The International Journal of Engineering And Science (IJES)
||Volume|| 2 ||Issue|| 01||Pages|| 250-253 ||2013||
ISSN: 2319 – 1813 ISBN: 2319 – 1805

Advanced Trends of Heterogeneous Computing with CPU-GPU Integration: Comparative Study

Ishan Rajani[1], G Nanda Gopal[2]
1. PG student at Department of Computer Engineering, Noble Group of Institution, Gujarat, India
2. Asst. Prof. at Department of Computer Engineering, Noble Group of Institution, Gujarat, India

----------**Abstract**----------
Over the last decades parallel-distributed computing becomes most popular than traditional centralized computing. In distributed computing performance up-gradation is achieved by distributing workloads across the participating nodes. One of the most important factors for improving the performance of this type of system is to reduce average and standard deviation of job response time. Runtime insertion of new tasks of various sizes to different nodes is one of the main reasons of Load unbalancing. Among the several latest concepts of Parallel-Distributed Processing CPU-GPU Utilization is focused here. How the ideal portion of the CPU can be utilized for GPU process and visa-versa. This paper also introduces the heterogeneous computing work flow integration focused on CPU-GPU. The purposed system exploits the coarse-grain warp level parallelism. It is also elaborated here that by using which architectures and frameworks developers are racing in the field of heterogeneous computing.

Index Terms: *Heterogeneous Computing, Coarse-Grained warp level parallelism, standard deviation of job response time*

Date of Submission: 28th December, 2012 ⟺ Date of Publication: Date 20th January 2013

I. Introduction

The goal of proper work utilization in emerge techniques of multi core processing is one of the most important aspects from the massively parallel systems (MMP). Emerging parallel-distributed technology has picked its highest progress in processing power, data storage capacity, circuit integration scale etc. in last several few years but still it doesn't satisfy the increasing need of advanced multi core processors. As we have chosen participation of CPU ideal source in the activities of GPU, for this GPU computing has emerged in recent years as a viable execution platform for throughput oriented applications or codes at multiple granularities by using several architecture developed by different GMA manufacturers.

In general GPUs started its era as independent units for code execution but after that initialization phase of graphics unit several developments has done for CPU-GPU integration. Various manufacturers have developed thread level parallelism for their graphics Accelerators. For emerging a powerful computing paradigm General-Purpose computing on Graphics Processing Units (GPGPU) has been enhanced to manage hundreds of processing cores [6][13]. It helps to enhance performance of operations by mostly focusing on Single instruction multiple data (SIMT) models which has concept very similar to single instruction multiple data (SIMD) architecture of Flynn's classification. For this NVIDIA has mentioned architecture named unified graphics computing architecture (UGCA), here on the basis of that concept, and by the researches done on recent decades we have proposed architecture of that concept with entire work flow.

II. Related Work

With the concept of general purpose graphics units, manufacturer of GMA like NVIDIA, Intel, ARM and AMD etc. has developed specified architecture for this integration. For example, AMD/ATi GPU developed Brook+ for general purpose programming and, NVIDIA has developed Compute Unified Device Architecture (CUDA) which provides greater programming environment for general purpose graphics devices.

GPGPU programming paradigm of CUDA increases performance, but still it doesn't satisfies extreme level scheduling of tasks allocated to GPU [12]. For establishing such kind of situation a better

way is to exploit a technique which utilizes ideal GPU recourses which will reduce the complete runtime of the it's processing. For upgrading their source utilization process recently other developers also released some integrated solutions for above mentioned system. Some most popular of them are Intel's Sandy Bridge, AMD's Fusion APUs, and ARM's MALI.

This paper proposes Heterogeneous environment for CPU-GPU integration. CPU-GPU Chip integration offers several advantages than traditional systems like, reducing communication cost, proper memory resource utilization of both because rather than using separate memory space it also focuses on the shared memory mechanism, warp level parallelism for faster and efficient performance[5] [7]. Warps can be defined as the groups of threads; most of the warp structures include 32 threads into one warp structure [7]. Researches on warp mechanism for threading also noticed that for efficient control flow execution on GPUs via dynamic warp formation and large warp micro architectures, in which a common program counter is used for execution of all the threads of a single warp together [5]. The scheduler selects warps which is ready to execute, issues the next instruction to the active threads of that warp.

III. Design

As we have discussed CUDA is used to establish virtual kernel between distinguished sources of CPU-GPU it is also mentioned in given architecture, these kernels will use preloaded data after decomposition from the memory repositories like buffer memory or temporary cache memory. The key design problem is caused by the fact that the computation results of the CPU side are stored into the main memory that is different from the device memory. To overcome this problem, in our architecture preserve two different memory addresses in a pointer variable at a time. An efficient design architecture of the proposed UGCA architecture is shown in figure1.

Mainly UGCA contains two phases in its workflow. First phase of process includes task decomposition module and mapping of all decomposed tasks also it manages the distribution ratio to the kernel configuration information. And second phase is designed to translate the Parallel Thread Execution (PTX) code into the LLVM code. Here, PTX is pseudo assembly language code used in CUDA. As shown in fig. 1 second procedure is designed to translate PTX code into the LLVM intermediate representation. On GPU device, runtime system passes the PTX code through the CUDA device driver, which means that the GPU executes the kernel in the original manner using the PTX-JIT compilation.

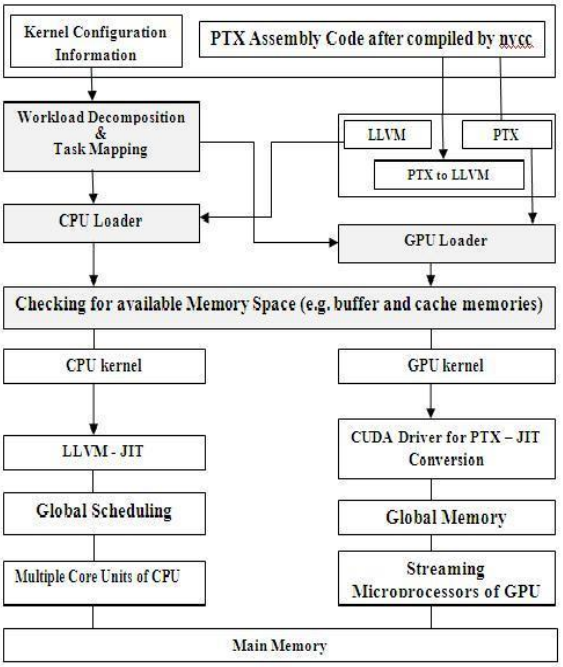

Figure 1.Unified Graphics Computing Architecture

UGCA uses PTX translator provided for converting PTX instructions into LLVM instruction register. LLVM instruction register is used for a kernel context. Proposed system has the advantage that initialization of GPU, memory checking and, PTX to LLVM compilation are performed in parallel.

IV. Workflow

In the first phase of UGCA architecture, Nvcc compiler translates code written in CUDA, into PTX, then graphics driver indicates CUDA compiler which translates PTX to LLVM which can be run on processing core. The most important functionality of Workload distribution module is it generates two additional execution configurations for each (CPU and GPU). Then for performing further operation this module delivers the generated execution configuration, to the CPU and GPU loaders.

As shown in figure 1 top module involves all the configuration information of both CPU and GPU kernels and basic PTX assembly code which will further converted into LLVM code for intermediate representation. The input of workload distribution module is the kernel configuration information and the output specifies two different portions of the kernel space, here dimension of grid is used for

workload distribution module. Proposed work distribution can split the kernel according to the granularity of thread block. It also determines the amount of the thread blocks to be detached from the grid considering the dimension of the grid

and workload distribution ratio. We are focusing on coarse warp level parallelism, so in this condition sometimes number of threads doesn't follow preferred bock size, bound tests should be inserted to kernel [5]. Otherwise threads may access memory out of index. For declaring block boundries some preprogrammed semantics are available in cuda library.

```
_global_ void kernel(int n, int* x )
{
  int idx =  threadidx.x  +  blockidx.x*blockdim.x;
  if(i < n)
  {
    //  core programming
   }
}
```

How many block should be covered in a block can be declared by inbuilt schematics of CUDA library like threadidx and blockidx. In above mentioned code, threadidx.x indicates the number associated with each thread in a block. Block size of thread can be mapped with number of threads multiples of 32 like 192, 256 etc. Similarly block size can be declared for two dimension, three dimension and so on. Convention mentioned above is very useful for two dimensional matrices [9]. Here blockidx.x refers to the label associated with a block grid. The blockidx.x is similar to the thread index except it refers to the number associated with the block. Lets take an example for elaborating the concept of blockdim.x, suppose we want to load an array of 10 values in to a kernel using two blocks, each one having size of 5 threads in each. A major problem to do this kind of operation is that, our thread index may only goes for 0 to 4 as mentioned threads in one block. Here to overcome this problem we can use a third parameter given in CUDA stated as threaddix.x. This holds the size of the block, like in our case threadidx.x = 5.

A program written for CUDA application should assign memory spaces in the device memory of the graphics hardware. Then their memory addresses are used for the input and output data. In this UGCA model, data can be copied between the host memory and the dedicated memory on the device. For this, the host system should preserve pointer addresses pointing to the locations in the device memory. The key design problem is caused by the fact that the
Computation results of the CPU side are stored into the main memory that is different from the device memory.

V. Comparison

As we have discussed for above architecture, CUDA has been provided a massive parallelism for GPU as well as non GPU programming also [1]. Now let us compare with other parallel programming environment according to their performance. CUDA environment is only available to NVIDIA GPU device, whereas the Open CL standard may be implemented to run on any vendor's hardware, such as traditional CPUs, and GPUs [2] [3]. One of the most effective advantages of OpenCL compare to CUDA is its programming portability. But while comparison is subject of matter then CUDA gives more speedups by using its thread based warp level parallelism [4]. Because of the multi-vendor compatibility of OpenCL makes enough difference between both of these that pose significant challenges towards realizing a robust CUDA to OpenCL source to source translation [2]. Figure 2, shows comparison of both CUDA and OpenCL for execution of various algorithms like fast Fourier transformation, double-precision matrix multiplication [9], molecular dynamics, and simple 3D graphics programs.

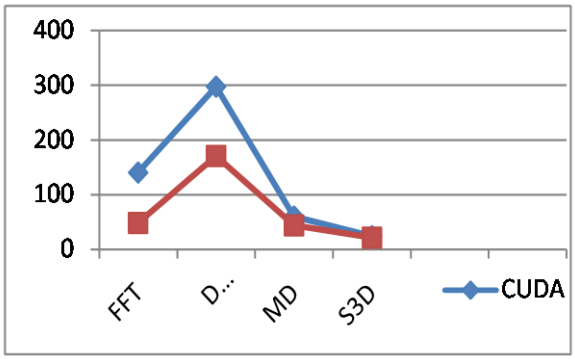

Figure 2. GFLOPS required to execute different algorithms on CUDA and OpenCL

Performance of various environments according to required GFLOPS is analyzed by results. NVIDIA Tesla C2050 GPU computing processors perform around 515 GFLOPS in double precision calculations, and the AMD FireStream 9270 peaks at 240 GFLOPS [14][19]. In single precision performance, Nvidia Tesla C2050 computingprocessors perform around 1.03 TFLOPS and the AMD FireStream 9270 cards peak at 1.2 TFLOPS.

VI. Conclusion

The paper has introduced architecture of the concept which is widely adopted for coarse grained warp level parallelism and also going to be adopted as the advanced concept of warp level parallelism. For example Success of Proposed

UGCA framework can be exampled with NVIDIA GeForce 9400 GT device. After the drastic success of mentioned graphics device. This architecture is going to be focused for more future enhancements. We believe the cooperative heterogeneous computing can be utilized in heterogeneous multi-core processors which are expected to include even more GPU cores as well as CPU cores.

As future work, we will first develop a dynamic control scheme on deciding the workload distribution ratio, warp scheduling, also plan to design more effective thread block distribution technique considering data access patterns and thread divergence technique.

References

[1] Ziming Zhong, Rychkov, V., Lastovetsky, A. "Data Partitioning on Heterogeneous Multicore and Multi-GPU Systems Using Functional Performance Models of Data-Parallel Applications", Cluster Computing (CLUSTER), 2012 IEEE International Conference on , vol., no., pp.191-199, 24-28 Sept.

[2] Sathre, P.; Gardner, M.; Wu-chun Feng; "Lost in Translation: Challenges in Automating CUDA-to-OpenCL Translation", Parallel Processing Workshops (ICPPW), 2012 41st International Conference on , vol., no., pp.89-96,10-13,Sept.,2012

[3] Scogland, T.R.W., Rountree B., Wu-chun Feng, de Supinski, B.R.; "Heterogeneous Task Scheduling for Accelerated OpenMP", Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International , vol., no., pp.144-155, 21-25May2012.

[4] Okuyama, T., Ino, F., Hagihara, K., "A Task Parallel Algorithm for Computing the Costs of All-Pairs Shortest Paths on the CUDA-Compatible GPU", Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium, vol., no., pp.284-291

[5] Goli, M., Garba, M.T., GonzalezVélez, H., "Streaming Dynamic Coarse-Grained CPU/GPU Workloads with Heterogeneous Pipelines in FastFlow" High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on , vol., no., pp.445-452, 25-27 June 2012

[6] Manish Arora, "The architecture and Evolution of CPU-GPU Systems for General Purpose Computing " , By University of California, San Diago

[7] Veynu Narasiman, Michael Shebanow, Chang Joo Lee, " Improving GPU Performance via Large Warps and Two-Level Warp Scheduling ", The University of Texas at Austin

[8] Rafiqul Zaman Khan, Md Firoj Ali: "A comparative study on parallel programming tools in parallel distributed computing system: MPI and PVM", Aligarh Muslim University

[9] Ali Pinar and Cevdet Aykanat; "Sparse Matrix Decomposition with Optimal LoadBalancing ", TR06533 Bilkent, Ankara, Turkey

[10] NVIDIA, Nvidia cuda sdks.

[11] N. Bell and M. Garland. Cusp; "Genetic parallel algorithms for sparse matrix and graph conputations", 2010

[12] J. Nickolls and W. Dally. The gpu computing era. Micro, IEEE, 30(2):56-69, april 2010

[13] S. Huang, S. Xiao, W. Feng; "On the Energy Efficiency of Graphics Processing Units for Scientific Computing"

[14] Andrew J. Page: "Adaptive Scheduling in Heterogeneous Distributed Computing Systems, National University of Ireland, Maynooth

[15] Getting Started with cuda http://blogs.nvidia.com/

[16] Nvidia GeForce Graphics Card, http://www.nvidia.com/object/geforce_family.html

[17] Nvidia Tesla GPGPU system, http://www.rzg.mpg.de/computing

[18] The Supercomputing Blog http://supercomputingblog.com/cuda/cuda-tutorial-1-getting-started/

[19] Articles on advanced computing *http://www.slcentral.com/articles/*