International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 11 | Nov -2016 www.irjet.net p-ISSN: 2395-0072

Framework of rCUDA: An Overview

Mohamed Hussain[1], M.B.Potdar[2], Third Viraj Choksi [3]

[1]1 Research scholar, VLSI & Embedded Systems, Gujarat Technological University, Ahmedabad, India
[2] Project Director, Bhaskaracharya Institute for Space Applications and Geo-Informatics(BISAG), Gandhinagar, India
[3] Project Scientist, Bhaskaracharya Institute for Space Applications and Geo-Informatics(BISAG), Gandhinagar, India

---

Abstract - *CUDA is a programming module developed and produced by NVIDIA in 2006.This programming model allows the programmer to use the power of GPU in general purpose computation. However using the GPU in the system has some drawbacks, where it will increase the acquisition cost and power consumption, in addition, it requires more space to install the new hardware. To overcome those drawbacks of using GPU some visualization techniques are used. Those visualization techniques allow sharing the GPU resource between the machines in the cluster.by implementing one of those techniques the client machine (don't have GPU processor) will able to access the remote GPU resources on the server machine. Some of those techniques enable the client application to distribute its loads between all the GPUs in the cluster which increase the throughput of the system. Using those visualization techniques will reduce the number of the GPUs in the cluster. This will reduce the energy consumption, maintenance and upgrade the system it will be easier, only add anew GPU machine to the cluster.*
In this paper, we will explain rCUDA framework which is one of the visualization techniques used for remote GPU accessing, also the architecture and the overhead presented when applying rCUDA in the system.in the demo, live application is used to present the overhead of the system and compere between CUDA and rCUDA. The reason of using rCUDA because of its fidelity comparing to other visualization techniques and its ability to share the GPU between the clients by using different context like multiplexing.

Key Words: GPU, CUDA, GPU virtualization, rCUDA.

1.INTRODUCTION

GPU-accelerator computing consists of GPU processor to reduce the execution time of applications as it has an ability to execute massive part of the code in parallel. This type of processing is required in applications which have a high execution time and large number of literation such as some equations which used in finance, chemical, physics, computational fluid dynamic, computational algebra and image analysis.

However, using GPU in the system presents several drawbacks such as, increase power consumptions, large space to add the new hardware and high acquisition cost. In addition to that GPU, utilization is relatively low in the system. The desktop computers for gaming consume around 500W which is a non-negligible amount of energy. In computers with GPU, the power consumption will increase around 30% [2]. To overcome the drawbacks of using the GPU processor in any system, remote GPU virtualization mechanisms can be used. The virtualization allows the applications which are executed on a computer that doesn't have GPU processor to use the remote GPU installed in another machine. In another word this mechanism allows sharing the GPU processor installed in one machine between all machines connected to cluster. This will increase the overall GPU utilization by allowing all computers connected to nodes to share the GPU resources, thus reducing the drawbacks in the system.

There are several visualization frameworks which allow the applications installed in client PC to access the remote GPU such as GridCuda, DS-CUDA, gVirtuS, vCUDA , GViM and rCUDA .

In this paper, we are going to use the rCUDA framework due to the fact that rCUDA intercepts all the CUDA calls, in addition to its ability to track the state of the memory area used by the application in GPU processor , also it has a good fidelity in sharing GPU comparing to other frameworks. [4]

2. BACKGROUND ON CUDA PROGRAMMING

CUDA stands for computing unified device architecture, it is an extension of the C programming language developed and introduced by NVidia in 2006.Using CUDA programming module allows the programmer to take advantage of the massive parallel computing power of NVidia graphic cards, in order to use it for general purpose computation.[6] in this programming module the programmer will divide the code into two parts ,the first part of the code will be executed normally on the CPU while the second part is executed on the GPU. The programmer decides which part of the program will execute in GPU and which one will execute in CPU.

The designs of CPU and GPU are significantly different which make the way of executing the instructions totally different. The CPU consist of four or eight cores, but the GPU has hundreds or

thousands of cores. That makes the GPU able to execute hundreds or thousands of threads or processes in parallel.

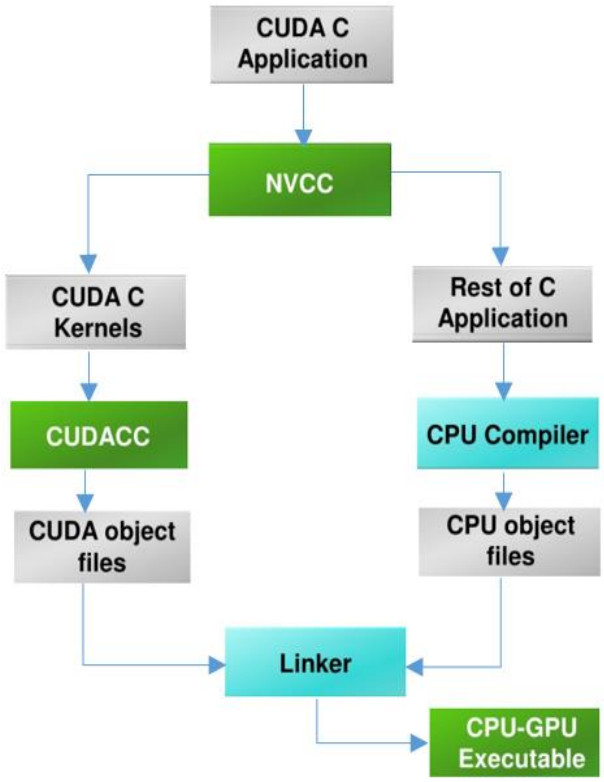

Fig -1: CUDA compilation flow [6]

CUDA programming langue provides C/C++ library in addition to a compiler (NVCC) to compile the code. The source code will be a combination of CPU and GPU codes. At the compilation time the NVCC compiler will automatically separate the code in the source file and send CPU part to common C/C++ compiler and GPU part to CUDACC [5]. The compilation flow of the code is as illustrated in the figure below Figure (1).

3. RCUDA FRAMEWORK

"rCUDA (remote CUDA) is a middleware which enables sharing remote CUDA-Compatible devices concurrently and transparently"[1]. rCUDA framework is a client-server distributed architecture consisting of two software modules one is the client middleware and the second is the server middleware. The client middleware has wrappers to CUDA runtime which is responsible for passing the API calls from client application to remote server middleware. The server middleware runs on the machine which hosts the GPUs (the server can host one or more GPUs).the server will execute the API calls received from client and send back the result. The sever always work with more than one client, for that different GPU contexts like Multiplexing are implemented.

Client-server communication in network will be through TCP/IP protocol stack or InfiniBand Verbs API. TCP/IP protocol is always with Ethernet connection and infiniBand networking for high-speed links. In virtual machines where client and server are both in the same machine the client can't directly access the GPU, It should invoke wrappers to be able to access the GPU in the server side, which requires a virtual network on the host machine.

rCUDA provides full compatibility supported by CUDA. It can also implement all functions in the CUDA run time and Driver API, although it has some limitations when it's used for graphics interoperability.
By implementing rCUDA in the system the execution time will increase by less than 4% [1] which relatively small considering the cost and power consumption when a separate GPU processor is installed in each machine. Another to reduce the impact of remote CUDA is by increasing the throughput of the system where rCUDA framework enables the applications to distribute their loads between all the machines that have GPUs in the cluster, this method makes the overhead value negligible.

To distribute the load between the nodes the rCUDA has been integrated with SLURM scheduler, this integration provides an overall reduction in the execution time of job batches between 25% and 45%, depending on exact composition of the job batch. [7].

4. RCUDA ARCHITECTURE

rCUDA Architecture consists of two parts server side and client side, these parts communicate through the network, to enable client machine in rCUDA to connect to server:

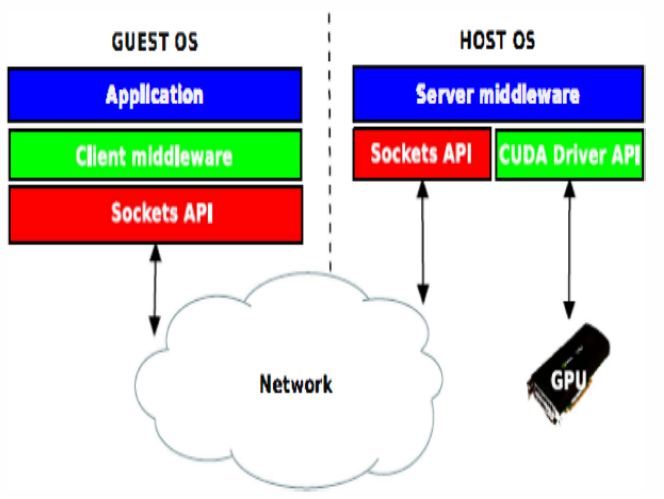

Fig -2: rCUDA Architecture [3]

- **The client middleware**: consists of a collection of wrappers these wrappers will be responsible for forwarding the API calls from the application in the client computer to server middleware and returning back the result at run time.
- **The server middleware:** will be installed on the server that hosts the GPU. The server will receive the requests from client, interpret and execute them. To enable the server to handle different requests from different users the server middleware will apply the GPU multiplexing techniques as mentioned previously.

The communication between the client and server in rCUDA framework, will be through the network. rCUDA uses TCP sockets with some customized application-level protocols to make the communication between client-server middleware's more effectively.

5. rCUDA DEMO

The purpose of application in the Demo [1] is to present the overhead of using rCUDA in the system. To do so, two types of image filters are implemented as explained below:

5.1 Color Image to Grayscale Conversion:

In any computer system to present any colored image, we use four parameters called RGBA to represent only one pixel of the image. Where 'R' indicates how much red color used in the pixel, 'G' for green color, 'B' for blue and 'A' for the opacity of the picture.

To convert the colored image to grayscale image the formula which recommended by NTSC (National Television System Committee) equation (1) is used. (Note: the value of Alpha in the equation is ignored)

$I = 0.299 * R + 0.587 * G + 0.114 * B$ (1)

According to this formula, the authors developed the program and installed it on the client machine. The application will be execute twice, once by local CUDA and the second time by rCUDA.

5.2 Image Blurring

Image blurring is a type of filtering to reduce the edge content of the colors and make transition from one color to another color very smoothly, to accomplish that the relation between the pixels and their neighbors should be defined, for example if we have the pixel B we want to bluer it, we have to consider the values of the pixel and it's neighbors.

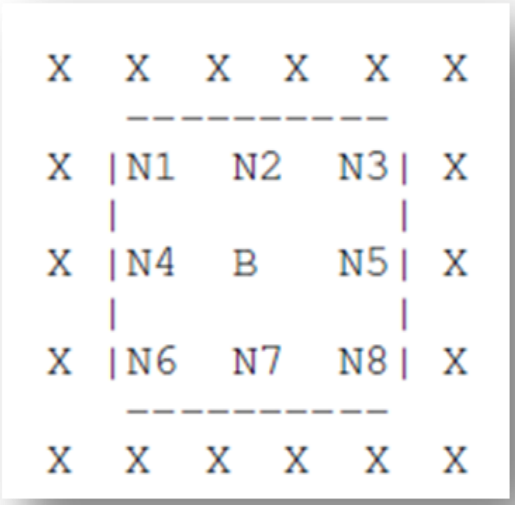

Fig -3: Matrix representing the images pixel [1]

And for that we use this formula (2):

$$blur(B) = B * db + \sum_{i=1}^{8} N[i] * d[i] \qquad (2)$$

6. Analysis of Demo:

In the demo two types of filters are applied for each image. First, the image will convert to grayscale using CUDA (local GPU), and then using rCUDA (remote GPU). This will be for the top right part of the picture. In the bottom right part the image will blur using blur filter. This filter also will convert the image using CUDA (local GPU) and then rCUDA (remote CUDA).the conversation time for CUDA and rCUDA it will store separately.

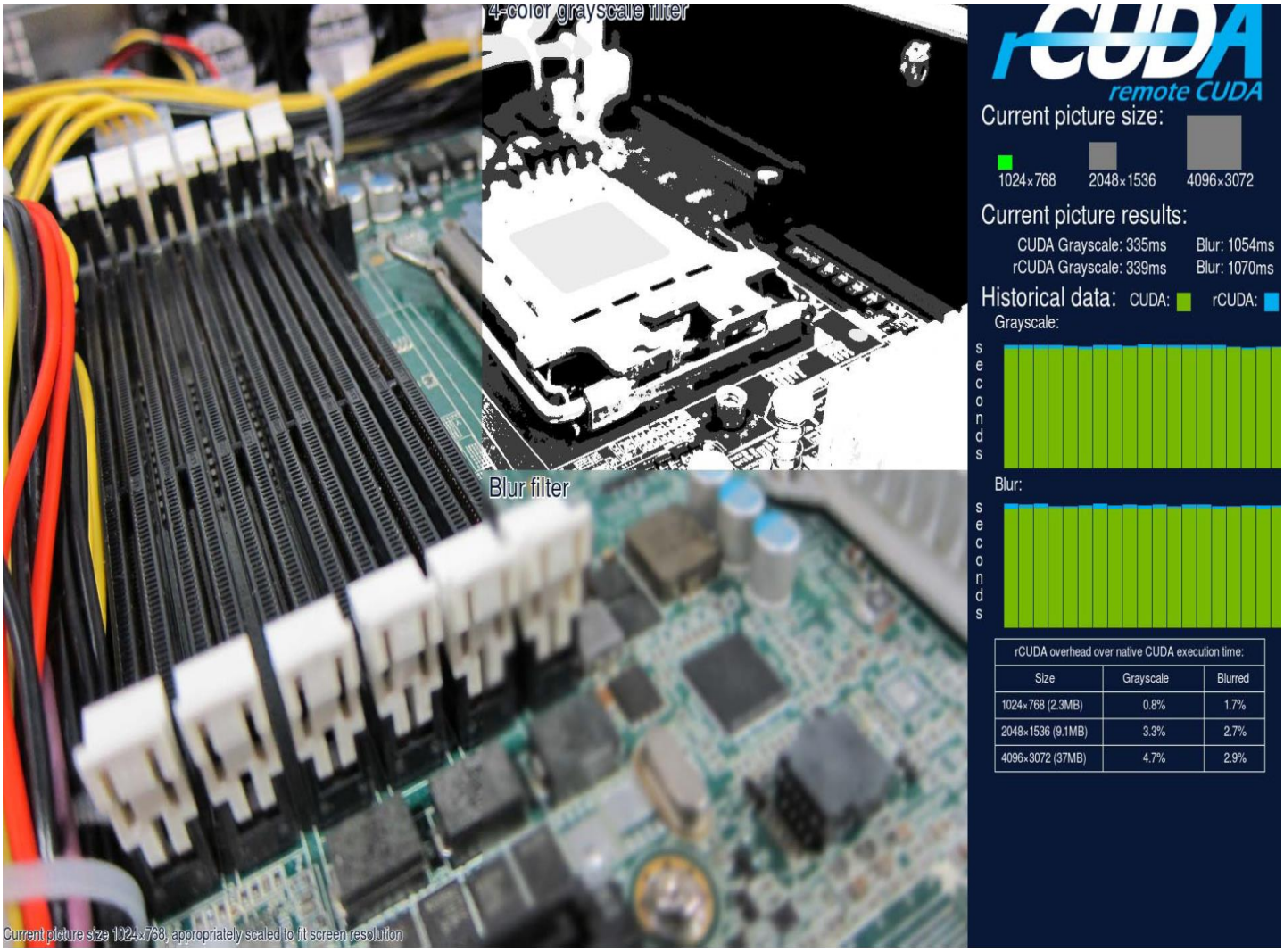

Fig -4: screen shot of the Demo [1]

The application presents some details on the right side of the screen. It shows the time requires to convert the image to grayscale and blur the image.

The green bar shows the average time of converting last 20 images using CUDA (local GPU).the blue part of the bar it shows the overhead presented when implement rCUDA framework on the system. The table on the right bottom of the screen represents the size of each image and the percentage of the overhead of each presented by each filter.

Figure (4) presents the screen shot of the demo, in the demo 245 images were used each image has three different sizes: 1024x768 (2.4MB), 2048x1536 (9.4MB), and 4096x3072 (37.7MB).

7. PERFORMANCE RESULTS

The performance of rCUDA influences by three main parameters:

1. Data transferring: implementing rCUDA increase the time required to transfer data, where the client PC will send the data through the network. Sending the data through network introduce more delay on the system. The value of this delay depends on the bandwidth of the network.

2. Computation time: the time employed by CUDA kernel in GPU to execute is same for CUDA and rCUDA, only the overhead comes from transferring the data in the network.

3. CUDA calls: when the application sends the call using rCUDA the call pass through the network to reach the rCUDA server. For that, the overhead presented by rCUDA depends on the network latency.

The result in the Chart (1) it shows the average of executing the last ten images in local and remote GPU. The maximum Relative Standard Deviation (RSD) observed was 0.077. This one achieved with converting an image to the grayscale of size 1024x768. The maximum overhead achieved also by the grayscale filter with maximum image size used 4096x3072.

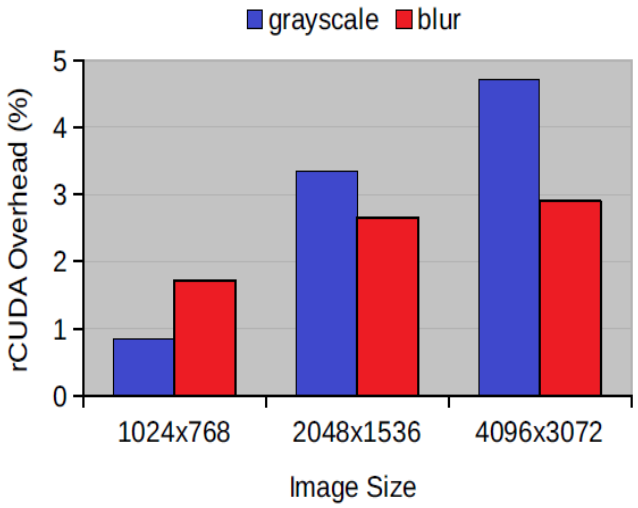

Chart -1: CUDA overhead over CUDA when running grayscale and blur filters [1].

From figure (5) the overhead of converting smallest image size using blur filter is higher compering to the remaining bigger size. This is because the blue filter has larger number of calls to CUDA API larger than grayscale one. The time require to send the calls through the network represent higher overhead than time spent in computations.

8. BENEFITS OF USING REMOTE GPU VIRTUALIZATION

There are many benefits introduced when we implement the rCUDA, those benefices may reflect on the performance of the system or the acquisition cost:

1. Reduce power consumptions: the power consumption will reduce due to reduce the number of GPUs in the cluster.

2. Acquisition cost: by sharing the GPU processor between the machines is not required to install GPU processor in each machine which will reduce the cost of the system.

3. More GPUs are available for a single application: where rCUDA framework enable the application to distribute the load between all GPUs in the cluster.

4. Overall GPU utilization is also increased.

5. Upgrading the system it will be easier: to increase the output of the system we only need to add a new GPU to the cluster.

6. More than one machine can access the same GPU concurrently: rCUDA implement multiplexing technique to allow more than one machine to access the GPU.

4. CONCLUSION AND FUTURE WORK

Using GPU processor in the machines it will come with some drawbacks, like increasing in the cost of the system, the power consumption and the space required to install the new hardware. To overcome all those drawback a visualization technique is implemented in the system. This visualization technique enable the GPU resource to be shared between all the devices in the cluster.

In this paper the rCUDA framework are used. The reason of using rCUDA it's fidelity in using GPU, ability to intercepts all CUDA calls and multiplexing the GPU resource between the clients.

For future work, we are going to implement this system in one of embedded boards, which are using Linux environment. Implement this framework in the embedded systems it will give the system ability to use the GPU processor, without requirement to use embedded board has GPU processor.

6. REFERENCES

[1]C. Reaño, F. Pérez and F. Silla, "On the Design of a Demo for Exhibiting rCUDA," Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on, Shenzhen, 2015.

[2]J. Duato, A. J. Peña, F. Silla, R. Mayo and E. S. Quintana-Ortí, "rCUDA: Reducing the number of GPU-based accelerators in high performance clusters," HPCS and Simulation(HPCS), 2010 International Conference on, Caen, 2010.

[3]M. S. Vinaya, N. Vydyanathan and M. Gajjar, "An evaluation of CUDA-enabled virtualization solutions," Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on, Solan, 2012.

[4]F. Silla, J. Prades, S. Iserte and C. Reaño, "Remote GPU Virtualization: Is It Useful?" 2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), Barcelona, 2016.

[5] Y. F. Huang and W. C. Chen, "Parallel Query on the In-Memory Database in a CUDA Platform," 2015 10th International Conference on P2p, Cloud and Internet Computing (3PGCIC), Krakow, 2015.

[6] www.nvidia.com

[7] http://rcuda.net/index.php/what-s-rcuda.html