

**Development of 3D-CFD code for Heat Conduction Process using
CUDA**

Dissertation

Submitted in partial fulfilment of

Evaluation of

Master of Technology

Computer Engineering

By

Yogesh Bhadke

MIS No: 121222007

Under the guidance of

Dr. Vandana Inamdar

and

Dr. Vikas Kumar (C-DAC)

Dr. Supriyo Paul (C-DAC)



Department of Computer Engineering and Information Technology

College of Engineering, Pune

Shivajinagar, Pune – 411005

June-2014

**DEPARTMENT OF COMPUTER ENGINEERING AND
INFORMATION TECHNOLOGY,
COLLEGE OF ENGINEERING, PUNE**

CERTIFICATE

This is to certify that the dissertation titled

Development of 3D CFD Code for Heat Conduction Process Using CUDA

has been successfully completed

By

Yogesh D. Bhadke

MIS No: 121222007

and is approved for the partial fulfillment of the requirements for the degree of

Master of Technology in Computer Engineering

Dr. Vandana Inamdar
Project Guide,
Department of Computer
Engineering
and Information Technology,
COEP, Pune

Dr. J. V. Aghav
Head,
Department of Computer Engineering
and Information Technology,
COEP, Pune

Dr. Vikas Kumar
Project Guide,
C-DAC, Pune

Dr. Supriyo Paul
Project Guide,
C-DAC, Pune

June 2014

Acknowledgement

I would like to express my gratitude to all those who have provided timely guidance and helping hand in making my project a success.

I'm thankful to Center for Development of Advanced Computing (C-DAC) Pune for selecting me for project. I'm thankful to **Dr.Vikas Kumar** for giving me opportunity to work for the project. I'm deeply indebted to project guide **Dr.Supriyo Paul** whose guidance, stimulating suggestions and encouragement helped me in all the time of during project discussions & coding phase.

I want to thank the Department of Computer engineering, College of Engineering, Pune for giving me opportunity to do this project as part of our fourth semester, to do the necessary research work and to use departmental resources. I extend my sincere thanks to project guide **Dr. Vandana Inamdar** who encouraged me for this project.

From this project work my guides have brought the best out of me. I even want to thank my department colleagues for all their help, support, interest and valuable hints.

Abstract

Heat conduction is natural phenomenon which is governed by three dimensional, transient partial differential equation. The partial differential equation can be solved by many numerical method such as finite difference method, finite volume method, finite element method etc. These methods require heavy computation to solve the system of algebraic equations.

Graphics processing unit (GPU) can be used to handle the computation of CPU as a co-processor so the GPU will save a lot of time for computation. The dominant proprietary framework for GPU computing is CUDA, provided by NVidia. It can be used to solve computation intensive task.

The objective of this work is to develop a heat conduction code on CUDA platform, which will solve the system of algebraic equations using GPU framework.

List of Figures

Fig. 1.1	Modern GPU architecture	5
Fig. 4.1	Computational Mesh where width and height of cell is shown by Δx and Δy	14
Fig. 5.1	Flow Diagram for ADI and Douglas method	17
Fig. 5.2	Physical domain and physical domain after grid generation	21
Fig. 5.3	Set of XY-plane of physical domain chopped across Z-direction.	21
Fig. 5.4	Computation of Tridiagonal system for row is assigned to 1 thread for X-Sweep.	23
Fig. 5.5	Computation of Tridiagonal system for column is assigned to 1 thread for Y-Sweep	23
Fig. 6.1	Time comparison between ADI, Douglas and ADI CUDA	25
Fig. 6.2	Time comparison for Grid size	25
Fig. 6.3	Speedup	26
Fig. 6.4	Heat map for initial stage	26
Fig. 6.5	Heat map for intermediate stage	27
Fig. 6.6	Heat map at convergence state	27
Fig. 6.7	Validation of ADI and Douglas serial code	28
Fig. 6.8	Validation of ADI-CUDA code.	28

List of Tables

Table 6.1	System specification	24
Table 6.2	Test data	24
Table 6.3	Execution time, No of iteration for convergence in ADI, Douglas and ADI-CUDA method	24

Contents

Acknowledgement	ii
Abstract	iii
List of Figures	iv
List of Tables	v
1 Introduction	1
1.1 Heat Transfer	1
1.2 Heat Conduction	1
1.3 Heat Conduction Equation	2
1.3.1 Steady and unsteady heat transfer	2
1.3.2 Multi-dimensional heat transfer	2
1.4 Boundary and Initial Conditions	3
1.5 GPU	4
1.5.1 GPU Computing	4
1.5.2 GPU Architecture	5
1.6 CUDA	5
1.6.1 Memory hierarchy	6
1.6.2 CUDA threads, block, grid	7
1.6.3 Kernel	7
2 Literature Survey	8
2.1 Literature review	8
3 Problem Statement	10
3.1 Research Gap	10
3.2 Problem Statement	10
3.3 Motivation	10
3.4 Objectives	10
4 Methodology	12
4.1 General Steps	12
4.2 Partial Differential Equation	12
4.2.1 Numerical method	13
4.2.2 Finite difference method	13

5	Implementation	16
	5.1 Alternating Direction Method	16
	5.2 Douglas Method	19
	5.3 Thomas Algorithm	20
	5.4 ADI-CUDA	20
	5.4.1 Mapping strategy	21
6	Results	24
7	Conclusion and Further work	29
	7.1 Conclusion	29
	7.2 Further work	29
	References	

Chapter 1

INTRODUCTION

1.1 Heat Transfer

Heat is the form of energy that can be transferred from one system to another as a result of temperature difference. A thermodynamic analysis is concerned with the amount of heat transfer from one equilibrium state to another equilibrium as a system undergoes a process. The Heat transfer is a stream of thermal engineering that concerns the creation, use, conversion, and transactions of thermal energy and heat amid physical systems. Heat transfer is categorized into assorted mechanisms such as thermal conduction, thermal convection, thermal radiation and transfer of energy by phase change [12]. As these mechanisms have different characteristics, they occur simultaneously in the same system.

The basic condition for the heat transfer is temperature difference. There can be no net heat transfer amid two mediums that are at the alike temperature. The rate of heat transfer in a particular direction depends on the magnitude of the temperature gradient (the temperature difference per unit length or the rate of change of temperature) in that direction. The larger the temperature gradient, the higher the rate of heat transfer [12].

1.2 Heat Conduction

Heat transfer by conduction is the flow of thermal energy inside solids and non-flowing fluids, driven by thermal non-equilibrium usually measured as a heat flux (vector), i.e. the heat flow per unit time at a surface [12].

Conduction is the transfer of energy from the more energetic particles of a substance to the adjacent less energetic ones as a result of interactions between the particles.

Conduction can occur in solids, liquids, or gases. In gases and liquids, conduction occurs due to the diffusion of the molecules across their random motion. In solids, it is due to the combination of vibrations of the molecules in a lattice and the energy transferred by free electrons. The rate of heat conduction across a medium depends on the geometry of the medium, its thickness, and the physical of the medium, as well as the temperature difference across the medium [12].

1.3 Heat Conduction Equation

Heat conduction equation is the parabolic partial differential equation (PDE). Heat transfer have direction as well as magnitude. The rate of heat conduction in a particular direction is proportional to the temperature gradient. Heat conduction in a medium is three-dimensional and depends on time. That is, $u=u(x, y, z, t)$ here u is the temperature function over space variable x, y, z and time t . The temperature in a medium varies along position as well as time [12].

1.3.1 Steady and unsteady heat transfer

Heat transfer problems are usually categorized as being steady or transient. Steady state conduction is a form of conduction that happens when the temperature difference is constant [12], so that after an equilibrium the temperatures in the object does not change any further. In steady state conduction, the heat going in the object is equal to heat coming out. In steady state conduction, temperature is independent of time and can be represented in the form like $u=u(x, y, z)$.

Transient conduction occurs as the temperature inside an object changes as a function of time. In transient conduction temperature function can be represented like $u=u(x, y, z, t)$.

1.3.2 Multidimensional heat transfer

Heat transfer problems are additionally categorized as being one-dimensional, two-dimensional, or three-dimensional, depending on the relative magnitudes of heat transfer rates in various direction and the level of accuracy required [13].

The temperature dissipation across the medium at a particular time can be delineated by a set of three coordinates $x, y,$ and z in the rectangular or Cartesian co-ordinate system [12].

$$\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2} = 0 \quad (1.1)$$

Where $\alpha = \frac{k}{\rho c_p}$

k is the thermal conductivity

ρ is the density of the material and

c_p is the specific heat capacity

The equation in two-dimensions takes the form:

$$\frac{\partial u}{\partial t} - \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = 0 \quad (1.2)$$

And in three dimensions:

$$\frac{\partial u}{\partial t} - \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) = 0 \quad (1.3)$$

1.4 Boundary Conditions and Initial Condition

The differential equations do not include any information regarding the conditions on the surfaces such as the external temperature or a specified heat flux. Yet we understand that the heat flux and the temperature dissipation in a medium depend on the conditions at the surfaces and the description of a heat transfer problem in a medium is not finished unless a maximum description of the thermal conditions at the bounding surfaces of the medium is specified. The mathematical expressions for the thermal condition at the borders are called as Boundary conditions [12].

To delineate a heat transfer problem completely, two boundary conditions have to be given for every single direction of the coordinate system where the heat transfer is significant. Therefore, we have to give two boundary conditions for one-dimensional problem, four boundary conditions for two-dimensional problem, and six boundary conditions for three-dimensional problems [13]. For example, for a three dimensional box of size L^3 the boundary conditions can be described as:

$$T(x=0, y, z, t) = T_{x1} \quad T(x, y=0, z, t) = T_{y1} \quad T(x, y, z=0, t) = T_{z1}$$

$$T(x=L, y, z, t) = T_{x2} \quad T(x, y=L, z, t) = T_{y2} \quad T(x, y, z=L, t) = T_{z2}$$

Initial condition is a mathematical expression for the temperature dissipation of the medium at the beginning of the simulation. Note that we demand merely one initial condition for a heat conduction problem as the conduction equation is first order in time.

For example $u(x, y, z, t=0) = 0$

Analysis of multidimensional heat conduction PDE is complicated and requires numerical analysis by high end computer with great computation power. So now a days there is hurdle in increase in computation power (GHz) of workstation so one think of using the Graphics Processing Unit to offload the heavy computation task [13].

1.5 GPU

GPU stands for Graphics Processing Unit (GPU) a chip with computing capabilities hosted on a video card mostly used for 3D acceleration in games or codec acceleration in movie editing. NVidia pioneered GPU computing in 2007. Now a days GPU are power energy-efficient datacentres in universities, government labs, universities, enterprises and small and medium business around the world.

1.5.1 GPU computing

GPU-computing is the adoption of the GPU plus CPU to speed-up engineering, enterprise and scientific applications. GPU speed-up the application by offloading the heavy computation task on the GPU whereas rest part of the execution is handled by the CPU.

If we compare CPU and GPU based on the number of cores then Intel core i7 CPU have 6 cores and 1.17 billion transistors and NVidia GTX 580 SC GPU have 512 cores and 3 billion transistors. A CPU consists of the few core optimised for the sequential processing while GPU is built from thousands of smaller and efficient cores designed for processing multiple tasks parallelly. As compared to CPU, GPU's are optimised for data-parallel throughput computation, architecture of GPU is tolerant of memory latency and more transistors are dedicated for computation [17].

1.5.2 GPU architecture

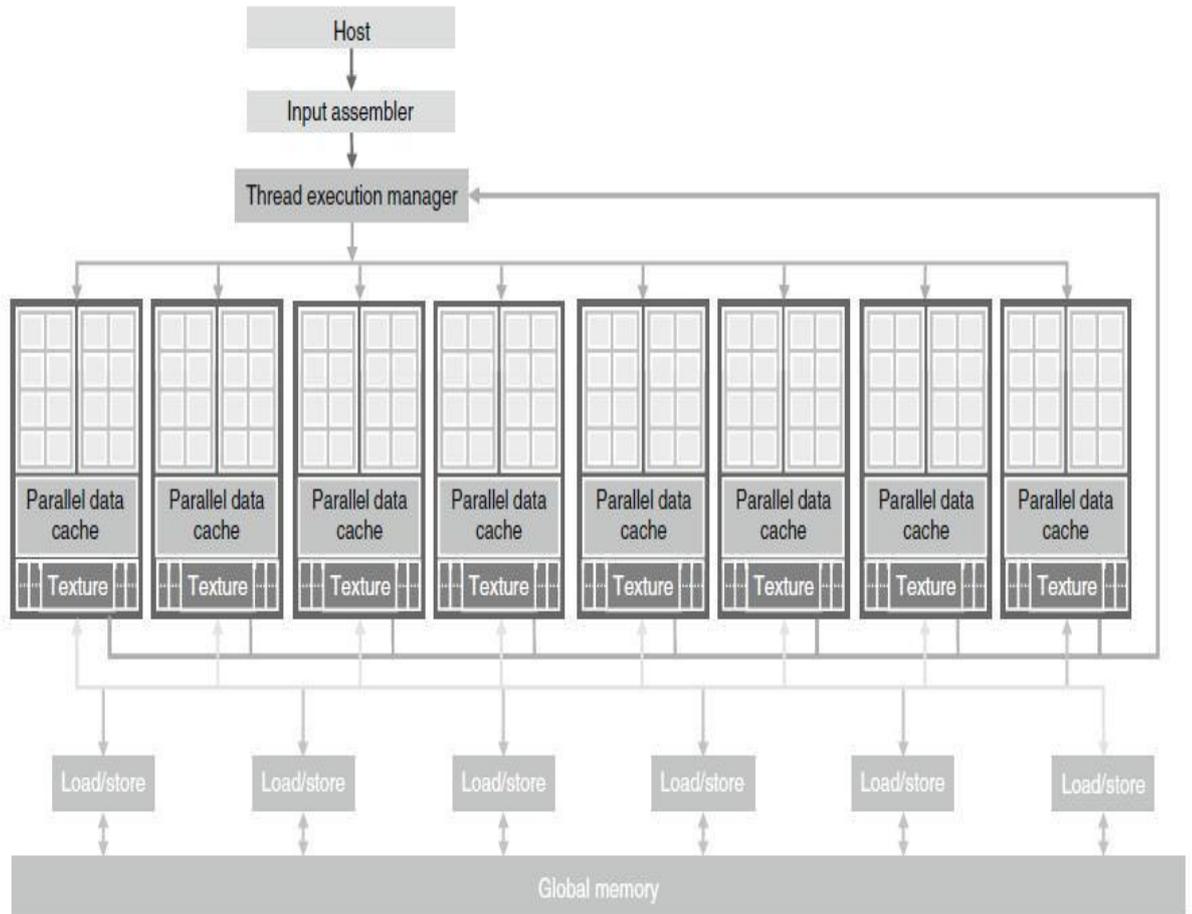


Fig. 1.1 Modern GPU architecture

GPUs are developed as array of threaded streaming multiprocessors. Every streaming multiprocessor contains number of streaming processors that share control logic and instruction cache. Now, every single GPU comes alongside graphics double data rate (GDDR) DRAM, that is called as globe memory as shown in Fig. 1.1 Modern GPU architecture. This global memory differs from the one in CPU. In 2006, NVIDIA industrialized parallel computing framework CUDA to use GPU effectively for general purpose computations [17].

1.6 CUDA

CUDA is a scalable parallel programming model and software environment for parallel computing. CUDA stands for Compute Unified Device Architecture. It is minimal

extension to the C/C++ environment. In CUDA context, GPU is addressed as the Device CPU as host and Kernel is the function that runs on the device. Parallel portions of an application are executed on the device as kernels, one kernel is executed at a time and many threads execute each kernel. CUDA threads are extremely lightweight have very little creation overhead. In CUDA programming environment we can use thousands of thread to achieve maximum efficiency. In CUDA parallel computing, many threads execute in parallel and all execute the same instruction at the same time [17].

1.6.1 Memory hierarchy

Every CUDA enabled GPU provides several different types of memory. These different types of memory have different properties such as access latency, address space, scope and lifetime.

- **Registers** are the fastest memory accessible without any latency on each clock cycle just as on a regular CPU. A thread's register cannot be shared with other kernels.
- **Shared Memory** is comparable to L1 cache memory on a regular CPU. It resides close to the multiprocessor and very little access time. It is shared among threads from the same block.
- **Global Memory** available on the device but off-chip from the multiprocessors so that access to the global memory can be 100 times greater than shared Memory. All threads from all blocks have access to the Global memory and used for inter-block communication between thread.
- **Local Memory** is thread specific private memory stored where global memory is stored. Arrays declared inside a thread are stored in local memory.
- **Constant memory** resides off-chip from multiprocessors and is mostly read-only. The host code writes to the device's constant memory before launching the kernel and the kernel may then read this memory. All thread have access to the shared memory. Constant memory access is cached so that subsequent reads from constant memory can be very fast.
- **Texture memory** is another variety of read-only memory that can improve performance and reduce memory traffic when reads have certain access patterns.

1.6.2 CUDA threads, blocks, grid

Threads in CUDA are extremely lightweight and they have their own registers and program counter [16]. All threads share a memory address space called Global Memory and threads within the same block share access to a very fast shared memory that is more limited in size while threads across the block uses very slow global memory for communication. Within the same block thread share the instruction stream and execute instruction in parallel. When the thread execution diverges then the different branches of execution are run serially until the divergent section is completed then at that point all threads in the block can resume their execution in parallel again. CUDA devices run many threads simultaneously. For example, NVidia Tesla C2075 has 14 multiprocessors, each of which have 32 cores so that 448 threads may be running simultaneously. Threads running under CUDA must be grouped under blocks, a block can hold at most 512 or 1024 threads. Now a days GPU based on kepler architecture can support 2048 threads also. Grid is the blocks of threads which may be one, two or three dimensional as programmer prefers.

1.6.3 Kernel

Kernel is the main unit of work that the main program running on the host computer offloads to the GPU for the computation on the device. In CUDA, launching a kernel requires specifying three things

- The dimensions of the grid
- The dimension of the Blocks
- The kernel functions to run on that device.

Kernel function is specified by declaring `__global__` in the code and a special syntax is used in the code to launch these functions on the GPU with specification of the block and grid dimensions. These kernel functions acts as the entry point for the GPU computations and kernel function can call another function called as device function declared using `__device__` keyword. Both function have the return type as the void.

Chapter 2

LITERATURE SURVEY

2.1 Literature Review

Federio E. Teruel and Rizwan-uddin implemented a parallel numerical code for the simulation of the heat conduction between packed spheres [9]. The complex geometry motivates the authors to build the tool because modelling of such geometry is computationally expensive. This tool helps to compute the temperature distribution of any kind of arrangement or configuration of packed spheres which are constrained to different boundary condition. The authors used the finite volume method for discretization. The authors created a parallel version of the code because number of sphere to be simulated can become very large increasing the computational load. Each sphere is handled by a single processor. The authors observed that the parallel code was giving good performance when the number of spheres was greater than 50.

Yuzhi Sun and Indrek Wichman in a technical note [11] presented the theoretical solution to the heat conduction in one-dimensional three-layer composite slab. Eigen function expansion and finite difference method was used to find out and compare the solution of the above mentioned heat conduction problems.

Xiaohua Meng et al [1] designed a CPU based heat Conduction algorithm by leveraging CUDA. The algorithm implements the calculation of displacement and velocity of each particle on GPU in parallel. To simulate process of one-dimensional heat conduction on the one-dimensional particles structure system they implemented CPU based serial algorithm of the Runge-kutta method. In parallel version of the algorithm, they used the cellular element method to organize the data in parallel fashion. This method divides the computational domain into a series of thread grids, each grid contains multiple particles and each particle's state is only relevant with two adjacent particle's state. Their parallel algorithm process divided into three parts: 1. Initialize and read each particles state's information, which is done on the CPU. 2. Establish the mapping from particles to thread grid. This part is also completed in CPU. 3. State of each particle is calculated. This step is performed on GPU. The third part is very time consuming (around 80% of the total time).

Cohen and Molemaker [3] implemented a second order finite volume code for CFD simulations on CUDA. They studied buoyancy driven flow with this implementation and observe over 8x performance gain on a single GPU over a 8-core CPU based system.

Jacobsen et. al. [4] studied a mixed MPI-CUDA implementation of incompressible flow computation on a GPU cluster. They implemented a dual-level parallelism to solve Navier-Stokes equation for simulating buoyancy driven flow. CUDA is used for fine grain data-parallelism executing on each GPU and MPI for coarse grain parallelism across the whole cluster. They observe over 130x performance gain using 128 GPUs on 64 nodes over a CPU based implementation.

Tomasz P. Stefanski, Timothy D. Drysdale [7] implemented the 3D ADI FDTD method using graphics processing unit to accelerate the performance. They used parallel cyclic reduction algorithm to solve the tridiagonal system of equations. The performance obtained from parallel implementation is 8 times that of the serial implementation of the 3D ADI-FDTD method.

Chapter 3

Problem Statement

3.1 Research Gap

According to literature review many authors have tried to implement the heat conduction equation in one-dimensional and in two-dimensional they also got good results. Implementation in the three-dimensional and in multidimensional is much complicated as compared to the one or two dimensional heat conduction equation. One author headed for the three-dimensional heat conduction equation and implemented the parallel algorithm on the GPU, he used the parallel cyclic reduction algorithm to solve the tridiagonal system of equations and the performance received by him was 8X.

As I'm also heading towards the implementation of three-dimensional heat conduction equation my goal is to design an effective parallel program strategy which can best utilise the memory hierarchy in CUDA, fine-grained (thread) and coarse grained (block) parallelism. I need to select the best tridiagonal solver according to my parallel program strategy so that my performance should be at least greater than 8X.

3.2 Problem Statement

To develop a parallel CFD code for three-dimensional transient Heat conduction process on the CUDA platform.

3.3 Motivation

Basically I was curious about parallel computing and want to learn the same so I selected the NVidia's CUDA framework for parallel computing because NVidia is the pioneer in the utilising the GPU for general purpose parallel computing. One of the reason for selecting Computational Fluid Dynamics (CFD) domain is that CUDA has major application in the CFD domain. Problems in the CFD involves huge computation task so these problem can be well efficiently solved by the combination of the CPU and GPU.

3.4 Objectives

- To examine the best algorithm to solve partial differential equation.
Many finite difference method are available to solve the partial differential equation like explicit method, implicit method, semi implicit method. Choose the best among them to solve the equation.

- To develop a sequential code for three-dimensional transient heat conduction equation in C language.
- To modify the sequential code into parallel code to work on CUDA platform.
Develop a new parallel program strategy to implement the parallel program in context of the CUDA framework to yield the best performance.
- To validate the above code with a standard CFD solver.
Accuracy of the results obtained from the serial and parallel code must be validated with the reference results obtained from the standard CFD solver.

Chapter 4

METHODOLOGY

4.1 General Steps

To solve any CFD problem following general steps are used.

1. The Geometry (physical bounds) of the problem is defined.
2. The problem domain is divided into discrete cells called as nodal network or mesh. The mesh may be uniform or non-uniform.
3. At boundary grid points, apply the boundary conditions. Any derivatives in boundary conditions are replaced by one sided finite difference approximations involving values at boundary and interior nodes.
4. Collect the algebraic conditions at all the grid point interior as well as boundary to obtain a system of algebraic equations in terms of unknown values of the variable at these nodes.
5. Solve the resulting equations in terms of unknown values of the variable at these nodes.

The solution obtained at the grid points can be interpolated and processed to obtain the desired physical quantities in the so-called post processing step of the simulation.

4.2 Partial Differential Equation

Partial Differential Equation (PDE) is a differential equation that contains the number of unknown multivariable functions. A partial differential equation is the equation which contains the partial derivative shown in the Eq. (1.1) in which u is treated as the function of x and t .

Partial differential equation are often used to construct model of the most basic theories underlying physics and engineering [14]. Partial differential equation are used to model the multidimensional systems. The Heat equation is a parabolic partial differential equation which represents the variation of temperature with respect to time over a given region.

4.2.1 Numerical methods

Two ways are available to solve partial differential equations one is by analytical way and other is by numerical Method. Heat conduction problems with simple physical bounds are solved in analytical way but in reality the problem involves the complex geometry with complex boundary conditions which cannot be solved by analytical method. In that case high performance computers are used to get accurate approximate solution by using numerical method.

In analytical method, governing differential equation are solved with boundary conditions which yields the temperature functions at every point in the medium. In the numerical method, differential equation are substituted by the algebraic equation and for the 'n' unknown temperature values in the medium, 'n' algebraic equations are solved simultaneously.

Numerical formulation for the heat conduction problem can be obtained by the finite difference method, finite volume method and finite element method. Each method have its own pros and cons and each used in practice.

4.2.1 Finite difference method

In every numerical method, continuous partial differential equations are replaced by the discrete approximations [14]. Here discrete conveys the meaning that solution is known only for definite number of points in the physical domain. These number of definite points in the domain is selected by the user or programmer of the method. Increase in the number of discrete points in the domain increases the accuracy and resolution of the numerical solution.

The set of discrete points in the physical domain where discrete solution is computed is called as Mesh .These points are called as the nodes and if adjacent point are connected by the lines then the resulting structure looks similar to the net or mesh. $\Delta x, \Delta t$ are the key parameters in the mesh which are local distance between adjacent points in the space and local distance between the adjacent time steps.

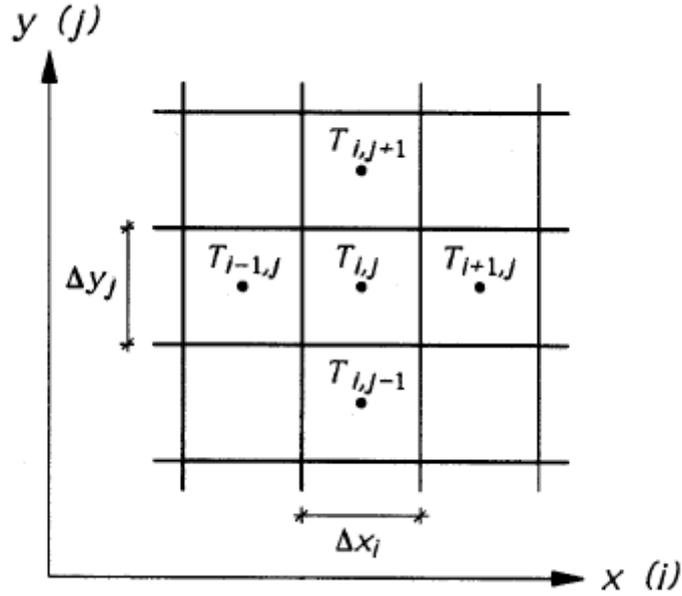


Fig. 4.1 Computational Mesh where width and height of cell is shown by Δx and Δy .

In Fig. 4.1 two-dimensional plane is divided across x and y directions with width and height Δx and Δy for cell (i, j) respectively. The temperature at the cell (i, j) is denoted as the $u_{i,j}$.

In Finite difference method, each derivative of partial differential equation is substituted by its truncated Taylor series expansion while equation is discretized term by term. Time derivative is linear

$$\frac{\partial u}{\partial t} = \frac{u(t+\Delta t) - u(t)}{\Delta t} \quad (4.1)$$

For Spatial order derivative at least second order development is needed to approach second-order derivatives

$$\frac{\partial^2 u}{\partial t^2} = \frac{u(x+\Delta x) - 2u(x) + u(x-\Delta x)}{(\Delta x)^2} \quad (4.2)$$

First order derivatives are usually approximated by the central difference to avoid bias when selecting between forward difference and backward difference.

$$\frac{\partial u}{\partial t} = \frac{u(x+\Delta x) + u(x-\Delta x)}{2(\Delta x)} \quad (4.3)$$

When regular mesh is used, finite difference method produces the highly structured systems of equations. Major advantage of finite difference method is that the simple formulation of method. Disadvantage is that the method demands the simple geometry with a structured grid i.e. method becomes complicated in a non-rectangular geometries.

Finite difference method can be implemented in explicit or implicit manner, explicit method is relatively simple to set up and implement than implicit method. In explicit method there is stability constraints over time step which in some cases results in selecting the very small time step which results in longer computation time. In case of implicit approach, stability constraints over time step is maintained over large values of time step which results in lesser computation time. Implicit method more complicated to program and the set up.

Finite difference method can also be implemented from Crank-Nicolson method which is semi-implicit method because it takes the average of the explicit and implicit method. When the Crank-Nicolson method used for multidimensional heat equation, the number of unknown in difference equation tends to increase with the dimension so the difference equation becomes more complicated and time consuming to solve. So solution of the problem is to use splitting method called ADI which produces the tridiagonal system of equation in each split which can be efficiently solved by Thomas algorithm.

Chapter 5

IMPLEMENTATION

5.1 Alternating Direction Implicit Method

In this section we show how ADI method is used to solve a three-dimensional heat conduction equation. We also introduce TDMA algorithm which is underlying basic block of ADI method.

For the three-dimensional heat equation (Eq. (1.3)) the difference equation is given as

$$\frac{u_{i,j,k}^{n+1} - u_{i,j,k}^n}{\Delta t} = \frac{u_{i+1,j,k}^{n+1} - 2u_{i,j,k}^{n+1} + u_{i-1,j,k}^{n+1}}{(\Delta x)^2} + \frac{u_{i,j+1,k}^{n+1} - 2u_{i,j,k}^{n+1} + u_{i,j-1,k}^{n+1}}{(\Delta y)^2} + \frac{u_{i,j,k+1}^{n+1} - 2u_{i,j,k}^{n+1} + u_{i,j,k-1}^{n+1}}{(\Delta z)^2} \quad (5.1)$$

When the set of simultaneous equation is solved using Crank-Nicolson method for two dimensional difference equation (Eq. (1.2)) which results in coefficient matrix which is penta-diagonal. The solution for penta-diagonal system of equations is very time consuming. In case of three-dimensional heat equation (Eq. (1.3)), corresponding solution of difference equation results in coefficient matrix which is hepta-diagonal in nature when solved using Crank-Nicolson method which is also very time consuming. One way to overcome this is to use the splitting method. This method is known as the Alternating Direction Method or ADI.

In ADI mainly the computation is split into the two steps in case of two-dimension case and in three steps in case of three-dimensional case. In case of three-dimensional case, in first step implicit method is applied in X-direction and explicit method in Y-direction and Z-direction producing an intermediate solution. In second step, implicit method is applied in Y-direction and explicit method in X-direction and Z-direction. In third step, implicit method is applied in Z-direction and explicit method is applied in X-direction and Y-direction. Flow diagram for ADI is shown in Fig.5.1

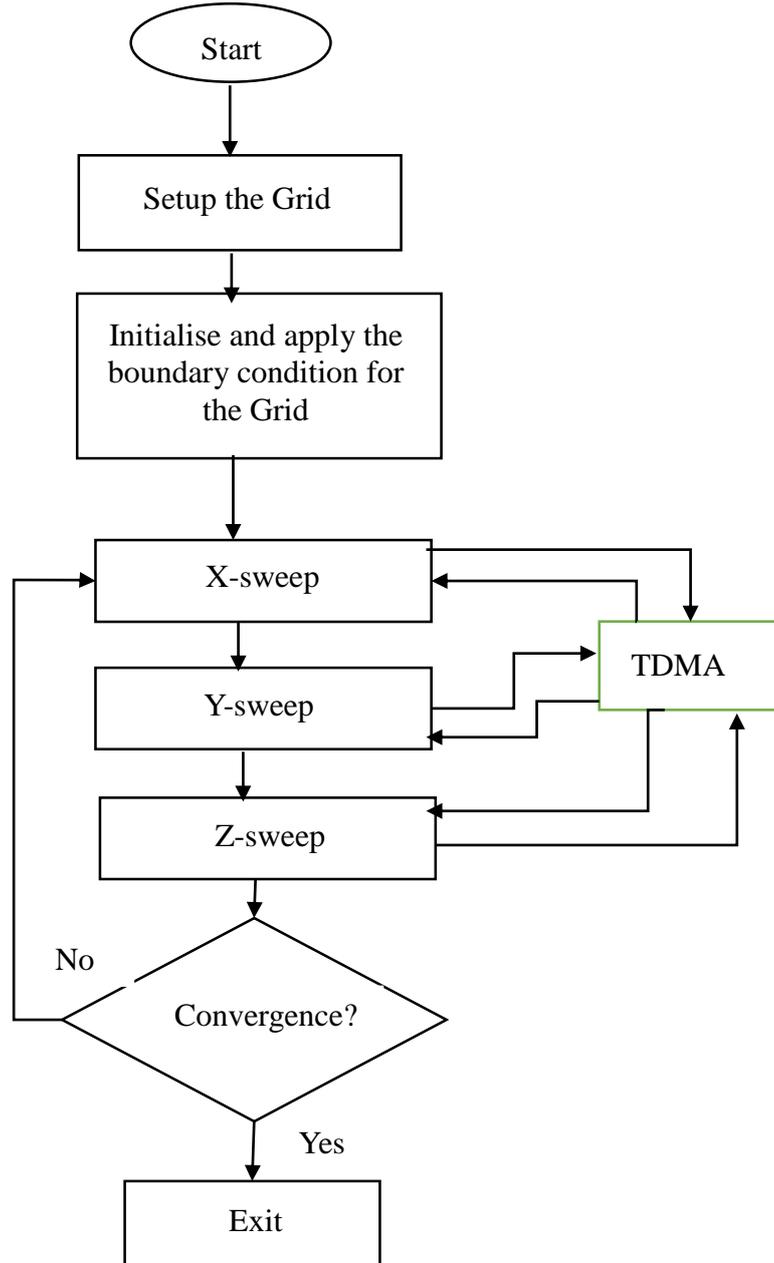


Fig.5.1 Flow Diagram for ADI and Douglas method

The finite difference equation of model equation in the ADI formulation are

$$\frac{u_{i,j,k}^{n+1/3} - u_{i,j,k}^n}{\frac{\Delta t}{3}} = \alpha \left[\frac{\delta_x^2 u_{i,j,k}^{n+1/3}}{(\Delta x)^2} + \frac{\delta_y^2 u_{i,j,k}^n}{(\Delta y)^2} + \frac{\delta_z^2 u_{i,j,k}^n}{(\Delta z)^2} \right] \quad (5.2)$$

$$\frac{u_{i,j,k}^{n+2/3} - u_{i,j,k}^{n+1/3}}{\frac{\Delta t}{3}} = \alpha \left[\frac{\delta_x^2 u_{i,j,k}^{n+1/3}}{(\Delta x)^2} + \frac{\delta_y^2 u_{i,j,k}^{n+2/3}}{(\Delta y)^2} + \frac{\delta_z^2 u_{i,j,k}^{n+1/3}}{(\Delta z)^2} \right] \quad (5.3)$$

$$\frac{u_{i,j,k}^{n+1} - u_{i,j,k}^{n+2/3}}{\frac{\Delta t}{3}} = \alpha \left[\frac{\delta_x^2 u_{i,j,k}^{n+2/3}}{(\Delta x)^2} + \frac{\delta_y^2 u_{i,j,k}^{n+2/3}}{(\Delta y)^2} + \frac{\delta_z^2 u_{i,j,k}^{n+1}}{(\Delta z)^2} \right] \quad (5.4)$$

$$\begin{aligned}
\text{Where } \delta_x^2 u_{i,j,k}^n &= u_{i+1,j,k}^n - 2u_{i,j,k}^n + u_{i-1,j,k}^n \\
\delta_y^2 u_{i,j,k}^n &= u_{i,j+1,k}^n - 2u_{i,j,k}^n + u_{i,j-1,k}^n \\
\delta_z^2 u_{i,j,k}^n &= u_{i,j,k+1}^n - 2u_{i,j,k}^n + u_{i,j,k-1}^n
\end{aligned}$$

Boundary conditions are applied on the domain boundaries and for any typical node (i, j, k) the algebraic equation is given as

$$-d1 u_{i+1,j,k}^{n+1/3} + (1 + 2d1)u_{i,j,k}^{n+1/3} - d1u_{i-1,j,k}^{n+1/3} = (1 - 2d2 - 2d3)u_{i,j,k}^n + d2(u_{i,j+1,k}^n + u_{i,j-1,k}^n) + d3(u_{i,j,k+1}^n + u_{i,j,k-1}^n) \quad (5.5)$$

$$-d2 u_{i,j+1,k}^{n+2/3} + (1 + 2d2)u_{i,j,k}^{n+2/3} - d2u_{i,j-1,k}^{n+2/3} = (1 - 2d1 - 2d3)u_{i,j,k}^{n+1/3} + d1(u_{i+1,j,k}^{n+1/3} + u_{i-1,j,k}^{n+1/3}) + d3(u_{i,j,k+1}^{n+1/3} + u_{i,j,k-1}^{n+1/3}) \quad (5.6)$$

$$-d3 u_{i,j,k+1}^{n+1} + (1 + 2d3)u_{i,j,k}^{n+1} - d3u_{i,j,k-1}^{n+1} = (1 - 2d1 - 2d2)u_{i,j,k}^{n+2/3} + d1(u_{i+1,j,k}^{n+2/3} + u_{i-1,j,k}^{n+2/3}) + d2(u_{i,j+1,k}^{n+2/3} + u_{i,j-1,k}^{n+2/3}) \quad (5.7)$$

$$\text{Where } d1 = \frac{\alpha \Delta t}{3(\Delta x^2)} \quad d2 = \frac{\alpha \Delta t}{3(\Delta y^2)} \quad d3 = \frac{\alpha \Delta t}{3(\Delta z^2)}$$

ADI is the implicit approach where the unknown must be obtained by means of a simultaneous solution of the difference equation applied at all grid point at a given time level. If we write these system of equation in matrix form then it will look like

$$\begin{bmatrix} b1 & c1 & 0 & 0 & 0 \\ a2 & b2 & c2 & 0 & 0 \\ 0 & a3 & b3 & c3 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & a5 & b5 \end{bmatrix} \begin{bmatrix} u1 \\ u2 \\ u3 \\ \vdots \\ u5 \end{bmatrix} = \begin{bmatrix} d1' \\ d2 \\ d3 \\ \vdots \\ d5' \end{bmatrix} \quad (5.8)$$

This coefficient matrix in Eq. (5.8) is a tridiagonal matrix, defined as having nonzero elements only along the three diagonals. The solution of the system of equations denoted by coefficient matrix involves manipulation of tridiagonal arrangement; such a solution can be obtained by using Thomas algorithm which becomes almost the standard for the treatment of tridiagonal systems of equations.

Drawback of ADI method is that method is unconditionally stable (means irrespective of how large the time step (Δt) is, method produces the stable results) for two-dimensional case but for generalisation to three –dimensional case it is not unconditionally stable. so there is an alternative to ADI method is available which is Douglas Method.

5.2 Douglas Method

Douglas method is unconditionally stable and method is in complete analogy with ADI method so the steps for the Douglas method remains same as that of ADI steps.

The finite difference equation of three-dimensional heat equation Eq.(3) in the Douglas method formulation are

$$\frac{u_{i,j,k}^{n+1/3} - u_{i,j,k}^n}{\Delta t} = \alpha \left[\frac{1}{2} \frac{\delta_x^2 u_{i,j,k}^{n+1/3} + \delta_x^2 u_{i,j,k}^n}{(\Delta x)^2} + \frac{\delta_y^2 u_{i,j,k}^n}{(\Delta y)^2} + \frac{\delta_z^2 u_{i,j,k}^n}{(\Delta z)^2} \right] \quad (5.9)$$

$$\frac{u_{i,j,k}^{n+2/3} - u_{i,j,k}^n}{\Delta t} = \alpha \left[\frac{1}{2} \frac{\delta_x^2 u_{i,j,k}^{n+1/3} + \delta_x^2 u_{i,j,k}^n}{(\Delta x)^2} + \frac{1}{2} \frac{\delta_y^2 u_{i,j,k}^{n+2/3} + \delta_y^2 u_{i,j,k}^n}{(\Delta y)^2} + \frac{\delta_z^2 u_{i,j,k}^n}{(\Delta z)^2} \right] \quad (5.10)$$

$$\frac{u_{i,j,k}^{n+1} - u_{i,j,k}^n}{\Delta t} = \alpha \left[\frac{1}{2} \frac{\delta_x^2 u_{i,j,k}^{n+1/3} + \delta_x^2 u_{i,j,k}^n}{(\Delta x)^2} + \frac{1}{2} \frac{\delta_y^2 u_{i,j,k}^{n+2/3} + \delta_y^2 u_{i,j,k}^n}{(\Delta y)^2} + \frac{1}{2} \frac{\delta_z^2 u_{i,j,k}^{n+1} + \delta_z^2 u_{i,j,k}^n}{(\Delta z)^2} \right] \quad (5.11)$$

Where $\delta_x^2 u_{i,j,k}^n = u_{i+1,j,k}^n - 2u_{i,j,k}^n + u_{i-1,j,k}^n$

$\delta_y^2 u_{i,j,k}^n = u_{i,j+1,k}^n - 2u_{i,j,k}^n + u_{i,j-1,k}^n$

$\delta_z^2 u_{i,j,k}^n = u_{i,j,k+1}^n - 2u_{i,j,k}^n + u_{i,j,k-1}^n$

Boundary conditions are applied on the domain boundaries and for any typical node (i, j, k), the algebraic equation is given as

$$-d1 u_{i+1,j,k}^{n+1/3} + (1 + 2d1)u_{i,j,k}^{n+1/3} - d1u_{i-1,j,k}^{n+1/3} = (1 - 2d1 - 2d2 - 2d3)u_{i,j,k}^n + d2(u_{i,j+1,k}^n + u_{i,j-1,k}^n) + d3(u_{i,j,k+1}^n + u_{i,j,k-1}^n) + d1(u_{i+1,j,k}^n + u_{i-1,j,k}^n) \quad (5.12)$$

where $d1 = \frac{\alpha \Delta t}{2(\Delta x)^2}$ $d2 = \frac{\alpha \Delta t}{(\Delta y)^2}$ $d3 = \frac{\alpha \Delta t}{(\Delta z)^2}$

$$-d2 u_{i,j+1,k}^{n+2/3} + (1 + 2d2)u_{i,j,k}^{n+2/3} - d2u_{i,j-1,k}^{n+2/3} = (1 - 2d1 - 2d2 - 2d3)u_{i,j,k}^n + d1(u_{i+1,j,k}^{n+1/3} + u_{i-1,j,k}^{n+1/3}) + d2(u_{i,j+1,k}^n + u_{i,j-1,k}^n) + d3(u_{i,j,k+1}^n + u_{i,j,k-1}^n) \quad (5.13)$$

Where $d1 = \frac{\alpha \Delta t}{2(\Delta x)^2}$ $d2 = \frac{\alpha \Delta t}{2(\Delta y)^2}$ $d3 = \frac{\alpha \Delta t}{(\Delta z)^2}$

$$\begin{aligned}
& d3 u_{i,j,k+1}^{n+1} + (1 + 2d3)u_{i,j,k}^{n+1} - d3u_{i,j,k-1}^{n+1} = (1 - 2d1 - 2d2 - 2d3)u_{i,j,k}^n + \\
& d1(u_{i+1,j,k}^{n+1/3} + u_{i+1,j,k}^{n+1/3}) + d2(u_{i,j+1,k}^{n+2/3} + u_{i,j-1,k}^{n+2/3}) + d3(u_{i,j,k+1}^n + u_{i,j,k-1}^n) \quad (5.14)
\end{aligned}$$

Where $d1 = \frac{\alpha \Delta t}{2(\Delta x^2)}$ $d2 = \frac{\alpha \Delta t}{2(\Delta y^2)}$ $d3 = \frac{\alpha \Delta t}{2(\Delta z^2)}$

5.3 Thomas Algorithm

Gaussian elimination is the standard method for the solving the system of linear, algebraic equation. Thomas algorithm is essentially the result of applying Gaussian elimination to the tridiagonal system of equations. Thomas algorithm is an efficient solver for Eq. (7) and it has two sweeps forward elimination and backward substitution. Specifically we wish to eliminate the lower diagonal term (the a's) as follows.

In the first sweep, the lower diagonal is eliminated by

$$c'_0 = \frac{c_0}{b_0}, c'_i = \frac{c_i}{b_i - c'_i a_i} \quad i = 1, 2, \dots, M - 1 \quad (5.15)$$

$$d'_0 = \frac{d_0}{b_0}, d'_i = \frac{d_i - d'_{i-1} a_i}{b_i - c'_i a_i} \quad i = 1, 2, \dots, M - 1 \quad (5.16)$$

The tridiagonal system i.e. a system of equations with finite coefficient only on the main diagonal (the b's), the lower diagonal (the a's) and the upper (the c's).

The second sweep solves unknowns from the last point of the domain to the first as

$$u_{M-1} = d'_{M-1}, u_i = d'_i - c'_i u_{i+1} \quad i = M - 2, M - 3, \dots, 0 \quad (5.17)$$

Thomas algorithm is serial in nature. In order to calculate c'_i, d'_i and u'_i their immediately preceding terms c'_{i-1}, d'_{i-1} and u'_{i-1} have to calculate ahead.

5.4 ADI-CUDA

In this section we are talk about strategy required to convert serial code into parallel code in CUDA.

For parallel implementation, as we are discussing about three dimension we consider only symmetrical or non-symmetrical cubes. In CUDA we are talking about the lots of threads so we have to arrange CUDA threads according to our need.

5.2.1 Mapping strategy

As shown in Fig.5.2 the physical domain i.e. the cube is divided into discrete cells and the resultant cube after meshing is also shown in the same Fig.5.2.

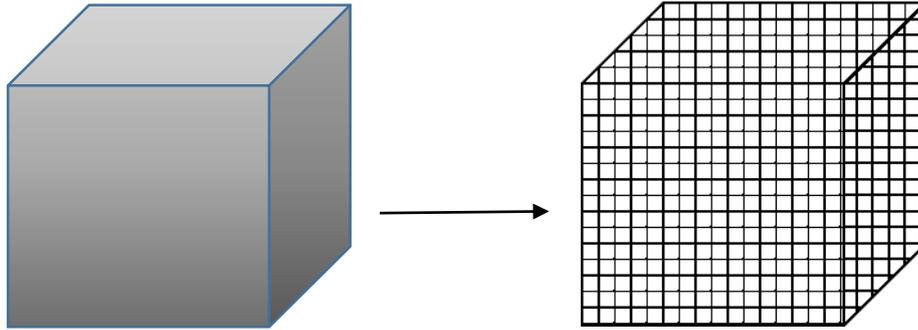


Fig. 5.2 Physical domain and physical domain after grid generation

For Mapping CUDA threads we consider the cube as the set of XY-planes chopped across the Z-direction as shown in Fig.5.3. For performing parallel implementation we have to consider set of 1D blocks with set of 1D threads in it. We can assume the block in the CUDA as the one XY plane in the physical domain. One XY plane consists of the number of rows and number of column and set of XY-planes can be seen across Z-direction. We are using the set of 1D threads, one thread can handle one row or column for calculating the temperature values.

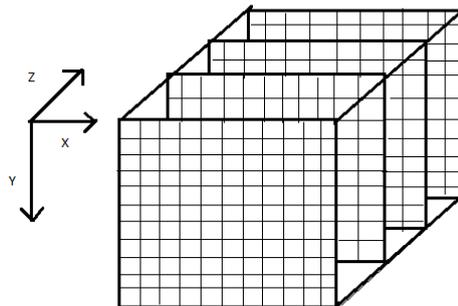


Fig. 5.3 Set of XY-plane of physical domain chopped across Z-direction.

In CUDA environment, there is no effective way available for inter-block synchronisation so we have to create every new call to GPU when we require the synchronisation among the threads from different block. So in our design we have created GPU call for X-sweep, Y-Sweep, and Z-Sweep and also for the checking convergence of the solution and for copy temperature values between successive time steps.

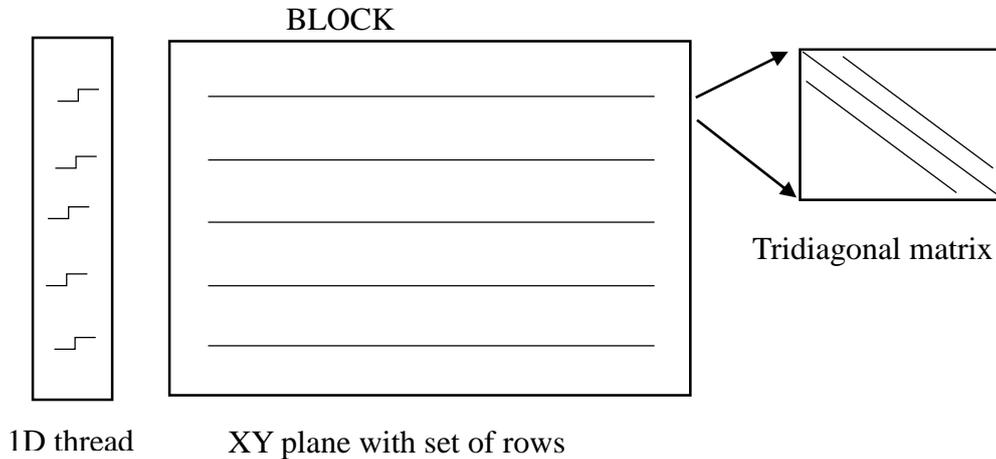


Fig. 5.4 Computation of Tridiagonal system for row is assigned to 1 thread for X-Sweep.

In X-Sweep, the number of thread generated for kernel launch is equal to the number of rows across every plane in the discretized domain as shown in Fig.5.4. One thread can handle the one row completely to generate the coefficient matrix of the tridiagonal system and again gives call to the tridiagonal system solver to solve the tridiagonal system and write back the results from tridiagonal system solver. Each thread stores the coefficient for the tridiagonal matrix in its private memory which is in turn stored in the local memory.

In Y-Sweep, the number of thread generated for the kernel launch is equal to the number of column across every plane in the discretized domain as shown in Fig. 5.5. Here also one thread can handle one column completely and call tridiagonal solver and write back results.

In Z-Sweep, the number of thread generated for kernel launch is equal to the number of rows in one plane only. As in Z-Sweep, temperature values lies across blocks in the grid there is no use for generating the threads across the block.

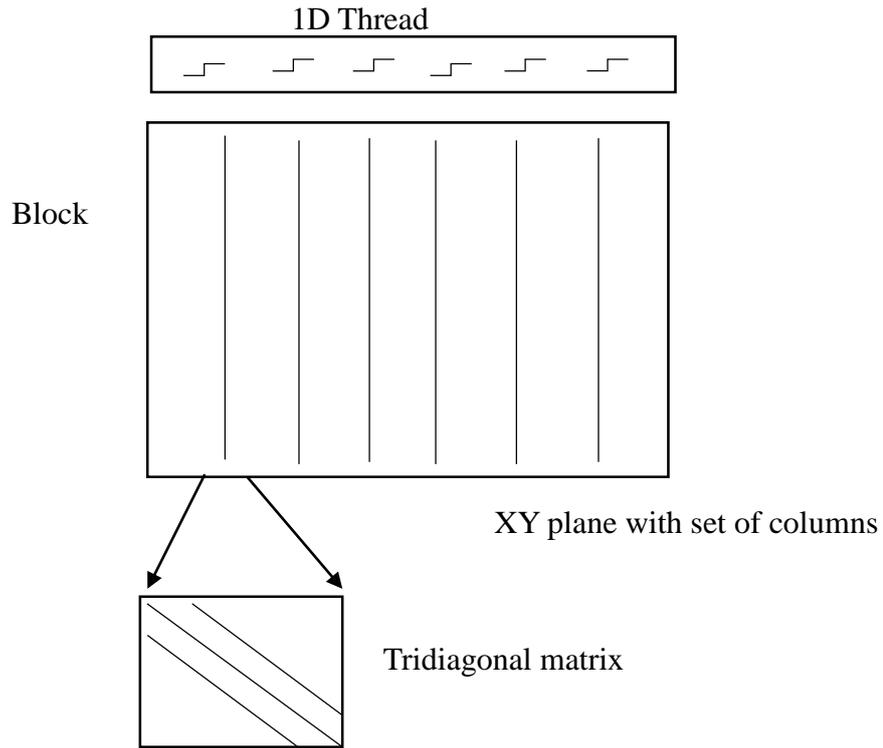


Fig. 5.5 Computation of Tridiagonal system for column is assigned to 1 thread for Y-Sweep

Convergence of the solution is said to be achieved when the maximum temperature difference between the each discrete point of the domain is almost zero or constant in successive time steps

Chapter 6

RESULTS

ADI serial and Douglas code which is written in C-language and the ADI-CUDA which is written in CUDA-C is tested on the system with following specification

Processor	Intel Xeon E5-1607 3.00GHz X 4
RAM	16GB
OS	Ubuntu 12.04 LTS 64-bit
GPU	GeForce GTX 480
Compute capability	2
CUDA version	5.5

Table 6.1 System Specification

Following are the boundary conditions for Grid

Temperature at left end (°c)	80
Temperature at Right end (°c)	20
Temperature at Top end (°c)	60
Temperature at Bottom end (°c)	30
Temperature at Front end (°c)	70
Temperature at Back end (°c)	20
Density(ρ) (kg/m ³)	7800
Sp. Heat(C) (J/K)	473
Conductivity(κ) ($\frac{W}{m \cdot K}$)	43
Time step (second)	0.001

Table 6.2 Test data

Following results shows comparison between ADI, Douglas and ADI-CUDA method wrt execution time and No. iteration taken to converge the solution also the speed-up is shown wrt ADI and ADI-CUDA

Grid size	ADI		Douglas		ADI Parallel		Speedup
	Time to execute	No of Iteration for convergence	Time to execute	No of iteration for convergence	Time to execute	No of iteration for convergence	
10X10X10	57	910000	74	943000	158	866000	0.36075949
20X20X20	689	912000	899	948000	418	866000	1.64832536
30X30X30	2971	912000	3676	948000	694	866000	4.28097983
40X40X40	7372	912000	9201	948000	953	866000	7.73557188
50X50X50	14432	912000	18932	948000	1304	866000	11.0674847

Table 6.3 Execution time, No of iteration for convergence in ADI, Douglas and ADI-CUDA method

From the above table one can easily infer that irrespective of grid size number of iteration for particular method are constant.

Following graphs shows the results obtained by executing the of ADI, Douglas, ADI-CUDA program on the system with system specification mentioned in Table 6.1

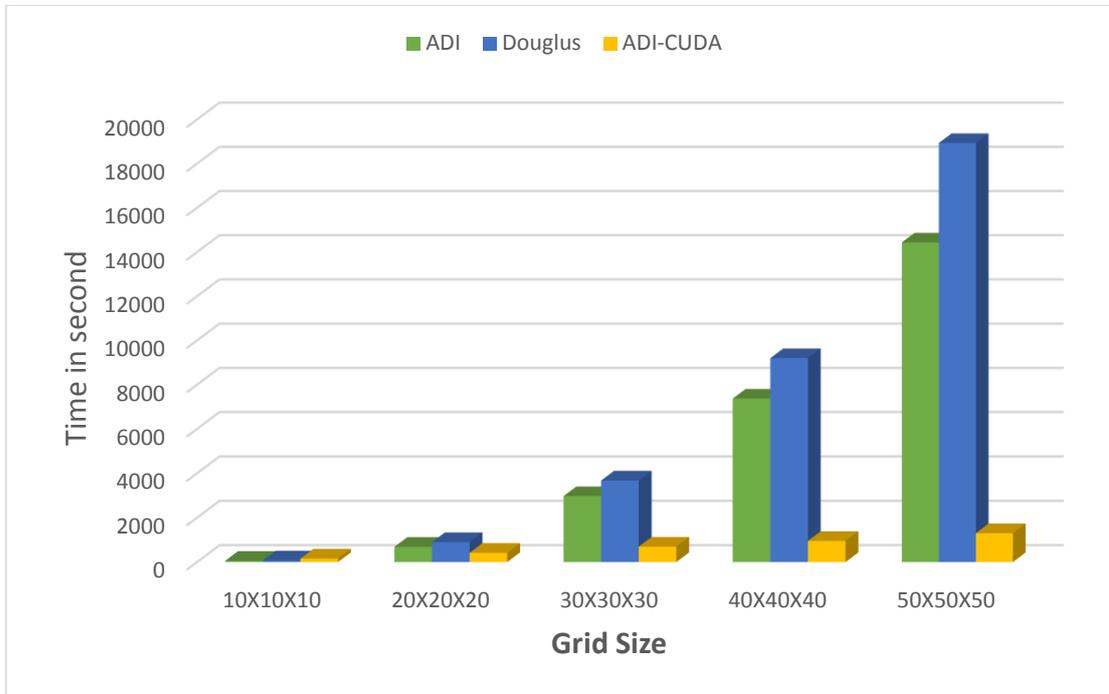


Fig. 6.1 Time comparison between ADI, Douglas and ADI CUDA

Graph from Fig.6.1 implies that for any grid size the time taken for execution is greatest for Douglas method, intermediate for ADI method and minimum for ADI-CUDA method.

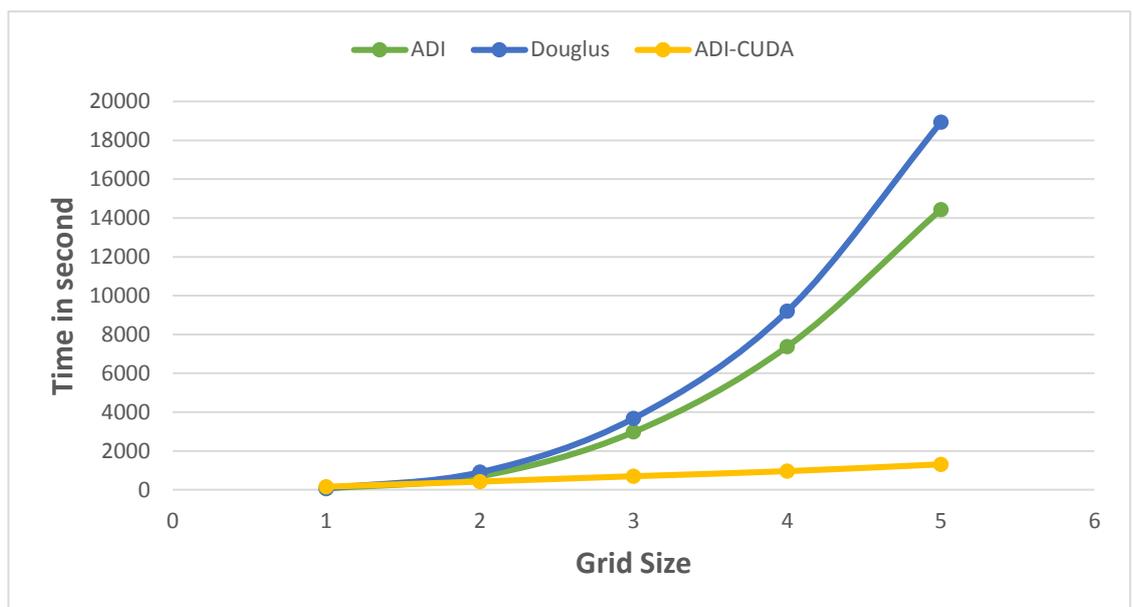


Fig. 6.2 Time comparison for Grid size

Graph from Fig.6.2 implies that as the Grid size increases the time taken for execution also increases exponentially.

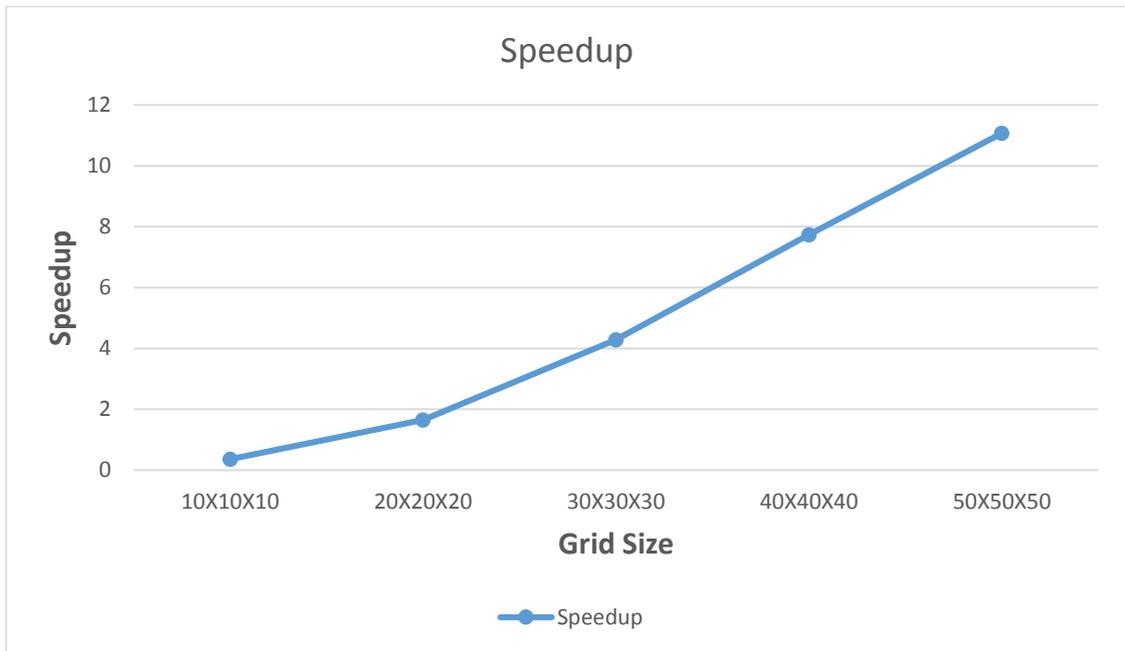


Fig. 6.3 Speedup

Graph from Fig.6.3 implies that as the grid size increases there is increase in speedup also.

Heat Map for sample grid size 10X10X10 at initial stage , at intermediate stage and at convergence state resp.

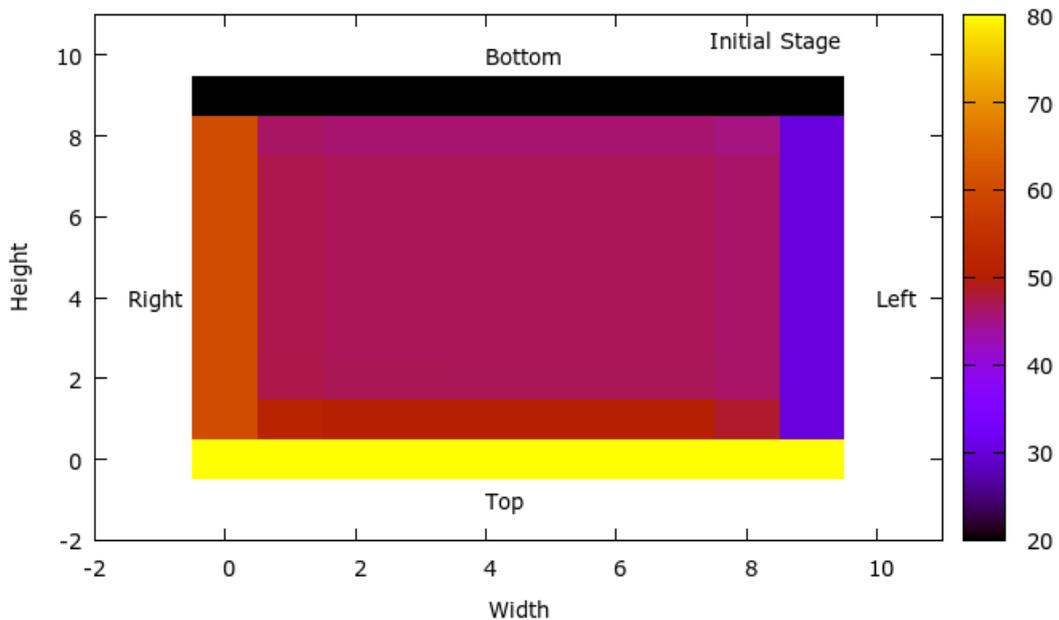


Fig. 6.4 Heat map for initial stage

Heat map is a graphical representation of the temperature values contained in a matrix form as a color

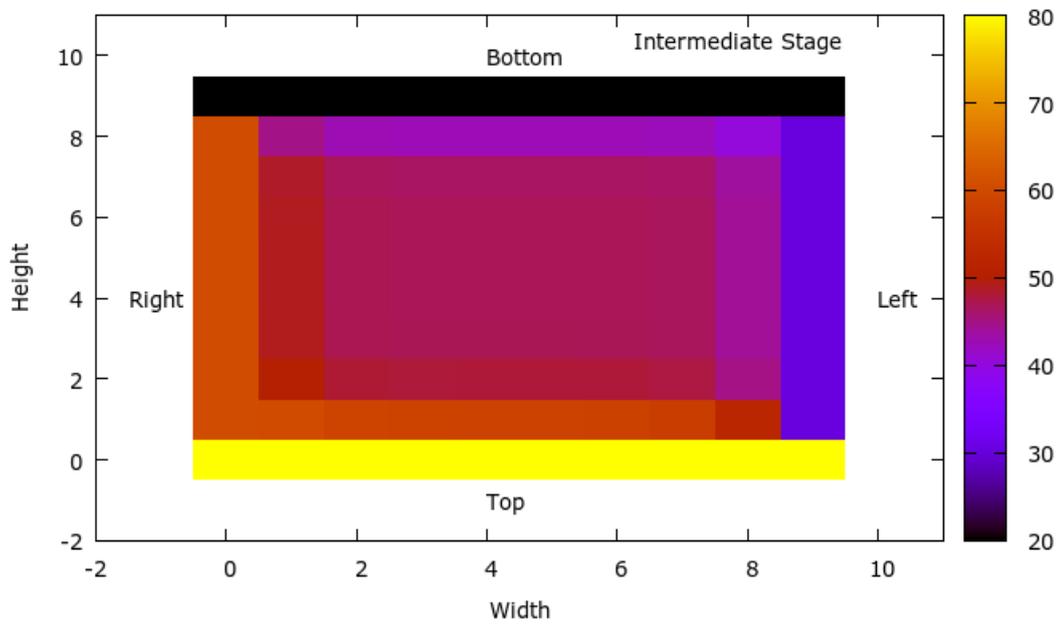


Fig. 6.5 Heat map for intermediate stage

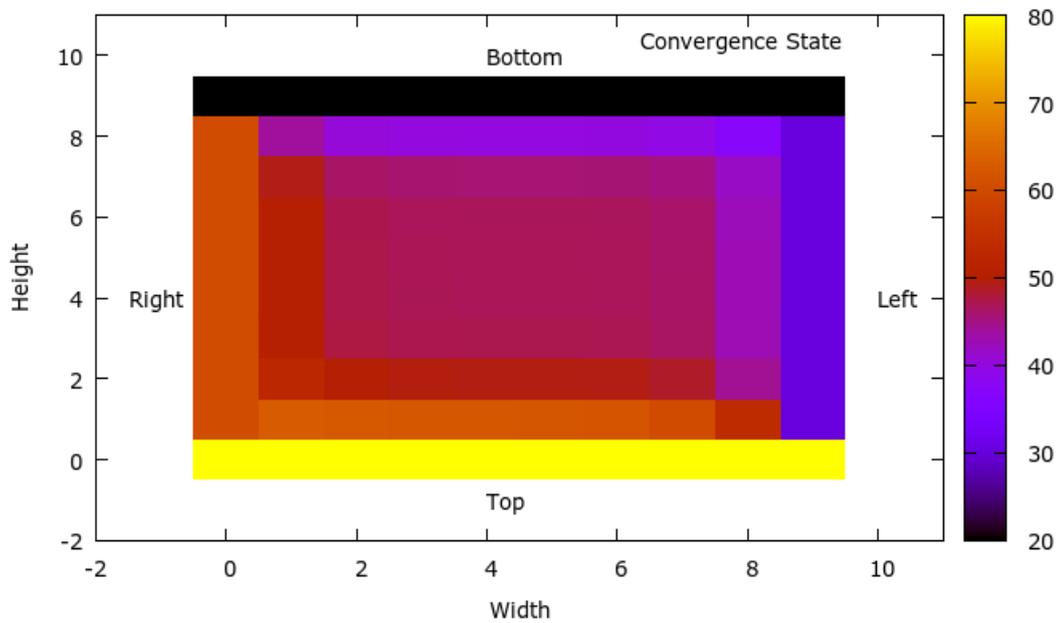


Fig.6.6 Heat map at convergence state

Validation of results with the reference datum

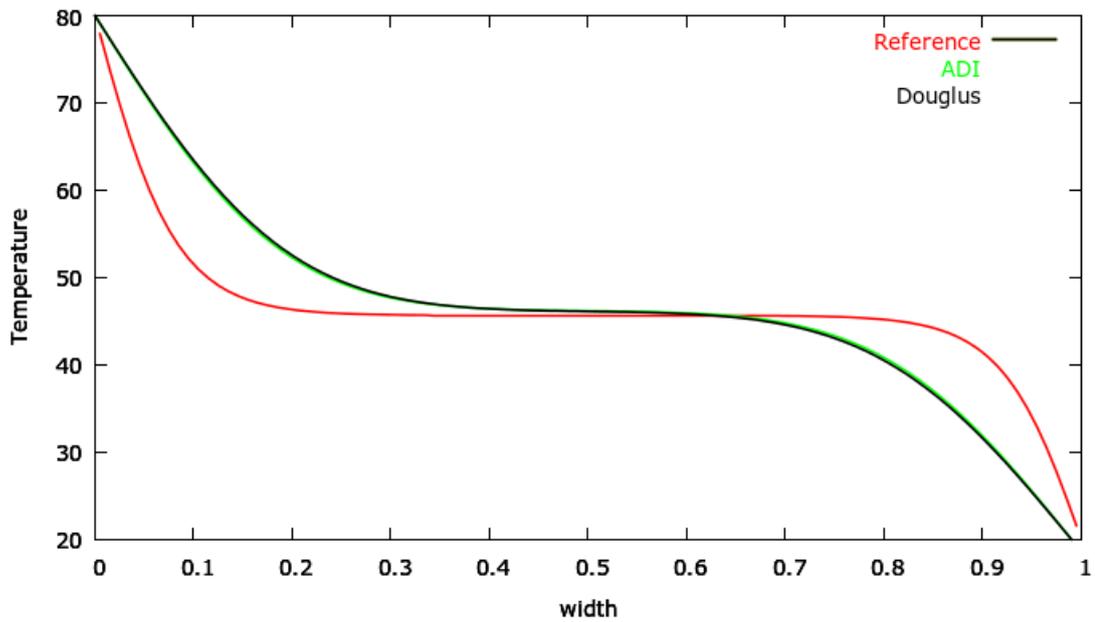


Fig. 6.7 Validation of ADI and Douglas serial code

As shown in the Fig. 6.7 the line of the ADI and Douglas method are overlapping which implies that the results are totally identical because Douglas method is designed in analogy with ADI method. Although the lines of the ADI and Douglas method are deviating from the reference line still the results are acceptable because the equilibrium temperature is same and also system specification and method used for computation is not known for the reference results.

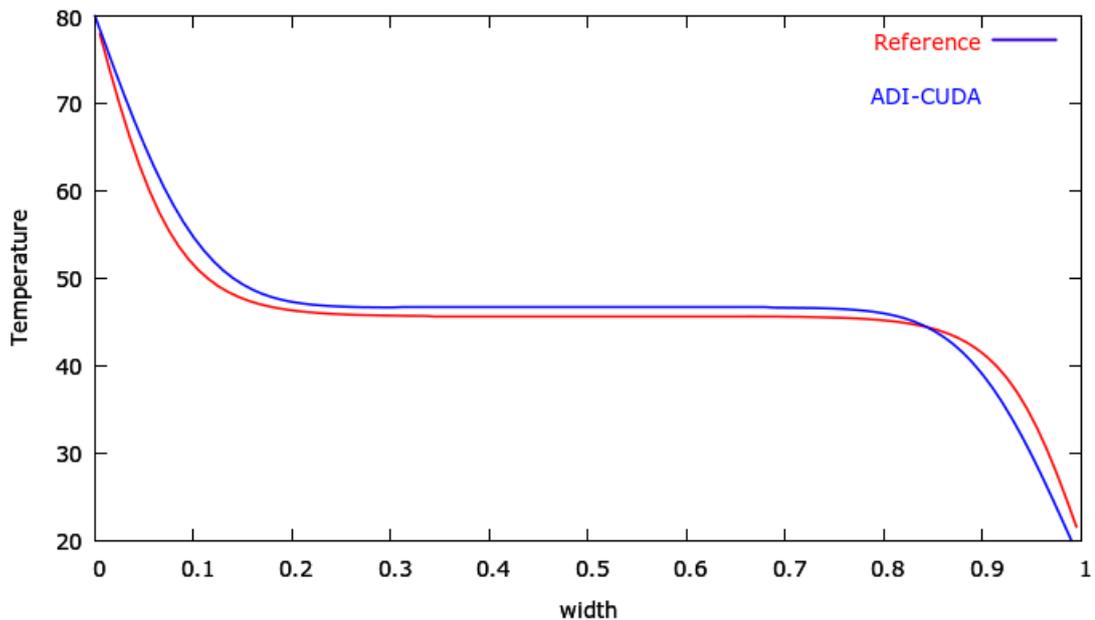


Fig. 6.8 Validation of ADI-CUDA code.

Interestingly from Fig. 6.8 the results of the parallel implementation are in complete analogy with the reference line and the accuracy of the results in parallel ADI-CUDA method is more as compared to the serial ADI and Douglas method.

Chapter 7

CONCLUSION and FURTHER WORK

7.1 Conclusion

Alternating Direction Implicit method is an alternative to the Crank-Nicolson method which is computationally expensive when extended to multidimensional. ADI method shows the high level of data parallelism which can be exploited using the high end GPU.

Speedup up to the 11X is obtained in this project, which is increasing with respect to the grid size. In parallel implementation local memory which is private memory of the CUDA thread is best utilised in this project. As there is no significant inter-block synchronisation available in CUDA, we were forced to split out kernel call for X-sweep, Y-sweep, Z-sweep which is significant overhead in case of parallel implementation.

7.2 Further Work

If in future any significant inter-block synchronisation is available then again it will be possible to improve the speedup with the same or another strategy.

3D heat conduction PDE which is used in this project is almost similar to some of the PDEs which are used in computational finance. With some change in this project one can easily extend this project for computational finance application.

Speedup of this project is possible to increase if the memory layout of the data stored in the GPU is tuned with the memory access during X-sweep, Y-sweep and Z-sweep in parallel implementation as the memory access in these sweeps is widely scattered.

References

- [1] Xiaohua Meng, Dasheng Qin, Yuhui Deng, “*Designing a GPU based Heat conduction algorithm by leveraging CUDA*”, Fourth International Conference on Emerging Intelligent Data and Web Technologies,2013
- [2] Tomasz P. Stefa_ski, Timothy D. Drysdale,” *Acceleration of the 3D ADI-FDTD Method Using Graphics Processor Units*”, IEEE IMS 2009,2009
- [3] Jonathan Cohen, M. Molemaker, “*A Fast Double Precision code using CUDA*”, Proceedings of Parallel CFD, 2009
- [4] Dana Jacobsen, Julien Thibault, Inanc Senocak, “*An MPI CUDA Implementation of massively parallel Incompressible Flow Computations on Multi-GPU Cluster*”, Aerospace Sciences Meeting, 2010
- [5] Duy Minh Dang, “*Pricing of Cross-Currency Interest Rate Derivatives on Graphics Processing Units*”, IEEE, 2010
- [6] OlafSchenk, Matthias Christen, Helmar Burkhart, “*Algorithmic performance studies on graphics processing units*” J. Parallel Distrib. Compute. 68 (2008) 1360–1369
- [7] Shuai Chen, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Kevin Skadron, ”*A performance study of general-purpose applications on graphics processors using CUDA*”, J. Parallel Distrib. Comput. 68 (2008) 1370–1380
- [8] Paulius Micikevicius,”*3D Finite Difference Computation on GPUs using CUDA*”, *GPGPU2*, Washington D.C., US, March 8, 2009.
- [9] Federico E. Teruel, Rizwan-uddin ,” *Parallel Numerical tool for the Heat Conduction problem n three-dimensional packed sphere*“, INREC10
- [10] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips, “*GPU Computing*”, Proceedings of the IEEE Vol.96 No. 5 May 2008
- [11] Yuzhi Sun, Indrek S. Wichman ,”*Transient heat conduction in one dimensional composite slab*”, International Journal of Heat and Mass transfer,2003
- [12] Yunus A. Cengel,”*Heat and mass Transfer*”, 2nd Ed, Mc-Graw-Hill Education,ch.01-05,2007
- [13] John D. Anderson, ”*Computational Fluid Dynamics The Basics with Applications*”, Springer, ch.04,1992

- [14] Klaus Hoddmann, Steve Chaing, "*Computational Fluid Dynamics Volume I*", 4th Ed., EES, Ch.02,03,2000
- [15] Jayathi Y. Murthy, "*Numerical Methods in Heat, Mass and Momentum Transfer*", Draft Notes, Ch.02,03,2002
- [16] Jason Sanders, Edward Kandrot, "*CUDA by example An introduction to General purpose computing*", Addison Wesley, 2010
- [17] D. B. Kirk and W. mei W. Hwu, "*Programming Massively Parallel Processors: A Hands-on Approach*", Morgan Kaufmann, 2010
- [18] *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, ver. 2.0, 6/7/2008, http://www.nvidia.com/object/cuda_develop.html