Research Journal of Computer and Information Technology Sciences ______________________ISSN 2320 – 6527
Vol. **11(1),** 1-6, June **(2023)** *Res. J. Computer and IT Sci.*

Review Paper

Concept of High Performance Computing

Ashish Kumar*, Pawan Patnaik and Sargam Gupta
Department of Computer Science and Engineering, Bhilai Institute of Technology, Durg, C.G., India
ashishsahu96sky@gmail.com

Available online at: www.isca.in
Received 3rd September 2021, revised 13th February 2023, accepted 15th May 2023

Abstract

For any computational problem there can be more than one solution with different computational resource demands and execution time. One of the prominent factors while considering the performance of various solutions is execution time. High Performance computing techniques and models deals with the challenges of handling problems at massive scale using computing infrastructures, tools, techniques and parallel algorithm designing programming skills. With the advent of new HPC paradigms and significant improvements in processor design, it's now feasible to employ HPC techniques on many new compute intensive domains. The game changer in HPC has been the developments of the last decade with the introduction of GPUs and FPGAs which together have revolutionized the HPC computing space. This paper presents a comprehensive review of the three major computational models used in HPC – multi core, cluster and GPGPU.

Keywords: GPGPU, HPC, MPI, OpenMP, CUDA.

Introduction

Necessity is the mother of invention. We are constantly under the greed for more and more computing power to handle the challenges of the Bigdata revolution. Access to high computational power allows us to explore possibilities across many different compute intensive fields. The field of HPC have been addressing the issues of massive computational power for handle these kinds of compute intensive workloads. Traditional HPC have focused around aggregating computational power by using clusters of computational nodes which communicate through well defined message passing paradigms like MPI. In the last 2 decades, developments in VLSI and processor technology have equipped our desktop processors to be multi cores. This opened up new parallel computing paradigms like multi core computing on shared memory architectures. Both compilers and programmers leveraged this increased computational capabilities of multi core processors.

GPU and CPU are two processing units[1]. GPUs have been used as a dedicated piece of hardware for accelerating image and graphics workloads. The HPC space experienced a game changing transition when GPUs were used as a code acceleration device for general purpose computations tasks too. GPGPU is the term used when GPUs are employed for general purpose programming[2].

Sharing the workload of CPU and taking away the burden of the parallel part of the code to be executed in the multiple cores of the GPU. This idea of using GPUs for offloading some of the computational workload of the CPU proved to be a very promising design optimization for many domains like data analytics, artificial intelligence, machine learning.

The idea of GPGPU was materialized by the introduction of CUDA. Initially abbreviated as Common Unified Device Architecture, CUDA was a C –based library from NVidia that offloaded the parallel sections of the c code to the GPU cores while the sequential part was executed by the CPU. An efficient work sharing compiler nvcc developed by Nvidia was used to compile the CUDA programs written in C.

Core is small and individual processor built into a big CPU. It is an integrated circuit that implements independent physical execution at same time. If system has 2 cores then it executes two separate executions at a same time that enhance the speed of computing.

Enhance the speed of system need multi-cored system CPU is best for serial processing but it has limited numbers of core maximum 128 cores CPU that only handle 128 different executes at a time. Now GPU comes under the picture with many numbers of core to operate many numbers of execution at a same time and have features of High throughput, good for parallel operation. There are many companies who are in the market of GPU – Nvidia, AMD, Asus, etc. Nvidia GPU RTX-3090 has a core size 10496 and CPU Amperes Altra max has max 128 core.

Methodology

To understand the concept and methods first come to understand the threads. Basically threads are the highest level of code execution by processors it is virtual component that manage the task. If you have lines of code (program) to execute but it takes a finite time to execute that program wholly.

Here a technique to execute that code less than that finite time is threading that program in different levels and run those levels separately at same time in that core that is threading. Number of threading is greater than two is multi-threading.

Now threading in single core leads concurrency behavior means it execute the threads in sequential manner if one program has 2 threads (T1 and T2) those threads can run one by one if start with T1 but after some time it Executes T2 as well and come back to T1.

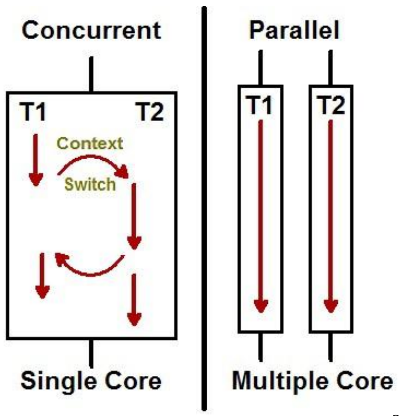


Figure-1: Concurrency vs parallelism[3].

Real life example of concurrency: We can take an example of railway counter there is only one ticket clerk but the line is too long. Then clerk announced to create two lines one for men and other for women (Threading). That is threading big line into two small lines and executes to provide a ticket sequentially one man after one woman and so on. The line thought it is short to proceed.

Threading is useful when we crawling the web page the site load their information simultaneously load the interface but it is very useful when system has multi- core that is basically a parallelism. In parallelism number of program can execute on different threads and saves the computing time that makes the system high performance computing system.

Real life example of parallelism: If there is two clerk for ticket booking and that work for two threads as above one for men and other for women then the booking of tickets for lines may execute with very fast. Here we can relate the clerk as Core, whole line as Program and line for men and women are Threads. For high performance computing system should have multi-core, multi-threads for better performance.

There is big role of memory also we can call it as RAM (Random Access Memory) that are volatile in nature mean if the program exhausted or the system turned off them their stored memory can lost. Means it free those spaces that they occupied during the session. There are two types of model named[4] i. Shared memory model: In shared memory model the system has only one memory for many number of cores (CPU) like our pc, laptops that share one memory for their multi-core architecture.
ii. Distributed memory model: In distributed memory model the CPU has their own local memory. Means every core has their own memory and those systems are interconnected virtually.

Now lets take an example to understand how we can increase the performance of computing using parallelism.

One set of program has 40% dependent and 60% are independent: Here dependent program can execute only by sequential process but that independent program can execute parallel. Now consider the system has 2 core to run for parallel. If execute whole program using sequential method that takes time called Tseq is 100% x. For parallel operation 40% are dependent that execute only by sequentially but 60% are execute as parallelly with 2-cores means 30% - 30% percent each cores. So parallel time Tpara 40% x+60% x/2 = 70% x unit time.

Now Performance = Tseq/Tpara = 100% x/ 70% x = 1.4285
This means if we perform some program parallelly then we can enhance the system performance by 42.84% (as per example) this is basically AMDAHL'S law. AMDAHL’S law is used to check the system performance when their programs executes parallelly with respect to when all programs executes sequentially.

There are techniques/APIs available to enhance the performance are OpenMP, MPI, and CUDA.

OpenMP[5]: OpenMP has evolved as the de-facto standard accepted by the industry and the academia for multicore computing model that employs the fork-join thread model for parallel execution[6]. By itself OpenMP is a standard and it is implemented by many vendors across various operating systems in languages are Fortran and C/C++. It is the set of compiler directives and that provides supports for parallel programming in shared memory architecture[7]. OpenMP is a library for parallel programming in SMP. OpenMP identifies parallel regions as block of codes that may execute in parallel. Developers insert compilers directives into their codes to instruct the OpenMP runtime library to execute the region in parallel. OpenMP included header file for program #include<omp.h>

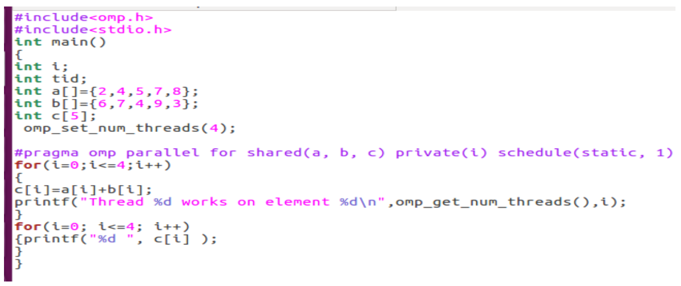
Open MP has two types of threads: i. Master thread execute for sequential section that is execute from beginning to end with thread id is 0. ii. Slave threads executes for parallel section that are forked with directives called #pragma opm parallel into number of threads as core available in that system with thread id 1 to N. Where N is the number of core.

OpenMP fork the set of program in number of threads as have core on that system using command in that region if system have 2 core it created two threads for parallel operation.

```
#pragma omp parallel{}
```

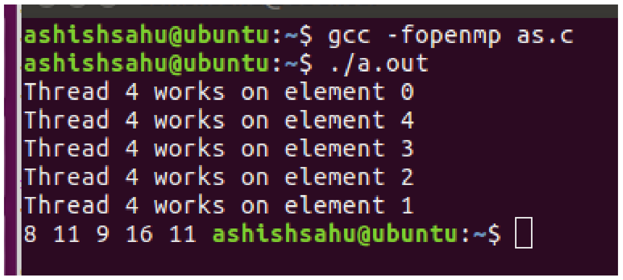
And there is other option if you have 2 core system and want to create a 4 threads by using: omp_set_num_threads(4);
Compile the program using: gcc -fopenMP <filename.c>
To execute the program: ./a.out

OpenMP program:



```
#include<omp.h>
#include<stdio.h>
int main()
{
int i;
int tid;
int a[]={2,4,5,7,8};
int b[]={6,7,4,9,3};
int c[5];
 omp_set_num_threads(4);

#pragma omp parallel for shared(a, b, c) private(i) schedule(static, 1)
for(i=0;i<=4;i++)
{
c[i]=a[i]+b[i];
printf("Thread %d works on element %d\n",omp_get_num_threads(),i);
}
for(i=0; i<=4; i++)
{printf("%d ", c[i] );
}
}
```

Figure-2: Code for Adding two sets of data using OpenMP.



```
ashishsahu@ubuntu:~$ gcc -fopenmp as.c
ashishsahu@ubuntu:~$ ./a.out
Thread 4 works on element 0
Thread 4 works on element 4
Thread 4 works on element 3
Thread 4 works on element 2
Thread 4 works on element 1
8 11 9 16 11 ashishsahu@ubuntu:~$ 
```

Figure-3: Result of addition and allotment of threads.

MPI[8]: MPI is abbreviated as Message Passing Interface system developed for distributed and parallel computing in distributed memory model. In the MPI model number of computers connected through a network and distribution middleware that enables computers to coordinates and to share the resources. MPI is a communication protocol for parallel programming. It has library of routines used to create parallel program in C and FORTAN languages. A set of libraries exist for using standard on HPC. User can write program in C/C++, FORTRAN which can be portable used to communicate between source and receiver through routines.

Blocking call: Blocking call blocks (stops) the executions until the operation finishes. If someone sends a message to the receiver but the message don't pass until the receiver acknowledge that message. It executes synchronously. Real life example of Blocking call: Phone call, when one is trying to connect to the other one through a phone call one can't send its message until other picks that phone call.

Non-Blocking call: Non-Blocking call don't block (stops) the execution and pass that message to the receiver without their acknowledgement. And sender not waiting for receiver response sender continues their other program after just send a message. It execute asynchronously.

Real life example of Non-Blocking call: When sender sends a text message to the receiver there is no need to acknowledge that message. Sender can execute other works after send a message.

MPI communicates to other computers with the help of MPI Routines. MPI routines are basically a technique to pass the information from one end to other end.

MPI routines are[9]: MPI_Send, MPI_Reduction, MPI_Bcast, MPI_Scatter, MPI_Recv, MPI_Gather, MPI_Allgather, etc.

Send – MPI_send() function is used to send data from one node to other node it has six arguments that are void*data: Address of data
Int count: length of data has to send
MPI_datatype: Type of data like int, float
Int destination: World rank (other node) where to send data
Int tag: tag should be match to send or receive data from one node to other node
MPI_comm: It is communicator

Receive – MPI_Recv() function is used to receive data from one node to other node it has seven arguments that are void*data: Address of data
Int count: length of data has to send
MPI_datatype: Type of data like int, float
Int source: World rank (other node) where from send data
Int tag: tag should be match to send or receive data from one node to other node
MPI_Comm: It is communicator
MPI_Status*: It is status for only used in receive function

Broadcast: Broadcast function is used to send a same message (red block) to different nodes from one node.

Real life ex: If one person wants to send their marriage invitation card to their friends that is broadcasting invitation.

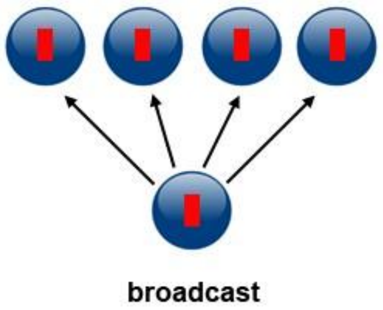


Figure-4: Broadcast routine[10].

Scatter: Scatter function is used to send different – different messages to different – different nodes from one node.
Real life ex: University result of every student.

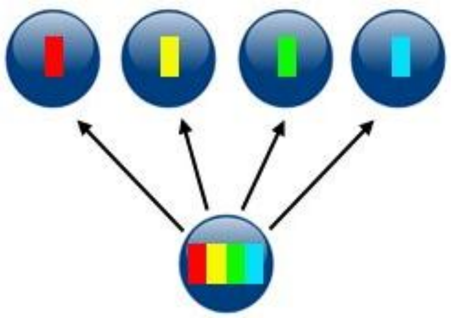


Figure-5: Scatter routine[10].

Gather: Gather function is used to gather different messages from different nodes to one node.

Real life ex: Collecting everybody's data in a single database

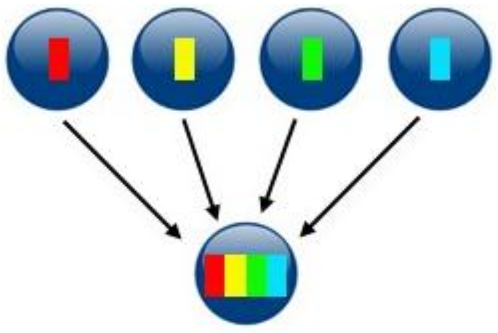


Figure-6: Gather routine[10].

Reduction: Reduction function is used to make reduction of that data using arithmetic operations.
Here sum of all node is 16 that reduces number of nodes to one single node.

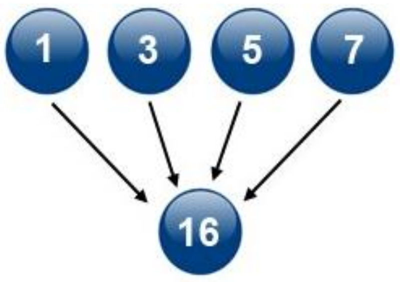


Figure-7: Reduction routine[10].
All gather: All gather function is used to gather all messages from different nodes to every nodes.

Real life ex: There are 3 friends everybody has different units of syllabus but everyone needs whole 3 units so they sends everybody's copies to each other that they have all 3 units to everyone.

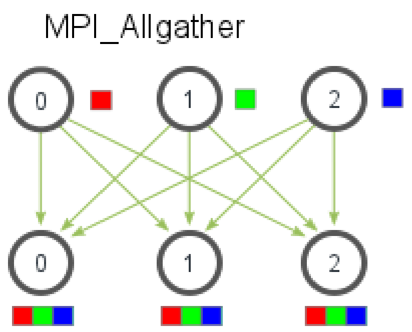


Figure-8: Allgether routine[11].

MPI executes as processes to multi computers that have their own memory and interconnected through a network

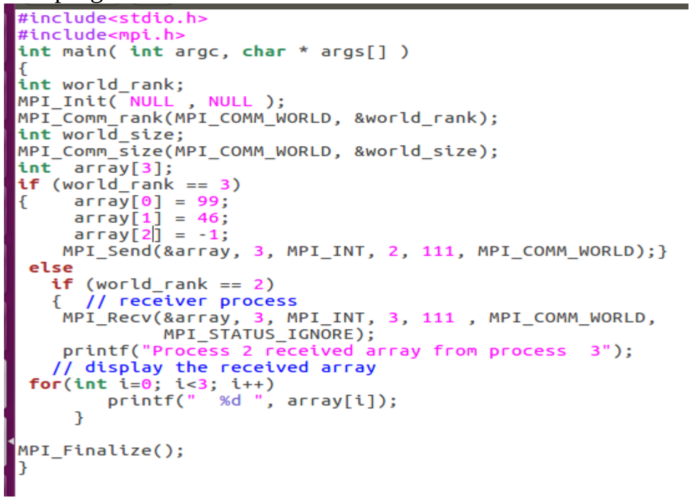
Steps to install MPI on ubuntu ..
sudo apt-get install mpich
if its installed properly then the command
man mpicc will show the manual for mpicc

To compile MPI code -
mpicc <filename.c>

To run MPI code -
mpirun -np x ./a.out

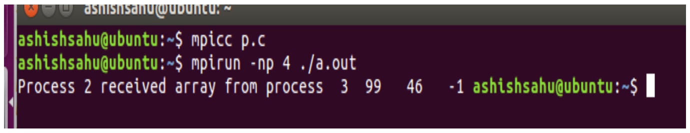
x (it is the no. of processes that we spawn) you can use any number like4, 5 , 6 etc

MPI program[12]:



```
#include<stdio.h>
#include<mpi.h>
int main( int argc, char * args[] )
{
int world_rank;
MPI_Init( NULL , NULL );
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
int  array[3];
if (world_rank == 3)
{    array[0] = 99;
     array[1] = 46;
     array[2] = -1;
    MPI_Send(&array, 3, MPI_INT, 2, 111, MPI_COMM_WORLD);}
 else
   if (world_rank == 2)
   {  // receiver process
    MPI_Recv(&array, 3, MPI_INT, 3, 111 , MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process 2 received array from process  3");
   // display the received array
 for(int i=0; i<3; i++)
        printf("  %d ", array[i]);
     }

MPI_Finalize();
}
```

Figure-9: Code sending data using send routine MPI.



```
ashishsahu@ubuntu:~$ mpicc p.c
ashishsahu@ubuntu:~$ mpirun -np 4 ./a.out
Process 2 received array from process  3  99   46   -1 ashishsahu@ubuntu:~$
```

Figure-10: Result of send routine.

CUDA: CUDA is abbreviated as Compute Unified Device Architecture[13]. It is a platform and application of programming interface model created which is used parallel computing schemes to faster the performance of the system. It is created by the Nvidia Company. CUDA is essentially for C/C++ with few extensions that allow one to execute function on GPU using many threads in parallel. It is general purpose graphics processing unit. It is a platform which used GPU for general purpose and parallel computing. CUDA execute the program with number of threads on GPU with high speed computing.

For high speed computing we have to use Graphical Processing Unit (GPU) instead of only using Central Processing (CPU) Unit because GPU has more number of cores than CPU and compute at high speed and better throughput comparatively CPU. GPU can forked one program into number of threads and executes those threads parallelly within less time. CPU is Host and GPU is Device

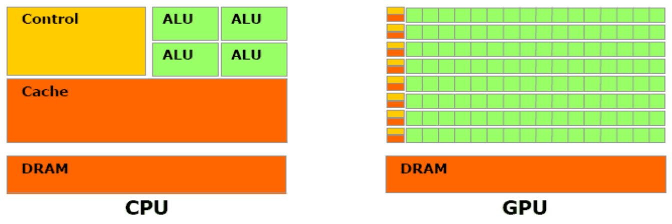


Figure-11: Comparison of CPU and GPU Cores[14].

Architecture of GPU[15]: GPU is divided into grids and every grids again divided into block now Blocks have number of threads to executes.
It is basically GRIDS >> BLOCKS >> THREADS
In example picture consider Grid 1 here consider block (1, 1) has 3 dimensional spaces 16 Threads in one block, 6 block in Grid1 so grid one has total number of threads are 6x16 = 96 Threads. So it has many applications[17].

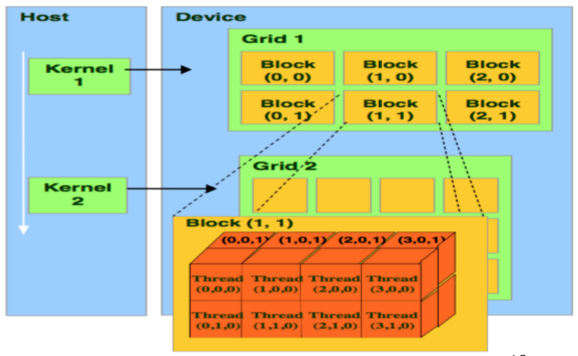
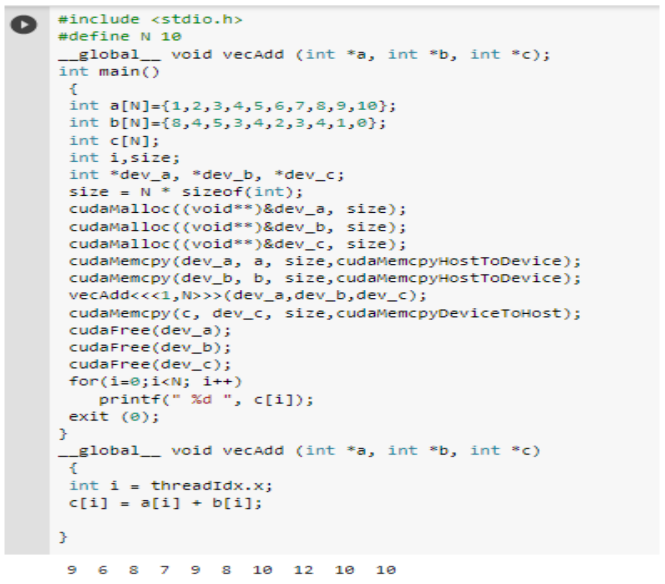


Figure-12: GPU's Kernel, Host and Device[16].

CUDA code execution:
Allocating memory spaces for data on Host by using cudaMalloc.
Copy data from CPU to GPU using cudaMemcpy HostToDevice.
CPU initiates the GPU compute kernel (programming for GPU)
GPU's CUDA cores execute the kernel in parallel.
Copy back the result from GPU to CPU cudaMemcpy DeviceToHost.

CUDA program:



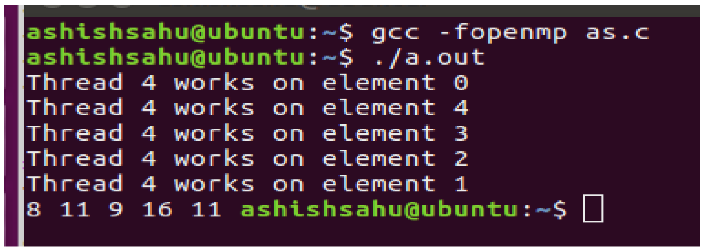
```
#include <stdio.h>
#define N 10
__global__ void vecAdd (int *a, int *b, int *c);
int main()
 {
 int a[N]={1,2,3,4,5,6,7,8,9,10};
 int b[N]={8,4,5,3,4,2,3,4,1,0};
 int c[N];
 int i,size;
 int *dev_a, *dev_b, *dev_c;
 size = N * sizeof(int);
 cudaMalloc((void**)&dev_a, size);
 cudaMalloc((void**)&dev_b, size);
 cudaMalloc((void**)&dev_c, size);
 cudaMemcpy(dev_a, a, size,cudaMemcpyHostToDevice);
 cudaMemcpy(dev_b, b, size,cudaMemcpyHostToDevice);
 vecAdd<<<1,N>>>(dev_a,dev_b,dev_c);
 cudaMemcpy(c, dev_c, size,cudaMemcpyDeviceToHost);
 cudaFree(dev_a);
 cudaFree(dev_b);
 cudaFree(dev_c);
 for(i=0;i<N; i++)
    printf(" %d ", c[i]);
 exit (0);
}
__global__ void vecAdd (int *a, int *b, int *c)
 {
 int i = threadIdx.x;
 c[i] = a[i] + b[i];

}
```

```
 9   6   8   7   9   8   10   12   10   10
```

Figure-13: Code and result of adding two sets of data using CUDA.

Results and discussion

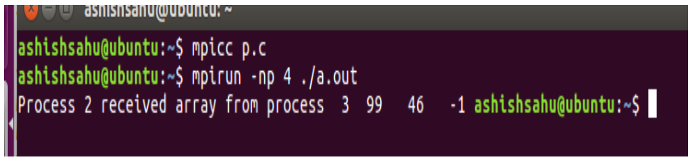
Result of OpenMP: It work on shared memory architecture based on threads of program to execute and it is directive based programming. OpenMP program written to add two arrays for 4 slave threads and one master slave. So with the use of OpenMP one program forked with number of threads and execute those program parallelly to enhace system performance.



```
ashishsahu@ubuntu:~$ gcc -fopenmp as.c
ashishsahu@ubuntu:~$ ./a.out
Thread 4 works on element 0
Thread 4 works on element 4
Thread 4 works on element 3
Thread 4 works on element 2
Thread 4 works on element 1
8 11 9 16 11 ashishsahu@ubuntu:~$
```

Figure-14: Result of addition and allotment of threads.

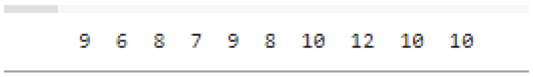
Result of MPI: It is work on distributed memory architecture based on processes of program to execute and it is message passing style, MPI used multi-cored interconnected computer to executes program to enhance the performance of system.



```
ashishsahu@ubuntu: ~
ashishsahu@ubuntu:~$ mpicc p.c
ashishsahu@ubuntu:~$ mpirun -np 4 ./a.out
Process 2 received array from process  3  99   46   -1 ashishsahu@ubuntu:~$
```

Figure-15: Result of send routine.

Result of CUDA: It is work on GPU in threads, comparatively high throughput, CUDA allocate the memory location for program in GPU and execute those program and send back to CPU. If someone thought it takes more time if we send codes on GPU from CPU and send back to CPU after execution but wait it not takes time to copying and if we use millions of transaction to executes then it gave high throughput because GPU uses numbers of cores to execute program that's why it is very fast.



```
9  6  8  7  9  8  10  12  10  10
```

Figure-16: Result using CUDA.

Conclusion

Traditionally HPC space has mostly multi core and cluster computing models but GPUs have revolutionized the HPC space by providing speed-ups up to factors of 100. To leverage the true computational power of GPU, a programmer has to well aware of the underlying GPU architecture, the best algorithms for data decomposition and the best practices of the GPU programming model involved. GPU clusters sharing the workload among multiple GPUs and communicating through MPI are the most commonly used model in modern data centers for today's' HPC workloads. Newer GPUs are being designed by vendors and are opening new avenues of research.

References

1. Caulfield, B. (2009). What's the Difference Between a CPU and a GPU?. NVIDIA. URL: https://blogs. nvidia. com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu.
2. Stephan Perkins (2021). GPGPU: Definition, Differences & Example. https://study.com/academy/lesson/gpgpu-definition-differences-example.html 21/04/2021
3. Shivprasad Koirala (2021). Concurrency vs Parallelism. https://www.codeproject.com/Articles/1267757/Concurrency-vs-Parallelism 23/04/2021
4. Gabriel southern (2021). Main difference between shared memory and distributed memory. https://stackoverflow.com/questions/36642382/main-difference-between-shared-memory-and-distributed-memory 28/04/2021
5. Arnab Chakraborty (2021). What is OpenMP?. https://www.tutorialspoint.com/what-is-openmp 30/04/2021
6. Ashwini Ms. and Bhugul M. (2017). Parallel computing using OpenMP. *IJCSMC*, 6(2).
7. Blume, H., von Livonius, J., Rotenberg, L., Noll, T. G., Bothe, H., & Brakensiek, J. (2008). OpenMP-based parallelization on an MPCore multiprocessor platform–A performance and power analysis. *Journal of Systems Architecture*, 54(11), 1019-1029.
8. LLNL (2021). Message passing interface. https://hpc-tutorials.llnl.gov/mpi/ 01/05/2021
9. MPI (2021) Message passing interface. https://wstein.org/msri07/read/Message%20Passing%20Interface%20(MPI).html 02/05/2021
10. LLNL (2021). Collective communication routines. https://hpc-tutorials.llnl.gov/mpi/collective_communication_routines/ 05/05/2021
11. Wes Kendall (2021). MPI scatter gather all gather. https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/ 08/08/2021
12. Andrey V Tabakov and Alexey A Panzikov (2019). Using relaxed concurrent data structure for contention minimization multithreaded MPI program. *Journal of Physics*, 1399(3), 033037
13. Fred OH (2021). What is CUDA?. https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/ 13/05/2021
14. Sikdar (2021). Sandipan Sikdar. https://medium.com/@sikdar_sandip/cpu-vs-gpu-1e1264204920 15/05/2021
15. Ghorpade, J., Parande, J., Kulkarni, M., & Bawaskar, A. (2012). GPGPU processing in CUDA architecture. arXiv preprint arXiv:1202.4347.
16. CUDA (2021). Thread and block heuristics in cuda programming. http://cuda-programming.blogspot.com/2013/01/thread-and-block-heuristics-in-cuda.html 20/05/2021
17. Samel, B., Mahajan, S., & Ingole, A. M. (2016). Gpu computing and its applications. *International Research Journal of Engineering and Technology*, 3(04).