# Survey on Parallel Computing and Performance Modelling in High Performance Computing

Chaitali S.Jadhav
P.G Student M.E Computer science & Engineering,
Shriram Institute of Eng. & Technology, Paniv, Maharashtra, India
*Email id:chaitalijadhav999@gmail.com*

Dr. Deshmukh Pradeep K.
Principal Computer science & Engineering
Shriram Institute of Eng. & Technology
Paniv, Maharashtra, India
*Email id:-principalsietc@gmail.com*

Prof. Yevale Ramesh S.
Head of CSE Dept.
ME(CSE)*
*Email-id:ryevale33@gmail.com*

Prof. Dhainje Prakash B
Vice Principal at Shriram institute,Paniv India

***Abstract*—** The parallel programming come a long way with the advances in the HPC. The high performance computing landscape is shifting from collections of homogeneous nodes towards heterogeneous systems, in which nodes consist of a combination of traditional out-of-order execution cores and accelerator devices. Accelerators, built around GPUs, many-core chips, FPGAs or DSPs, are used to offload compute-intensive tasks. Large-scale GPU clusters are gaining popularity in the scientific computing community and having massive range of applications. However, their deployment and production use are associated with a number of new challenges including CUDA. In this paper, we present our efforts to address some of the issues related to HPC and also introduced some performance modelling techniques along with GPU clustering.

***Keywords:****Accelerators,GPU,HPC etc.*

*****

## I. INTRODUCTION

In the eras of eighty it was believed computer performance was best improved by creating faster and more efficient processors. This idea was challenged by parallel processing, which in essence means linking together two or more computers to jointly solve a computational problem. Since the early 90s there has been an increasing trend to move away from expensive and specialized proprietary parallel supercomputers towards networks of computers. Parallel computing mainly involving clusters. Clusters use intelligent mechanisms for dynamic and network-wide resource sharing, which respond to resource requirements and availability. These mechanisms support scalability of cluster performance and allow a exile use of workstations, since the cluster or network-wide available resources are expected to be larger than the available resources at any one node/workstation of the cluster. These intelligent mechanisms also allow clusters to support multiuser, time-sharing parallel execution environments, where it is necessary to share resources and at the same time distribute the workload dynamically to utilize the global resources efficiently. Scalable computing clusters, ranging from a cluster of PCs or workstations, to SMPs, are rapidly becoming the standard platforms for high-performance and large-scale computing. The main attractiveness of such systems is that they are built using a orderable, low-cost, commodity hardware fast LAN such as Myrinet, and standard software components such as UNIX, MPI, and PVM parallel programming environments. These systems are scalable, i.e., they can be tuned to available budget and computational needs and allow efficient execution of both demanding sequential and parallel applications.

## II. RELATED DATA

In this paper we will summarize how the parallel computing is beneficial with the advances in various tools available and along with the high performance computing. And some concepts in GPU and issues in HPC are of more considerable. Firstly there are some advantages of parallel computing are summarized here.

The important advantages of parallel computing are given below.

1. Programmability
A set of ready-to-use solutions for parallelization will considerably increase the productivity of the programmers: the idea is to hide the lower level details of the system, to promote the reuse of code, and relieve the burden of the application programmer. This approach will increase the programmability of the parallel systems.

2. Reusability
Reusability is a hot-topic in software engineering. The provision of skeletons or templates to the application programmer increases the potential for reuse by allowing the same parallel structure to be used in different applications. This avoids the replication of efforts involved in developing and optimizing the code specific to the parallel template. In it was reported that a percentage of code reuse rose from 30 percent up to 90 percent when using skeleton-oriented programming. Since the programmer will have more time to

spend in optimizing the applications itself, rather than on low-level details of the underlying programming system.

3. Portability
Providing portability of the parallel applications is a problem of paramount importance. It allows applications developed on one platform to run on another platform without the need for redevelopment.

4. Efficiency
There could be some connecting trade-offs between optimal performance and portability/programmability. Both portability and efficiency of parallel programming systems play an important role in the success of parallel computing.

**A.GPU**

Generally the graphics processing unit (GPU) is a specialized unit and highly parallel microprocessor designed to offload and accelerate 2D or 3D rendering from the central processing units. GPUs can be found in a wide range of systems, from desktops and laptops to mobile phones and super computers [12]. As we see the evolution of GPU hardware architecture has gone from a very firstly the single core, fixed function hardware pipeline implementation made solely for graphics, to a set of highly parallel and programmable cores for more general and easy purpose computation. The trend in GPU technology has undoubtedly adding more programmability and parallelism to a GPU core architecture that is ever evolving towards a general purpose more CPU resembling core. A graphics processing unit (GPU) is a dedicated parallel processor optimized for parallel floating point computing power found in a modern GPU is orders of magnitude higher than a CPU [13]. Recently, NVIDIA refreshed their Fermi-based gaming card, the GTX580,by adding one more SM and offering a slightly higher memory bandwidth. Now, the architecture of many-core GPUs are starting to look more and more like multi-core, general purpose CPUs [14]. In that respect, Fermi can essentially be thought of as a 16-core CPU with 32-way hyper-threading per core, with a wide vector width. The General Purpose GPU (GPGPU) had come a long way .But GPGPU was far from easy back then, even for those who know graphics programming languages such as OpenGL. Developers had to map scientific calculations onto problems that could be represented by triangles and polygons. GPGPU was practically found more difficult to those who hadn't know the latest graphics In 2003, a team of researchers leading by Ian Buck unveiled Brook, firstly adopted programming model to extend C with data parallel constructs. They have used the concepts like streams, kernels and reduction operators, the Brook compiler and runtime system exposed the GPU as a general-purpose processor in a high- level language. Here the most important and considerable thing was the programs designed by Brook were very much easy to write than hand-tuned GPU code, and they were faster seven times than similar code which was existing currently. The massively parallel hardware architecture and high performance of floating point arithmetic and memory operations on GPUs make them particularly well-suited to many of the same scientific and engineering workloads that occupy HPC clusters, leading to their incorporation as HPC accelerators. Beyond their appeal as cost-effective HPC accelerators, GPUs also have the potential to significantly reduce space, power, and cooling demands, and reduce the number of operating system images that must be managed relative to traditional CPU-only clusters of similar aggregate computational capability. Although successful use of GPUs as accelerators in large HPC clusters can confer the advantages, they present a number of new challenges in terms of the application development process job scheduling and resource management, and security. After evolution of parallel programming the concept of high - performance computing evolved with GPUs. The high performance computing landscape is shifting from collections of homogeneous nodes towards heterogeneous systems, in which nodes consist of a combination of traditional out-of-order execution cores and accelerator devices. Large-scale GPU clusters are gaining popularity in the scientific computing community. However, their deployment and production use are associated with a number of new challenges. In this paper, we have introduced some of the challenges with building and running GPU clusters in HPC environments.

**B. GPU CLUSTER PROGRAMMING**

There are mainly three principal components used in a GPU cluster:
Host nodes, GPUs, and interconnect. Since the expectation is for the GPUs to carry out a substantial portion of the calculations, host memory and network interconnect performance characteristics need to be matched with the GPU performance in order to maintain a well-balanced system. The GPU Code Development Tools what we have are of two abstraction levels as High abstraction subroutine libraries that provide commonly used algorithms with auto generated or self-contained GPU kernels, e.g., CUBLAS, CUFFT, and CUDPP. And another is Low abstraction lightweight GPU programming toolkits, in which the programmers write GPU kernels entirely by themselves with no automatic code generation, e.g., CUDA and Open CL.

1. CUDA C
CUDA is a scalable parallel programming model and software model developed for parallel computing which is widely deployed through thousands of applications and published research papers and supported by an installed base of over 375 million CUDA-enabled GPUs in notebooks, workstations, compute clusters and supercomputers. Currently, NVIDIA's CUDA toolkit is widely used GPU programming toolkit which are available. The CUDA programming model is focused entirely on data parallelism, and provides convenient lightweight programming abstractions which allow programmers to express kernels in terms of a single thread of execution, which is expanded at runtime to a collection of blocks of tens of threads that cooperate with each other and share resources, which expands further into an aggregate of tens of thousands of such threads running on the entire GPU device. Since CUDA uses language the work of packing and unpacking GPU kernel parameters and specifying various runtime kernel launch parameters is largely taken care of by the CUDA compiler. This makes the host side of CUDA code relatively uncluttered and easy to read. The CUDA toolkit

provides a variety of synchronous and asynchronous APIs for performing host-GPU I/O, launching kernels, recording events, and overlapping GPU computation and I/O from independent execution streams. Let's have a look over a basics of CUDA Programming. It is consists of following hierarchy.

**Software stack-**

CUDA can scales up to 100s of cores and 1000s of parallel threads. The focus is totally on parallel algorithms rather than parallel computing. It can also be deployed on heterogeneous systems (CPU+GPU).Now coming to CUDA Kernels in which the application is running in parallel portion is executed on a device known as kernels. Only one kernel can execute at a time. But there may be many threads can execute on each kernel. threads are very lightweight. CUDA kernels are executed by an array of threads. The main feature of CUDA is thread cooperation. The second part comes as the
**Data management-**The data management involves management of memory by managing data. CPU and GPU having separate memory spaces. Host codes which are considered asCPU manages the device which is nothing but GPU memory. It firstly allocates the memory or releases it. Then it copies data to and from device. And at last applies to the global device memory. Here are some functions which are used in memory allocation and release like cudaMalloc, cudaMemset and cudaFree.

Next and the last part of the hierarchy is **Execution of codes on GPU-**

It is mainly kernel with C. Though there are some functions which are having some restrictions like it cant access host memory and having no any ststic variable.The function arguments automatically copied from host to device.Kernels designated by function qualifires like _global_

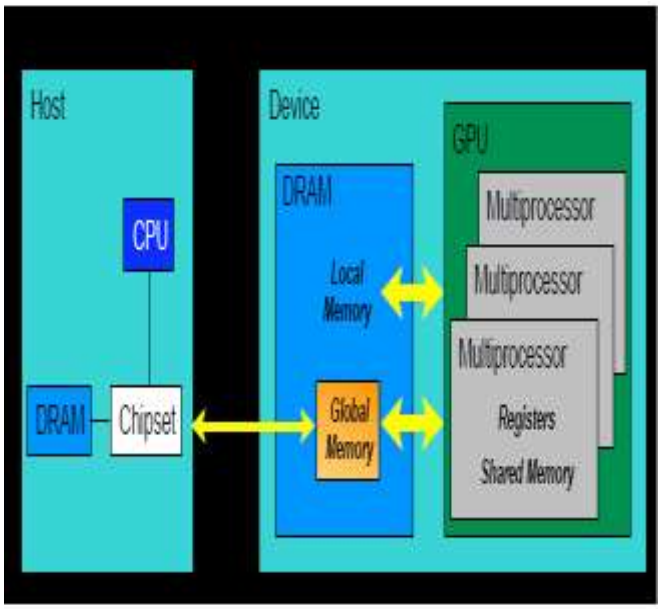


Fig 1: Data management in CUDA

.

2. Open CL

The Open CL programming model is based on the notion of a host device, supported by an application API, and a number of devices connected through a bus. These are programmed using Open CL C. The host API is divided into platform and runtime layers. Open CL C is a C-like language with extensions for parallel programming such as memory fence operations and barriers.Open CL is a newly developed industry standard computing library that targets not only GPUs, but also CPUs and potentially other types of accelerator hardware. Once the Open CL kernels are compiled, the calling application must manage these kernels through various handles provided by the API. In practice, this involves much more code than in a comparable CUDA-based application, though these operations are fairly simple to manage. Most Open CL programs follow the same pattern. Given a specific platform, select a device or devices to create a context, allocate memory, create device-specific command queues, and perform data transfers and computations. Generally, the platform is the gateway to accessing specific devices, given these devices and a corresponding context, the application is independent of the platform. Open CL uses all computational resources in the system. Open CL applications works in a way that firstly the serial code executes in a Host CPU Thread and after that parallel code executes in many devices (GPU) threads across multiple process elements. Open CL decomposes task into work-items which defines N-dimensional computation domain. And executes a kernel at each point in computation domain. Open CL execution model consists of the application runs on a Host which submits work to the Devices. Work-item is the basic unit of work on an Open CL device. Kernel is the code for a work-item which is basically a C function. Program:

Collection of kernels and other functions.The application runs on a Host which submits work to the Devices**.** Many operations are performed with respect to a given context there are many operations that are specific to a device. For example, program compilation and kernel execution are done on a per-device basis. Performing work with a device, such as executing kernels or moving data to and from the device's local memory, is done using a corresponding command queue. A command queue is associated with a single device and a given context.

**C .HPC**

High performance computing as we see it in very simple way is nothing but collection of computers, networks, algorithms and environments to make such system usable. So very big hardware architectures and high performance of floating point arithmetic and memory operation make them suited on to many same scientific and engineering workloads which occupy HPC clusters. HPC is nothing but the aggregation of computing powers which delivers higher performance than typical machine, together. As we take a look over HPC it is becoming more important and popular emerging trends for national economics because HPC also called as supercomputing, and which is linked to economic compressive and competitive and scientific advances. HPC consisting of some parallel architectures as Single instruction Single data

sequential machine, Multiple instruction Single data, Single instruction multiple data, and Multiple instruction multiple data.TESLA is an important concept in HPC which is a high-end GPU oriented to general-purpose computing. HPC come up with clusters. Cluster is nothing but all of those components working together to form one big computer. The HPC cluster needs several computers, nodes, one or more interconnected networks and software that allows the nodes to communicate with each other. HPC comes with communication overhead as latency and bandwidth.

**Latency-** Start up time for each message transaction which is in order of 1 µs.

**Bandwidth-** The rate at which the messages are transmitted across the nodes /processors whose order is of 10 Gbits / Sec.

HPC having applications in the fields like simulation of physical phenomenon like climate modelling and galaxy formation. Also in field of Data mining in gene sequencing. And also for visualisation purposes in reducing large data sets into pictures for scientific understandings.

Performance issues in HPC can be summed up as following equations: **Speed up**

$$\textbf{Speed up} = \frac{\textbf{Time for sequential code}}{\textbf{Time for parallel code}}$$

$$Sp = \frac{Ts}{Tp} \qquad 1 \le Sp \le P$$

**Efficiency**

$$\textbf{Ep} = \frac{\textbf{Sp}}{\textbf{P}} \qquad \textbf{0 < Ep < P}$$

**EP = 1 = > SP = P 100% Efficient**

**D. Performance modelling**

There are mainly three models for performance modelling as per our review are simulation, analytical modelling, machine learning.

1. Simulation

A simulator is a system representation which able to map, step-by-step, the behavior of the target system. Simulators are widely used for carrying out performance studies of existing hardware and software platforms, and also analyze platforms that either do not exist, or are not available. The accuracy of the output information provided by a simulator depends on many factors, which vary upon number of details. Some Simulators of GPU-based accelerators are accelerators are GPGPU-Sim [8], [9] and Barra [10], [11]. Both are simulations of NVIDIA and uses CUDA implementation.

2. Analytical Model

An analytical model is an abstraction of a system in the form of a set of equations. These equations try to represent and comprise all of the characteristics of the system.

3. ML Model

ML model is mainly designed to predict execution times of a particular application on a particular hardware platform with a specific configuration may take as training dataset the execution times of applications running in the same or similar hardware for a variety of different configurations. Generally all these models are different from each other.

## III. CONCLUSION

There are many emerging technologies in parallel programming along with HPC. The appearance of new computing devices and the design of new algorithms in different fields of science and technology is forcing a fast evolution of HPC. Designing and developing programs that use currently available computing resources efficiently is not an easy task. As stated in present-day parallel applications differ from traditional ones, as they have lower instruction-level parallelism, more challenging branches for the branch-predictors and more irregular data access patterns. The goal of these architectures is twofold: providing an unified address space that eliminates the need to interchange data with an external accelerator using a system interconnect and improving power efficiency reducing the total transistor count. A majority of the models discussed in this review have been designed for CUDA, the most mature development environment for GPGPU. However the vendor neutrality of Open CL and its availability. For non-GPU accelerators is increasing its adoption by HPC programmers. In the past, Open CL tools produced less efficient codes than their CUDA counterparts, but this is no longer true with the most current versions of Open CL ,SDKs. So this paper is an overview of all these concepts.

## IV. ACKNOWLEDGMENT

The survey is based on IEEE Transaction paper under the title A Survey of Performance Modelling and Simulation Techniques for Accelerator-based Computing .published in IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. And still the work is going on this topic.

## V. REFERENCES

[1] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring GPGPU workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in IEEE Int. Symp. on Workload Characterization (IISWC), 2010, pp. 1–10.

[2] N. Brunie, S. Collange, and G. Diamos, "Simultaneous branch and warp interweaving for sustained GPU performance," SIGARCH Comput. Archit. News, vol. 40, no. 3, pp. 49–60, 2012.I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in

Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[3] R. Jain, The Art of Computer Systems Performance Analysis. Wiley,1991.

[4] G. Wilson. Parallel Programming for Scientists and Engineers. MIT Press, 1.Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stove, "GPU Cluster for High Performance Computing,", in Proc. ACM/IEEE conference on Supercomputing, 2004.

[5] H, Takizawa and H. Kobayashi, "Hierarchical parallel processing of large scale data clustering on a PC cluster with GPU co-processing," J. Supercomputer., vol. 36, pp. 219--234, 2006.

[6] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, S. Buijssen, M. Grajewski, and S. Tureka, "Exploring weak scalability for FEM on a GPU-enhanced cluster," Parallel Computing, vol. 33, pp. 685-699, Nov 2007.

[7] M. Arora, S. Nath, S. Mazumdar, S. Baden, and D. Tullsen, "Redefining the Role of the CPU in the Era of CPU-GPU Integration,"Micro, IEEE, vol. 32, no. 6, pp. 4–16, 2012.

[8] "GPGPU-Sim Online Manual," http://gpgpu-sim.org/manual/ index.php5/GPGPU-Sim 3.x Manual/, Apr. 2013.

[9] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in IEEE Int. Symp. on Performance Analysis of Systems and Software.ISPASS 2009., pp. 163–174.

[10] S. Collange, M. Daumas, D. Defour, and D. Parello, "Barra: a parallel functional simulator for GPGPU," in IEEE Int. Symp. On Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010, pp. 351–360.

[11] "Barra GPU Simulator," https://code.google.com/p/barra-sim/, Apr. 2013.

[12] Wikipedia: Video Card. Retrieved November 2010. http ://en.wikipedia.org/wiki/Video_card

[13] Wikipedia: Graphics Processing Unit. Retrieved November 2010.http :/ / en . wikipedia . org / wik i / Graphics processing uni t

[14] AMD Fusion whitepaper. 2010 AMD.http ://sites.amd.com/us/Documents/48423B_fusion_whitepaper _WEB.pdf