International Journal of Linguistics and Computational Applications (IJLCA) ISSN 2394-6385 (Print)
Volume 2, Issue 1, January - March 2015 ISSN 2394-6393 (Online)

A Framework for Solving Linear Systems of Equations using Openmp and Cuda

Sai Sruthi#1, Arivazhagan N*2

1Dept. of Information Technology, SRM University, Chennai, India.
ammupambalath@gmail.com
2 Assistant Professor, Dept. of Information Technology, SRM University, Chennai, India.
arivazhagan.n@ktr.srmuniv.ac.in

Abstract— Explaining frameworks of linear equations are presumably a standout amongst the most investigative applications of linear algebra. Direct techniques for registering the arrangement of such frameworks can be exceptionally extravagant because of high memory necessities and computational expense. This is a decent motivation to utilize iterative techniques which processes just an estimate of the arrangement. This paper overhauls one of the iterative strategies for direct frameworks of linear systems of equation on a cuda platform and openmp platform. It analyzes how quick the parallel code runs than that of serial code furthermore looks at the cpu and gpu based parallel computing.

Keywords— *Graphics processing unit, Compute unified development architecture, Parallelization, Congugate gradient*

1. Introduction

From physical science and building to macro econometric displaying, explaining extensive straight frameworks of mathematical statements is a typical issue. For substantial direct frameworks, the direct routines are not productive as a result of high memory and computational requests. Iterative routines are to be considered in such cases. These days Graphics Processing Units contain elite numerous center processors fit for high FLOP rates and information throughput being genuinely broadly useful parallel processors. Since the first thought of Mark Harris numerous applications were ported to utilize the GPU for figure serious parts and they get speedups of few requests of extent contrasting with comparable executions composed for ordinary CPUs.

As of now there are a few models for GPU processing: CUDA (Compute Unified Device Architecture) created by NVIDIA, Stream created by AMD and new rising standard, OpenCL that tries to bring together distinctive GPU general figuring API executions giving a general system to programming advancement crosswise over heterogeneous stages comprising of both CPUs and GPUs. I utilized the C CUDA augmentation to add to a library that executes iterative direct frameworks solvers. OPENMP, a CPU computing method is also used here.

Parallel machine projects are more hard to compose than consecutive ones, on the grounds that concurrency presents a few new classes of potential programming bugs, of which race conditions are the most widely recognized. Correspondence and synchronization between the distinctive subtasks are regularly a percentage of the best deterrents to getting great parallel system execution. The most extreme conceivable pace up of a solitary program as an after effect of parallelization is known as Amdahl's law.

Amdahl's law states that, if f is the fraction of the code parallelized, and if the parallelized version runs on a p-processor machine with no communication or parallelization overhead, the speedup is given by,

$$\frac{1}{(1 - f) + (f/p)}$$

2. Openmp vs Cuda

2.1. OpenMP

Openmp is a usage of multithreading, a parallel execution plan whereby the expert string (a progression of in-structions executed sequentially) forks a defined number of slave strings and an undertaking is isolated among them. Openmp is essentially intended for imparted memory multiprocessors, utilizing the SPMD model (Single Program, Multiple Data Stream): all the processors have the capacity straightforwardly get to all the memory in the machine, through a sensibly immediate association. Projects will be executed on one or more processors that impart some or the greater part of the accessible memory. The system is ordinarily executed by numerous autonomous strings that impart information, yet might likewise have some extra private memory zones.

2.2. NVIDIA architecture

GPUs fit none of the customary execution models proposed by Flynn scientific classification, since their structural planning is very diverse even from the SIMD execution model. As NVIDIA GPUs are generally accessible and their programming surroundings arrived at a stable adaptation, we chose to spotlight on these architectures. As a physical format, NVIDIA GPUs are composed as Streaming Multiprocessors (SM) with basic scalar processors (SP) on chip. Inside the gadget, strings have the capacity access information from numerous memory spaces:

- Local memory: per-thread, private, for temporary data (implemented in external DRAM);
- Shared memory: for low-latency access to data shared by cooperating threads in the same SM (implemented on chip);
- Global memory: for data shared by all threads of a computing application (implemented in external DRAM).

2.3. *Cuda*

On February 2007, NVIDIA Corporation has discharged the beginning form of the CUDA SDK. It gives two APIs: the C runtime for CUDA which is conveyed through the cudart dynamic library and the CUDA driver API which is conveyed through nvcuda dynamic library. CUDA gives all the method for a parallel programming model with the disposition of two sorts of imparted memory: the on-chip imparted memory that can be impacted by strings of a piece executing on a SM and the worldwide memory got to autonomously by the pieces running on the GPU.

Contrasted and other GPU programming systems, for example, Opencl, the CUDA dialect can be viewed as abnormal state since it needn't bother with all the low level introductions and changes. On the other hand, managing the whole programming structure so as to oversee gadget gets to, memory exchanges between the gadget can even now be a monotonous work, then again the Openmp programming model permits a straightforward and incremental parallelization. Subsequently, we decided to make a source-to-source interpreter for Openmp C code to Cuda.

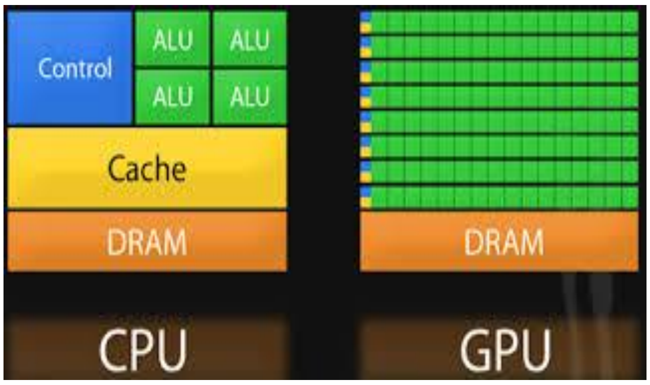

Fig.1: CPU and GPU architecture

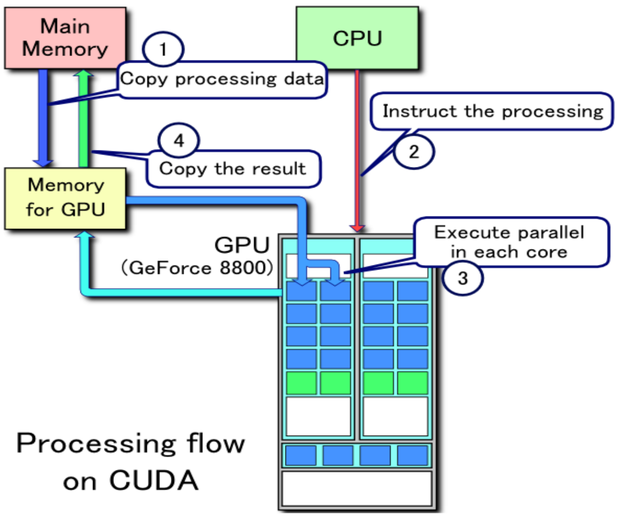

Fig.2: Processing flow of CUDA

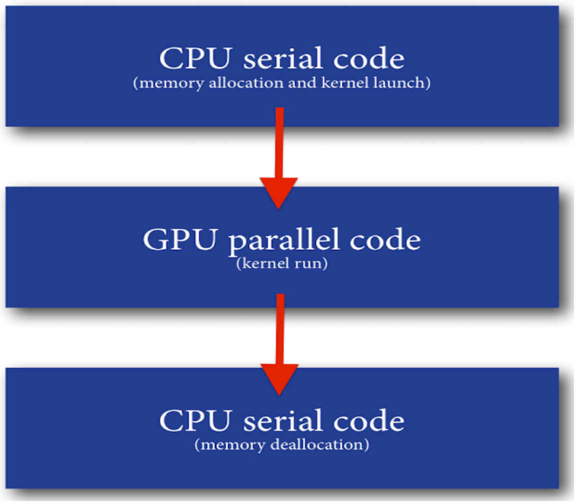

Fig. 3: Execution of code in CUDA

3. Congugate Gradient

In arithmetic, the conjugate slope system is a calculation for the numerical arrangement of specific frameworks of direct comparisons, in particular those whose framework is symmetric and positive-unequivocal. The conjugate slope strategy is regularly executed as an iterative calculation, pertinent to inadequate frameworks that are so vast it is not possible be taken care of by a direct execution or other direct routines, for example, the Cholesky disintegration.

Expansive scanty frameworks regularly emerge when numerically fathoming incomplete differential mathematical statements or enhancement issues. The conjugate inclination system can likewise be utilized to take care of unconstrained improvement issues, for example, vitality minimization. It was essentially created by Magnus Hestenes and Eduard Stiefel.

Suppose we want to solve the following system of linear equations

$$Ax = b$$

for the vector x where the known *n*-by-*n* matrix A is symmetric (i.e. $A^T = A$), positive definite (i.e. $x^TAx > 0$ for all non-zero vectors x in R^n), and real, and b is known as well. We denote the unique solution of this system by $\mathbf{x}_*$.

Code in MATLAB

```
function [x] = conjgrad(A,b,x)
  r=b-A*x;
  p=r;
  rsold=r'*r;

  for i=1:1e6
    Ap=A*p;
    alpha=rsold/(p'*Ap);
    x=x+alpha*p;
    r=r-alpha*Ap;
    rsnew=r'*r;
    if sqrt(rsnew)<1e-10
        break;
    end
    p=r+rsnew/rsold*p;
    rsold=rsnew;
  end
end
```

4. Iterative Method Implementation Using Openmp

Conjugate gradient methodis one of the methods of Krylov subspace method. Writing the codes in parallel make the method to run fast. OpenMP is one of the parallel computing which mainly runs on CPU platform. The principle wellsprings of parallel overhead in any imparted memory implementation are: worldwide hindrances, genuine/false offering impacts and burden parity.

On a NUMA framework extra overheads originate from non-ideal information position which will trigger remote memory gets to. On account of the CG calculation what's more OpenMP, there are no major false offering impacts as most stores are fundamentally stride-1 gets to which delineate well to the static parts generated by OpenMP circle mandates.

When parallelizing iterative solvers like the CG-algorithm, there are three basic types of operations to consider: vector operations, inner products, sparse matrix vector product.

5. Iterative Method Implementation using Cuda

At the point when actualizing the code in cuda stage, it runs quicker than openmp, since it is executing in GPU. It has numerous times strings than that of CPU. Each one string will be executing the free code and that makes the calculation to run productively and quick. Serial code executes on host while parallel code executes on device.

CUDA's modifying model likewise accept that both the host and the gadget keep up their own particular DRAM, alluded to as host memory and gadget memory, separately. Subsequently, a project deals with the worldwide, steady, and composition memory spaces obvious to pieces through calls to the CUDA runtime. This incorporates gadget memory allotment deallocation, and in addition information exchange in the middle of host and gadget memory.

The general flow of a solver implemented is:

- Allocate memory for matrices and vectors in the host memory;
- Initialize matrices and vectors in the host memory;
- Allocate memory for matrices and vectors in the device memory;
- Copy matrices from host memory to device memory;
- Define the device grid layout:
  - Number of blocks
  - Threads per block
- Execute the kernel on the device;
- Copy back the results from device memory to host memory
- Memory cleans up.

6. Overall Performance Optimization Strategies

Performance optimization revolves around three basic strategies:

- Maximizing parallel execution;
- Optimizing memory usage to achieve maximum memory bandwidth;
- Optimizing instruction usage to achieve maximum instruction throughput.

Expanding parallel execution begins with organizing the calculation in a manner that uncovered however much information parallelism as could reasonably be expected. At focuses in the calculation where parallelism is broken on the grounds that a few strings need to synchronize so as to impart information between one another, there are two cases: Either these strings have a place with the same square, in which case they ought to utilize __syncthreads() and offer information through imparted memory inside the same bit call, or they fit in with diverse pieces, in which case they must offer information through worldwide memory utilizing two different piece summons, one for composing to and one for perusing from worldwide memory.

Concerning improving guideline use, the utilization of number-crunching directions with low throughput ought to be minimized. This incorporates exchanging exactness for pace when it doesn't influence the finished result, for example, utilizing characteristic rather than standard capacities or single-accuracy rather than twofold exactness.

7. Conclusion

In this paper, I tried to conclude that parallel will be better when compared with serial computing. Even though OpenMP is also a parallel way to execute, it has some disadvantages like thread problem, cache problem which sometimes makes the algorithm worst than serial code. In order to recover from that, CUDA can be used. Since cuda is a GPU based computing it will surely be executing faster than OpenMP.

Many applications of linear algebra can be then be converted into parallel. Applications through image processing, computer science etc can be used. More over the code can be later implemented using OpenCL.

Acknowledgment

This research paper is made possible through the help and support from everyone, including: parents, teachers, family, friends, and in essence, all sentient beings. First and foremost, I would like to thank Professor Arivazhagan N, SRM University, for his most support and encouragement. Finally, I sincerely thank to my parents, families, and friends, who provide the advice and support. The product of this research paper would not be possible without all of them.

References

[1] Girish Sharma, Abhishek Agarwala and Baidurya Bhattacharya, "A fast parallel Gauss Jordan algorithm for matrix inversion using CUDA", at SciVerse ScienceDirect Computers and structures, 2013.
[2] Marcus M, Minc H. Introduction to linear algebra. New ed. New York: Dover Publications; 1988.
[3] NVIDIA, CUDA C Programming Guide, Version 4.0, 2011.
[4] AMD, ATI Stream Computing - Technical Overview. AMD, Tech. Rep. 2008
[5] Saad, Y, "Iterative Methods for Sparse Linear Systems", PWS Publishing Company, 1996
[6] Khronos OpenCL Working Group, The OpenCL Specification - Version 1.0. The Khronos Group, Tech. Rep. 2009.
[7] Mei W, Hwu W, Kirk D. Video lectures for ECE 498AL, University of Illinois [m4v Video].
[8] Fathi Vajargah B. Different stochastic algorithms to obtain matrix inversion. Appl Math Comput 2007;189:1841–6.
[9] Jie Shen, Jianbin Fan, Henk Sips and Ana Lucia Varbanescu, "Performance Gaps between OpenMP and OpenCL for Multi-core CPUs", 2012 41st International Conference on Parallel Processing Workshops.
[10] Bogdan Oancea, Tudorel Andrei, Andreea Iluzia Iacob, "CUDA based iterative methods for linear systems", AWERProcedia Information Technology & Computer Science 2012.
[11] Gantmacher FR. Applications of the theory of matrices. 1st ed. Dover Publications; 2005.
[12] S.F. McGinn and R.E. Shaw, "Parallel Gaussian Elimination Using OpenMP and MPI", in Proc. of the 16th Annual International Symposium on High Performance Computing Systems and Applications, pp. 169-173, 2002.
[13] NVIDIA Corporation, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi," 2009.
[14] Michael J. Wolfe, High Performance Compilers for Parallel Computers, Addison-Wesley Publishing Company, Redwood City, California, 1996.
[15] R. Membarth, F. Hannig, J. Teich, M. Körner, and W. Eckert, "Frameworks for Multi-core Architectures: A Comprehensive Evaluation Using 2D/3D Image Registration," in ARCS'11, pp. 62–73, 2011.
[16] NVIDIA, "CUDA C Programming Guide." http://developer.download.nvidia.com/compute/DevZone/docs/html /C/doc/CUDACProgramming Guide.pdf, 2012.

Sai Sruthi,currently pursuing MTech in Department of Information Technology from SRM University, Chennai. I hold a Bachelor Degree in Information Technology from KMCT College of Engineering(Under Calicut University). I am a member of IEEE.

N.Arivazhagan is Asst.Professor of Department of information technology, SRM University, Kattankulathur. He has been serving more than 25 years of teaching and 1 year industry experience.He is doing research in Content Based Image Retrieval.He holds a M.S Degree from Birla Institute of Technology, Pilani and M.B.A from Madurai Kamaraj University.