BEST: International Journal of Management, Information Technology and Engineering (BEST: IJMITE)
ISSN (P): 2348-0513, ISSN (E): 2454-471X,
Vol. 5, Issue 06, Jun 2017, 49-60

RUNNING APPLICATION USING NEURAL NETWORK ON CPU-GPU SYSTEM

SUMAN GOYAT[1] & A. K. SONI[2]

[1]*Research Scholar, Computer Science and Engineering, Sharda University, Greater Noida, Uttar Pradesh, India*

[2]*Professor, Computer Science and Engineering, Sharda University, Greater Noida, Uttar Pradesh, India*

ABSTRACT

Scheduling decisions to assign some applications to CPU and others to GPU, at local PC location is very crucial for optimum utilization of devices such as CPU and GPU, if they are available in a PC or Laptop. If we allow the operating system to make global scheduling decisions and assign some applications to a slower device, we may both increase overall system throughput +t and decrease individual application runtimes. We shall use an application developed using Neural Network, to be executed on a system having CPU and GPU. This is implemented in Cuda-C language. It shows the performance improvement drastically, when major portion of application is run on GPU and few steps are executed on CPU. Cuda C has the functions to handle it. It is an extension of C – Language.

KEYWORDS: Scheduling Decisions, Historical Data, Artificial Neural Network, Weighted Averages, Cuda– C Language, Cudamalloc and Cudamemcpy

INTRODUCTION

GPU (Graphical Processing Unit) is becoming very popular now days due to its highest performance, as compared to using CPU only in the systems. CPU was used in older systems, for the execution of tasks and processing the information and GPU is included, to improve performance of the system. GPU computing has become very popular in recent years, as an execution platform for highest throughput, in various application oriented approaches. A GPU can be defined as a single chip processor, used for processing two and three dimensional transform and a various other operations in computer graphics. In the case of GPU, processors are implemented as moderately parallel platform and it contains several ALUs (Arithmetic Logic Units), several pixel pipelines (around 16), several vertex pipelines (around 6) and fast access to a relatively small amount of, on card memory (32 to 512 MB), as in case of single CPU systems. Modern 3-D graphic cards contain one or more of these GPUs which improve overall throughput of the system.

Graphics processing units are used in many fields of computation, due to its effectiveness. Artificial neural network is one of the main contexts which offer some attractive solutions to many real world problems. ANN is an area of computation that has been largely unexplored, in comparison to other fields of GPU applications. We have introduced CPU with GPU, both used in single systems by, discussing our work in the area of ANN. We have used one of the major problems of miscarriage, throughout pregnancy cases. We collected various informations, regarding these cases from a nursing home. We summed up various parameters relating to these conditions, which are favoured for the chance of miscarriage. Then, we examined how the GPU offers us, a parallel platform where, large number of calculations were done simultaneously, and using shared components such as last level caches and memory controllers that are evolved to improve the performance of CPU-GPU systems [6]. We then looked at the collaborative CPUGPU execution schemes, to see how it will impact on the available solutions, in artificial intelligence. Lastly, we discussed the future work directions and research

opportunities, for CPU-GPU systems [2].

The first attempt, at using GPUs for non-graphics computations, used corner cases of the graphics APIs. To use graphics APIs for general purpose computation, programmers mapped program data carefully to the available shaded buffer memory and operated the data via the graphics pipeline. There was limited hardware support for general purpose programming; however, for the correct workload, large speedups were possible [3]. The initial success attained for a few non-graphics workloads on GPUs, had prompted vendors, to add explicit hardware and software support. This enabled a somewhat wider class of general purpose problems, to execute on GPUs. NVIDIA's CUDA [10] and AMD's CTM [1] solutions added hardware, to support general purpose computations and exposed the massively multi-threaded hardware, via a high-level programming interface. The programmer is given an abstraction of a separate GPU memory address space, similar to CPU memory, where data can be allocated and threads launched, to operate on the data. The programming model is an extension of C, providing a familiar interface to non-expert programmers. Such general-purpose programming environments for GPU programming have bridged the gap between GPU and CPU computing and led to wider adoption of GPUs, for computing applications [8].

Our approach is different from using algorithm that assigns jobs which are to be run on CPU and others, on GPU. We are using a single job i.e. an application whose execution steps are broken to be executed on CPU and GPU. In our case Kernel doesn't need to do any assignment of jobs. In order to understand our approach, it is a good idea to look briefly at the algorithm steps, for assignment of jobs.

Algorithm Steps

Algorithm provides the whole computation that has been done over a platform, where a large number of operations were performed on a data set at a time using both GPU and CPU. Historical information available for execution time of particular application helps us to get solution, whether to allocate on GPU or on CPU. Even if an application would run faster, assigned to a particular device, if there were enough applications ahead of it, in the queue for that device, it may finish faster assigned to the slower device because, that device is free in comparison with queue, for the faster processor. It needs certain calculations on predicted execution time, on a device and the time it will take to finish, on each of the available devices.

Algorithm is based on the available information, about the parameters used in our example. The steps are as follows:

- Four parameters are taken as input. X1 is the age in years/100, X2 is the no. of previous births, X3 is the no. of induced abortions, and i.e. the deliberate interruption of pregnancy, X4 is the spontaneous abortion.
- Neural network is normalized with the random weights being normalized as per the Expected output (EO). This data is the previous data that was available from a nursing home. EO states the % for normal pregnancy, e.g. if EO is 15, it means 15% are the chances, of having normal pregnancy.
- A new data was given as input to 4 parameters. The program calculates the O/P value, which tells us the % chances of getting pregnant.
- We used ANN (Artificial Neural Network) logic to do our job. We took 20 lines of historical data and ran it in parallel on GPU, so that the time of processing them serially on CPU was reduced almost 20 times, when run on

GPU. We are used C++ Language, on windows on 64 bit processor PC, having NVIDIA GPU.

As an application finishes, update was made on the historical database, with the runtime information, calculating the average runtime and the standard deviation and repeat the algorithm from the beginning. The scheduling algorithm described above continues to improve, as more data would be entered into the historical database and each application was penalized, at most once when it was assigned to a slower device, in order to build the database. Due to the fact that, the application runtimes were averaged into, the previous runtime for a device, outlying points that could be caused by unknown factors to the scheduler.

APPLICATION RUN TIME PREDICTION

If an application has been assigned to a device at least once, with a given set of inputs and the same application was subsequently ran, with the same input size, or the scheduler has been assigned minimum twice to a given device with different inputs and was again ran with another different input, the scheduler makes the prediction based on a linear least-squares calculation, using the input sizes as one parameter and the previous runtimes as another. In current CPU/GPU architectures, the applications that launch kernels run themselves on the CPU and the scheduler that runs also uses a CPU thread to coordinate the scheduling. When scheduling kernels, to run on the CPU, the kernels utilize standard operating system threads and therefore contribute, to the overall CPU usage.

LATEST IN GRAPHICS PROCESSING UNIT

NVIDIA Pascal, is the world's most advanced GPU architecture delivering company, which gives truly game-changing performance, innovative technologies and immersive next-generation virtual reality features. Pascal provides its performance, to ensure with brilliant game play with captivating sights and sounds, it's a whole new way to gaming applications [14]. Pascal-powered graphic cards, provide superior performance and powerful efficiency, which was built to be using ultra-fast Fin FET and supporting DirectX™ 12 features, to deliver the fastest, smoothest and the most power-efficient gaming experiences, in today's life. NVIDIA GeForce GTX 1080, the flagship Pascal GPU, also features high-bandwidth GDDR5X technologies for incredible gaming experiences. The next version of this GeForce® GTX 1080 Ti is NVIDIA's new flagship gaming graphic cards, based on the NVIDIA Pascal™ architecture that captures most of the real time applicability, to the various applications including Screen display and gaming. In the latest addition to the ultimate gaming platform in graphics, this card was packed with extreme gaming horsepower, next-gen 11 Gbps GDDR5X memories and a massive 11 GB frame buffer as discussed via the developer [14]. Advancements in GPU technology in cars, has helped push self-driving technology, which will also provide self assisting display in the latest version of automobiles, for example Audi Q8, Mercedes Benz etc, in 2018.

NVIDIA and Radeon, are the two competitive of each other in the field of developing latest and fastest graphic cards, for the GPGPU (General Purpose Graphical Processing Unit). The PS4 and Xbox One are the other versions, which were released in 2013, they both use GPUs based on AMD's Radeon processor, with version HD 7850 and 7790 respectively. The GeForce 10 series of cards were under this generation of graphic cards. They were made using the 16 nm manufacturing process, which improves upon previous micro architectures. The Polaris 11 and Polaris 10 GPUs, from AMD were made with a 14 nm process. Their release results in a big increase, in the performance per watt of AMD video cards [10]. In Nvidia's Kepler, line of GPUs was followed by the Maxwell line, manufactured on the same process. 28 nm chips by Nvidia, were manufactured by TSMC, the Taiwan Semiconductor Manufacturing Company that was

manufactured, using the 28 nm process [14]. In comparison with 40 nm technology from the past, the new manufacturing processes, allowed a 20 percent boost in performance, while drawing less power. Pascal was the newest generation of graphic cards by Nvidia, released in 2016 [14].

AMD's Radeon HD 6000 Series cards, released in 2010 and 2011, had different version provide and different performance measures, based on the requirement of the developer. Then, AMD released their 6000M Series discrete GPUs, to be used in mobile devices. The Kepler line of graphic cards, by Nvidia, came out in 2012 and was used in the 600 series, 700 series and 800 series, of graphics cards, by Nvidia [13].

FILTERING TECHNIQUE TO INCREASE PARALLEL COMPUTATION SPEED

If we talk of parallel applications whose performance models were not linear, we may have to use a filtering technique to extract subsets of observations that were close to the query point and show strong linearity. Image filtering is one of the important preprocessing steps, in image processing. In the graphics application based on 3D or 2D images, it is required to filter so that, unwanted information can be neglected. Advancement in the technology has brought development of filtering in both spatial and frequency domain of the images. In Spatial filtering, the filtering operations were performed directly, on the pixels of an image. Spatial filtering includes various techniques like Gaussian blurring, edge detection, denoising, etc. In frequency domain filtering, the process was done, over the distribution of the signal strength. This paper concentrates on accelerating image filtering with Graphical Processing Unit (GPU), such that the speedup and performance of the filtering process could be enhanced. GPU is an efficient way, to accelerate image filtering and uses NVidia's Compute Unified Device Architecture (CUDA) technology, for parallel computing. Traditional CPU can run only a few complex threads concurrently. GPU allows a concurrent execution of hundreds or thousands of simpler threads, in parallel execution of all data sets, to generate solution. Types of filtering includes two filtering techniques, namely, Gaussian blur filter and sober edge detection filter, for grayscale images on GPU using CUDA programming language.

The closeness of data point to the query point in filtering was defined by the distance function of the filter. Since the range and distribution of variables used for run-time predictions are unknown, we need distance function that should normalize distances, with respect to the query point. In addition, the extent to which individual variables influence the run time was also different, as per the distribution and range of the variables [15]. Therefore, the importance of each variable, with respect to the run time should also be incorporated into the distance function, so that, it mainly gives the best of filtering techniques used. For each dimension, whether that is 2D or 3D, a local distance is computed that denotes how distant is the value of a variable is from the corresponding variable in the query point and data pointy, and is normalized to 1. The denominator used in the local distance function represents the range of the variable based on the observed data in the distribution and range measures. The low cost GPU certainly do possesses tremendous processing power, which can be harnessed with suitable parallelized modifications in the conventional algorithms, such that they become suitable for GPU implementation. Although not all algorithms would possesses this inherent ability of parallelization, but those which do possesses this scope of parallelization can be further exploited with the use of GPU‘s. Median filtering is suitable for parallel implementation as its algorithm is quite simple to understand. Also its GPU implementation leads to considerable speedups [15].

ARTIFICIAL INTELLIGENCE AND GPU

Graphics processing units are being considered for many fields of computations, due to their many attractive

traits, for example, ANNs. Artificial neural networks, give attractive solutions to many real world problems. The field of, artificial neural networks, is an area of computation, that has been largely unexplored in comparison to the other fields of GPU application. Because of these facts, the marriage of ANNs and the GPU would seem to be a worth-while area of research; to establish or invalidate particular architectures amenability, to GPU simulation, to provide a fair and unbiased comparison of CPU and GPU execution of these architectures and to establish some general traits, that make ANNs more or less amenable to simulation on a GPU [12].

EVALUATION

Pregnancy Estimation

This was a program in Cuda C. It uses Artificial Neural Network - Back Propagation

// Four parameters were taken as input.

// X1 is age in years/100, X2 is no. of previous deliveries, X3 is no. of

// induced abortions i.e., the deliberate interruption of pregnancy, X4 is spontaneous abortion.

// Neural network was normalized, with the random weights being normalized,

// as per the Expected output (EO). This data was collected from the

// nursing home. EO finds the % probability of normal pregnancy, for

// e.g., if EO is15, it means, 15% chances of getting normal pregnancy.

// A new data was given as input to 4 parameters. The program calculates O/P value,

// which tells us the % chances of getting pregnant.

// We had used ANN (Artificial Neural Network) logic, to do our job. We took 20 lines of historical data

// and ran it in parallel, on GPU so that, time of processing them serially on CPU was reduced almost 20 // times when ran on GPU. We did it using C++ Language, on windows, on 64 bit processor PC,

// having NVIDIA GPU.

// Four parameters were taken as input.

// X1 was the age in years/100, X2, the no. of previous deliveries, X3 being the no. of

// induced operations, X4 being the spontaneous operation.

// Neural network was normalized, with the random weights being normalized

// as per the Expected output (EO). This data was the previous data, available from the

// nursing home. EO says the % in which the pregnancy will be normal.

// e.g. if EO is.15, it means 15% chances of getting normal pregnancy.

// Simulate Neural Network - Back Propagation

// A new data was given as an input to 4 parameters. Program calculates O/P value,

```
// which tells us the % chances of getting pregnant.
#include<conio.h>
#include<stdio.h>
#include<math.h>
#include<iostream.h>
#include<fstream.h>
#include<time.h>
voidNue_Net (float z [] [4], int N)
{
Double Pos_e7;
Double eta=0.9;
Double x1, x2, x3, x4;
Double w15=0.2, w16=-0.3;
Double w25=0.4, w26=0.1;
Double w35=-0.5, w36=0.2;
Double w45=-0.6, w46=0.3;
Double w57=-0.3, w67=-0.2;
Double w05=-0.4,w06=0.2,w07=0.1;
double Tot_w15=0, Tot_w16= 0;
double Tot_w25=0, Tot_w26= 0;
double Tot_w35=0,Tot_w36=0;
double Tot_w45=0,Tot_w46=0;
double Tot_w57=0,Tot_w67=0;
double Tot_w05=0,Tot_w06=0,Tot_w07=0;
double EO;
double i5, i6, o5, o6, i7, o7, e7, e6, e5;
for (int k=0; k<=N-1; k++)
 {
inti=0;
```

```
x1=z[k][0]/100; x2=z[k][1]/100; x3=z[k][2]/100; x4=z[k][3]/100;EO=z[k][4]/100;

clrscr();

printf("\n\n x1=%6.2f, x2=%6.2f, x3=%6.2f, x4=%6.2f",x1,x2,x3,x4);

// getch();

w15=0.2; w16=-0.3;

w25=0.4; w26=0.1;

w35=-0.5;w36=0.2;

w45=-0.6;w46=0.3;

w57=-0.3;w67=-0.2;

w05=-0.4;w06=0.2;w07=0.1;

Do

{

I++;

 i5=x1*w15+x2*w25+x3*w35+x4*w45+w05;

 i6=x1*w16+x2*w26+x3*w36+x4*w46+w06;

 o5 = 1/ (1+POW (2.71828,-1*i5));

 o6 = 1/ (1+POW (2.71828,-1*i6));

 i7= o5*w57+o6*w67+w07;

 o7 = 1/ (1+POW (2.71828,-1*i7));

 e7 = o7*(1-o7)*(EO-o7); //

 e6= o6*(1-o6)*e7*w67;

 e5= o5*(1-o5)*e7*w57;

 w15=w15+eta*e5*x1;

 w16=w16+eta*e6*x1;

 w25=w25+eta*e5*x2;

 w26=w26+eta*e6*x2;

 w35=w35+eta*e5*x3;

 w36=w36+eta*e6*x3;

 w45=w45+eta*e5*x4;
```

```
 w46=w46+eta*e6*x4;
 w57=w57+eta*e7*o5;
 w67=w67+eta*e7*o6;
 w07=w07+eta*e7;
 w06=w06+eta*e6;
 w05=w05+eta*e5;
If (e7 < 0)
 Pos_e7 = -1 * e7;
Else
 Pos_e7 = e7;
I++;
 } while (Pos_e7 > 0.0001);
Print ("\n\n Count = %ld ", i);
printf("\n\n x1=%4.2f, x2=%4.2f, x3=%4.2f, x4=%4.2f",x1,x2,x3,x4);
printf("\n\n Expected Outut(o7) %4.2f : ",EO);
printf("\n\n Actual output (o7)=%4.2f", (float)(o7));
printf("\n\n w15,w16,w25=%4.2f,%4.2f,%4.2f",(float)(w15),(float)(w16),(float)(w25));
getch();
 w15=w15+eta*e5*x1;
 w16=w16+eta*e6*x1;
 w25=w25+eta*e5*x2;
 Tot_w15=Tot_w15+w15; Tot_w16= Tot_w16+w16;
 Tot_w25=Tot_w25+w25; Tot_w26= Tot_w26+w26;
 Tot_w35=Tot_w35+w35; Tot_w36= Tot_w36+w36;
 Tot_w45=Tot_w45+w45; Tot_w46= Tot_w46+w46;
 Tot_w57=Tot_w57+w57; Tot_w67= Tot_w67+w67;
 Tot_w05=Tot_w05+w05; Tot_w06= Tot_w06+w06;
 Tot_w07=Tot_w07+w07;
 w15=Tot_w15/N; w16= Tot_w16/N;
```

```
 w25=Tot_w25/N; w26= Tot_w26/N;

 w35=Tot_w35/N; w36= Tot_w36/N;

 w45=Tot_w45/N; w46= Tot_w46/N;

 w57=Tot_w57/N; w67= Tot_w67/N;

 w05=Tot_w05/N; w06= Tot_w06/N;

 w07=Tot_w07/N;

getch();

 }

// end of for(k) loop

}

voidmain()

{

clock_t t;

constint N=20;

floatz[5][4];

longinti;

charline[80],c;

int j, val,k, n;

floatx[4];

double i5, i6, o5, o6, i7, o7, e7, e6, e5;

double w15, w16;

double w25, w26;

double w35,w36;

double w45,w46;

double w57,w67;

double w05,w06,w07;

ifstream fin;

fin.open("Preg.txt");

clrscr();
```

```
 t = clock();
// time(&t1);
for (k=0;k<=N-1;k++)
 {
 n=0;
fin.getline(line,80);
 j=0;
val=0;
 c=' ';
while (c!=';')
 {
if (c != ',')
 {
val=(line[j]-'0') + val*10;
j++;
 c=line[j];
 }
else
 {
 z[k][n]=val;
 n++;
j++;
val=0;
 c=line[j];
 }
 }
 z[k][n]=val;
}
```

```
for (k=0;k<=N-1;k++)
 {
for (n=0;n<=4;n++)
 {
cout<< z[k][n] <<",";
 }
cout<<'\n';
 }
Nue_Net(z,N);
clrscr();
double x1, x2,x3, x4;
 x1=0.46;x2=6;x3=2;x4=1;
// Classification
 i5=x1*w15+x2*w25+x3*w35+x4*w45+w05;
 i6=x1*w16+x2*w26+x3*w36+x4*w46+w06;
 o5 = 1/(1+pow(2.71828,-1*i5));
 o6 = 1/(1+pow(2.71828,-1*i6));
 i7= o5*w57+o6*w67+w07;
 o7 = 1/(1+pow(2.71828,-1*i7));
 t = clock() - t;
doubletime_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds
// time_t t1,t2;
// time(&t2);
printf("\n\n Classification Output ---------------------------->");
printf("\n\n x1=%6.2f, x2=%6.2f, x3=%6.2f, x4=%6.2f",x1,x2,x3,x4);
printf("\n\n Actual Output with normalized weight(o7)=%6.2f", (float)(o7));
// printf("\n Execution time %9.2f ",difftime(t2,t1));
printf("\nTime taken %.2f seconds to execute \n", time_taken);
getch();
```

```
getch();

} // endofmain()
```

REFERENCES

1. J. T. Adriaens et al. The case for gpgpu spatial multitasking. High Performance Computer Architecture, 2012.
2. S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," IEEE Micro, vol. 31, no. 5, pp. 7–17, 2011.
3. J. Gummaraju, L. Morichetti, M. Houston, B. Sander, B. R. Gaster, and B. Zheng, "Twin peaks: a software platform for heterogeneous computing on general-purpose and graphics processors," in Proc. of the 19th International Conference on Parallel Architectures and Compilation Techniques, Sep. 2010, pp. 205–216.
4. G. Diamos, A. Kerr, S. Yalamanchili, and N. Clark, "Ocelot: a dynamic optimization framework for bulk-synchronous applications in heterogeneous systems," in Proc. of the 19th International Conference on Parallel Architectures and Compilation Techniques, Sep. 2010, pp. 353–364
5. KHRONOS Group, "OpenCL - the open standard for parallel programming of heterogeneous systems," 2010. [Online]. Available: http://www.khronos.org
6. V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," in Proc. of the 37th Annual International Symposium on Computer Architecture, 2010, pp. 451–460.
7. Nvidia, "Cuda Zone," 2009, https://developer.nvidia.com/category/zone/cudazone.
8. G. F. Diamos and S. Yalamanchili, "Harmony: an execution model and runtime for heterogeneous many core systems," in Proc. of the 17th international symposium on High performance distributed computing, 2008, pp. 197–200.
9. W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt, "Dynamic warp formation and scheduling for efficient GPU control how," in Proc. of the 40th Annual International Symposium on Microarchitecture, 2007, pp. 407–420.
10. NVIDIA Corporation. CUDA Toolkit 4.0. http://developer.nvidia.com/ category/zone/cuda-zone.
11. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In Eurographics, 2005.
12. Wikipedia. Graphics processing unit. http://en.wikipedia.org/ wiki/Graphics processing unit, 2005.
13. https://www.revolvy.com/main/index.php?s=Graphics%20processing%20unit.
14. http://www.nvidia.in/graphics-cards/geforce/pascal
15. https://lagunita.stanford.edu/c4x/Engineering/CS149/asset/pa5.pdf