi core central processing unit (CPU) and graphics processing unit (GPU) features that can be used to run parallel programs. The use of such GPUs is often referred to as general purpose computations on graphics processing units (GPGPU) [3].

HPC, which was once only the domain of theoretical scientists, computer scientists and software developers, has now become increasingly important as a primary support tool for research in many fields of science.

The use of HPC in modeling complex physical phenomena such as forecasting and weather dynamics, fluid dynamics, interactions between atoms and molecules, astronomical calculations, engineering designs, market economy modeling,

interactions of brain cells, etc., are very common and have become a trend among researchers in these fields [2]. HPC is also now used in industry to improve products, reduce production costs and reduce the time needed to develop new products.

With the amount of data generated that grows exponentially, known as "Big Data", this increases the need for massive data analysis, where HPC is needed.

Recently the HPC is used by researchers in social media, semantics, geology, archeology, advanced materials, urban planning, graphics, genomics, brain imaging, economics, game design, and even music. This list will continue to grow as more and more fields of science are introduced into the possible use of HPC. The development trend of artificial intelligence (artificial intelligence) in various fields at this time also requires the use of HPC [4]. In particular, there are many *related works*, i.e., research studies utilizing GPGPU in their computations: Harmon et al. [5], researchers at NVIDIA and the National Institutes of Health (NIH), developed GPU-optimized AI models helping researchers to study COVID-19 in CT scans of a chest to better understanding and detecting infections; Gunraj et al. [6] developed COVID-Net CT-2 AI model, fed using a number of large and diverse datasets, which was created over several months with the University of Waterloo. The model was trained using NVIDIA RTX 6000 GPU, and it has 96% detection accuracy of COVID-19 detection in CT scans across a diverse number of scenarios; Amaro et al. [7] simulated SARS-CoV-2's spike protein movement to understand its behavior and access mechanism to the human cell. They utilized Nanoscale Molecular Dynamics (NAMD) and the Visual Molecular Dynamics (VMD) codes on OLCF Summit's 24,576 NVIDIA V100 GPUs running classical MD simulation of viruses with 305 million atoms. Since their computation generated an enormous amount of data (~200 TB), they used AI to identify the intrinsic features from the simulations and extract important information, hence they were able to simulate the virus and its mechanisms in unprecedented detail never done before. The team won the ACM Gordon Bell Special Prize for High Performance Computing-Based COVID-19 Research [8]. There are still many more works related with GPGPU computing, not to be mentioned here.

Those related works, which give significant impacts to the science, knowledge and community, which are also being the parts of the battle against COVID-19 pandemic, are the main motivation of conducting this research, hoping that it can give, at least, a small contribution to the science and knowledge discovery.

The goal of our research is to build HPC using GPGPU, which will further investigate its computing performance and capability, making it a useful tool for students and faculty members to support their research, particularly in doing heavy and complex scientific computations.

II. METHODOLOGY

Here our research methodology is described using strategy approach in brief as shown in Figure 1.

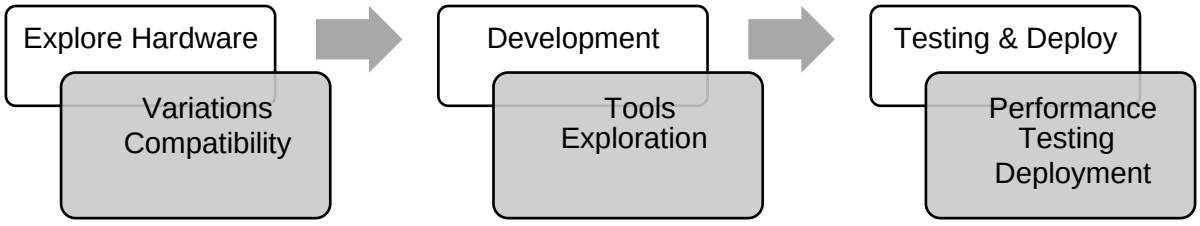

Figure 1 Research Methodology

Firstly, our strategy is to explore the variation of hardware available in the market, by looking up a reference in [9], several choices of general-purpose computing capable (not only graphics) of GPU products were found, based on consideration of their hardware compatibility to the system, such as PCIe 3.0 x16 or newer, compatibility pair with the main CPU, and software compatibility, such as operating system and their respective drivers.

Secondly, at this stage, the research methodology is done by an exploration of the features and capabilities of various tools (API / Library / Applications). These tools are useful for optimal use of the GPGPU HPC system environment to be built. The following are the results of the exploration.

A. HPC Frameworks

These are essential as main frameworks of the environments:

1) *NVIDIA CUDA® Toolkit 11.3:* NVIDIA CUDA® (Compute Unified Device Architecture) [10] is a parallel computing platform and programming model developed by NVIDIA for general computing on GPUs. With CUDA, developers can dramatically accelerate computing applications by harnessing the power of the GPU. In GPU-accelerated applications, the sequential part of the workload runs on the CPU which is optimized for single-threaded performance - while the intensive part of the application is calculated to run on thousands of GPU computing cores

in parallel. With CUDA, developers can program in popular languages such as C, C++, Fortran, Python, R and MATLAB and express parallelism through extensions in the form of several basic keywords. CUDA Toolkit version 11.3 from NVIDIA provides everything needed by developers to develop applications that are accelerated by the GPU. CUDA Toolkit consists of libraries, compiler, development tools, and binary runtime.

2) *OpenCL 3.0:* OpenCL™ (Open Computing Language) [11] is a standard for cross-platform, accelerating parallel programming for supercomputers, cloud servers, desktop computers, laptops, smartphones, and embedded platforms. OpenCL 3.0 aligns the OpenCL roadmap to provide the functionality needed by developers to be widely used by hardware vendors, and significantly increases deployment flexibility by empowering OpenCL implementations that are appropriate to focus on functionality relevant to the current computing market. OpenCL 3.0 also integrates parts of functionality into the main specifications, with the new OpenCL 3.0 programming specification, uses the new integrated specification format, and introduces extensions for asynchronous data copy for embedded processors.
3) *NVIDIA HPC SKD™ with OpenACC:* Open Accelerators (OpenACC) [12] is a programming standard for parallel programming created by Cray, CAPS, NVIDIA and PGI. OpenACC was created to simplify parallel programming on heterogeneous CPU / GPU systems. NVIDIA HPC SDK [13] with OpenACC gives scientists and researchers the convenience of high-performance computing more easily by inserting a "hint" or directive compiler into C11, C++ 17 or Fortran code. With the NVIDIA OpenACC compiler, program code can be run on GPU and CPU. Besides, the NVIDIA HPC SDK has a library for the GPU complete with its development tools. With this, researchers can focus more on their science/research while developing their scientific computing programs.
4) *NVIDIA DirectCompute*: DirectCompute [14] is an API created by Microsoft for GPGPU, which is part of DirectX as a layer for direct access to GPU hardware. DirectCompute is supported by NVIDIA for GPUs equipped with DirectX 10 and DirectX 11 capabilities.
5) *NVIDIA cuDNN*: NVIDIA CUDA® Deep Neural Network library (cuDNN) [15] is a library that uses GPUs for deep neural networks. cuDNN provides tuned-up implementations for standard routines such as forward & backward convolution, pooling, normalization, and activation layers.

B. HPC Libraries

These libraries are advanced computing toolkits. They are most popular, readily available and suitable for GPGPU accelerated computing environments.

1) *Tensorflow*: TensorFlow [16] is a powerful library for numerical computing, specifically and optimized for Machine Learning projects. TensorFlow was created by the Google Brain team and is used in many services such as Google Cloud Speech, Google Photos, and Google Search. TensorFlow is an open source that began in November 2015 and is currently the most popular library for Deep Learning. TensorFlow is used for many things such as image classification, natural language processing, time series forecasting, etc.
2) *Keras*: Keras [17] is a high-level Deep Learning API that can be used easily to build, train, evaluate, and run various types of neural networks. Keras runs on top of the libraries that can be selected, namely TensorFlow, Microsoft Cognitive Toolkit (CNTK), and Theano.
3) *PyTorch*: PyTorch [18] is a tool and framework for Machine Learning and Deep Learning created by Facebook's artificial intelligence division. One of the PyTorch applications is for large-scale image analysis. PyTorch can be used to implement complex algorithms. PyTorch can process computing in a GPU environment.
4) *GROMACS*: GROningen MAchine for Chemical Simulations (GROMACS) [19] [20] is a molecular dynamics application designed to simulate Newtonian equations of motion for systems containing hundreds to millions of particles. GROMACS is designed to simulate biochemical molecules such as proteins, lipids, and nucleic acids that have many complex bond interactions. GROMACS can be used in an HPC environment that utilizes a combination of GPU-CPU performance.

Lastly, performance testing is done to the GPGPU system to measure its capability, especially for scientific computation which is the main purpose of the research. Here is the strategy:

(i) Testing CPU numerical computation performance of integer and floating points operations: native arithmetic operations, General Matrix-to-matrix Multiply (GEMM), data compression, and cryptographic computations.
(ii) Testing GPU numerical computation performance of integer and floating points operations and comparing it with the CPU: GEMM, Fast Fourier Transform (FFT), N-Body Simulation (SNBODY), linear system solvers (non-iterative and iterative), dynamic parallelism, binomial options pricing computation, convolution, sorting algorithms, and particle dynamics.

III. SET-UP

The GPU is installed in relatively powerful hardware, utilizing a quite fast and state-of-the-art AMD Ryzen 9 3900X CPU, clocked at a frequency of 3.8 GHz (4.6 GHz Max Boost), which is manufactured using 7-nm lithography technology. The CPU has 12 computing cores and is capable of running 24 simultaneous threads (simultaneous multithreading (SMT) capable), with 64-bit instruction set extension. The system is equipped with 64 GB high-clocked rate (3200MHz) DDR4 random access memory (RAM), thus capable to handle multiple requests simultaneously. A very fast 1 TB large capacity solid state drive (SSD) M.2 PCIe NVMe is installed to enhance I/O high data streams. This high specification hardware will ensure the system to run smoothly even under high load.

With consideration of price per performance measure and our research budget, the decision was ended up by choosing NVIDIA GeForce® RTX 3080 Specification and GPU with brand iGame GeForce RTX 3080 Advanced OC 10G-V, as shown in Figure 2.

Name	iGame GeForce RTX 3080 Advanced OC 10G-V
GPU Chipset	NVIDIA GeForce® RTX 3080
Architecture	NVIDIA second generation RTX architecture
GPU Code Name	GA102-200-KD-A1
Manufacturing Process	8nm lithography process
CUDA Cores	8704
Core Clock	Base: 1440 MHz; Boost: 1710 MHz
One-Key OC	Base: 1440 MHz; Boost: 1785 MHz
Memory Clock	19 Gbps
Memory Size	10 GB
Memory Bus Width	320 bit
Memory Type	GDDR6X
Memory Bandwidth	760 GB/s
Power Connector	3*8 Pin
Power Supply	10+6+4
Thermal Design Power (TDP)	370 Watts
Display Ports	3*DP+HDMI
Fans Type	13 blades automatic
Heat Pipe Number/Spec	5-φ8
Power Max	800 W
DirectX	DirectX 12 Ultimate/OpenGL4.6
NV technology Support	NVIDIA DLSS, NVIDIA G-SYNC, 2nd Gen Ray Tracing Cores
Slot Number	3 slots
Size	315.5*131*60 mm
Weight	1.6 kg (N.W)

Figure 2. iGame GeForce RTX 3080 Advanced OC 10G-V Specification and GPU [21]

For the scientific computing purpose, NVIDIA CUDA Toolkit version 11.3 [10] is installed, enabling the GPU to perform general purpose computation (GPGPU) using the application programming interface (API) provided.

Architecture of the NVIDIA "Ampere" RTX 3xxx is shown in Figure 3 which shows a block diagram of each Streaming Multiprocessor (SM) that consists of CUDA cores, Tensor cores and Ray Tracing (RT) cores. It can be seen on the figure that the architecture has single precision integer (INT32) and single precision floating points (FP32) registers along with their tensor cores. The double INT32 and double FP32 serve as double precision integer (INT64) and floating point (FP64), respectively. It also has hybrid floating point-integer (FP32+INT32) registers [22].

Figure 3. NVIDIA Ampere Streaming Multiprocessor Architecture [22]

IV. Results of Performance Testing and Analysis

Because the HPC system that was built is intended for scientific computing, the tests carried out were CPU and GPU performance to perform numerical operations on integer, floating point and cryptography. The performance of the HPC environment is a combination of CPU and GPU performance as well as memory bandwidth. The following is the analysis of performance of the components.

A. CPU

In this HPC system, as stated before, the CPU used is the AMD Ryzen 9 3900X which has 12 cores [23]. This processor has the AMD Zen2 (Matisse) architecture with Simultaneous Multi-Threading (SMT) technology [24] [25], where each core can run two logical threads, so this processor has a total of 24 logical threads. The working frequency of this CPU is 3.8 GHz (default), and can be boosted up to 4.6 GHz.

The test was carried out using a synthetic benchmark from Geekbench 4.0 [26], namely Single Precision General Matrix-to-matrix Multiply (SGEMM), performing matrix multiplication operation as follows:

$$C_{ij} \leftarrow C_{ij} + \sum_{k=1}^{n} A_{ik} B_{kj} \qquad (1)$$

with $i = [1, n]$ and $j = [1, n]$, where *A*, *B* and C are square matrices with size $n = 512$. The test result shows the CPU performance is 820.0 GFLOPS (Giga (10^9) Floating Point Operations Per Second) of running operation (1).

The next test was performed using SiSoft Sandra software [27] to measure the performance of native arithmetic operations using the highest performing processor's instruction sets (AVX2, FMA3, AVX), which gave the following results:

- Aggregate Native Performance: 410.17 GOPS
- Dhrystone Integer Native AVX2: 549.8 GIPS
- Dhrystone Long Native AVX2: 557.70 GIPS
- Whetstone Single-float Native AVX/FMA: 339.99 GFLOPS
- Whetstone Double-float Native AVX/FMA: 281.87 GFLOPS

where GOPS stands for Giga Operations Per Second, and GIPS stands for Giga Instructions Per Seconds.

To measure the performance of integer operations in real world applications, a data compression application was also conducted using 7-Zip [28] to measure the integer instruction rate using the ZIP algorithm, with the results: 81,238 GIPS for the compression process and 88,411 GIPS for the decompression process.

The performance in cryptography was tested using VeraCrypt [29], which is based on TrueCrypt software, which uses the Advanced Encryption Standard (AES) encryption algorithm [30] [31], in which embedded in the processor, known as AES-NI extension [32]. The test result shows that the CPU performance for data encryption is 8.9 GB/s.

B. GPU

The main component in this HPC system, as described above, is the NVIDIA GeForce RTX 3080 GPU with GA102 graphics processor (GA102-200-KD-A1) which has the Ampere architecture. This GPU has 8704 CUDA (FP32) cores (shading units), 272 texture mapping units (TMU), 96 render output units (ROP) and 68 raytracing acceleration cores (RT). There are also 272 tensor cores to accelerate machine learning or deep learning computations. The working frequency of this GPU is 1.44 GHz (default), and can be boosted up to 1.71 GHz. The tests carried out included the performance of cryptography, integer, floating point and memory bandwidth using SiSoft Sandra software. The test results show that the encryption performance with Crypto AES-256 algorithm is 92 GB/s, Crypto SHA256 [33] is 317 GB/s. Meanwhile, the floating-point test result for FP32 with SGEMM computation shows 12590 GFLOPS performance, Single Precision Fast Fourier Transform (SFFT) computation shows 889 GFLOPS performance, and Single Precision N-Body Simulation (SNBODY) computation shows 11317 GFLOPS performance. The memory bandwidth test results show 622 GB/s internal performance, 18.7 GB/s upload performance, 19.8 GB/s download performance, and 157 ns global latency (In-Page Random Access).

Fast Fourier Transform (FFT) [34] [35] is an optimized algorithm of Fourier Transform, which computes a transformation of signals or time series from time domain into frequency domain, $\omega = 2\pi\nu$, as follows:

$$X(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \qquad (2)$$

and the inverse transform

$$x(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} X(\omega) e^{j\omega t} d\omega \qquad (3)$$

where $x(t)$ and $X(\omega)$ are the signal, or time series, and its transform, respectively, and here $j = \sqrt{-1}$ is the imaginary number. Then the FFT is performed by discretizing the signal into *N* evenly spaced time series x_n and applying the discrete version of Eq. (2) in the form of

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi j}{N} nk} \qquad (4)$$

and decompose Eq. (4) into even and odd indices of x_n into

$$X_k = \left(\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi j}{N/2} mk} \right) + e^{-\frac{2\pi j}{N} k} \left(\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi j}{N/2} mk} \right) = E_k + e^{-\frac{2\pi j}{N} k} O_k \qquad (5)$$

Here E_k and O_k are the even and odd part of the transform, respectively, which are calculated recursively for every E_k and O_k by decomposing the transform by half (N/2) in every recursion step until only one data point left (details of the computation are given in [34] and [35]). For our testing purpose, the computation was done using single precision floating point.

The N-Body Simulation [36] performed is based on the computation of the Newtonian gravitational interaction of N particles. The gravitational potential arising on the i-th particle from the j-th with mass m_i and m_j, respectively, is

$$V(r_{i,j}) = -\frac{G m_i m_j}{r_{i,j}} \qquad (6)$$

Therefore, the force acting on every particle is the gradient of the potential, which is computed using

$$\mathbf{F} = \nabla V(r_{i,j}) = \left(\hat{\boldsymbol{x}} \frac{\partial}{\partial x} + \hat{\boldsymbol{y}} \frac{\partial}{\partial y} + \hat{\boldsymbol{z}} \frac{\partial}{\partial z} \right) V(r_{i,j}) \qquad (7)$$

Every particle moves according to the Newtonian law of motion, governed by the gravitational force according to Eq. (7), a vector $\mathbf{F} = (F_x, F_y, F_z)$ in three dimensions acting on every particle (see [36] for details of the computation). In our case, the simulation was computed using single precision floating point.

Some test code samples of NVIDIA CUDA Toolkit package, which are coded in C++, were compiled. The source code is compiled using Visual Studio 2019 C++ compiler [37]. Some of the important test code samples were executed on the system, resulting in the following.

Tests utilizing cuBLAS library [38], which is a highly optimized implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA CUDA runtime, were performed using single precision GEMM (SGEMM) and double precision GEMM (DGEMM), which results shown in Figure 4. However, the test utilized not all SMs of the GPU hence resulted in lower performance GFLOPS than using SiSoft Sandra software.

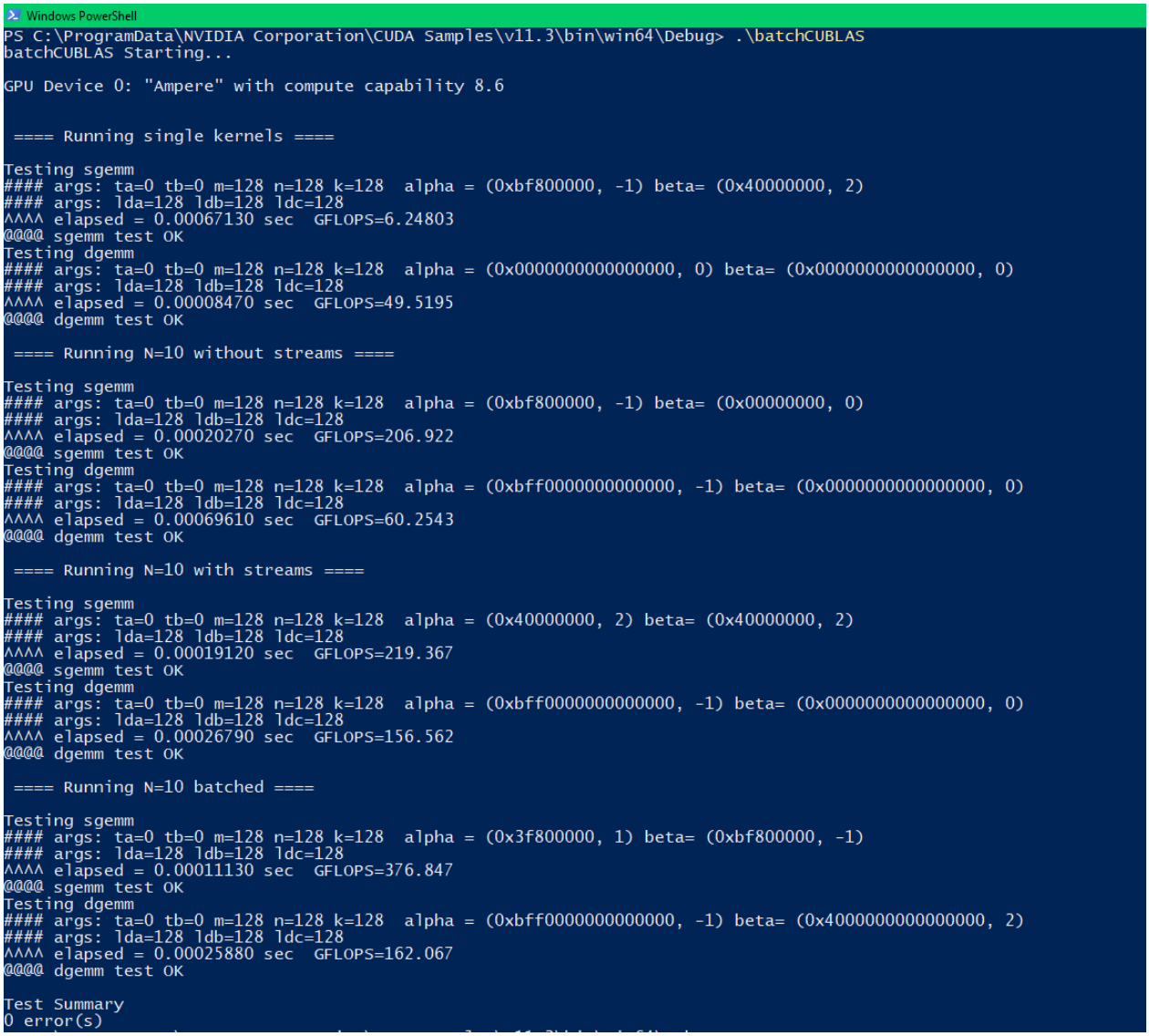

Figure 4. cuBLAS performance of GEMM computation

In this test, cuBLAS Integer, FP32 and TensorFloat-32 GEMM computation using Warp Matrix Multiply and Accumulate (WMMA) API were performed on the GPU's tensor cores, as shown in Figure 5.

p-ISSN : 2443-2210
e-ISSN : 2443-2229

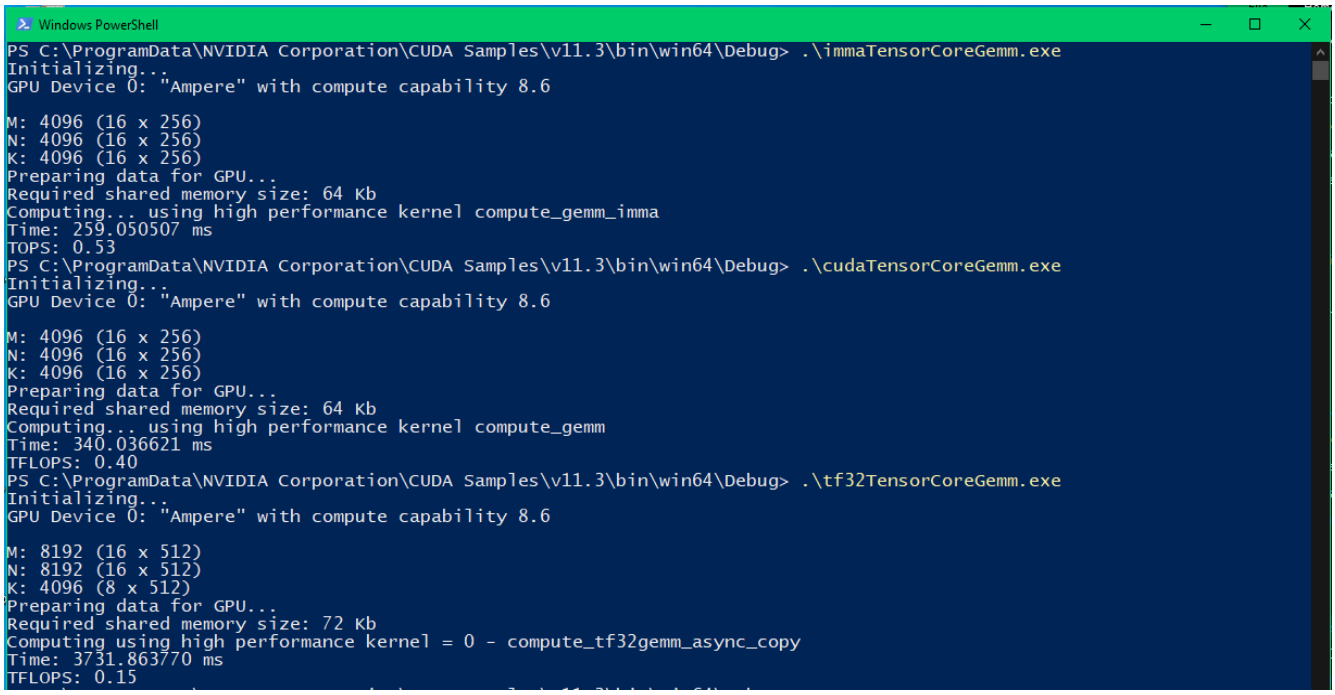

Figure 5. Integer, FP32 and TensorFloat-32 GEMM computation on GPU's tensor cores

Here simple matrix multiplications were performed using CUDA kernel and cuBLAS (see Figure 6), in the form of

$$C_{ij} = \sum_{k=1}^{r} A_{ik} B_{kj} \qquad (8)$$

with $i = [1, m]$ and $j = [1, n]$, where matrices *A*, *B*, and *C* have sizes $m \times r$, $r \times n$, and $m \times n$, respectively. The results show cuBLAS outperformed CUDA kernel.

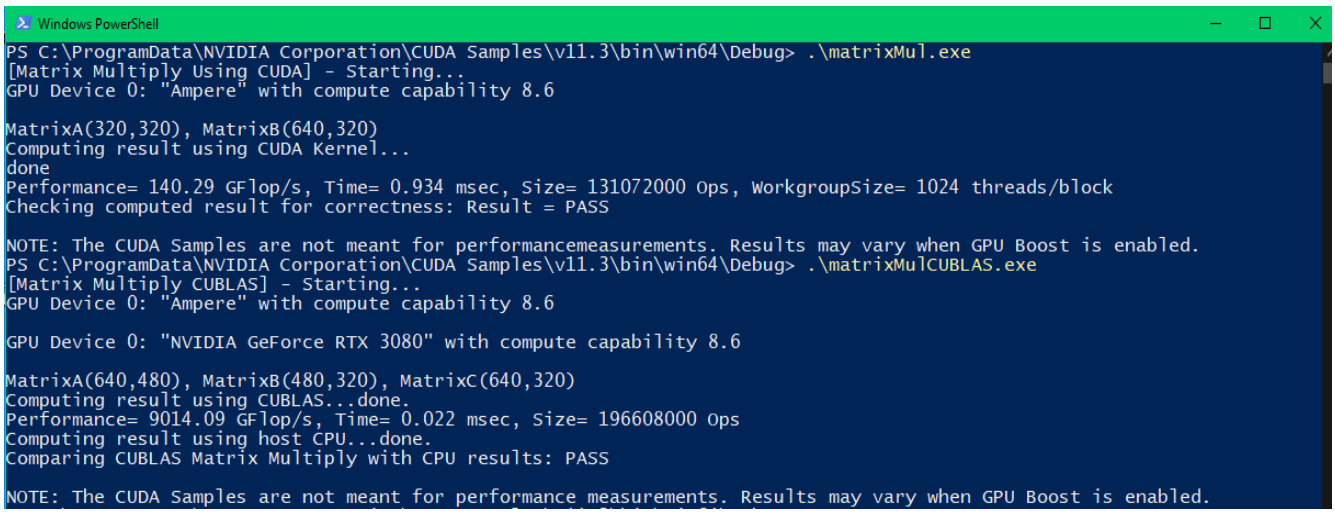

Figure 6. CUDA kernel and cuBLAS simple matrix multiplications

Operations of matrix transpose

$$A_{ij}^{\mathrm{T}} = A_{ji} \qquad (9)$$

were performed using various methods utilizing all of the CUDA cores. The GPU performed very well with very high throughput, 36 GB/s up to 350 GB/s (see Figure 7).

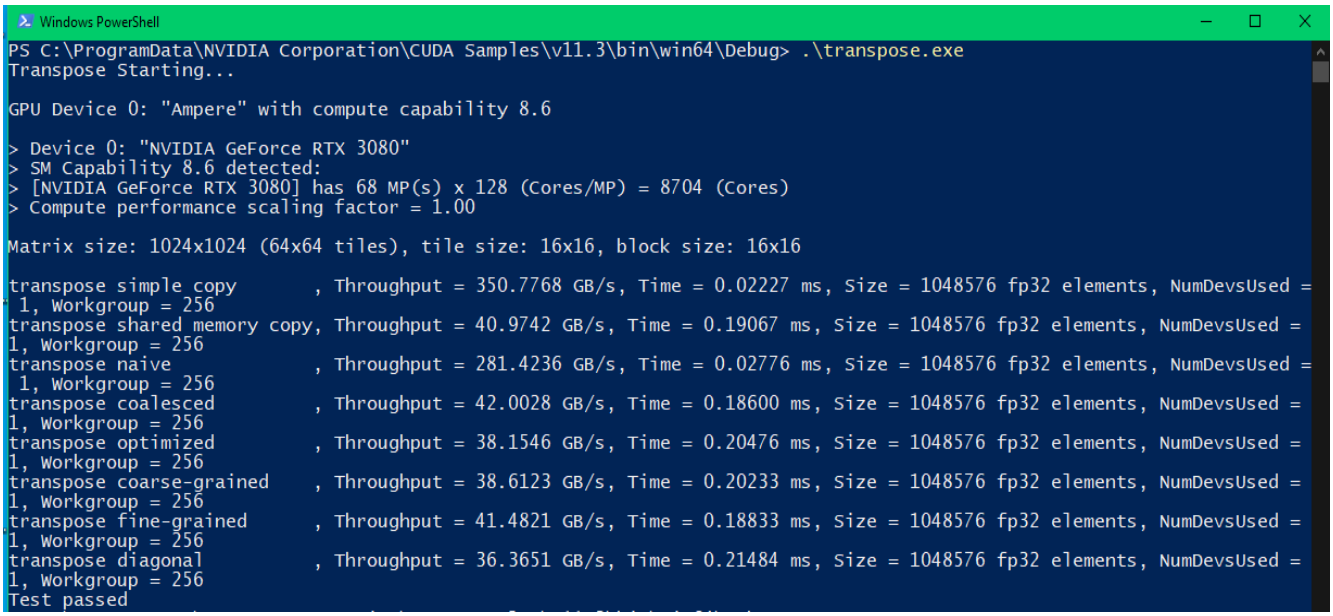

Figure 7. Matrix transpose operations using various methods

A linear system solver cuSolverDn was performed to solve a large and dense linear system

$$\mathbf{Ax} = \mathbf{b} \qquad (10)$$

which performs QR factorization and LU with partial pivoting applied to a matrix **A** (general matrix, sparse or dense, symmetric or non-symmetric), as shown in Figure 8. For symmetric or Hermitian matrices, there is Cholesky factorization, while for symmetric indefinite matrices, Bunch-Kaufman factorization is provided [39]. The computation of the solver using the GPU produces remarkably a very precise solution (with negligible residual in the order of 10^{-16}).

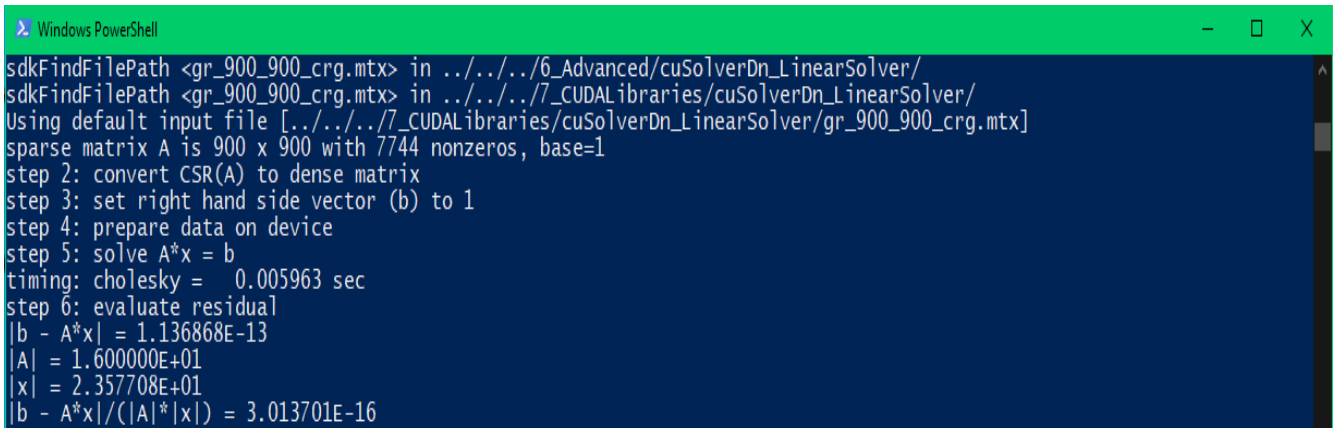

Figure 8. Solving a dense linear system using cuSolverDn

A test to compute the solutions of sets of relatively large (100 million matrix elements) sparse linear systems by fast re-factorization (LU factorization) using CPU and GPU [39] was performed. Remarkably, the result shows that both CPU and GPU produce the same results (with a negligible difference in the order of 10^{-17} due to machine precision, see Figure 10).

In Figure 9 CUDA dynamic parallelism was performed on the GPU applying recursive fashion.

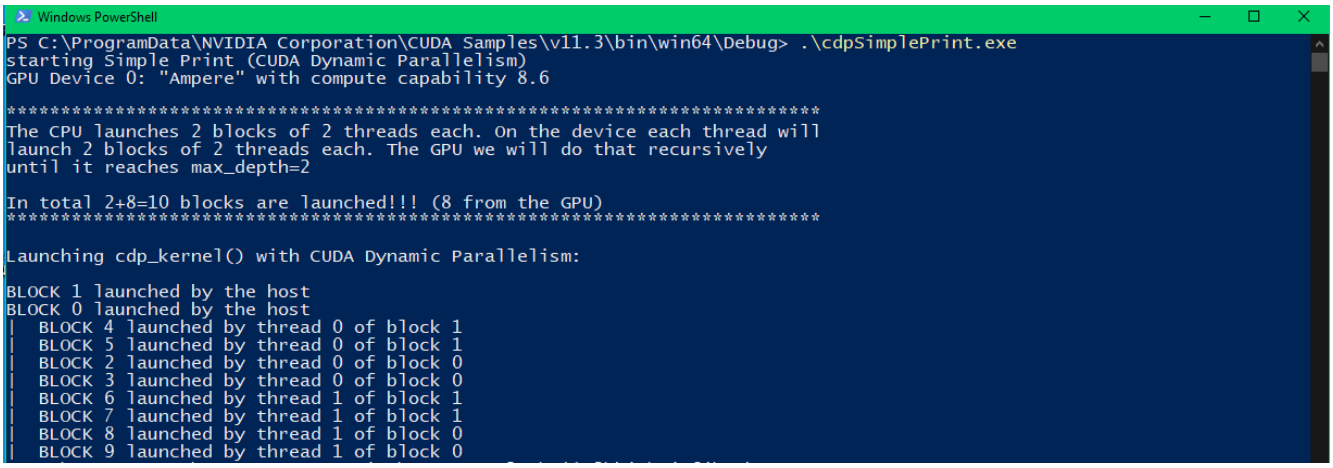

Figure 9. Recursive CUDA dynamic parallelism

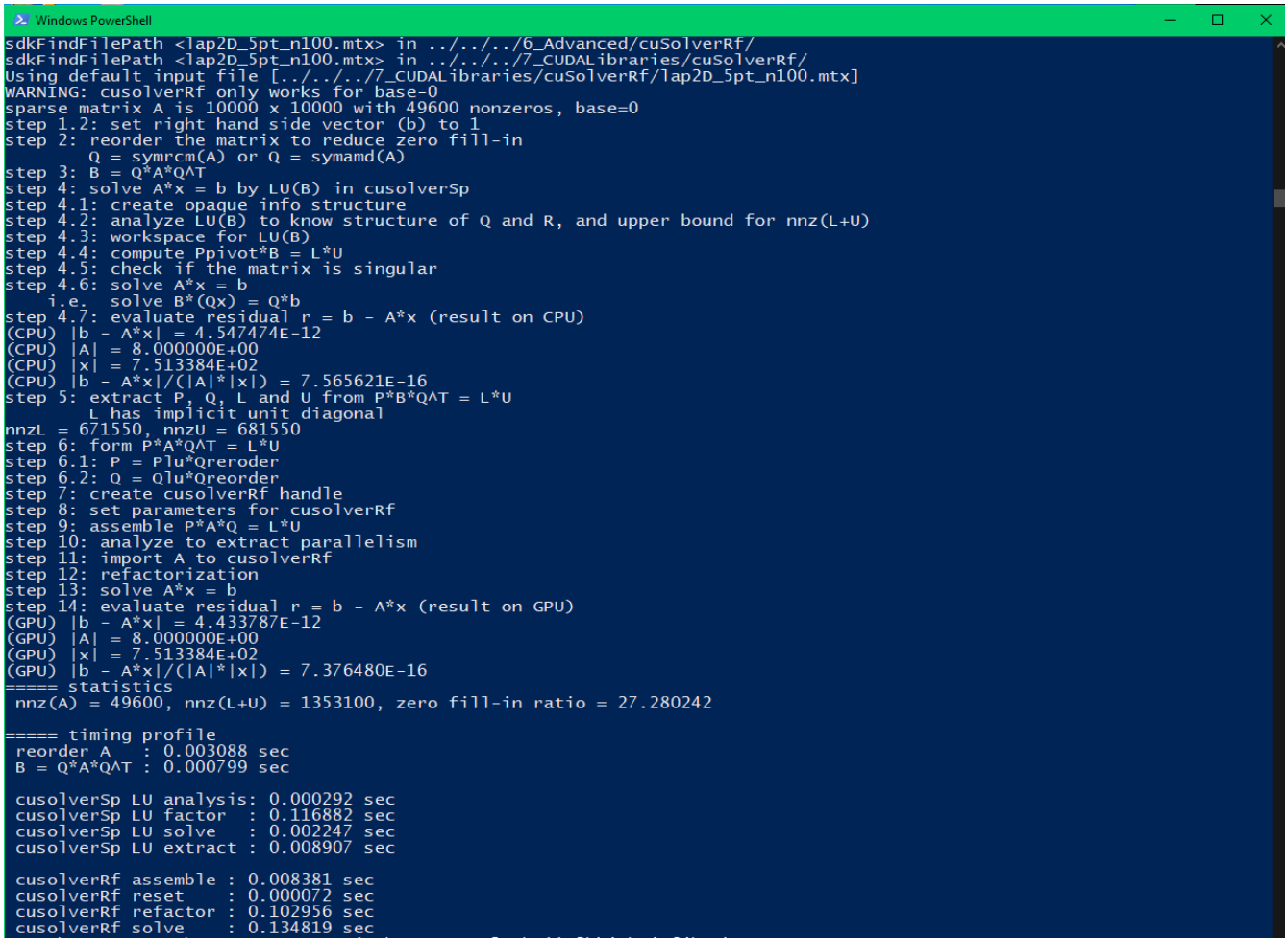

Figure 10. Computing solution of sets of sparse linear systems by fast re-factorization using CPU and GPU.

A computation of binomial options pricing model [40] [41] for the valuation of options was performed, using CPU and GPU, as shown in Figure 11. The algorithm uses a "discrete-time" tree-based model of the price variation over time of the underlying financial instrument, i.e., stocks, ETF, etc., with cases where the Black-Scholes closed-form formula [42] [43] is wanting.

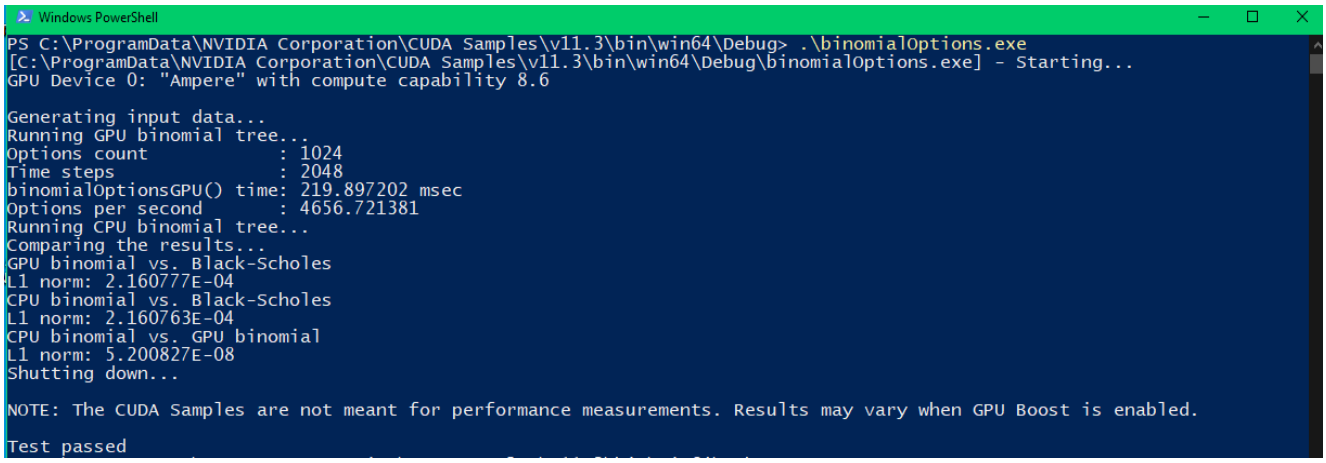

Figure 11. Computation of binomial options using CPU and GPU

The Black-Scholes option pricing model computation is based on a partial differential equation describing the evolution of option price over time, $V(t)$, on the relationship with the underlying asset price, $S(t)$, asset volatility, σ, and the force of interest (continuously compounded annualized risk-free interest rate), r, which is expressed as

$$\frac{\partial V(t)}{\partial t} + \frac{1}{2}\sigma^2 S(t)^2 \frac{\partial^2 V(t)}{\partial S(t)^2} + rS(t)\frac{\partial V(t)}{\partial S(t)} - rV(t) = 0 \qquad (11)$$

The solution of Eq. (11) is $V(S,t)$, as a function of the underlying asset S, at time t; in particular $C(S,t)$ is the price of a European call option and $P(S,t)$ the price of a European put option; T, time of option expiration, with $\tau = T - t$, the time to maturity; and K, the strike price (exercise price) of the option. Black and Scholes solved Eq. (11) (see [42] and [43] for the details) using boundary conditions: (i) $C(S,t) = 0$, (ii) $C(S,t) \to S$ as $S \to \infty$, and (iii) $C(S,T) = (S - K, 0)$, to get the value of a call option for a non-dividend-paying underlying stock in terms of the Black–Scholes parameters,

$$C(S(t),t) = N(d_1)S(t) - N(d_2)Ke^{-r(T-t)} \qquad (12)$$

and its corresponding put option price, based on put-call parity with discount factor $e^{-r(T-t)}$,

$$P(S(t),t) = Ke^{-r(T-t)} + C(S(t),t) - S(t) = N(-d_2)Ke^{-r(T-t)} - N(-d_1)S(t) \qquad (13)$$

where

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left(\ln\left(\frac{S(t)}{K}\right) + \left(r + \frac{\sigma^2}{3}\right)(T-t)\right) \qquad (14)$$

and

$$d_2 = d_1 - \sigma\sqrt{T-t} \qquad (15)$$

with $N(z)$ denotes the standard normal cumulative distribution function, that is,

$$N(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z} e^{-x^2/2}\,dx \qquad (16)$$

Both fair call and put prices for a given set of European options as in Eqs. (12) and (13) were computed in CPU and GPU for eight million options. The throughputs of CPU and GPU are 0.375 GOptions/s and 5.7548 GOptions/s, respectively, showing that the GPU is more than 15 times faster than the CPU. Remarkably, both CPU and GPU produce approximately the same result, with negligible difference in the order of 10^{-7} (see Figure 12).

A computation of iterative solver algorithm of $N \times N$ linear system Eq. (10), namely conjugate gradient method [44], was also performed (see Figure 13). In this case, the coefficient matrix $\mathbf{A}$ is symmetric, $\mathbf{A}^{\mathrm{T}} = \mathbf{A}$, and positive-definite, $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0$, for all non-zero vectors $\mathbf{x} \in \mathbb{R}^n$. The algorithm is to minimize the function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{b} \qquad (17)$$

when its gradient is zero,

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0} \qquad (18)$$

which is actually Eq. (10). This iterative solver is efficient, particularly using parallel implementation in the GPU. The details of the derivation of the algorithm are presented in [45] chapter 10, and [46] chapter 2.

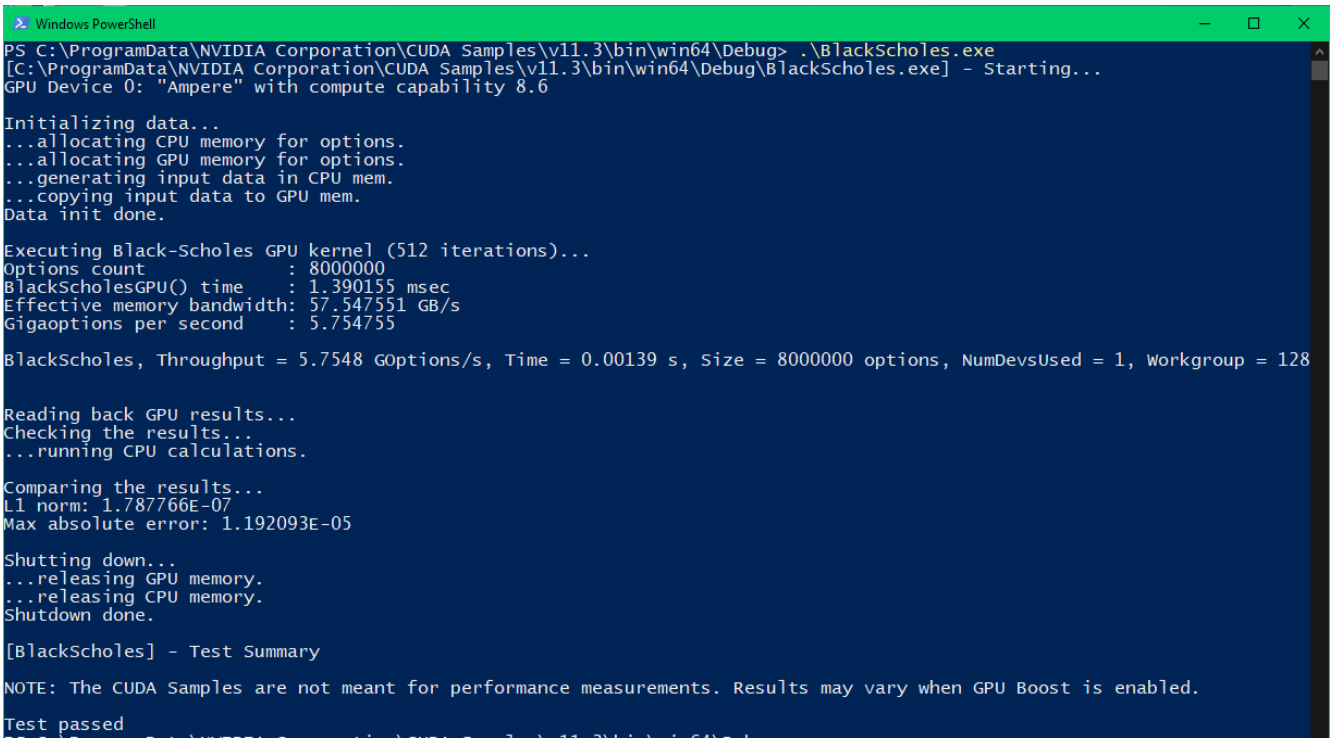

Figure 12. Computation of Black-Scholes option pricing model

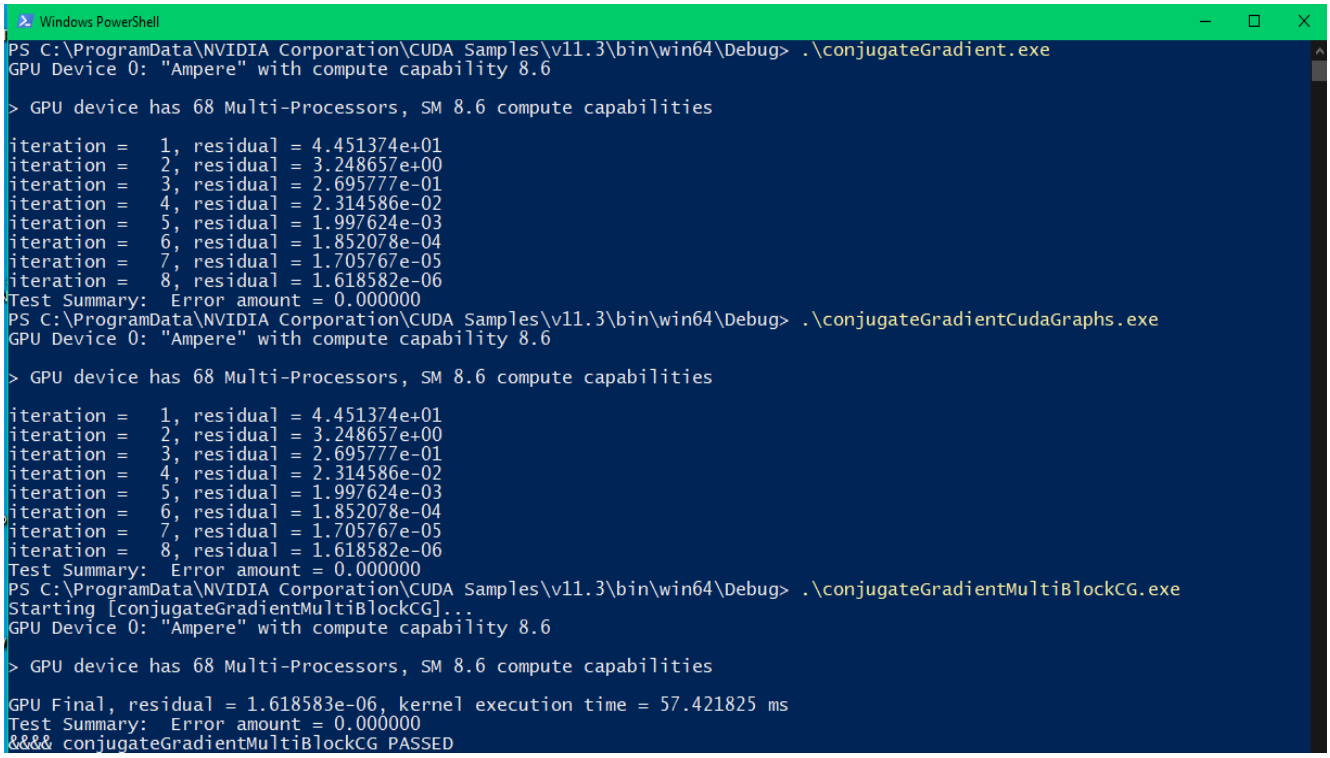

Figure 13. Computation of conjugate gradient method of $N \times N$ linear system

Computation of FFT-based 2D convolution was also performed [47]. The 2D convolution, denoted as $\mathbf{C} = \mathbf{A} * \mathbf{B}$, which is expressed as

$$C(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} A(x-u, y-v)B(u,v)\, du\, dv \qquad (19)$$

and in discrete form

$$c_{p,q} = \sum_{r=0}^{m-1} \sum_{s=0}^{n-1} a_{p-r,q-s} b_{r,s} \qquad (20)$$

which can be done efficiently by using 2D FFT:

$$\mathbf{C} = FFT^{-1}(FFT(\mathbf{A}) \cdot FFT(\mathbf{B})) \qquad (21)$$

where $FFT(\mathbf{X})$ and $FFT^{-1}(\mathbf{X})$ denotes 2D FFT and its inverse, respectively, with input $\mathbf{X}$ as two-dimensional array, and the " $\cdot$ " operator denotes element wise multiplication.
The computation took a 2048 × 2048 pixels image as the input data and convolved it with a convolution kernel, running on GPU and CPU as reference. Both GPU and CPU produced approximately the same result, with negligible difference in the order of 10^{-7} (see Figure 14).

Various sorting algorithms, namely quick sort, merge sort and radix sort were also tested. The radix sort is using Thrust, a highly optimized parallel algorithms library applied to GPU [48] [49], which was the best performance (see Figure 15).

p-ISSN : 2443-2210
e-ISSN : 2443-2229

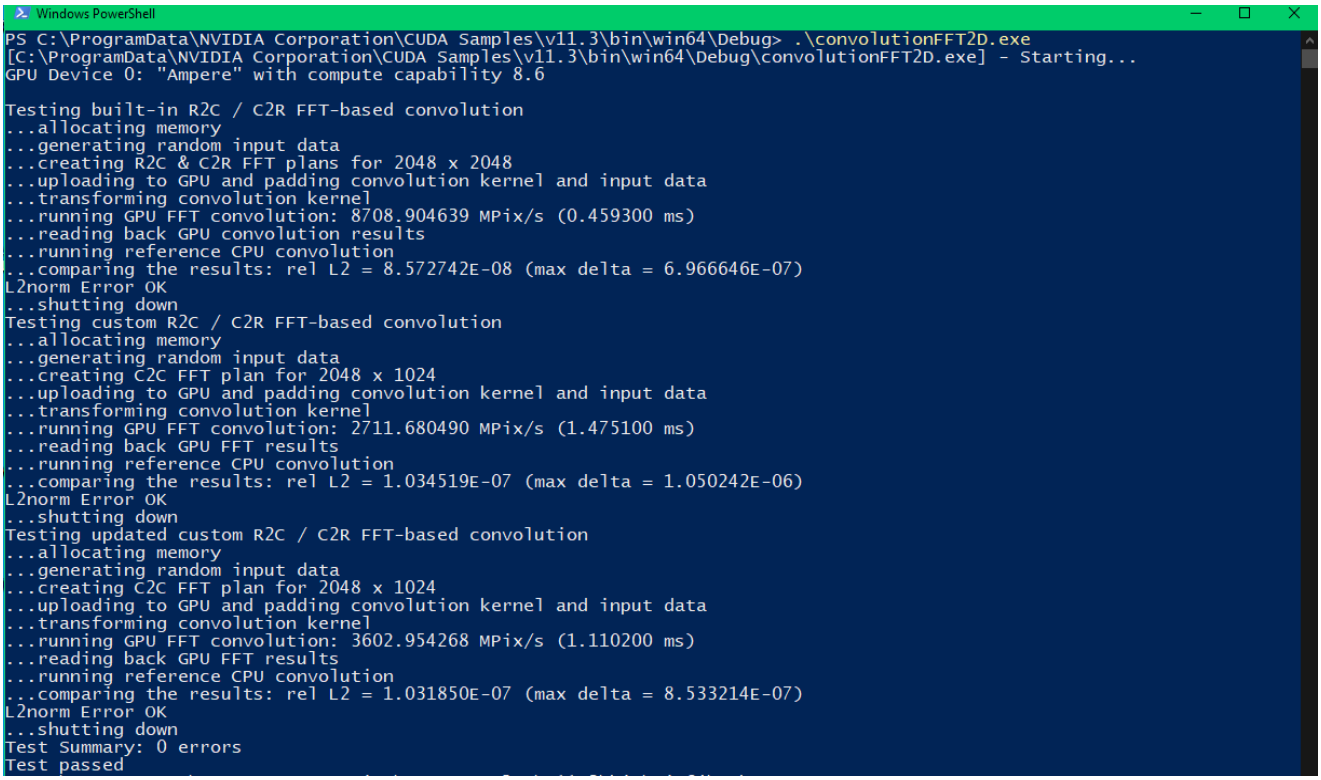

Figure 14. Computation of FFT-based 2D convolution

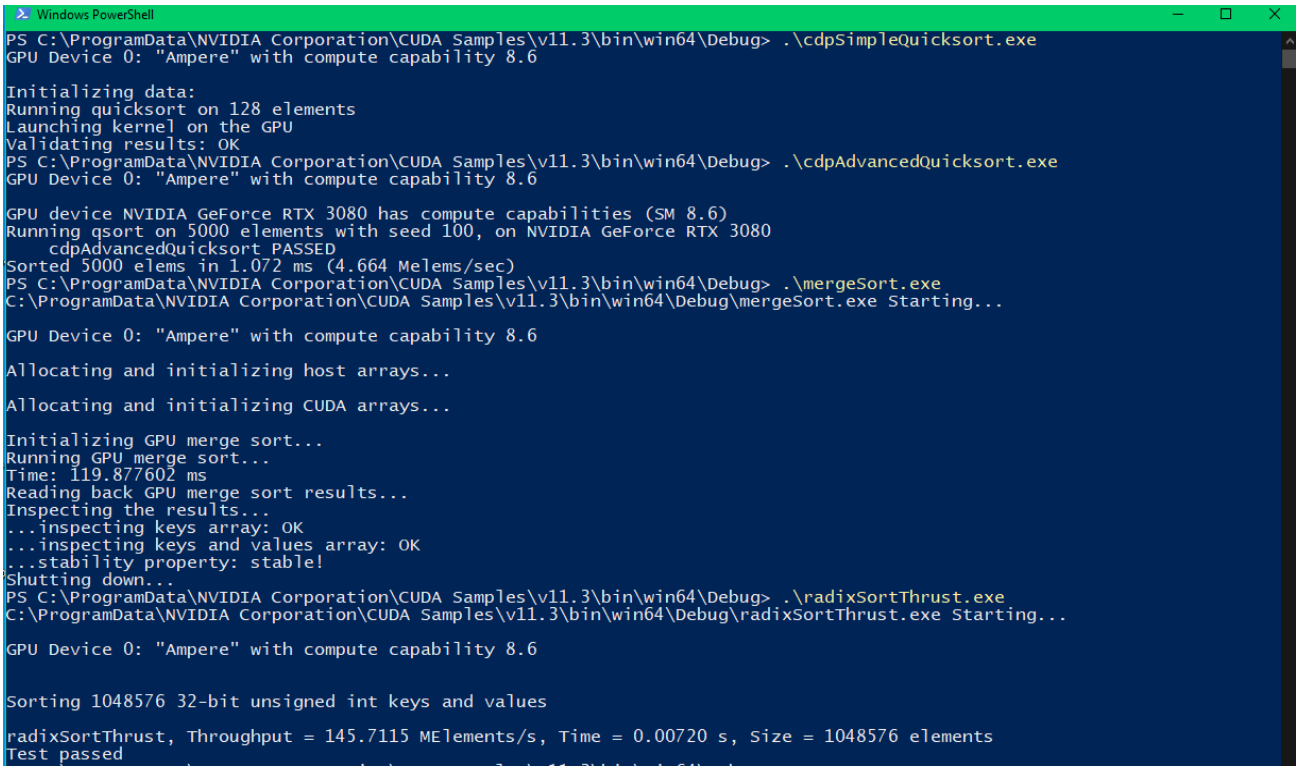

Figure 15. Computation of various sort algorithms

The N-body simulation of particle dynamics was also tested using the compiled NVIDIA sample code (CUDA N-body) running on the GPU. The simulation computed 69632 particles dynamics, including their interactions, governed by Newtonian laws of force and gravitation, Eqs. (6) and (7). A snapshot of the particle dynamics is captured and shown in Figure 16.

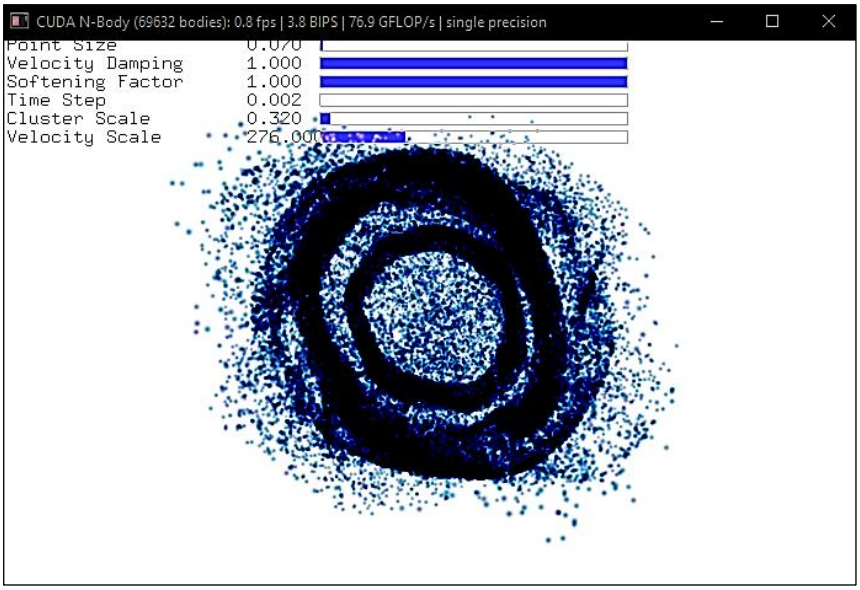

Figure 16. A snapshot of N-body particle simulation. The color is inverted.

Simulation of many particle dynamics under earth gravity is also performed, where every particle is a ball and can collide with other particles and the confined walls. The particle dynamics are governed by Newtonian laws of motions under earth gravity, rigid body kinematics, and conservation of energy and momentum [50]. A snapshot is shown in Figure 17.

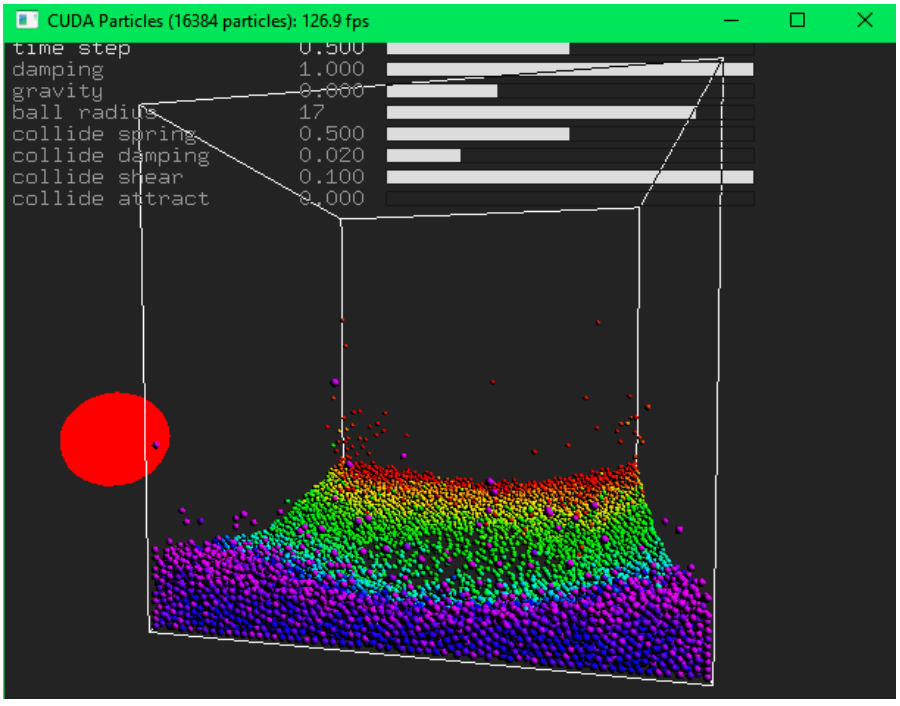

Figure 17. A snapshot of simulations of ball particles under earth gravity

V. DISCUSSION

The results of tests performed on GPU and CPU confirm that the performance of the GPU is superior to the CPU. What it means is the performance in term of scientific/numerical computation, i.e., computations which involve "number crunching", with massive streams of numerical data (integers or floating points), and their operations in standard arithmetic and some elementary mathematical functions, such as trigonometric functions, transcendental functions, hyperbolic functions, logarithmic functions, powers and roots. Other advanced functions can be computed using optimized algorithms involving many elementary functions. However, it does not mean that the GPU can replace the CPU, because the CPU can do many things that the GPU cannot, like I/O managements, threads scheduling, branch predictions, memory management, and many more complex tasks. The superiority of the GPU over CPU is only on massive parallel numerical computations, because it is designed to do so in the first place, that is, graphics operations are just mathematical operations acting on numbers.

TABLE 1.
SUMMARY OF NUMERICAL COMPUTATION TEST RESULTS

Test	Result
GEMM	The GPU is more than 15 times faster than the CPU performance. The GPU's tensor cores utilization greatly improves the performance.
Options pricing model computation	The GPU is more than 15 times faster than the CPU performance.
Cryptographic computation	The GPU is more than 10 times faster than the CPU performance.
FFT, convolution, linear system solvers	GPU and CPU produce similar precision.
N-body, particle dynamics simulations	The GPU can handle computation involving a large number of objects at once.

Computational capabilities of the GPU were tested using various numerical algorithms, as summarized in Table 1. The GEMM computation is essential, since matrix-to-matrix multiplication is the important operation in linear algebra, and being a part in many advanced algorithms, such as deep learning (neural networks) [51], machine learning (SVD, regressions, etc.) [52], signal processing and time series analysis [53], finite element analysis [54], differential equations solvers [46], molecular dynamics simulations [19] [7], physics simulations [36], bioinformatics [55], computer graphics [56], image processing and computer vision [57], computational finance [58], and many more. Most linear system solver algorithms use GEMM as a part of them. The linear system solver algorithms themself are building blocks of many other complex algorithms [46]. As the computation of FFT and convolution were tested on the GPU, the results were accurate with good precision, implying the capability of the GPU to do computations involving FFTs. The FFT is widely used for applications in science, technology, engineering, mathematics, music and multimedia. There are many important applications of the FFT [59]. According to Strang, the FFT is the most important algorithm of our lifetime [60]. Even IEEE magazine Computing in Science & Engineering included the FFT into the "Top 10 Algorithms of 20th Century" [61]. The tensor cores utilization greatly improves our GPU performance, suitable to run applications using libraries which can take advantage of tensor cores, namely NVIDIA cuDNN [15], TensorFlow [16], Keras [17] and PyTorch [18]. Simulations of particles dynamics and N-body in the GPU show that the GPU is able to handle many body complex simulations involving a large number of objects at once, thus capable to run applications such as molecular dynamics simulations, such as GROMACS [19], etc.

The availability of many libraries which support the GPU is also an advantage, opening a great opportunity to utilize the GPU to its maximum potential. Most of the libraries are highly optimized for massive parallelism and multithreaded computations provided by the GPU computing cores. Additionally, the continuous support from the GPU manufacturer are very beneficial, as the drivers and toolkits are updated regularly to address bugs issues and to improve the performance.

With the reason for those advantages and benefits, deployment of our HPC using the GPGPU system is ready, which serves as a scientific computing platform that will be used to accommodate computational projects of students and members of the faculty.

VI. CONCLUSION

The HPC using GPGPU system is developed, which is capable of running computations with massive parallelism taking advantage of many cores it has. The performance testing to the GPU, applying various important algorithms was successfully passed. The test showed that the GPU has superior performance than the CPU, in terms of numerical computations, hence the CPU-GPU tandem pair will be beneficial for the HPC system. Our HPC using GPGPU as a scientific computing platform is readily deployed.

Our future works are to add access for users to run their computational jobs on the system, locally or remotely, and adding user management and job scheduler. More importantly, our HPC system will be able to facilitate computational tasks of research projects of faculty members and students.

ACKNOWLEDGEMENTS

We thank the Faculty of Information Technology, Universitas Kristen Maranatha for funding this research. Lembaga Penelitian dan Pengabdian Kepada Masyarakat (LPPM) Universitas Kristen Maranatha is also acknowledged for facilitating the administrative process necessary for this research.

REFERENCES

[1] R. Garabato, A. More and V. Rosales, "Optimizing Latency in Beowulf Clusters," *CLEI Electronic Journal*, vol. 15, no. 3, 2012.
[2] M. Norman, *Parallel Computing for Data Science, with Examples in R, C++ and CUDA*, New York: Chapman & Hall CRC, 2016.
[3] T. Soyata, *GPU Parallel Program Development Using CUDA*, Chapman & Hall/CRC Computational Science, 2018.
[4] NVIDIA, "Deep Learning," NVIDIA, (2020). [Online]. Available: https://developer.nvidia.com/deep-learning. [Accessed 2 Jan 2020].
[5] S. A. Harmon, T. H. Sanford and S. Xu et al., "Artificial intelligence for the detection of COVID-19 pneumonia on chest CT using multinational datasets," *Nature Communications*, vol. 11, no. 4080, 2020.
[6] H. Gunraj, A. Sabri, D. Koff and A. Wong, "COVID-Net CT-2: Enhanced Deep Neural Networks for Detection of COVID-19 from Chest CT Images Through Bigger, More Diverse Learning," *arXiv:2101.07433*, vol. https://arxiv.org/abs/2101.07433, 2021.
[7] R. E. Amaro et al., "AI-Driven Multiscale Simulations Illuminate Mechanisms of SARS-CoV-2 Spike Dynamics," in *Proceedings of SC20*, Virtual Event, November 16-19, 2020.
[8] O. Peckham, "Behind the Gordon Bell Prize-Winning Spike Protein Simulations," HPCwire, [Online]. Available: https://www.hpcwire.com/2021/03/11/behind-the-gordon-bell-prize-winning-spike-protein-simulations. [Accessed 1 Apr 2021].
[9] Techpowerup, "GPU Specs Database," Techpowerup, (2020). [Online]. Available: https://www.techpowerup.com/gpu-specs. [Accessed 15 Jun 2021].
[10] NVIDIA, "CUDA Toolkit 11.3," (2021). [Online]. Available: https://developer.nvidia.com/cuda-downloads. [Accessed 1 Mar 2021].
[11] The Khronos Group Inc, "OpenCL," The Khronos Group Inc., (2020). [Online]. Available: https://www.khronos.org/opencl/. [Accessed 2 Mar 2020].
[12] "OpenACC," (2020). [Online]. Available: https://www.openacc.org/get-started. [Accessed 2 Jan 2020].
[13] NVIDIA, "NVIDIA HPC SDK with OpenACC," NVIDIA, (2020). [Online]. Available: https://developer.nvidia.com/hpc-sdk. [Accessed 1 Mar 2020].
[14] "DirectCompute," (2020). [Online]. Available: https://developer.nvidia.com/directcompute. [Accessed 2 Jan 2020].
[15] NVIDIA, "NVIDIA cuDNN," NVIDIA, (2020). [Online]. Available: https://developer.nvidia.com/cudnn. [Accessed 1 Mar 2020].
[16] Google, "TensorFlow," (2020). [Online]. Available: https://www.tensorflow.org. [Accessed 1 Mar 2020].
[17] F. Chollet, "Keras," (2020). [Online]. Available: https://keras.io. [Accessed 1 Mar 2020].
[18] Facebook, "PyTorch," (2020). [Online]. Available: https://pytorch.org. [Accessed 1 Mar 2020].
[19] M. J. Abraham, , T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess and E. Lindahl, "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," *SoftwareX*, vol. 1 2, p. 19–25, 2015.
[20] G. Team, "GROMACS," (2020). [Online]. Available: http://www.gromacs.org. [Accessed 1 Mar 2020].
[21] Colorful, "iGame Colorful GeForce RTX 3080," Colorful, (2020). [Online]. Available: https://en.colorful.cn. [Accessed 1 Sep 2020].
[22] "NVIDIA details GeForce RTX 30 Ampere architecture," VideoCardz.com, (2020). [Online]. Available: https://videocardz.com/newz/nvidia-details-geforce-rtx-30-ampere-architecture. [Accessed 1 Sep 2020].
[23] AMD, "AMD Ryzen 9 3900X," Advanced Micro Devices, Inc. (AMD), (2020). [Online]. Available: https://www.amd.com/en/products/cpu/amd-ryzen-9-3900x. [Accessed 1 Sep 2020].
[24] D. M. Tullsen, S. J. Eggers and H. M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," *ACM SIGARCH Computer Architecture News* , vol. 23, no. 2, pp. 392-403, May 1995.
[25] G. Blokdyk, *Simultaneous multithreading A Complete Guide*, New York: 5STARCooks, 2018.
[26] "Geekbench 4," Primate Labs, (2020). [Online]. Available: https://www.geekbench.com/doc/geekbench4-cpu-workloads.pdf. [Accessed 2 Nov 2020].
[27] SiSoftware, "SiSoft SANDRA," SiSoftware, (2020). [Online]. Available: https://www.sisoftware.co.uk. [Accessed 2 Nov 2020].
[28] I. Pavlov, "7-zip," (2020). [Online]. Available: https://www.7-zip.org. [Accessed 2 Nov 2020].
[29] IDRIX, "VeraCrypt," IDRIX, (2020). [Online]. Available: https://www.veracrypt.fr. [Accessed 2 Nov 2020].
[30] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback and J. F. Dray Jr., "Advanced Encryption Standard (AES)," Federal Information Processing Standard 197 (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, USA, 2001.
[31] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti and E. Roback, "Report on the Development of the Advanced Encryption Standard (AES)," *Journal of research of the National Institute of Standards and Technology*, vol. 106, no. 3, pp. 511-77, 2001.
[32] E. G. AbdAllah, Y. R. Kuang and C. Huang, "Advanced Encryption Standard New Instructions (AES-NI) Analysis: Security, Performance, and Power Consumption," in *ICCAE 2020: Proceedings of the 2020 12th International Conference on Computer and Automation Engineering*, Sydney NSW Australia, 14–16 February 2020.
[33] NIST, "Proposed Revision of Federal Information Processing Standard (FIPS) 180, Secure Hash Standard," *Federal Register. National Institute of Standards and Technology*, vol. 59, no. 131, p. 35317–35318, 1997.
[34] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297-301, 1965.
[35] D. Takahashi, *Fast Fourier Transform Algorithms for Parallel Computers*, Singapore: Springer, 2019.
[36] K. Hawick, D. P. Playne and M. G. B. Johnson, "Numerical Precision and Benchmarking Very-High-Order Integration of Particle Dynamics on GPU Accelerators," in *Proceedings of International Conference on Computer Design (CDES 2011)*, Las Vegas, USA, 18-21 July 2011.
[37] Microsoft, "Visual Studio," Microsoft, (2020). [Online]. Available: https://visualstudio.microsoft.com. [Accessed 1 Jan 2020].
[38] NVIDIA, "cuBLAS," NVIDIA, (2020). [Online]. Available: https://docs.nvidia.com/cuda/cublas. [Accessed 1 Mar 2020].
[39] NVIDIA, "CUSOLVER Library," NVIDIA, (2020). [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUSOLVER_Library.pdf. [Accessed 1 Jan 2021].

[40] J. C. Cox, S. A. Ross and M. Rubinstein, "Option pricing: A simplified approach," *Journal of Financial Economics*, vol. 7, no. 3, pp. 229-263, September 1979.

[41] D. M. Chance, "A Synthesis of Binomial Option Pricing Models for Lognormally Distributed Assets," *Journal of Applied Finance (Formerly Financial Practice and Education)*, vol. 18, no. 1, pp. 38-56, 2008.

[42] F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637-654, 1973.

[43] J. C. Hull, *Options, Futures, and Other Derivatives, 10th Edition*, London: Pearson, 2018.

[44] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the National Bureau of Standards*, vol. 49 , no. 6, pp. 409-436, December 1952.

[45] W. Hackbusch, *Iterative Solution of Large Sparse Systems of Equations, 2nd Edition*, Switzerland: Springer, 2016.

[46] W. H. Press, S. A. Teukolsky and W. T. Vetterl, *Numerical Recipes, The Art of Scientific Computing, 3rd Edition*, Cambridge: Cambridge University Press, 2007.

[47] V. Podlozhnyuk, "FFT-based 2D convolution," NVIDIA Corporation, Santa Clara, 2007.

[48] J. Hoberock and N. Bell, "Thrust," NVIDIA, (2020). [Online]. Available: https://thrust.github.io. [Accessed 1 Jan 2021].

[49] J. Hoberock and N. Bell, "NVIDIA Developer: Thrust," NVIDIA, (2020). [Online]. Available: https://developer.nvidia.com/thrust. [Accessed 1 Jan 2021].

[50] J. Walker, D. Halliday and R. Resnick, *Fundamentals of Physics, 10th edition.*, Hoboken, NJ: John Wiley & Sons, 2014.

[51] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, Cambridge, Massachusetts: MIT Press, 2016.

[52] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning. Data Mining, Inference, and Prediction, 2nd Edition*, New York: Springer-Verlag, 2009.

[53] D. C. Montgomery, C. L. Jennings and M. Kulah, *Introduction to Time Series Analysis and Forecasting, 2nd Edition*, Hoboken, NJ: John Wiley & Sons, 2015.

[54] J. N. Reddy, *An Introduction to the Finite Element Method, 3rd Edition*, New York: McGraw-Hill Education, 2006.

[55] N. C. Jones and P. A. Pevzner, *An Introduction to Bioinformatics Algorithms*, Cambridge, Massachusetts: MIT Press, 2004.

[56] J. Vince, *Mathematics for Computer Graphics, 5th Edition*, London: Springer-Verlag, 2017.

[57] M. K. Bhuyan, *Computer Vision and Image Processing, Fundamentals and Applications*, Boca Raton, FL: CRC Press/Taylor & Francis Group, 2020.

[58] J.-C. Duan, W. K. Hardle and J. E. Gentle, *Handbook of Computational Finance*, Berlin Heidelberg: Springer-Verlag, 2012.

[59] D. N. Rockmore, "The FFT: an algorithm the whole family can use," *Computing in Science & Engineering*, vol. 2, no. 1, p. 60–64, January 2000.

[60] G. Strang, "Wavelets," *American Scientist*, vol. 82, no. 3, p. 250–255, May–June 1994.

[61] J. Dongarra and F. Sullivan, "Guest Editors Introduction to the top 10 algorithms," *Computing in Science & Engineering*, vol. 2, no. 1, p. 22–23, January 2000.