On the Comparative Performance of Parallel Algorithms on Small GPU/CUDA Clusters

N. P. Karunadasa & D. N. Ranasinghe
University of Colombo School of Computing, Sri Lanka
nishantha@opensource.lk, dnr@ucsc.cmb.ac.lk

***Abstract*—CUDA programmed GPUs are rapidly becoming a major choice in high performance computing and there are a growing number of applications which are being ported to the CUDA platform. However much less research has been carried out to evaluate the performance when CUDA is integrated with other parallel programming paradigms. We have developed a general purpose matrix multiplication algorithm and a Conjugate Gradient algorithm using CUDA and MPI. In this approach, MPI works as the data distributing mechanism between the GPU nodes and CUDA as the main computing engine. This enables the programmer to connect GPU nodes via high speed Ethernet without special technologies and also it helps the programmer to see the separate GPU nodes as they are and execute different components of a program in several GPU nodes.**

I. Introduction

CUDA is gaining its position as the choice of high performance computing[1] community gradually and there are growing amount of work being carried out around the world[2][3]. MPI has been the choice of high performance computing for more than a decade and it has proven its capability in delivering higher performance in parallel applications. CUDA and MPI use different programming approaches but both of them depend on the inherent parallelism of the application to be effective. CUDA runs on the GPU and the GPU is a magnitude order faster than the common CPU.But the the performance of the GPU depends on the application which is executed by the GPU. There are several factors dictate the processing speed of the GPU. One of them are the number of cores it has. The GPU, unlike the CPU uses less number of registers to store data temporally while they are processing. Therefore, GPU can uses more registers to data processing and that is one of the major reason to have many execution cores inside the GPU. In addition, the GPU has a faster memory bandwidth between device memory and the processing cores. However, any given algorithm won't gain the performance which are showed in the GPU specification because only the algorithm those are specially designed for the GPU environment will only enjoy the performance of the GPU. So that if the CUDA application has real parallel components, even a single GPU card is capable of delivering significant performance[4]. MPI typically runs on CPU clusters so that it does not have the support of hardware level performance acceleration like what CUDA has. However using MPI, we can execute different components of different programs in different CPUs in the cluster whereas we can only run one kernel at a time inside the GPU while we are using CUDA[5]. In other words MPI is excellent in distributing the parallel components within a parallel environment and CUDA has mastered in executing parallel components exploiting threads. In this paper we will describe how we integrate these capabilities of both programming approaches and how we can achieve superior performance in general purpose applications.

In this research work, we have experimented CUDA+MPI programming approach with two well-known algorithms and we have showed how we can achieve higher performance by means of

N. P. Karunadasa is a student of the University of Colombo School of Computing and following the Bachelor degree in Computer Science using MPI as computation distributing mechanism and CUDA as the main execution engine. However this CUDA+MPI programming paradigm is not the ideal approach for all parallel applications because there are instances where this programming approach delivers poor performance. In the Strassen algorithm, we have shown that effectively we can use CUDA+MPI approach whereas Conjugate Gradient algorithm, is less effective.

NVIDIA SLI technology can be used to connect multiple GPUs that are in one computer and as of the latest release of the CUDA sdk, all those SLI connected GPU cards can only be seen as one single GPU by the programmer. But we can connect GPU cards in different computers using ethernet and exploit CUDA+MPI model so that it enables the user to see different GPUs in different computer as separate processing engines. Hence the programmer can execute different kernels in one application on different GPUs at the same time.

II. CUDA AND MPI

A. CUDA

CUDA (Compute Unified Device Architecture)[7] is the programming language provided by NVIDIA to run general purpose applications on NVIDIA GPUs. The CUDA incorporates an Application Programmer Interface, a runtime, couple of higher level libraries and a device driver for the underline GPU.

B. MPI

MPI provides a standard set of subprogram definitions which allow parallel programs to be written using a distributed memory programming model.to allow more than one process to perform computations on a given set of data copies of this data must be sent to any process which requires it (to be saved on that process's memory). This is referred to as message passing.

III. ALGORITHMS AND IMPLEMENTATIONS

A. Strassens algorithm

Strassen's algorithm for matrix multiplication is an O(n$^{2.83}$) efficient approach. We consider two matrices *A* and *B* and the *A*, *B* matrices are divided in to 4 equal sized matrices creating 8 sub matrices of size n/2 if the size of the original matrices is n. The 7 Strassens Equations[9] are applied on above sub matrices creating 7 temporary sub matrices of size n/2.

$$P1 = (A11 + A22) * (B11 + B22) \tag{1}$$

$$P2 = (A21 + A22) * B11 \tag{2}$$

$$P3 = A11 * (B12 - B22) \tag{3}$$

$$P4 = A22 * (B21 - B11) \tag{4}$$

$$P5 = (A11 + A12) * B22 \tag{5}$$

$$P6 = (A21 - A11) * (B11 + B12) \tag{6}$$

$$P7 = (A12 - A22) * (B21 + B22) \tag{7}$$

The temporary sub matrices are used to calculate 4 sub matrices of result *C*.

$$C11 = P1 + P4 - P5 + P7 \tag{8}$$

$$C12 = P3 + P5 \tag{9}$$

$$C21 = P2 + P4 \tag{10}$$

$$C22 = P1 + P3 - P2 + P6 \tag{11}$$

When parallelizing above multiplication using a divide and conquer approach following tasks are done by the root master.

- Creation of *A* and *B* matrices.
- Scattering *A* and *B* in to 8 sub matrices.
- Do addition and subtractions on calculating *P1...P7* sub matrices
- Sending the added or subtracted sub matrices to the slave to do only the multiplication, while keeping sub matrices to do one multiplication locally.
- Do the multiplication on local sub matrices calculating say, *P7*.
- Receive the *P1...P6* sub matrices from the slave.
- Solve the 4 equations to calculate *C11..C22*, doing additions and subtractions.
- Gather the *C11...C22* sub matrices creating *C*.

The tasks of the slave would be,to receive the two added or subtracted matrices from master and recursively apply Strassens algorithm by becoming a sub master and do the multiplication using conventional method on the sub matrices. Finally the result is sent back to the master.

The matrix multiplication part of the algorithm is delivered to the GPU in each node because the GPU is capable of high performance matrix multiplication[10]. We implemented the GPU based matrix multiplication using basic CUDA language but the data transfer overhead between the GPU memory and the CPU memory will reduce the overrall performance of the application. However the data which is copied to the GPU memory is copied again to the cache of the each execution core cluster in the GPU. So it enhances the performance in a manner that it hides the latency between GPU memory and the host memory.

B. Conjugate Gradient method

Conjugate Gradient method can be recommended over simple Gaussian elimination if matrix *A* is very large and sparse. Theoretically the Conjugate Gradient algorithm will yield the solution of the system Ax=b in at most n steps. In practice however the algorithm is used as an iterative method to produce a sequence of vectors converging to the solution.

In this algorithm, the loop is the most compute intensive part and we were not able to find a single entity which is independent of other components of the algorithm in order to execute on the GPU using CUDA. Therefore small computation parts were transformed into CUDA and executed on the GPU. Because of the data dependency among computations, MPI root node requires to gather data and send the new data to slave nodes inside every cycle of the loop.

IV. Results and analysis

We have executed these programs in a CPU cluster with 6 nodes and on a 2 node GPU cluster. In CPU cluster there are 6 nodes each of which has two 3.0 GHz Intel Pentium 4 processors with 2GB RAM. In the GPU cluster there are two nodes where one node has a 2.4 GHz Intel Quad Core processor with 3GB RAM and the other one

```
Input  x ,A, b, M, e, f
r  ← b - Ax
v ← r
c ← (r, r )
for k=1 to M do
        if (v, v)^{1/2} < f then exit loop
        z ← Av
        t ← c/(v, z)
        x ← x+ tv
        r ← r-tz
        d ← (r, r)
        if d < e then exit loop
        v ← r+(d/c)v
        c ← d
        output k ,x ,r
end do
```

Fig. 1. Conjugate Gradient algorithm

has a 3.0 GHz Intel Dual Core processor with 1GB RAM.The quad core processor node is named as *A* and the dualcore processor node is named as *B*. Each GPU node has a NVIDIA 8800 GT graphic card and with 768MB graphic memory. We executed each program 10 times in each scenario and calculated average execution time.

A. Results

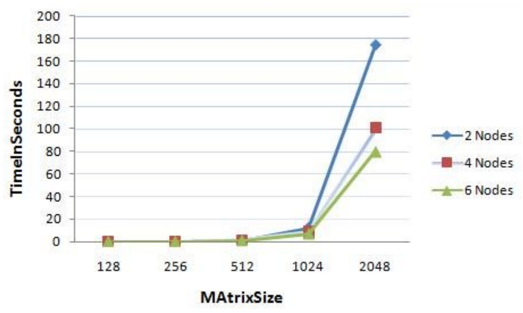

Fig. 2. Strassen's on CPU cluster with MPI

According to figure 3, Strassen's algorithm performs better in the *A* computer than the *B*. this is because the *A* computer has higher hardware specification than *B* regardless GPU card it hosts. The host processor and the host memory plays a critical role in GPGPU computing, because the GPU is not capable of doing anything by itself

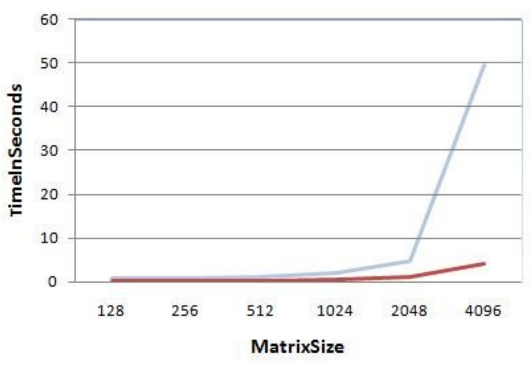

Fig. 3. Strassen's on each GPU node

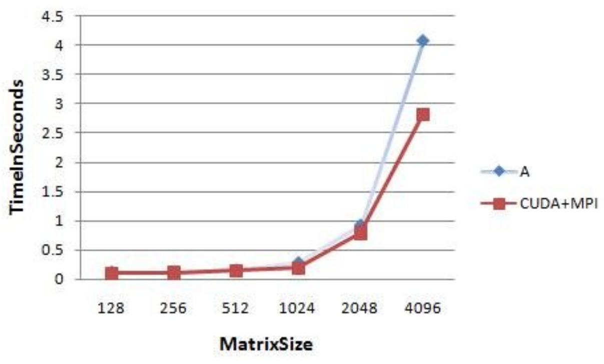

Fig. 4. Strassen's on single GPU node vs CUDA+MPI

but it needs the help of the CPU and the host memory in order to work. Figure 4 shows that how CUDA+MPI based Strassen's algorithm performs against single GPU based node. In this scenario, we have used the *A* computer as the single GPU node and algorithm was implemented with CUDA language. In that figure, CUDA+MPI program does not show significant performance gain because the *B* computer is comparatively slow in computing as mentioned earlier.

When comparing figure 2 and figure 4, the CUDA+MPI based Strassen's algorithm out performed the CPU cluster/MPI based one. The CUDA+MPI based program has more then 20 times faster than the six node CPU cluster version. This is to be expected due to the real power of the GPU in parallel processing.

In contrast the Conjugate Gradient algorithm as figure 1 based on a one huge loop and there are data

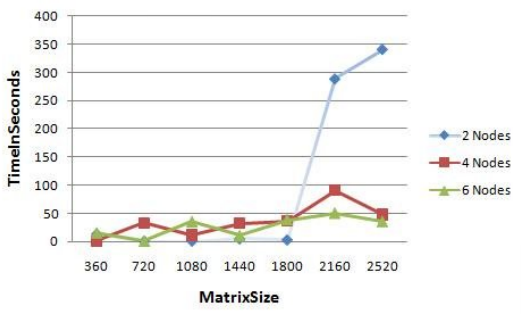

Fig. 5. Conjugate Gradient method in CPU cluster

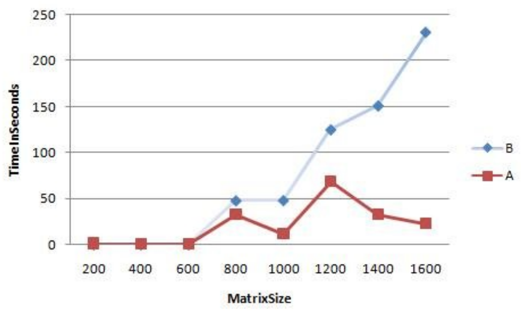

Fig. 6. Conjugate Gradient method on each GPU node

dependencies inside the loop. Therefore according to the firure 5 and 6,the algorithm doesn't give predictable performance improvement in the CPU clustering environment as well as on the GPU cluster. The most significant fact is that the normal MPI based implementation delivers higher performance than the CUDA+MPI version as the figure 7. By carefully examining the program, we note that this is because of the lack of second level parallelism in the Conjugate Gradient method. Therefore CUDA is not especially usefull in this scenario. Infact the GPU can handle some parts of the computations but because of the memory latency between the GPU memory and the CPU memory, it won't give much performance gain

B. Analysis

First of all we should consider the major difference between above two algorithms. In Strassen, it has a parallel implementated component within the matrix multiplication. Matrix multiplication requires considerable amount of processing power.

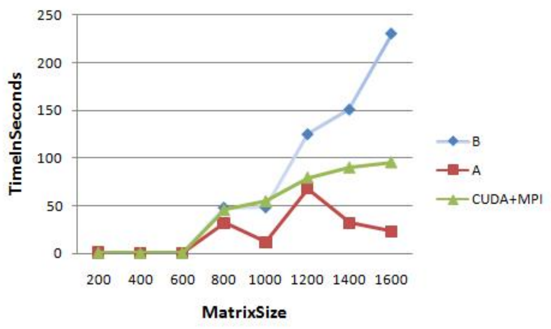

Fig. 7. MPI Conjugate Gradient method vs CUDA+MPI version

But in Conjugate Gradient method, after distributing data among nodes, it does not have a parallel component which requires considerable amount of processing and also does have the data dependency. In other words after expressing parallelism using MPI, Starssen has another level of parallelism which can easily be expressed using CUDA but the Conjugate Gradient algorithm does not have such second level parallelism. Therefore Strassen's gives higher performance than normal MPI version when it is combined with CUDA+MPI. But the Conjugate Gradient method does not deliver much higher performance than normal MPI when it executed with CUDA+MPI.

Threfore When we consider achieving performance using CUDA+MPI, there are two things that we should consider

- The first is, does the algorithm have two levels of parallelism?. If the algorithm has second level parallelism then the second factor should be considered.
- The second factor is that whether the second level parallelism have some compute intensive part that can be processed with in a GPU? If it has not a considerable amount of work which cannot be easily handled by the GPU, then there will not be much higher performance by executing that second level parallel components with CUDA as expected.

In our GPU cluster, there are two nodes each having one GPU card. These two machines have different hardware specifications except that they have identical GPU cards. Therefore we can connect heterogeneous computers those having GPU cards by means of CUDA+MPI. This is important because if we want to build a GPU cluster for high performance applications, we can get it done by adding GPU cards to existing heterogeneous CPU cluster.

V. Conclusion

Using CUDA+MPI we can accelerate parallel applications which have certain inherent parallelism characteristics. In such cases, the performance enhancement is more than that of by a MPI cluster. CUDA+MPI approach has also highlighted the fact that it will help to build high performance computing clusters at low cost. Finally we hope to enhance the capabilities of CUDA+MPI programming approach by introducing automatic load balancing in a cluster in which load is dynamically varying.

References

[1] H. Kasim , V. March1, R. Zhang, S. See.Survey on Parallel Programming Mode.Proceedings of the IFIP International Conference on Network and Parallel Computing (IFIP 2008)
[2] G.Vasiliadis, S.Antonatos, M. Polychronakis, E. Evangelos, P.Markatos, S. Ioannidis..Gnort: High Perormance Network Intrusion deection Using Graphics Processors.Institute of Computer Science, Foundation for Research and Technology Hellas,Greece.
[3] P. Harish, J. Narayanan.Accelerating large graph algorithms on the GPU using CUDA.Center for Visual Information Technology. International Institute of Information Technology Hyderabad, India.
[4] S. tomov, J.Dongarra.M. Baboulin..Towards Dense Linear Algebra for Hybrid GPU Accelerated Manycore Systems.
[5] NVIDIA CUDA.Compute Unified Device Architecture Programming Guide. Version 2.
[6] A. Richardson, A. Gray.Utilisation of the GPU architecture for HPC.EPCC, The University of Edinburgh
[7] T.R. Halfhill.Parallel Processing With CUDA, Nvidias High-Performance Computing Platform Uses Massive Multithreading. International Journal on Microprocessors,01/28/08-01.
[8] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, International Journal of Supercomputer Applications, Vol. 8, No. 3/4,1994.
[9] J. Green. Strassens Fast Multiplication of Matrices Algorithm and Spreadsheet Matrix Multiplications.
[10] S.Dinkins.performance and scalability analysis on paralleled matrix multiplication on shared memory.August 3, 2007