Parallel Programming Models: A Systematic Survey

Chougule Meenal D[1], Gutte Prashant H[2]

[1,2]*PG Student Information Technology Department,*
Walchand College of Engineering, Maharashtra, India

***Abstract*— This paper presents all parallel programming models available today. It reviews shared and distributed memory approaches. Hybrid programming models are also playing important role in High Performance Computing (HPC) era. This makes best use of both shared and distributed approaches. The study shows multi-core CPU's have given different impulse to shared memory model programming. Along with this, heterogeneous programming is explored, to exploit combined effects of CPUs and Graphics Processing Units (GPUs) Graphics Processing. This work introduces the contribution of Open standards such as Open Multi Processing (OpenMP), Message Passing Interface (MPI), Open Computing Language (OpenCL), Open Computer Vision (OpenCV) and OpenACC in parallel computing. This survey is accomplished with study of different programming languages according to parallel models.**

***Keywords*— Parallel Models, distributed programming, CUDA, openMP, GPU, OpenCV.**

I. INTRODUCTION

Earlier microprocessors brought giga (billion) floating-point operations per second (GFLOPS) to the desktop and hundreds of GFLOPS to cluster servers. But this trend has slowed since 2003 due to energy consumption and heat-dissipation issues [1]. This caused limitation on increase in CPU clock frequency. To overcome this problem a new design of processor came into picture termed as multi-core. In multi-core systems, two or more computing units are integrated on to a single microprocessor. Multi-core systems maintain execution speed while increasing number of cores. In contrast to this Many-core system having thousand numbers of cores tries to increase the throughput.

Initially all software applications are basically follow sequential execution flow. The program execution flow was first described by Von Neumann. The sequential execution models uses single core at a time but, this idea is not suitable today. So, the applications that can benefit from performance increases with each generation of new multi-core and many-core processors are the parallel ones [2]. Thus, parallel programming became most efficient way to improve performance of applications and most promising way to use underlying hardware.

Developing a parallel program involves dividing the program i.e. set of instructions into different subtasks. These subtasks are executed independently on different cores. Communication between tasks is carried if and whenever required. After completion of execution results are collected. There are various design patterns described to achieve parallelism. They differ in the achievable application performance and ease of parallelization. This paper defines different parallel programming models in section 1. Brief study and explanation of these models along with programming language is given in section 2, 3 and 4. Final section concludes the work.

II. PARALLEL PROGRAMMING MODELS

Throughout the years, there has been many number of parallel programming models proposed. Strictly speaking, a parallel programming model is an abstraction of the computer system architecture [3]. Parallel programming models and its associated implementations, i.e., the parallel programming environments defined by Mattson et al. [3]. Prominently, while building HPC applications, two main programming models are followed: a) OpenMP used in shared memory architecture, other is MPI used in distributed memory systems. These two models are termed as Pure Parallel Models [2].

Many-core processor architecture became so famous now-a-days called as GPU's. To exploit parallelism in such architecture Heterogeneous parallel programming is used. PGAS is another such variant of model which logically partitions global address space to use locality of reference. Hybrid models combine all of the above models into one.

Feature	Pthread	OpenMP	MPI
Memory model	Shared	Shared	Distributed + Shared
Worker Management	Explicit	Implicit	Explicit/ Implicit
Communication	Shared Address space	Shared Address space	Message passing

Fig.1 Comparison of programming Models

A. Pure Parallel Programming Models

As mentioned in the previous section pure models broadly divided into two types: Shared Memory Model and Distributed Memory model. These models are implemented using threads i.e. Portable Operating System Interface (POSIX) threads, OpenMP and MPI. Intel provides Thread Building Block (TBB), a C++ template library which is specially designed for utilizing multi-core architecture. Table 1, explains difference of pure parallel programming implementations.

1) POSIX Threads

POSIX threads are usually termed as Pthreads. In this programming structure two or more threads are used and manipulated independently. Each thread in execution has its own stack pointer, registers, scheduling properties and thread specific data. In UNIX threads are lightweight processes which are easy to manage and control. In 1995 a

standard was released [4]: the POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995), or as it is usually called Pthreads.

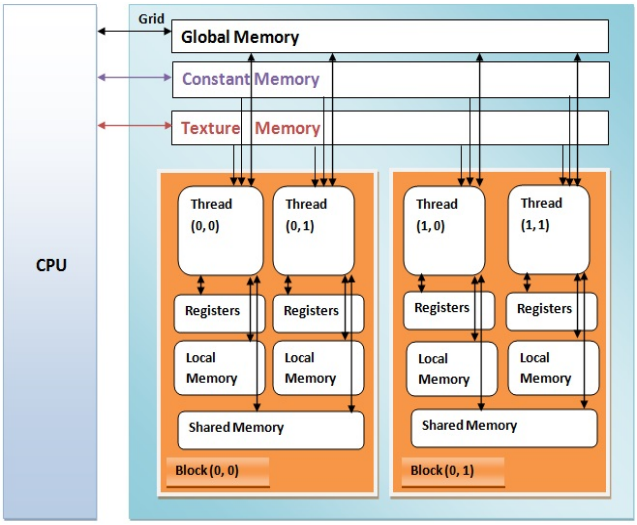

Fig 2: CUDA Memory Model

These Pthreads are implemented using a header or library (Pthread.h). This library is used for creating and destroying threads. Some functions are also useful to co-ordinate thread activities using locks, mutex, critical sections, Semaphore and some conditional variables. This model is especially appropriate for the fork/join parallel programming pattern [3].

In this model heap memory which is dynamically allocated and global variables are shared among all threads. So, when multiple threads access the shared data, programmers have to be aware of race conditions and deadlocks.

Pthread model is not well structured, is not recommended as a general purpose parallel program development technology. The most important thing to mention is that, number of threads are not directly related to number to processor cores. So, scalable application development is difficult in this model.

2) Shared Memory Model with OpenMP

As mentioned above task level parallelism is achieved in shared memory architecture using OpenMP. OpenMP is open standard Application programming interface (API) which supports shared memory multiprocessing programming. It is available in C, C++ and Fortran. Its current stable release is OpenMP 4.0 on July 23, 2013.

OpenMP and Pthread both are multithreaded programming libraries but, they differ with each other. Pthreads is purely implemented as a library, OpenMP is implemented as a combination of a set of compiler directives, pragmas, and a runtime providing both management of the thread pool and a set of library routines [2].The OpenMP model is highly structured and designed for HPC applications.

OpenMP is portable across the shared memory architecture. Workload partitioning and task-to-worker mapping require a relatively few programming effort. Programmers just need to specify compiler directives to denote a parallel region. OpenMP is specially suited for the loop parallel program structure pattern, although the SPMD and fork/join patterns also benefit from this programming environment.

3) Distributed Memory Model with MPI

MPI is a specification for message passing operations which is a natural way of communication in distributed memory architecture.

Originally, MPI was designed for distributed memory architectures, which were becoming increasingly popular at that time (1980s - early 1990s). This library is available in C, C++, Fortran and Java. Some of the well known MPI implementations are OpenMPI, MVAPICH, MPICH, GridMPI, LAM/MPI, and MRMPI. Message Passing is a parallel programming model where communication between processes is done by interchanging messages. This is a natural model for a distributed memory system.

Each working element is called as process. Workload partitioning and task mapping have to be done by programmers, similar to Pthread. Programmers have to manage what tasks to be computed by each process. In this model, the processes executed in parallel have separate memory address spaces. Communication occurs when part of the address space of one process is copied into the address space of another process. MPI operations are broadly classified into two types, point to point and collective operation.

When Java first appeared there was immediate interest in its possible uses for parallel computing, and there was a little explosion of MPI and PVM “bindings” for Java. MPJ Express is an open source Java message passing library that allows application developers to write and execute parallel applications for multi-core processors and compute clusters/clouds. MPICHG2 and GridMPI are used for MPI implementations in grid environment. MPI is basically useful for task-parallel computations and for applications where the data structures are dynamic, such as unstructured mesh computations.

B. Heterogeneous Parallel Programming Models

Today’s computer systems are having one or more CPU’s and one or more GPU’s.

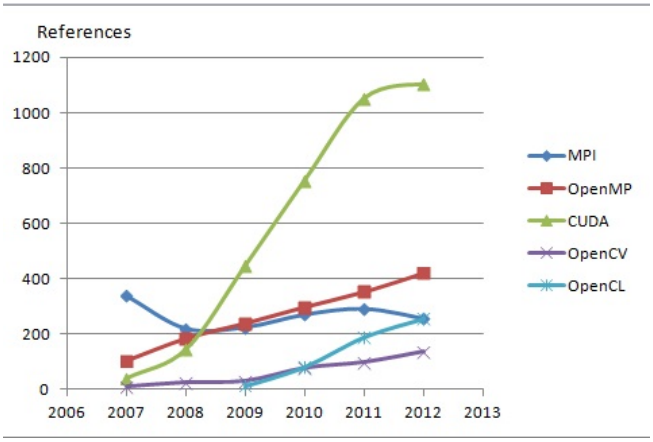

Fig 3: No. Of Hits in Scopus database

In 2003 the Siggraph/Eurographics Graphics Hardware workshop, held in San Diego, showed a shift from graphics to non-graphics applications of the GPUs [5]. Then everyone came to know that, GPU's can be used for general purpose and scientific applications.

As mentioned earlier GPU's mainly designed for throughput oriented applications. GeForce, Tegra, Tesla, Quadro are the GPU's provided by NVIDIA.

An APU is an Accelerated Processing Unit which integrates the CPU (multi-core) and a GPU on the same die. To implement application on such architecture Brook [6] and cg [7] languages are used. Then NVIDIA introduced Compute Unified Device Architecture (CUDA) for doing parallel programs [8].

To support many-core architecture Intel provides Array Building Block (ArrBB), a C++ library. Its stable release is in August 2011[9] and Direct Compute [9] is an API also supports many-core architectures. There are API's available as an Open standard like OpenCL, OpenCV, and OpenACC which are also useful.

1) CUDA

The computation of tasks is done in GPU by a set of threads that run in parallel. The GPU architecture for threads consists of two-level hierarchy, namely *block* and *grid.* Block is a set of tightly coupled threads where each thread is identified by a thread ID, while grid is a set of loosely coupled of blocks with similar size and dimension [3].A CUDA program consists of one or more phases that are executed on either the host (CPU) or a device such as a GPU [1].

Worker management in CUDA is done implicitly. Programmers do not manage thread creations and destruc tions. They only specify the dimension of the grid and block required to process a certain task. But, workload partitioning and worker mapping in CUDA is done explicitly. Programmers have to define the workload to be run in parallel by using the function "Global Function" and specifying the dimension and size of the grid and of each block. [2] The stable release of CUDA is CUDA 6.0 on November 2013. The CUDA memory model is given in Fig 2.

2) Direct Compute:

DirectCompute is Microsoft's API which is used for GPU programming [10]. 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. In particular, it was introduced thanks to the new Shader Model 5 provided in DirectX 11, which allows computation independently of the graphic pipeline, therefore suitable for GPGPUs. The advantage of DirectCompute is that, it supports only in windows platform.

3) OpenCL, OpenCV and OpenACC

OpenCL is Open Computing Language. OpenCL is first free cross-platform standard for heterogeneous parallel computing. OpenCL 2.0 is the latest significant evolution of OpenCL standard, designed to further simplify cross-platform programming while enabling a rich range of algorithms and programming patterns to be easily accelerated [11].

OpenCV was designed for computational efficiency and with a strong focus on real-time applications. It has C/C++, Python, Java interfaces and written in C++. OpenACC is a programming standard designed to simplify parallel programming of heterogeneous CPU/GPU systems. It is having collection of compiler directives to be offloaded from host CPU to an accelerator.

C. Hybrid Parallel Programming Model

This programming model is a modern software trend for the current hybrid hardware architectures [2].The idea behind this is, use message passing over distributed nodes and shared memory within single node. To do this one can use above mentioned standard and API's. CUDA, OpenCL and OpenACC can be used to implement multi CPU and multi GPU programming. OpenMP or Pthreads and MPI can also be combined to exploit shared and distributed memory models.

Combining CUDA and MPI is useful for parallelizing programs in GPU clusters. MPI is used to control the application, the communication between nodes, the data schedule, and the interaction with the CPU. Meanwhile, CUDA is used to compute the tasks in the GPU [12], [13].

III. CONCLUSIONS

This paper mainly reviews different parallel programming models. From study it is seen that, there is vast change in processor architecture since last ten years. . The improvised and new processor systems tend to use different models and API's. GPGPU's are becoming most promising systems for high performance computing. The increasing significance of parallelism in the computational field can be mapped on computing literature over the last decade. Thus, Fig. 3 represents the number of hits in the Scopus database [14] for some keywords: MPI, OpenMP, CUDA, OpenCL and OpenCV.

By referring to the above chart we can see that, MPI is the oldest and most widely used distributed programming model. CUDA changed the meaning of parallel computing drastically over the year from 2009 till now. Open standard industries are also contributing major share in parallelism. OpenCL, OpenACC are new bees in this area but playing good role. OpenCV is also showing its results so its use is increasing per year.

REFERENCES

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

[2] Javier Diaz, Camelia Mun˜oz-Caro, and Alfonso Nin˜o, "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era" IEEE Tranc. On Parallel and Distributed Systems, Vol. 23, No. 8, August 2012

[3] T.G. Mattson, B.A. Sanders, and B. Massingill, Patterns for Parallel Programming. Addison-Wesley Professional, 2005.

[4] POSIX1003.1FAQ,http://www.opengroup.org/austin/papers/posix_faq.html, Oct. 2011.

[5] M. Macedonia, "The GPU Enters Computing's Mainstream,"Computer, vol. 36, no.10, pp. 106-108, Oct. 2003.
[6] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston,and P. Hanrahan, "Brook for GPUs: Stream Computing onGraphics Hardware," Proc. SIGGRAPH, 2004.
[7] 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.
[8] CUDAZone, http://www.nvidia.com/object/cuda_home_new.html, Oct. 2011.
[9] MicrosoftDirectXDeveloper Center,http://msdn.microsoft.com/en-us/directx/default, Oct. 2011.
[10] P.B. Hansen, Studies in Computational Science: Parallel Programming Paradigms. Prentice-Hall, 1995.
[11] Khronos Group, http://www.khronos.org/opencl, Jan, 2014
[12] 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.
[13] 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.
[14] http://www.scopus.com Jan 2014