International Journal of Information, Engineering & Technology

Volume 11, Issue 4, PP 19-25, ISSN: 2360-9194, January, 2021
Double Blind Peer Reviewed International Research Journal
garcjournalssubmit@gmail.com
Journal Series: Global Academic Research Consortium (garc)

Parallel Programming Models: An Overview of Hybrid and Heterogeneous Models

Ahmed Ali Umar[1]; Engr. Mallam Terab Ali[2]; Muhammad H. Abdulfattah[3]; Falmata Bukar Wakil[4]

[1]Department of Computer Engineering, University of Maiduguri, Borno State
[2 &4]Department of Computer Engineering Technology Ramat Polytechnic Maiduguri P.M.B. 1070 Borno State.
[3]Computer and ICT Centre, University of Maiduguri. mhabdulfattah@unimaid.edu.ng

Correspondence Author: Email emfance@gmail.com

Abstract: *The increasing relevance of parallelism in the computational field can be illustrated by considering its effect on the computing literature over the last decade. Since the switch of the microprocessor industry to the multi-core model, and the introduction of GPGPU, parallelism has become a key player in the software arena. Thus, the hybrid parallel programming approach is becoming increasingly popular. The platform model is viewed from a hierarchical and abstract perspective. Execution of an OpenCL program involves simultaneous execution of multiple instances of a kernel on one or more OpenCL devices. In addition, the easy availability of GPUs on multi-core systems is providing momentum to a new parallel programming model: heterogeneous programming. This supersedes pure GPU programming, allowing several multi-core CPUs and several GPUs to collaborate.*

Key words: *Charcoal Dust, Green moulding sand, metal casting.*

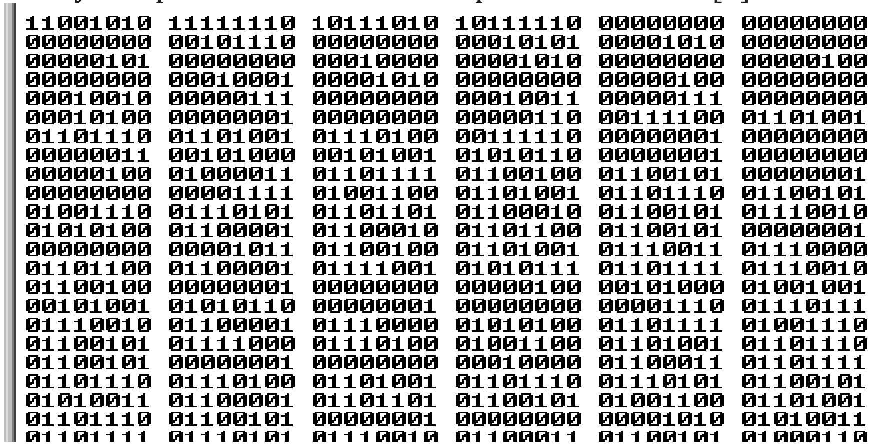
1.0 Introduction

The impressive rate of improvement and enhancement in technology has led to the dominance of microprocessor based computers in the computing arena. Microprocessors with single processing unit (CPU) led to a sustained performance increase and cost reduction in computer applications for more than two decades. However, this growth became stagnant around 2003 due to power constraints [1]. These shortfalls made it difficult to increase the clock frequencies and the number of tasks performed within each clock period. Additionally the number of data for processing continues to grow.

To remedy these limitations processor developers adopted a model where the processor has multiple processing units known as cores [3]. Today there are processors with dozens of cores available in the market and with the rapid growth of semiconductor technology it will be very common to have number of cores on processors going into hundreds and even thousands in the near future. This kind of processor is referred to as many core processors. However, to tap all the reward offered by these processors will not be an easy task, and one of the major challenges will be how to efficiently utilize the available computing power [4]. Obviously this gradual shift in computing will have a huge impact on software and its development. Almost all software applications are developed to execute in sequential order which is implemented on conventional single core processors. As such the era of sequential programming is gradually coming to an end [5], [6]. Hence, we can no longer rely on single core processors for improvement in performance but it can be achieved through multiple and many core processors by using parallel programming which has now taken a new star role on the stage [4].

Parallel computing can significantly increase the performance of applications by executing them on multiple processors. Unfortunately, the scaling of application performance is not equal to the scaling of peak speed, and the programming burden is still important. There is need for the applications to automatically scale to the number of processors. However, for this to come about the applications must be programmed to exploit parallelism. Thus, achieving scalable parallelism largely depends on the applications developer [7].

A parallel programming model is a construct that enables the expression of parallel programs which can be compiled and executed. The advantages of a programming model are usually judged on its generality: how well a range of different problems can be expressed and how well they execute on a range of different architectures. Therefore, the implementation of a programming model can take several forms such as libraries, language extensions, or invent new programming model. A good programming model can provide an essential bridge between hardware and software, which means that high-level languages can be efficiently compiled and executed on specific hardware [8].



```
11001010 11111110 10111010 10111110 00000000 00000000
00000000 00101110 00000000 00010101 00001010 00000000
00000101 00000000 00010000 00001010 00000000 00000100
00000000 00010001 00001010 00000000 00000100 00000000
00010010 00000111 00000000 00010011 00000111 00000000
00010100 00000001 00000000 00000110 00111100 01101001
01101110 01101001 01110100 00111110 00000001 00000000
00000011 00101000 00101001 01010110 00000001 00000000
00000100 01000011 01101111 01100100 01100101 00000001
00000000 00001111 01001100 01101001 01101110 01100101
01001110 01110101 01101101 01100010 01100101 01110010
01010100 01100001 01100010 01101100 01100101 00000001
00000000 00001011 01100100 01101001 01110011 01110000
01101100 01100001 01111001 01010111 01101111 01110010
01100100 00000001 00000000 00000100 00101000 01001001
00101001 01010110 00000001 00000000 00001110 01110111
01110010 01100001 01110000 01010100 01101111 01001110
01100101 01111000 01110100 01001100 01101001 01101110
01100101 00000001 00000000 00010000 01100011 01101111
01101110 01110100 01101001 01101110 01110101 01100101
01010011 01100001 01101101 01100101 01001100 01101001
01101110 01100101 00000001 00000000 00001010 01010011
01101111 01110101 01110010 01100011 01100101 01000110
```

Fig. 1 Shows Programming Codes

The availability of General Purpose computation on graphical processing units (GPGPUs) in actual multi-core systems has lead to the Heterogeneous Parallel Programming (HPP) model. HPP Seeks to harness the capabilities of multi-core CPUs and many-core GPUs. According to all these hybrid architectures, different parallel programming models can be mixed in what is called hybrid parallel programming. A well planned implementation of hybrid parallel programs can generate massive speedups in the otherwise pure homogeneous implementations [9]. The same can be applied to hybrid programming involving GPUs and distributed architectures [10], [11].

In this paper, parallel programming models are reviewed with special consideration to heterogeneous and hybrid programming models in relation to their suitability for High Performance Computing applications.

2.0 Heterogeneous Parallel Programming Models

In early 2001 NVIDIA released the first programmable GPU: GeForce3. In 2003 the Siggraph/ Eurographics Graphics Hardware workshop, held in San Diego, showed a change from graphics to non-graphics applications of the GPUs [12]. Hence, the concept of general purpose graphical processing unit was born. Nowadays, one or multiple host CPUs and one or more GPUs can be found in a single system. Thus, the name heterogeneous system and the emergence of a programming model oriented toward these systems.

Heterogeneous model is predicted to become a main approach due to the microprocessors industry interest in the development of Accelerated Processing Units (APUs). An APU combines the CPU (multi-core) and a GPU on the same die. In the first CPU+GPU systems, languages as Brook [13] or Cg [14] were used. However, NVIDIA has popularized (Compute Unified Device Architecture) CUDA [2] as the principal model and language to program their GPUs. More recently, the industry has worked together on the Open Computing Language (OpenCL) standard [17] as a common model for heterogeneous programming. In addition, different proprietary solutions, such as Microsoft's DirectCompute or Intel's Array Building Blocks (ArBB) [15], are available. Here these approaches are reviewed.

OpenCL is an open source program for general purpose parallel programming developed for CPUs, GPUs and other processors. It was first released in late 2008 by Khronos Group. Basically it separates between the devices and the host. The rationale behind OpenCL is to write functions that execute on OpenCL devices and APIs for creating and managing these kernels. The kernels are compiled for the targeted device during the execution of the application. This in turn enables the host application to take advantage of all the computing devices in the system. The OpenCL operation can be described in four interrelated models: the platform, execution, memory, and programming models. The platform model is viewed from a hierarchical and abstract perspective. Execution of an OpenCL program involves simultaneous execution of multiple instances of a kernel on one or more OpenCL devices. A kernel is the basic executable code unit. OpenCL defines a multilevel memory model similar to CUDA. OpenCL is designed to be used in GPUs as well as in other platforms like multi-core CPUs. Therefore, it can support both data parallel [16], and task parallel [17] programming patterns [11], which are well suited for GPUs and CPUs architectures, respectively.

CUDA is a parallel programming model developed by NVIDIA . The project started in 2006 and the first version was released in early 2007. The CUDA model is designed to scale applications with the increasing number of processor cores provided by the GPUs [2]. CUDA provides a development environment that allows developers to use C programming language. CUDA being a parallel system consists of a host and a computation device. The computation of tasks is done in the GPU by a set of threads running in parallel. The architecture of the GPU threads consists of a two-level hierarchy, i.e. the block and the grid. The block is a set of few tightly coupled threads, each thread is identified by a thread ID.

On the other hand, the grid is a set of loosely coupled blocks with similar size and dimension. There is no synchronization at all between the blocks, and an entire grid is handled by a single GPU. The GPU is organized as a collection of multiprocessors, with each multiprocessor responsible for handling one or more blocks in a grid. A block is never divided across multiple multiprocessors. Threads within a block can cooperate by sharing data through some shared memory, and by synchronizing their execution to coordinate memory accesses. The directory in which programming codes are stored is as shown in figure 3.

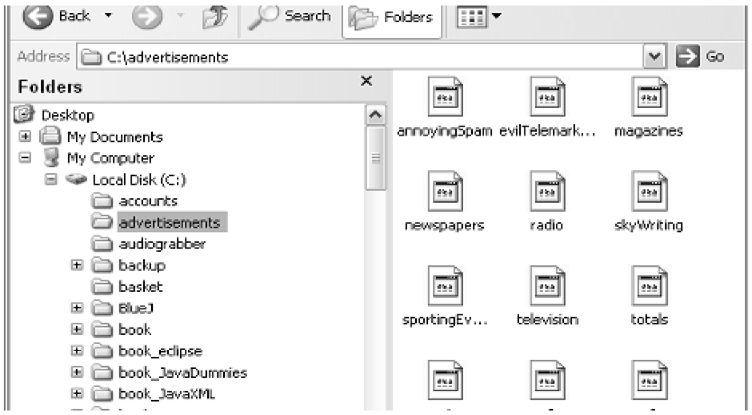


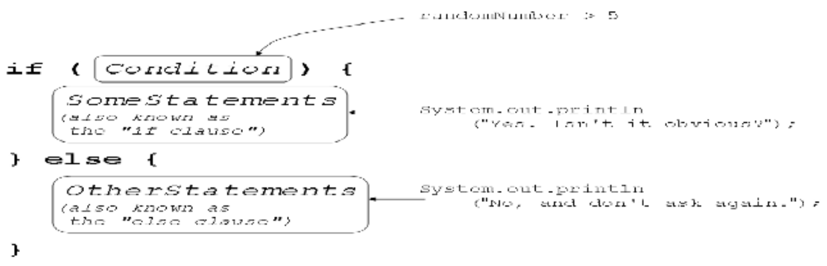
Fig. 3 Directory folders

DirectCompute is an approach adapted by Microsoft to GPU programming. DirectCompute is a part of the Microsoft DirectX APIs collection [18]. It is also known as DirectX11 Compute Shader. It was initially released with the DirectX 11 API, but runs on both DirectX 10 and DirectX 11 graphics processing units. It allows computation independently of the graphic pipeline, therefore suitable for GPGPUs. The main drawback of DirectCompute is that it only works on Windows platforms.

Array Building Blocks (ArBB) is also a heterogeneous programming model which provides a generalized vector-parallel-programming solution for data-intensive mathematical computation [19]. A user simply expresses computations as operations on arrays and vectors. ArBB comprises of a C++ standard library interface and a powerful runtime. A just-in-time (JIT) compiler supplied with the library translates the operations into target dependent code, where a target could be the host CPU or an attached GPU. At runtime, ArBB uses Intel's Threading Building Blocks [1], which contributes to abstract platform details and threading mechanisms for scalability and performance. Intel's ArBB can run data-parallel vector computations on a possibly heterogeneous system. By design, Intel ArBB prevents parallel programming bugs such as data races and deadlocks.

3.0 Hybrid Parallel Programming Models

Combining programming models is not a new idea; the main aim is to exploit the strengths of the models such as efficiency, memory savings, ease of programming, and scalability of the models involved. Rather than developing new languages, the already available programming models and tools can be mixed. This approach is known as hybrid (parallel) programming. This programming model is a modern software trend for the current hybrid hardware architectures. The basic idea is to use message passing across the distributed nodes and shared memory within a node. Hybrid programming can also involve the use of GPUs as source of computing power. GPU programming approaches, such as OpenCL, and CUDA could be used. However, since OpenCL directly supports multi-GPU and GPU+CPU programming its use is not specially extended in the hybrid programming field. The format for An If statement is shown in figure 2.



```
if ( Condition ) {
    SomeStatements
} else {
    OtherStatements
}
```

```
randomNumber > 5
```

```
System.out.println
    ("Yes. Isn't it obvious?");
```

```
System.out.println
    ("No, and don't ask again.");
```

(also known as the "if clause")
(also known as the "else clause")

Fig. 2 An If statement format

There are several models of hybrid parallel programming. In [18] Pthreads and MPI were combined to develop a parallel file compression program. In this program Pthreads was used to extend MPI, thereby improving the speed and efficiency of the program. Additionally, Pthreads is used to generate multiple tasks to be executed on a single processor simultaneously by using shared memory. MPI allows communication between the nodes. Another example were Pthreads and MPI are combined is found in [19], here the model is used for discovering bounded prime numbers. Lastly another example is in [20], where this model is used to develop a parallel version of RAxML cod for phylogenetic studies.

In [21] and [22] CUDA and Pthreads were combined to easily support multi-GPU parallelism. In this model a CPU thread is assigned to each GPU making each device to have its own context on the host. However, in the first example the programmer must to split the code for the GPU to have the same amount of work.

A hybrid model based on CUDA and OpenMP could be found in [28], and [29]. In CUDA model the CPU and GPU cannot share their memory which means that the GPU gets its data from the CPU. Thus, to exploit the efficiency of the GPU OpenMP is used which allows the CPU to generate as much data as possible.

Similarly the hybrid model of MPI/OpenMP programming takes advantage of the underlying features of both programming models. It mixes the explicit decomposition and task placement of MPI with the simple and fine-grain parallelization of OpenMP. This model represents the most widespread use of mixed programming on SMP clusters. The reasons are its portability and the fact that MPI and OpenMP are industry standards. But, it is not clear that this programming model will always be the most effective mechanism. Reasonable work has gone into studying this hybrid model [23], [24], [25].

However, some reasons also make the model inefficient such as; shared memory issues, it may introduce the drawbacks of Open MP, and the runtime libraries may have negative impact on the program's performance.

CUDA and MPI hybrid model is useful for parallelizing programs in GPU clusters. The programming environment and the hardware structure of the cluster node are different from traditional ones because of the heterogeneous model based on the CPU and GPU. This model separates process control tasks from data computing tasks. MPI controls the communication between nodes, the applications and the interactions with the CPU, while the CUDA computes the tasks in the GPU [26], [27]. However, this programming model is not ideal for parallel applications; there are situations where it delivers poor performance.

Broadly speaking, there is no general solution for hybrid parallelization even when considering the same architecture. In practice, the best solution largely depends on the individual characteristics of each application. Despite the challenges, the hybrid models offer an attractive way to harness the possibilities provided by current architectures.

4.0 CONCLUSION

Recent studies have shown that with current multi-core CPUs, computer clusters can blend distributed memory programming among the cluster nodes, and shared memory parallel programming within the cores of each node. Thus, the hybrid parallel programming approach is becoming increasingly popular. Therefore, the hybrid approach offers a way to harness the possibilities of current, and legacy, architectures and systems. In addition, the easy availability of GPUs on multi-core systems is providing momentum to a new parallel programming model: heterogeneous programming. This supersedes pure GPU programming, allowing several multi-core CPUs and several GPUs to collaborate.

References

[1] Diaz, J., C. Muñoz-Caro, and A. Niño, A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era. IEEE Transactions on Parallel and Distributed Systems, 2012. **23**(8): p. 1369-1386.

[2] D. Kirk and W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann, 2010.

[2] W. Hwu, K. Keutzer, and T.G. Mattson, “The concurrency challenge,” IEEE Design and Test of Computers, vol. 25, no. 4, pp. 312-320, July 2008.

[3] Cao YJ, Sun HY, Qian DP, Wu WG. Stable Adaptive Work-Stealing for Concurrent Many-core Runtime Systems. IEICE Transactions Information and Systems, 2012, E95D(5):I-IO. ACM Queue, vol. 3, no. 7, pp. 54-62, 2005.

[4] W-C. Feng and P. Balaji, “Tools and Environments for Multicore and Many-Core Architectures,” Computer, vol. 42, no. 12, pp. 26- 27, Dec. 2009.

[5] R.R. Loka, W-C. Feng, and P. Balaji, “Serial Computing Is Not Dead,” Computer, vol. 43, no. 9, pp. 6-9, Mar. 2010.

[6] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, The Sourcebook of Parallel Computing. Morgan Kaufmann Publishers, 2003.

[7] Vajda A. Practical Many-Core Programming. Programming Many-Core Chips, 2011:175-21l.

[8] K. Kedia, “Hybrid Programming with OpenMP and MPI, Technical Report 18.337J, Massachusetts Inst. of Technology May 2009.

[9] D.A. Jacobsen, J.C. Thibaulty, and I. Senocak, “An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters,” Proc. 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Jan. 2010.

[10] C.-T. Yang, C.-L. Huang, and C.-F. Li, “Hybrid CUDA, OpenMP, and MPI Parallel Programming on Multicore GPU Clusters,” Computer Physics Comm., vol. 182, no. 1, 2011.

[11] M. Macedonia, "The GPU Enters Computing's Mainstream," Computer, vol. 36, no.10, pp.
106-108, Oct. 2003.
[13] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan,
"Brook for GPUs: Stream Computing on Graphics Hardware," Proc. SIGGRAPH, 2004.
[14] W.R. Mark, R.S. Glanville, K. Akeley, M.J. Kilgard, "Cg: A System for Programming
Graphics Hardware in a C-Like Language," Proc. SIGGRAPH, 2003.
[15] Sophisticated Library for Vector Parallelism: Intel Array Building Blocks, Intel; http://software.intel.com/en-us/articles/intelarray-building-blocks, 2010.
[16] W.D. Hillis and G.L. Steele, "Data Parallel Algorithms," Comm. ACM, vol. 29, pp. 1170-1183, 1986.
[17] M. Quinn, Parallel Programming in C with MPI and OpenMP. McGraw-Hill, 2004.
[18] C. Wright, "Hybrid Programming Fun: Making Bzip2 Parallel with MPICH2 & pthreads on the Cray XD1," Proc. CUG, 2006.
[19] P. Johnson, "Pthread Performance in an MPI Model for Prime Number Generation," CSCI 4576 - High-Performance Scientific Computing, Univ. of Colorado, 2007.
[20] W. Pfeiffer and A. Stamatakis, "Hybrid MPI/Pthreads Parallelization of the RAxML Phylogenetics Code," Proc. Ninth IEEE Int'l Workshop High Performance Computational Biology, Apr. 2010.
[21] J.C. Thibault and I. Senocak, "CUDA Implementation of a Navier- Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows," Proc. 47th AIAA Aerospace Sciences Meeting, 2010.
[22] S. Jun Park and D. Shires, "Central Processing Unit/Graphics Processing Unit (CPU/GPU) Hybrid Computing of Synthetic Aperture Radar Algorithm," Technical Report ARL-TR-5074, US Army Research Laboratory, 2010.
[23] L. Smith and M. Bulk, "Development of Mixed Mode MPI/ OpenMP Applications," Proc. Workshop OpenMP Applications and Tools (WOMPAT '00), July 2000.
[24] R. Rabenseifner, "Hybrid Parallel Programming on HPC Platforms," Proc. European Workshop OpenMP (EWOMP '03), 2003.
[25] B. Estrade, "Hybrid Programming with MPI and OpenMP," Proc. High Performance Computing Workshop, 2009.
[26] Q. Chen and J. Zhang, "A Stream Processor Cluster Architecture Model with the Hybrid Technology of MPI and CUDA," Proc. First Int'l Conf. Information Science and Eng. (ICISE '09), 2009.
[27] J.C. Phillips, J.E. Stone, and K. Schulten, "Adapting a Message- Driven Parallel Application to GPU-Accelerated Clusters," Proc. ACM/IEEE Conf. Supercomputing, 2008.
[28] H. Jang, A. Park, and K. Jung, "Neural Network Implementation using CUDA and OpenMP," Proc. Digital Image Computing: Techniques and Applications, pp. 155-161, 2008.
[29] G. Sims, "Parallel Cloth Simulation Using OpenMP and CUDA," thesis dissertation, Graduate Faculty of the Louisiana State Univ. and Agricultural and Mechanical College, 2009.