***Introduction to Parallel Computing coursework:***

***a brief description of GPU technology and***

***GPU-based parallel programming***

Name: Kai Wang

Instructor: Dr. Lixin Tao

Seidenberg School, Pace University

Date: 2022/04/29

**Abstract**

Machine learning, big data analytics, artificial intelligence, and the Internet of Things are all emerging technologies that are benefiting human society at a rapid pace, and there is no doubt that their implementation relies heavily on the rapid processing of large amounts of data. Today, humankind is pushing the limits of Moore's Law to develop the manufacturing process of computer processors at high speed to meet the needs of processing vast amounts of data. However, to some extent, although the single-core performance improvement in the traditional Central Processing Unit (CPU) is significant, the benefit of this improvement is not obvious in massive data processing scenarios. Therefore, more effort is being put into the designs of processor architectures, with the expectation that different architectures will have the more powerful processing power for massive data.

With the development of the computer industry, a single-function processor has been playing an increasingly important role and is now one of the most critical processors for personal computers and large servers. This paper will briefly introduce the history, status, programming environment, applications, and challenges of the Graphics Processing Unit (GPU) and discuss how the theory study can help in efficient GPU-based parallel programming to support high-speed and massive cloud computing and machine learning.

**Contents**

[**Abstract** 1](#_Toc102152660)

[**Contents** 2](#_Toc102152661)

[**Background & Current Status** 3](#_Toc102152662)

[**Programming Environment** 7](#_Toc102152663)

[**Applications** 10](#_Toc102152664)

[**Challenges** 14](#_Toc102152665)

[**Discussion** 16](#_Toc102152666)

[**Conclusion** 18](#_Toc102152667)

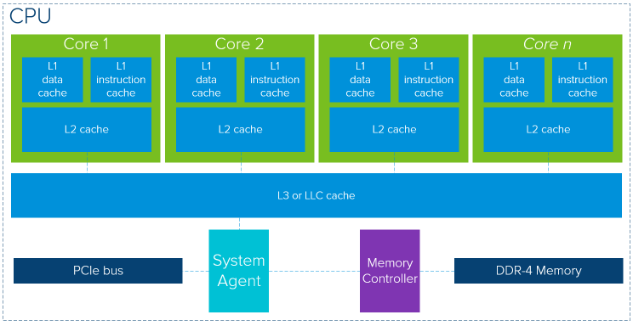
[**Acknowledgment** 19](#_Toc102152668)

[**References** 20](#_Toc102152669)

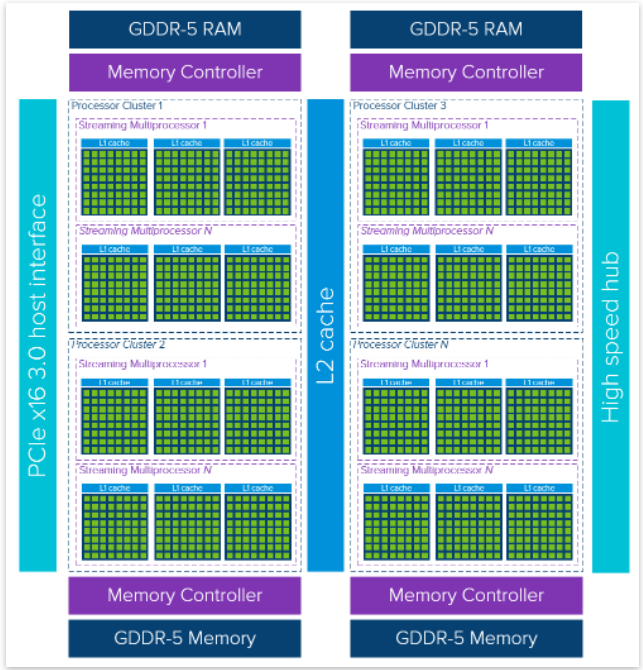
**Background & Current Status**

GPUs were originally designed for image processing, as the "G" in their name stands for "graphics." Furthermore, they were initially designed to perform a large number of floating-point operations to help computers render 3D images. It was used in a homogeneous scenario during the early time, only for processing video and video game graphics quickly. However, the modern GPUs are massively parallel and are fully programmable.

Compared to CPUs, GPUs are better structured to handle massive data. Going back to the original object of GPU processing, each frame contains a large number of pixel blocks for video and video games. To ensure smoothness and interaction with the player, each second of video usually includes more than 16 frames, and for today's games, players are demanding 30 to 144 frames (or more) per second of gameplay. Resolution aside, the large number of pixels contained in each frame needs to be calculated correctly, even if these calculations are usually elementary. For such a simple but massive amount of computations, the best way is usually parallel computing. In other words, the more processors that can be used for the computation, the faster the data tends to be processed. Nowadays, one of the most powerful CPUs, AMD's Threadripper Pro 3995WX contains 64 cores and has 128 thread processors available (Alcorn, 2022). However, Nvidia's GeForce GTX TITAN Z provides 5760 CUDA cores. Although the difference in single-core performance between these two processors is significant, single-core performance is not as important for scenarios such as many floating-point operations. Imagine a math PhD. and fifty middle school students competing to solve 10,000 elementary school level problems. The middle school students, who have the advantage in numbers, will clearly win. VMWARE, a could platform tech company provides two images that visually explain why the GPUs can embed such a large number of processors (Hagoort, 2022).



(Figure 1: a sample CPU architecture)



(Figure 1: a sample CPU architecture)

By comparing the above images, it is easy to see that for CPUs, each processor core is equipped with at least one cache, and beyond that, there are other layers of cache whose purpose is to ensure data throughput. Powerful single-core computing performance becomes redundant if limited by output throughput latency, so even though cache is often expensive and takes up processor space, cache often takes up a large portion of the space and cost in CPU architecture. For GPU architectures, however, there are often many processor cores sharing a single Cache, and the requirements for Cache size and quality are not as high as for CPUs. Even though the computation of large amounts of data implies huge data throughput, the busy and continuous computation allows a certain level of data access latency to be tolerated. In addition, the relatively simple computations handled by the GPUs do not require overly powerful single-core performance. Moreover, one thing should also be mentioned: GPU designs are much more diverse - varying widely from company to company and even from product line to product line. The developers involved are trying to find the most appropriate processor architecture for specific usage scenarios. But in general, GPUs have always used smaller transistor sizes to dramatically increase the number of processors, aiming to achieve increasingly larger data throughputs. In contrast, CPUs focus on instruction-level parallelism and reduced latency, as (Palacios & Triska, 2011) summarized.

Although GPUs have shown outstanding performance in many areas, this does not mean that CPUs can be replaced. CPUs still play an irreplaceable role in the execution of instruction-level operations and in some small but complex computing scenarios. Today, developers are investing more efforts to help CPUs and GPUs work better together to improve the overall performance of devices. Since the discussion of CPU and GPU heterogeneous technologies is beyond the main scope of this paper, related studies can be specifically referred to (Mittal & Vetter, 2015) 's study or research from other scholars.

**Programming Environment**

At the turn of the century, developers first tried to run the first non-graphical computation on a GPU, which was a matrix-matrix multiply (Du et al., 2012). Today, the vast majority of graphics cards on the market are programmable, and programming has become easier with the iteration of different programming languages. The main popular GPU programming languages on the market today include OpenCL, OpenACC, CUDA (Nvidia), ROCm (AMD), Xcelerit SDK, OpenMP 4, and OpenVIDIA (UToronto & Nvidia). In the following section, three of the most commonly used languages will be briefly introduced which are CUDA, OpenCL, and OpenACC (Levinas, 2021):

1. CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) that allows computing kernels to have direct access to the GPU's virtual instruction set. It allows the developer to program in C/C++ and Fortran or in Java, Python, and R language with third-party wrappers for general purpose processing. However, it can only be used for those CUDA-enabled GPUs.

1. OpenCL

OpenCL is one of the most general GPU programming tools. It is an open standard for parallel programming across heterogeneous platforms created by the Khronos Group. It works with CPU, GPU, digital signal processors, FPGA (field-programmable gate arrays), and other processors or hardware accelerators. Moreover, it is widely used by technology giants including Nvidia, AMD, Apple, IBM, Intel and others. It is based on C/C++, with third-party wrappers. It can be used in Python, Java, R language, GO, and JavaScript as well.

1. OpenACC

A user-driven directive-based parallel programming standard is mainly used in scenarios where it is desired to port code to various heterogeneous high-performance computing processors and is mainly used by scientists and researchers. It supports C/C++ and Fortran.

Single Instruction Multiple Thread (SIMT) is a low-level programming abstraction provided by the CUDA and the OpenCL language that allows fine-grained control over GPU systems. And both languages also allow the use of low-level features such as staging memory, warp operations, and block-level synchronization. As the GPU programming language is continuously updated, more and more functions and instructions become available. Experimental comparisons have been made regarding the efficiency of codes programmed in different languages to run on real devices. By and large, CUDA's code demonstrates better performance in most cases and shows more flexibility in some optimizations that OpenACC and OpenMP 4 are not yet able to implement. Due to block-level synchronization, CUDA can handle the race condition better and has higher data reusability. For OpenMP 4, its GPU support is not as mature as OpenACC. In addition, various combinations of experiments have shown that all compilers have slightly better support for C than for Fortran (Balogh et al., 2018).

Take Nvidia's CUDA as an example. Its three key abstractions (a hierarchy of thread groups, shared memories, and barrier synchronization) provide fine-grained data parallelism and thread parallelism, nested within coarse-grained data parallelism and task parallelism. The programmer's job then is to divide a total problem into independent subproblems that can be solved in parallel by a block of threads, and to divide each subproblem into more sophisticated parts that can be handled collaboratively by threads within the block, this way of implementation is automatically scalable. Since each task for the thread blocks can be scheduled in any order in any available multiprocessor unit group within the GPU, a compiled CUDA program can be executed on any number of multiprocessors (Nvidia Corporation, 2022).

GPU manufacturers and programming tools provide very convenient toolkits and learning materials for beginners and lightweight users. In many scenarios, functions provided by GPU manufacturers are available to help users quickly implement inter-process division of tasks and cooperation.

In summary, the programmable performance of GPUs has made good progress today. Each language also has its own relatively suitable scenario. While CUDA is often better and more popular in academia on CUDA-enabled GPUs, OpenCL is also popular for its broader applicability.

**Applications**

According to a document named *GPU-Accelerated Applications* that Nvidia released in 2020, accelerated computing has had a broad and beneficial impact on many industries, and there were over six hundred applications optimized for GPUs by that time. The GPU-based parallel computing technology are well established for applications including but not limited to: Computational Finance, Climate, Weather and Ocean Modeling, Data Science and Analytics, Artificial Intelligence (Deep Learning & Machine Learning), Public Sector, Designing, Life Sciences, Business Process Optimization (Nvidia Corporation, 2020). Again, the applications mentioned above in various fields involve massive data processing. Thanks to the GPU with its powerful parallel data processing capabilities, it has brought huge efficiency gains to many fields that rely on data processing.

When it comes to one of the hottest applications in the private sector regarding GPUs, digital cryptocurrency (e.g., Bitcoin) mining is definitely an excellent place to start. Since cryptocurrency mining is the process of solving simple mathematical problems (without involving overly complex algorithms) and is very computationally intensive, this section will use the example as a typical application of GPU in the massive data processing. Furthermore, the essence of cryptocurrency mining is about cryptography, where the application of blockchain technology is also being used in different fields, such as the Internet of Things.

Under the influence of multiple factors such as the Covid-19 epidemic and cryptocurrency market fluctuations, GPU prices are also experiencing huge fluctuations. According to data from the last few months, GPU prices have finally come down by 5~10% after some sharp rises, but most GPU devices are still priced 50~100% higher than the retail price suggested by the producers (Morales, 2022). The huge impact of the cryptocurrency industry on GPU prices reflects the demand for GPUs in the conversion industry. According to a study conducted by Ghimire and Selvaraj in 2018, the mining efficiency of GPUs is 5 to 200 times higher than that of CPUs while consuming only two-thirds of the power of CPUs (Ghimire & Selvaraj, 2018). Please see the form 1 below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Type** | **Hash Rate**  **R (Mhash/s)** | **Power Use**  **P (W)** | **Energy Efficiency**  **ε (Mhash/J)** | **Cost**  **($)** |
| Core i7 950 | CPU | 18.9 | 150 | 0.126 | 350 |
| Atom N450 | CPU | 1.6 | 6.5 | 0.31 | 169 |
| ATI 4850 | GPU | 101.0 | 110 | 0.918 | 45 |
| ATI 5770 | GPU | 214.5 | 108 | 1.95 | 80 |

(Form 1)

Alkaeed et al. (2020) 's research suggests the same thing, with their study showing that mining with CPUs is not always profitable from a revenue perspective, while GPUs are always more profitable. GPUs can also be used for machine learning thanks to their ability to multi-thread complex mathematical calculations. It is clearly the correct conclusion. In the field of machine learning and artificial intelligence, the highly concurrent computing power that GPUs can provide is also in high demand.

For machine learning, it simply means that by feeding the program a large amount of data for learning, the program eventually finds the correlation function between the data and the conclusion on its own. Without getting into specific learning techniques and machine learning models, try to imagine the program as a black box: for example, the researcher first puts together a large number of images of fish and a large number of images of dogs and tells the program which ones are fish and which ones are dogs at the time of input. The program then learns to compute based on the images and the results (the labels of the fish or dog) and eventually gains the ability to distinguish between fish and dog. At this point, if the researcher provides a new image, the program is expected to be able to identify whether the animal on the target image is a fish or a dog. It is a straightforward example. However, suppose the researcher then provides the program with a picture of a cat, and the program may conclude that "this is a dog." Because the above learning is not enough, in the previous learning, the program may have finished recognizing the dog by the fact that the dog has four limbs and the fish does not, but such a functional relationship is not enough for the program to recognize the cat which also has four limbs. So, the researcher needs to continue to provide pictures of cats for the program to recognize. If the program is required to obtain the ability to distinguish all animals by such learning, then a vast number of iterations are required to achieve the desired tolerance. It is easy to see that the process of machine learning requires a large amount of data for the program to learn. The massive amount of input data and even larger computations require powerful computing power in this process. Otherwise, this learning process will take an unacceptably long time. As with cryptocurrency mining, the parallel computing power of GPUs brings a huge efficiency boost to machine learning. With the GPU accelerated toolkit, many programs can improve their machine learning models to achieve higher efficiency (Jiang & Canny, 2017). For example, GPU and DNN-based (Deep neural networks) machine learning is advancing the pharmaceutical industry for analyzing high-content imaging systems, particularly phenotypic cell responses to chemical perturbations (Gawehn et al., 2018). Applications like this for GPU parallel computing power give researchers in various fields more powerful tools to deal with the various situations they face.

Another application that needs to be mentioned and is highly relevant to parallel computing technology is cloud computing. However, the application scenario of cloud computing is more diversified, and an entire GPU-Based architecture does not meet its requirements. For cloud computing-related scenarios, the cooperation between GPU and CPU becomes more critical. Of course, this still means that the accelerated computing power of the GPU is essential in this application. The issue of GPU collaboration with other devices will be mentioned in the next section.

**Challenges**

One of Neil Postman's well-known comments on technological change is, "All technological change is a trade-off. For every advantage a new technology offers, there is always a corresponding disadvantage ((Postman, 1998). Since power dominates the life-cycle system costs of supercomputers and servers, the first problem any GPU-based parallel computing system needs to face is energy consumption, although developers have long made many improvements to the architecture and circuitry. Another challenge associated with this is reflected in the instruction overhead. For most modern processors, much of the energy is consumed in the overhead of data supply, instruction supply, and control (Keckler et al., 2011).

In addition, as mentioned before in the introduction of GPU architecture, cache is much less important for GPUs than CPUs. However, this does not mean that GPUs do not need the cache. GPUs support high concurrency and are often characterized by high register read/write latency (Wong, 2008). As more and more domains add GPU-based parallel computing to their service systems, the collaboration between GPUs and other processors or devices becomes important. A study published by Freniere et al. (2016) related to an Amazon's cloud computing service platform mentions that communication latency between GPUs and CPUs is one of the challenges faced in improving the process of heterogeneous cloud servers, so much so that the criticality of GPUs in some scenarios becomes debatable.

On top of that, GPUs are not without rivals in high productivity computing, and field-programmable gate array (FPGA) is one of the challengers. Because FPGAs have a programmable hardware architecture of circuits, such devices can be reprogrammed as needed. Their computational speed, programmability, and flexibility have impacted many GPU-based parallel computing systems. In, for example, deep learning scenarios, the features of FPGAs provide researchers with low cost and high benefit. Its flexibility makes it well suited for applications in artificial intelligence (Intel Technologies, n.d.). Both studies by Kestur et al. (2010) and Jones et al. (2010) show that FPGAs are no less powerful than GPUs in many scenarios.

**Discussion**

The course Introduction to Parallel Computing is taught by ACM/IEEE Senior Member Dr. Lixin Tao, using Lester's (2013) textbook The Art of Parallel Programming. Much of the preceding section of this course avoids a standardized and portable message-passing standard designed to function on parallel computing architectures, the Message Passing Interface (MPI). The course begins with an extensive introduction to the concepts of Date Parallelism and Date Sharing, introducing students to the idea of inter-process communication and Synchronous Parallelism. Then it focuses on how to implement data partitioning and communication in a Distributed-Memory scenario with multiple computers. After introducing the various topologies of the current mainstream processing units, the course finally briefly introduced the basic functions and applications of MPI.

This paper (coursework) is the final project of the course. The instructor aims to guide students to focus on GPU hardware and GPU-based parallel computing system, thus helping them to understand the difference between parallel programming and non-parallel programs. Many of the problems presented in the course are still problems that every programmer needs to face and think about during parallel computing programming. For example, the exercises and tests related to the Jacobi relaxation procedure require students to coordinate the iteration levels of many processes and the need to perform aggregation and convergence tests on the results of parallel computations. Although the procedure may seem to differ from the concept of machine learning, it involves parallel processing of input data and testing for tolerance of computational results. In more detail, in machine learning, where massive amounts of data need to be processed and analyzed more quickly with the help of parallel computing, the program needs first to test the resulting mapping function to determine whether the iteration can be ended before a human evaluate the learning results.

Another example shows that the theoretical learning involved in the course is useful for understanding parallel computing techniques. Those test programs that simulate running on multiple computers demonstrate the communication overhead, which is something that can often be improved with cloud computing technologies and clustered servers. As mentioned in the previous section, many of today's servers and computing centers contain many heterogeneous devices, and the cost or delays of communication between devices are an issue that cannot be ignored. Even though a program's high parallelism can lead to efficiency improvement, if the programmers ignore the mapping between abstract processors and physical devices, it may not run as well as it should. When proposing a parallel computing solution for a specific problem or device, it is always necessary to adapt the process group structure to the actual situation. In addition to the communication overhead, the cost of process initialization reminds the programmer how to design the size and complexity of the subproblem size.

In summary, the course content and the research in this paper stop at the basic discussion, but many of the concepts and applications covered can be extended to specific challenges and practical solutions.

**Conclusion**

This paper briefly discusses the topics often involved in GPU devices and GPU-based parallel computing technologies, including programming environments, specific applications, and challenges, starting with the original design of GPUs and example architectures. Today, the programmability of GPUs and the available programming tools greatly facilitate industries to improve their data processing capabilities using parallel computing technologies. It is no exaggeration to say that the powerful thread concurrency of GPUs has disrupted many areas involving massive data processing. As a result, GPU-based parallel computing solutions are widely used in artificial intelligence (machine learning/deep learning), cloud computing, data processing, and other areas across many industries. Even though the associated technology has its challenges and even bottlenecks, the potential of GPUs is huge within the foreseeable future.

It is important to note that many of the studies mentioned in this paper are conclusions drawn by researchers under certain conditions, and some of the conclusions may no longer be applicable to the current environment as technology evolves rapidly. In addition, some generalized conclusions are not accurate due to the diversity of GPU architectures. Finally, it should be added that many specific application scenarios are often customized with GPUs or even multiple processors on the same chip. Heterogeneity of GPUs and CPUs (and more devices) is the trend in parallel computing technology (Shulga et al., 2016).

**Acknowledgment**

Thanks to [Dr. Lixin Tao](http://csis.pace.edu/~lixin/) for his guidance and Dr. Bruce P. Lester for freely providing the course materials and C\* simulator.

**References**

Alcorn, P. (2022, January 1). Best Workstation CPUs for 2022. Tom's Hardware. https://www.tomshardware.com/reviews/best-performance-cpus,5683.html

Alkaeed, M. Kh., Alamro, Z., Al-Ali, M. S., Al-Mohammed, H. A., & Khan, K. M. (2020). Highlight on Cryptocurrencies Mining with CPUs and GPUs and their Benefits Based on their Characteristics. 2020 IEEE 10th International Conference on System Engineering and Technology (ICSET), 67–72. https://doi.org/10.1109/ICSET51301.2020.9265386

Balogh, G. D., Reguly, I. Z., & Mudalige, G. R. (2018). Comparison of Parallelisation Approaches, Languages, and Compilers for Unstructured Mesh Algorithms on GPUs. In S. Jarvis, S. Wright, & S. Hammond (Eds.), High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation (pp. 22–43). Springer International Publishing. https://doi.org/10.1007/978-3-319-72971-8\_2

Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., & Dongarra, J. (2012). From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. Parallel Computing, 38(8), 391–407. https://doi.org/10.1016/j.parco.2011.10.002

Freniere, C., Pathak, A., Raessi, M., & Khanna, G. (2016). The Feasibility of Amazon's Cloud Computing Platform for Parallel, GPU-Accelerated, Multiphase-Flow Simulations. Computing in Science Engineering, 18(5), 68–77. https://doi.org/10.1109/MCSE.2016.94

Gawehn, E., Hiss, J. A., Brown, J. B., & Schneider, G. (2018). Advancing drug discovery via GPU-based deep learning. Expert Opinion on Drug Discovery, 13(7), 579–582. https://doi.org/10.1080/17460441.2018.1465407

Ghimire, S., & Selvaraj, H. (2018). A Survey on Bitcoin Cryptocurrency and its Mining. 2018 26th International Conference on Systems Engineering (ICSEng), 1–6. https://doi.org/10.1109/ICSENG.2018.8638208

Hagoort, N. (2022). Exploring the GPU Architecture | VMware. The Cloud Platform Tech Zone. https://core.vmware.com/resource/exploring-gpu-architecture

Intel Technologies. (n.d.). FPGA vs. GPU for Deep Learning Applications. Intel. Retrieved April 29, 2022, from https://www.intel.com/content/www/us/en/artificial-intelligence/programmable/fpga-gpu.html

Jiang, B., & Canny, J. (2017). Interactive Machine Learning via a GPU-accelerated Toolkit. Proceedings of the 22nd International Conference on Intelligent User Interfaces, 535–546. https://doi.org/10.1145/3025171.3025172

Jones, D. H., Powell, A., Bouganis, C.-S., & Cheung, P. Y. K. (2010). GPU Versus FPGA for High Productivity Computing. 2010 International Conference on Field Programmable Logic and Applications, 119–124. https://doi.org/10.1109/FPL.2010.32

Keckler, S. W., Dally, W. J., Khailany, B., Garland, M., & Glasco, D. (2011). GPUs and the Future of Parallel Computing. IEEE Micro, 31(5), 7–17. https://doi.org/10.1109/MM.2011.89

Kestur, S., Davis, J. D., & Williams, O. (2010). BLAS Comparison on FPGA, CPU and GPU. 2010 IEEE Computer Society Annual Symposium on VLSI, 288–293. https://doi.org/10.1109/ISVLSI.2010.84

Lester, B. P. (2013). The Art of Parallel Programming (Second Edition). Pearson Education, Inc. https://artofparallelprogramming.com/

Levinas, M. (2021, September 30). A Complete Introduction to GPU Programming With Practical Examples in CUDA and Python. Cherry Servers. https://www.cherryservers.com/blog/introduction-to-gpu-programming-with-cuda-and-python

Mittal, S., & Vetter, J. S. (2015). A Survey of CPU-GPU Heterogeneous Computing Techniques. ACM Computing Surveys, 47(4), 69:1-69:35. https://doi.org/10.1145/2788396

Morales, J. (2022, January 28). Will the Crypto Crash Affect GPU Prices and Availability? MUO. https://www.makeuseof.com/crypto-crash-gpu-prices-availability/

Nvidia Corporation. (2020). GPU-ACCELERATED APPLICATIONS. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/gpu-applications-catalog.pdf

Nvidia Corporation. (2022). CUDA C++ Programming Guide (Programming Guides). https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

Palacios, J., & Triska, J. (2011). A Comparison of Modern GPU and CPU Architectures: And the Common Convergence of Both. Undefined. https://www.semanticscholar.org/paper/A-Comparison-of-Modern-GPU-and-CPU-Architectures%3A-Palacios-Triska/dbe1750fdbe3908d0d99b9ea7c787a62b180740e

Postman, N. (1998). Five Things We Need to Know About Technological Change. 5.

Shulga, D. A., Kapustin, A. A., Kozlov, A. A., Kozyrev, A. A., & Rovnyagin, M. M. (2016). The scheduling based on machine learning for heterogeneous CPU/GPU systems. 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), 345–348. https://doi.org/10.1109/EIConRusNW.2016.7448189

Wong, H. T. (2008). Architectures and Limits of GPU-CPU Heterogeneous Systems.