**A review of**

**the features and applications of CUDA model**

# ABSTRACT

Application of CUDA in large scale computer graphics computing CUDA (Compute Unified Device Architecture), which is first raised by NVIDIA, is a developing environment using GPU processing general-purpose computer program. It focus on High-performance computing based on running tens of thousands of threads simultaneously and simplification in develop GPU- based program. CUDA environment is widely apply into multiple areas, such as Digital Image Processing, neural network computing and medical imaging. This paper mainly taking about the features of CUDA and how these features work together to providing fast parallel GPU-based computing, furthermore, it summarizing the performance among its applying.

Key words: CUDA, GPGPU, parallel computing, RSA

Contents

[ABSTRACT 2](#_Toc415551233)

[1. Introduction 4](#_Toc415551234)

[2. GPGPU computing 7](#_Toc415551235)

[2.1 GPU construction 7](#_Toc415551236)

[2.2 GPGPU mechanism 10](#_Toc415551237)

[3. CUDA model 12](#_Toc415551238)

[3.1 CUDA feature. 13](#_Toc415551239)

[3.2 CUDA compile system 16](#_Toc415551240)

[3.3 Nvidia support 18](#_Toc415551241)

[3.4 CUDA algorithm 18](#_Toc415551242)

[4. Conclusion 24](#_Toc415551243)

[Reference 25](#_Toc415551244)

# 1. Introduction

In traditional computer architecture, CPU bear the main processing task. And in the other side, GPU always cast like an external device. But along with the dramatically development in GPU performance, we eventually find that GPU can also afford processing task and even doing better than CPUs.

Using GPU to finish biomedical purpose computing have been start in a very long time, since XXXX. However, the biomedical purpose computing develop program with direct graphic language to achieve the highest performance of GPUs. So, their achievement is had to generalize to other field of working, in other words, it have limited portability hard to develop by general programmers.

In XXXX, one of the greatest GPU Corporation (https://developer.nvidia.com/about-cuda) release the CUDA platform that enables dramatic increase in generally computing performance by harnessing the power of the GPU. CUDA provide a massively multithreaded general-purpose architecture with up to 128 processor cores and thousands of threads in flight, programmable in C and capable of hundreds of billions of floating-point operations each second[1] [CUDA: SCALABLE PARALLEL PROGRAMMING FOR HIGH-PERFORMANCE SCIENTIFIC COMPUTING.6] . CUDA can and can only run in the GPU from NVidia later than G80, including the consumer grade GeForce series, the professional grade Quadro and the high performance computing core tesla series. Profit from the multiple core Desterilize and parallel design of computing core, GPU have significant advantage in the great scale simple computing and highly parallelization algorithm. Hundreds of GPU core work together in CUDA platform make an accelerated HPC (high-performance computing) application compelling platform.

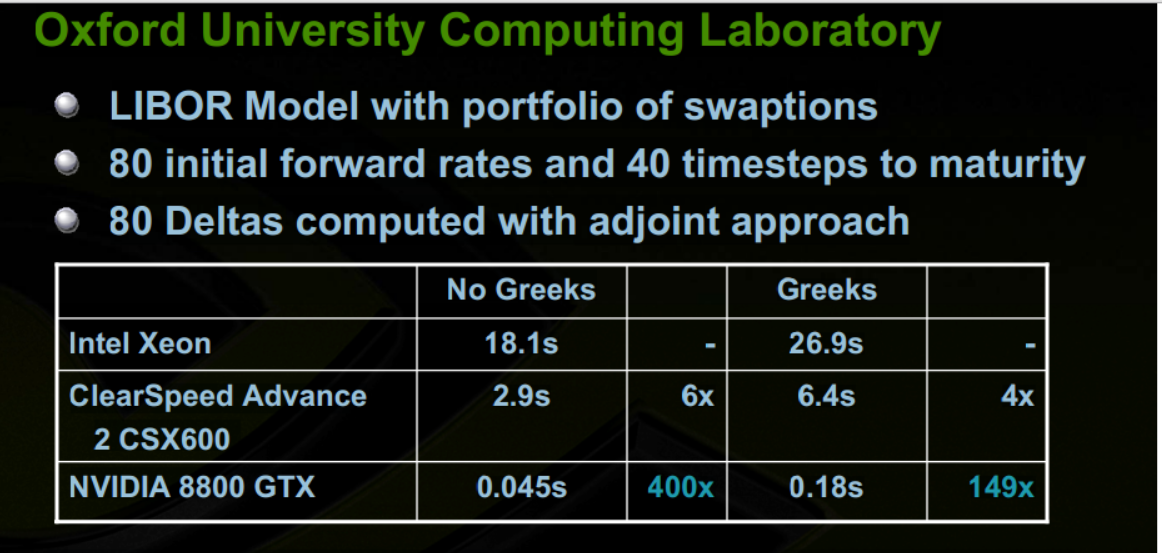


Table 1-1 The performance between GPU with CUDA and CPU

From labor application Oxford University Computing Laboratory, Mike Giles and Su Xiao Ke found NVIDIA 8800 GTX performing 400 times speed up in no Greeks computing and 149 times Greeks computing than the Xeon CPU, the most powerful CPU form intel in same generation. Also in the Pseudo-spectral simulation of 2D Isotropic turbulence computing, the performance speed up 17 times from CPU only to CPU+GPU work mode. CUDA can effectively express the parallelism in general purpose computing.

The massive multi core design from GPU enable the large scale parallel computing.

No knowledge of graphics programming is required—just the ability to program in a modestly extended version of C [2].

RSA key algorithm is the large integer arithmetic, which widely implement in worldwide application in security internet communication. We would explain and display the RSA algorithm parallelization mechanism CUDA realization, and gives and highlight the based on

Shared memory to achieve performance optimization techniques.

This paper would firstly introduced the GPU construction and how it can be processing the data better than CPU. Then, introduce the inner system and compile procedure when using CUDA model in programming. Finally, the article would give certain example in how CUDA apply into common and advanced algorithm and the how the CUDA computing performance.

# 2. GPGPU computing

## 2.1 GPU construction

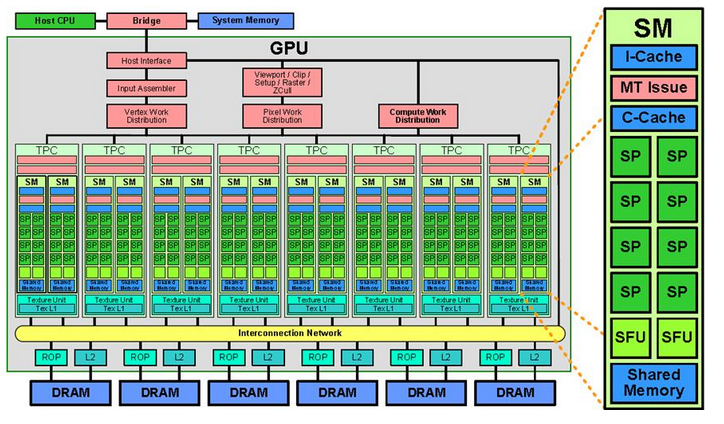
Contemporary graphic card achieving very strong computing capability, extremely high-speed memory and fast bandwidth. Most contemporary graphic card connect to mainboard with the PCI-E bus, PCI-E using multiple channel to adjust its slot and streaming speed. GPU also using GDDR memory to store data for processing, which implement the similar technology but transfer with much more frequency than normal on-board memory. The core part of graphic card is the GPU. GPU integrate tremendous billions of transistors, which is one hundred percent more than the quantity of transistors in CPU.

|  |  |  |
| --- | --- | --- |
|  | Intel Core i7-5960X | NVIDIA GeForce GTX980 |
| Lithography | 22nm | 28nm |
| Die(diced) size | 355.5 mm2 | 398mm2 |
| Processor Base Frequency | 3.0Ghz | 7.0Ghz(GM) |
| Transistors count | 2.6 billion | 5.2billion |
| Processor unit count | 8 | 2048 |
| TDP（Thermal Design Power） | 140w | 165w |

Table2-1: The comparison between latest GPU and CPU

However, as the table show, GPU have over 300% computing capability than CPU in one core. Furthermore is that GPU have 25500% more processor count inside processing data from memory, so it need 7Ghz graphic memory frequency to support its need in store, read and synchronize data. For CPU core, it only have several cores and designed with cached with Coherency and Consistency (CC). But in GPU, its caches are most read-only and it’s even hard to keep the CC in one GPU processor, therefore GPU can hardly keep the global data CC. Thus, GPU is designed to computing parallel data and algorithm.

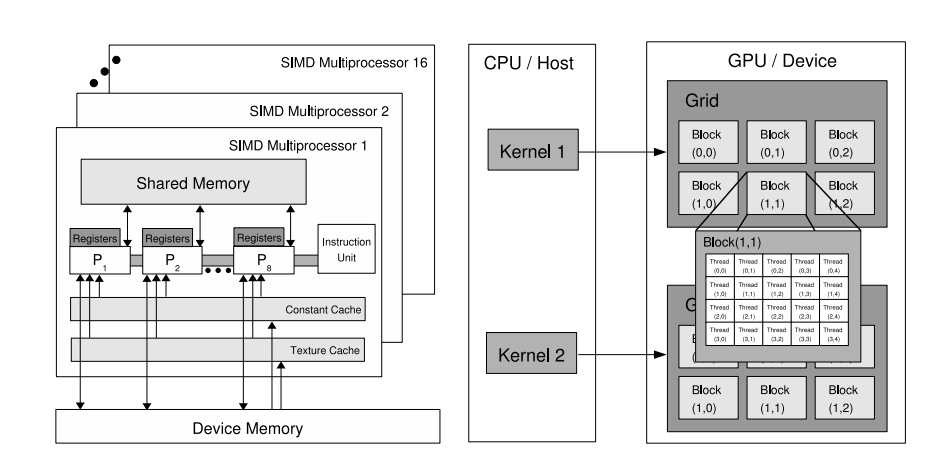
Born with these large processor and transistors quantity feature, GPU have a series complicated but well-optimized inner elements to finish its processing tasks. Given examples in the newest GM204 GPU core for NVIDIA. It named its processor as the Scalable Streaming Processor Array (SPA), they can processing task simultaneously and synchronous.



Graphic 2-1 GPU construction

## 2.2 GPGPU mechanism

General Purpose Graphic Process Unit (GPGPU) is one of the fastest developing area in computer science filed. It try to implement general purpose computing task with GPU instead of only processing graphics. As to all the Nvidia GPU supporting CUDA, their computing part are consist of many Multiprocessors with Single Instruction Multiple Data shard memory, every Multiprocessors are the combination of an array of stream processors, which processing the same instruction during certain giving time period, but with different data[3]. Every Multiprocessor can using four kinds of chip-based storage: The 32bit Registers, The high speed shared memory, The read-only constant cache and the read-only texture high speed cache[4].



Thus, in the CUDA applications, the GPU can run as the co-processor to the CPU, it can processing general purpose task, especially the high parallel and large scale scientific calculation.

# 3. CUDA model

CUDA is a GPGPU parallel programming model which first released by Nvidia Corporation in 2007, it contains the CUDA instruction set (ISA) and the GPU-based compelling engine of CUDA program. It utilizing and adjust C programing language as the programing language to support large scale high performance developing ability for its lower level and easy to understand feature. CUDA enable the developer an easier and more effective way to build highly productive program basing the very strong GPU computing ability. But, The CUDA developing and Application can only run with Nvidia GPU.

Therefore, the apple corporation propose the OpenCL model to offer a general platform and open-source model for GPGPU application developing. Nvidia also release the supporting drive for OpenCL-based drive program in June, 2009. The advantage of OpenCL is allow developer using the same code build their program in any OpenCL-supported platform. It reduce loads of complexities when transfer code from one platform to another, but OpenCL is superior in optimizing resource implementation than CUDA.

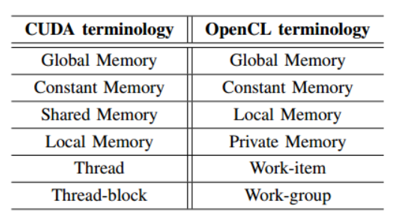


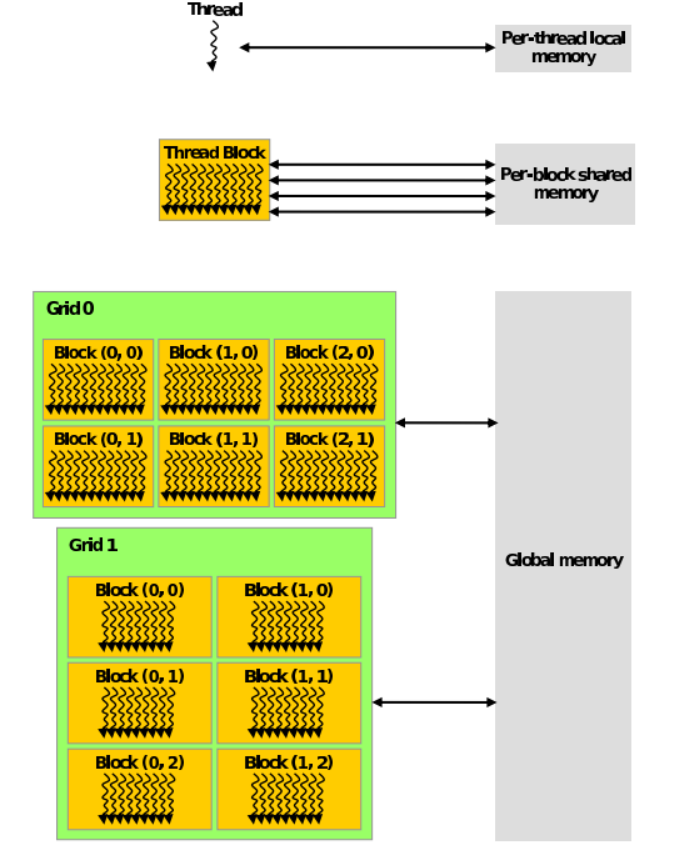
Table3-1.The comparison between CUDA and OpenCL [6]

## 3.1 CUDA feature.

Inside a typical CUDA application, the basic unit of processing is the thread, CUDA support three layered threads: The Grid, The block and the Individual Thread. The Grid can consist of either one, two or even three dimensional Blocks, The Grid can be processing in several Multiprocessors. However, the Block can only run in one single Multiprocessor, and only contains one or two dimensional Individual Thread. All of them construct the high speed and large scale parallel computing abilities of CUDA.

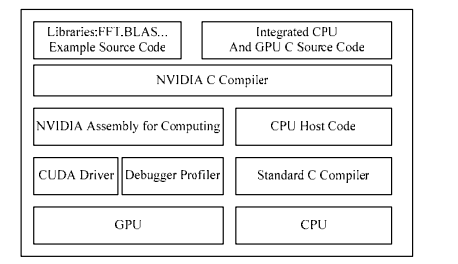
Before computing, the data are often required to transfer from memory to graphic memory through CUDA API, and then start kernel. The data transfer that through CUDA API memory management function does not need participation of SPA.

The data that managed by memory API and copied from memory to graphic memory exists in two different forms. One of the two form is Linear Memory. Linear Memory can be visited as Global memory in the CUDA program mode, and also can be visited in read-only way through binding to texture. The other form is CUDA Array, which optimize texture visit, and can only be visited in read-only way. In addition, data in constant memory also exists in graphic memory, which can be visited by accelerating constant cache.



Graph 3-1.One many-threaded Implementation of Differential Evolution for the CUDA Platform [5]

## 3.2 CUDA compile system

Nvcc is a complier drive, which can launch corresponding tool in different stages finishing the compiling work. CUDA using it to hidden the complicated detail while compiling. Furthermore is that Nvcc intimate some general compiler drive such as gcc instead of only suiting in the specific CUDA complier drive. For example, Nvcc giving the interface to compiling control, library path setting and macro self-definition. Nvcc also adjust its compiling path to user configuration with different CUDA mode especially in Virtual Machine System (VM System). Nvcc encapsulate several complier tool: ptxas，fatbin and cudafe in the System path C:\CUDA\open64\bin File package.

Grapg 3-2 The inner compile system of CUDA

When Nvcc begin to compiling the CUDA project, it will process a pre-operation to the .cu file, which including the unfold the macro and related invoke file, expanding the macro from pre-compiling C related program and merge the branch compiling result into target file. After pre-handling of the .cu file, the result first come to the front end of CUDA called CUDAfe. CUDAfe then separate the result file into self-adjustable complier switchers, which invoke specific inner compiler tool to compiling. CUDAfe would been invoked two times in the CUDA compiling procedure, the first one is separate the host code and device code into different .gpu file, the second one is doing “dead code analysis”, which is a compiler optimization to remove code which does not affect the program results.[http://en.wikipedia.org/wiki/Dead\_code\_elimination] and putting the result into Nvopencc. Nvopencc generate ptx (parallel thread execution) file and transfer to ptxas and finally into fatbin program.

As the same time, the non-C-based code in the CUDA project would translated into regular ANSI C source file, so that they can transfer to the normal C-based complier to achieve higher effective compiling. The host part of code would directly export with .c file. On the other hand, device code would linked to the host code, including its Cubin class as the global initial data array. It’s worth our attention that the kernel execution configuration also transform to the CUDA execution code, which in order to setup and start the complied kernel code. However, when utilizing CUDA driver API, it can only execute the ptx code or cubin object, while ignore the host code by nvcc compiling. On the other hand, each stages in nvcc compiling can set personal configuration with the filename, which means you can only compile object, library or resource file by just send them to the linker.

## 3.3 Nvidia support

CUDA drive API program designed has a little similarity with normal hard-core program. First CPU malloc the memory (most are DDR3), then transferring data into GPU high speed cache (most are GDDR5), so that GPU can process the data. However, driver API is the lower-level code library, which means it’s of great effectiveness but also programming with loads of complication.

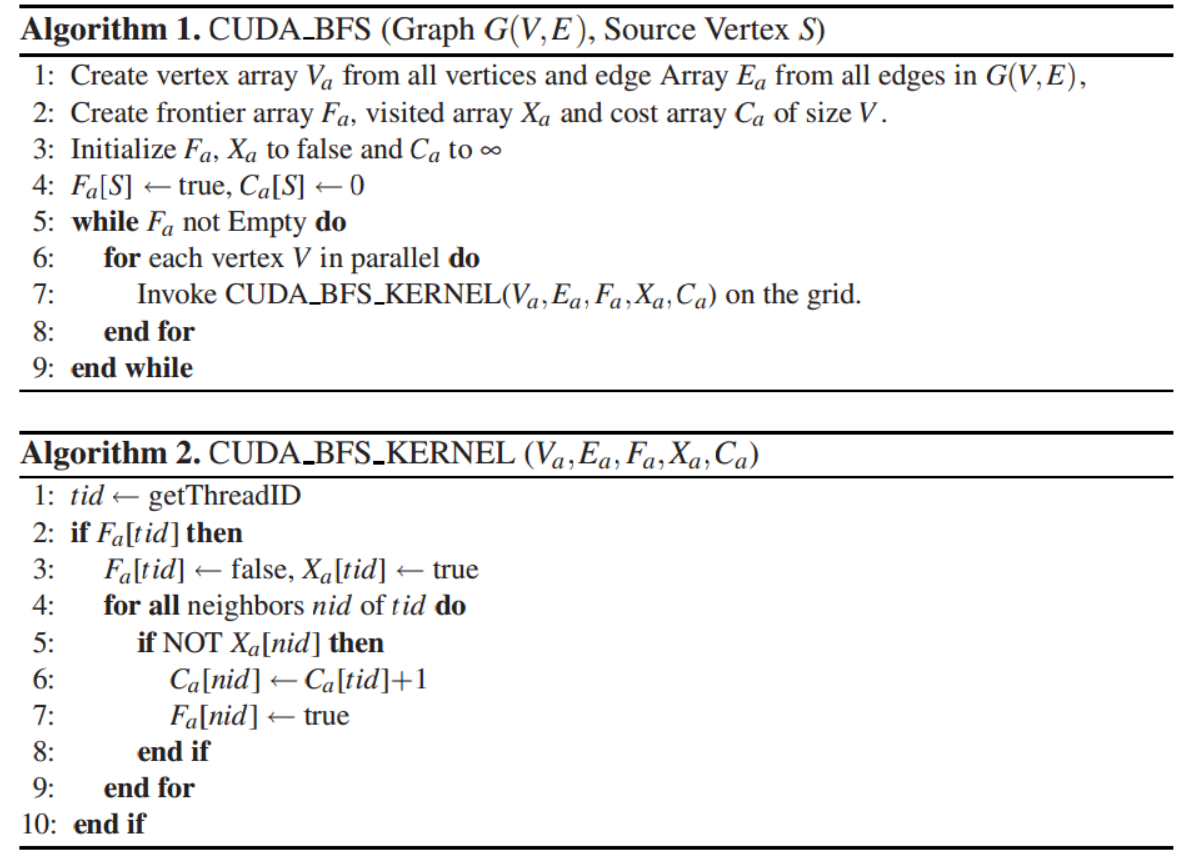
## 3.4 CUDA algorithm

1. CUDA implementation of BFS

Breadth-First-Search (BFS), which is the most known algorithm in graphic traversing, searching graphics from root node then to its neighbor with growing deep level. BFS is widely using in finding connected component from certain graphic, test the bipartite graph and window or grid clipping in the computer graphics display.

CUDA implement BFS with level synchronization. BFS search the graph levels by level, it won’t visit the next level until the current level is cleared, on the other side, BFS only visit frontier vertex, which means the node hasn’t been visited before. Generally BFS would maintain two array named “open” and “close”, one store the visited vertexes and one for the frontier vertexes. All the nodes in the “open” array and connected to the operated vertex would being processed at the current level. However in the CUDA algorithm, it do not maintain a queue for each vertex during our BFS processing since it will make repetition in multiple array indices and changing the grid configuration at every level of kernel execution in case of slows down the speed of processing.

When implement BFS with CUDA, we give one thread for each vertex in the graph. Create two Booleans arrays: frontier and visited with size |V|, we also keep another integer array to store the minimum route number to reach original source vertex S. In the beginning of every loop, each vertex would try to find its self in the frontier array or not. If true, it first update the cost of itself form Ca, then update all the neighbor vertices connected to it, the last is removing itself from the frontier array.



Graph 3-3 BFS realization with CUDA parallel model

RSA born in 1977, published by Ron Rivest, Adi Shamir and Leonard Adlemant. The user would first create the pair of public and private key, then he or she can transfer encrypted data and public key together to the receiver, if the receiver have the private key, so he or she can decode the data and finding out the important informations. RSA is one of the first practicable public-key cryptosystems and is widely used for secure data transmission.[http://en.wikipedia.org/wiki/RSA\_(cryptosystem)]. These cryptosystem using two keys: one is public visible for encryption, the other is private visible for decryption. These definitions was enlightened by the extremely difficulty of factoring the product of two large integers. The key parameter for RSA implement is the N, which is the product of two large prime number.

When RSA generate its public key, the focusing is the large integer modular exponentiation

\left(m^e\right)^d \equiv m \pmod{pq}



calculation. Given the inexistence of data relative in modular product and modular square， we can using CUDA model to design the GPGPU-based program to speed up the algorithm in generate public key. The key to speed up the algorithm is speed up the modular square processing. The modular calculation contains too much divide operation, which contains a series of addition and multiple operation.

We using Montgomery algorithm to reduce the complication of the divide operation. The advantage of Montgomery algorithm is the reduction in time of achieve module and simplify the complication of divide operation. Here is the pseudo code of Montgomery algorithm. Given the equation is:

So that we can use dichotomy to separate the series multiple into parallel calculation, which means we can first divide M by 2, if rest is 0, then set M=M/2, if not, then subtract M by 1, find the rest of the aforementioned equation.

To fit in the CUDA computing, we choose 32 as the k value, so that all operations could transform to 32bit multiple and addition operations. To minimize the visiting to global memory and synchronization latency, we store the most frequent accessed parameter:  result D, dividend N and all the middle variable into shared memory. Define as:

\_\_shared\_\_ int C[l + 1][ThreadNumInblock.x]；

\_\_shared\_\_ int D[l + 1][ThreadNumInblock.x]；

\_\_shared\_\_ int N[blockDim.x]；

for（int i = 0；i≤l；i++）{

ss[i][ThreadIndex.x] = 0；

ys[i][ThreadIndex.x] = y[i][column]；

cs[ThreadIndex.x] = 0；

}

\_\_syncthreads（）；

Which ThreadNumInblock is the Thread number in every thread blocks Thread Index is the index of current thread blockid is the index of current thread block. The column is the global index of current thread, equal to blockid.x\*ThreadNumInblock.x + ThreadIndex.x. So that the program can set up the synchronization point when the program is running.

# 4. Conclusion

CUDA is currently highly developed and popular GPGPU program architecture in whole world wide. In the world market, people can hardly find a video software not support CUDA acceleration. Furthermore, Including Elemental Technologies, MotionDSP and LoiLo, lots of Technology Company using CUDA model to build their product. CUDA model also sought after by finance company, which believe CUDA model can substantially speed up their real-time modeling and calculation. Most of super computer are based on the Nvidia GPU and running program upon CUDA model. This paper introduce the CUDA model with its feature and application. Explaining the inner structure and system to find out why CUDA is fast and easy to use. Besides, this paper offers real example in how to realize common and advanced general purpose algorithm in the CUDA parallel model.

# Reference

[1] Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2008). Scalable parallel programming with CUDA. *Queue*, *6*(2), 40-53.

[2] Sanders, J., & Kandrot, E. (2010). *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional.

[3] Nvidia.Nvidia CUDA compute unified device architecture programming guide version 2.0[EB/OL].[2008-06-07].http：//www.nvidia.com/object/cuda\_home.html.

[4] SUN Yinghong，TONG Yuanman，WANG Zhiying.CUDA based high performance implementation of RSA algorithm. Computer Engineering and Applications，2011，47（2）：84-87.

[5] Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2008). Scalable parallel programming with CUDA. *Queue*, *6*(2), 40-53.

[6] Fang, J., Varbanescu, A. L., & Sips, H. (2011, September). A comprehensive performance comparison of CUDA and OpenCL. In *Parallel Processing (ICPP), 2011 International Conference on* (pp. 216-225). IEEE.

[7] Harish, P., and Narayanan, P. J. “Accelerating large graph algorithms on the GPU using CUDA,” in: Proceedings of the 14th International Conference on High Performance Computing, Springer-Verlag, Goa, India, 2007, pp. 197-208.

[8] Kirk, D. (2007, October). NVIDIA CUDA software and GPU parallel computing architecture. In *ISMM* (Vol. 7, pp. 103-104).