BTP Project Report

Nandakishore MM

May 15, 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this report we introduce and describe the implementation of multigrid methods, GPU and parallel programming using CUDA and OpenMP and application variational principles for plane compressible flow calculations.

Multigrid methods in numerical analysis are a group of algorithms for solving differential equations using a hierarchy of discretizations. The main idea of multigrid is to accelerate the convergence of a basic iterative method by global correction from time to time, accomplished by solving a similar problem on a coarser grid. Multigrid methods are among the fastest solution techniques known today. In contrast to other methods, multigrid methods are general in that they can treat arbitrary regions and boundary conditions as they do not depend any special properties of the equation. There are many variations of multigrid algorithms, but the common features are that a hierarchy of discretizations (grids) is considered. The important steps are [2]-

- **Smoothing** reducing high frequency errors, for example using a few iterations of the GaussSeidel method.
- **Restriction** downsampling the residual error to a coarser grid.
- **Interpolation** or **prolongation** interpolating a correction computed on a coarser grid into a finer grid.

CUDA, which stands for Compute Unified Device Architecture, is a parallel computing platform and programming model created by NVIDIA and implemented by the graphics processing units (GPUs) that they produce. GPUs are highly parallel multi-core systems allowing very efficient manipulation of large blocks of data. This design is more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. A typical CFD problem involves solving for the flow variables using an iterative method on a grid. Each update on a grid points involves calculations using a few surrounding grid points. This procedure can be efficiently run in parallel. C/C++ programmers use 'CUDA C/C++', compiled with "nvcc", to program or convert existing code to run on the GPU.

OpenMP is an API for shared memory parallel programming in C/C++ and Fortran. It is ideal for running on multi core CPUs.

The classical problem of steady inviscid subsonic flow past an aerofoil can be formulated in two ways. The usual formulation is as a boundary value problem

consisting of a set non linear partial differential equations and a set of boundary conditions. However, it can also be formulated in terms of complementary variational principles. This method consists of replacing the infinitely dimensional variational problem by a finitely dimensional problem by means of finite differences, and an approximate maximizing function is then found by standard methods. This method is discussed in detail by H. Rasmussen, and N. Heys [1]

Chapter 2

Multigrid Methods

2.1 Model Problem

Consider a large sparse linear system of equations $A\mathbf{u} = \mathbf{f}$, where A is a $n \times n$ matrix. Let us denote $\mathbf{v}$ for the approximate solution obtained using some iterative method.

In later sections the jth component of $\mathbf{u}$ and $\mathbf{v}$ will be denoted as u_j and v_j. When associating $\mathbf{u}$ and $\mathbf{v}$ with a particular grid, say Ω^h (where h is the grid spacing), the notation $\mathbf{u}^h$ and $\mathbf{v}^h$ are used.

As an example the following model problem is considered in the following sections -

$$-u''(x) = f(x), \qquad 0 < x < 1 \tag{2.1}$$

$$u(0) = u(1) = 0 \tag{2.2}$$

This problem can be discretized as follows

$$-u_{j-1} + 2u_j - u_{j+1} = h^2 f_j, \qquad 1 \leq j \leq n-1 \tag{2.3}$$

$$u_0 = u_n = 0 \tag{2.4}$$

2.2 Residual and error

Define the algebraic error as

$$\mathbf{e} = \mathbf{u} - \mathbf{v}$$

The error is a vector and it magnitude can be measured by any of the standard vector norms. The Euclidean or 2-norm is defined by

$$\|\mathbf{e}\|_2 = \left\{ \sum_{j=1}^{n} e_j^2 \right\}^{1/2}$$

A computable measure of how well $\mathbf{v}$ approximated $\mathbf{u}$ is given by the residual

$$\mathbf{r} = \mathbf{f} - A\mathbf{u}$$

2.3 Basic Iterative Methods

2.3.1 Jacobi Iteration

The Jacobi iteration in component for can be expressed as

$$v_j^{(1)} = \frac{1}{2}\left(v_{j-1}^{(0)} + v_{j+1}^{(0)} + h^2 f_j\right), \qquad 1 \le j \le n-1. \tag{2.5}$$

where $v^{(0)}$ denotes the current approximation and $v^{(1)}$ denotes the new and updated approximation

2.3.2 Weighted Jacobi Iteration

The first step in weighted Jacobi iteration is to compute

$$v_j^{*} = \frac{1}{2}\left(v_{j-1}^{(0)} + v_{j+1}^{(0)} + h^2 f_j\right), \qquad 1 \le j \le n-1.$$

Then, the new approximation is given by

$$v_j^{(1)} = (1-\omega)v_j^{(0)} + \omega v_j^{*}, \qquad 1 \le j \le n-1. \tag{2.6}$$

2.3.3 Gauss-Seidel Iteration

The Gauss-Seidel iteration is similar to Jacobi iteration except that the components of the new iteration are used as soon as they are computed. This can be expressed as

$$v_j \longleftarrow \frac{1}{2}\left(v_{j-1} + v_{j+1} + h^2 f_j\right), \qquad 1 \le j \le n-1. \tag{2.7}$$

where $\longleftarrow$ stands for replacement or overwriting

2.3.4 Red-Black Gauss-Seidel Iteration

This is the same as Gauss-Seidel method except that all even components are updated first by the expression

$$v_{2j} \longleftarrow \frac{1}{2}\left(v_{2j-1} + v_{2j+1} + h^2 f_{2j}\right) \tag{2.8}$$

and then all odd components using

$$v_{2j+1} \longleftarrow \frac{1}{2}\left(v_{2j} + v_{2j+2} + h^2 f_{2j+1}\right) \tag{2.9}$$

The advantage of red-black over regular Gauss-Seidel is in terms of parallel processing. The red points need only the black points for updating and therefore be updated in any order. This can be done by several independent processors. The same is applicable for the black points.

2.3.5 Drawbacks of the basic iterative methods

Let us assume the initial guess to the above model problem consists of the following Fourier modes

$$v_j = \sin\left(\frac{jk\pi}{n}\right), \qquad 0 \le j \le n, \qquad 1 \le k \le n-1$$

where k represents the wave number. The modes in the lower half of the spectrum with wavenumbers in the range $1 \le k < n/2$, are called *law frequency* or *smooth modes*. The modes in the upper half of the spectrum , with $n/2 \le k \le n-1$ are called *high-frequency* or *oscillatory modes*. An analysis of the weighted Jacobi iteration will give the following expression for the convergence of various modes

$$\lambda_k = 1 - 2\omega \sin^2\left(\frac{k\pi}{2n}\right)$$

Convergence is better when $|\lambda_k|$ is small. From figure 2.1 [2], it can be seen that

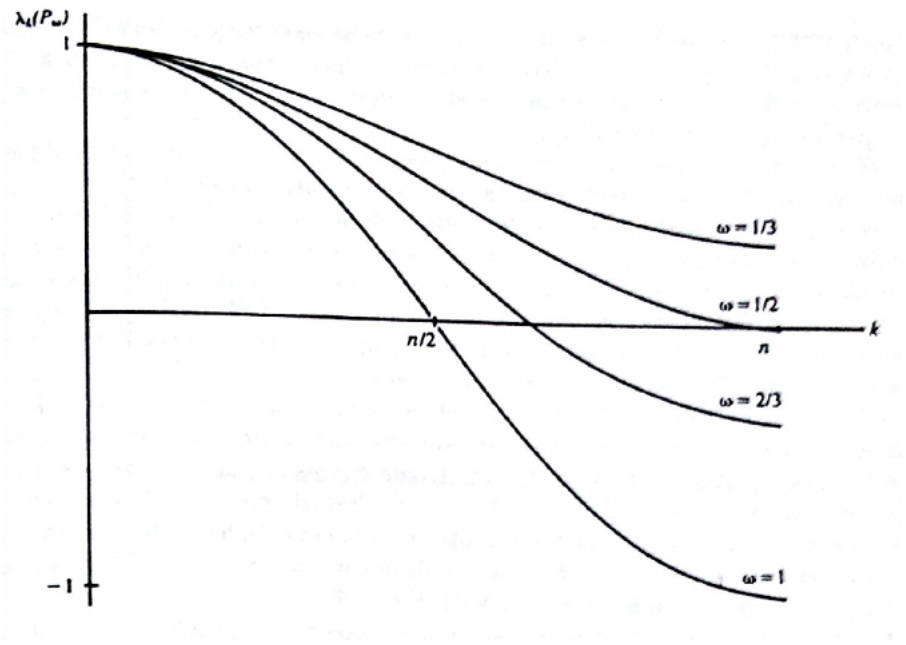

Figure 2.1: λ_k vs wave number k for different ω for weighted Jacobi iterations

while the high frequency modes have a nominal damping rate, the smooth mode damp very slowly. Many relaxation schemes possess this property of eliminating the oscillatory modes and leaving the smooth modes. This is called *smoothing property* and is a serious limitation of conventional relaxation methods. However this limitation can be overcome using multigrid methods.

2.4 Basic Multigrid methods

One of the ways to improve a relaxation scheme is to use a good initial guess. A well-known technique is to perform some preliminary iterations on a coarser grid which is inexpensive. Another important property of a coarser grid is how the various Fourier modes on the finer grid are represented on it. Consider the kth mode on the fine grid Ω^h evaluated at the even numbered grid points (Note: The grid points of the coarse grid Ω^{2h} are the even no. grid points on the fine grid Ω^h). If $1 \leq k < n/2$. its components can be written as

$$w^h_{k,2j} = \sin\left(\frac{2jk\pi}{n}\right) = \sin\left(\frac{jk\pi}{n/2}\right) = w^{2h}_{k,j}$$

We see that the smooth components on Ω^h is more oscillatory on the coarse grid Ω^{2h}. This can be useful in overcoming the smoothing property of the relaxation schemes.

2.4.1 Residual equation

If $\mathbf{v}$ is the approximation to the exact solution $\mathbf{u}$, then the error is given by

$$\mathbf{e} = \mathbf{u} - \mathbf{v}$$

Then

$$A\mathbf{e} = A\mathbf{u} - A\mathbf{v}$$

$$A\mathbf{e} = \mathbf{f} - A\mathbf{v}$$

$$A\mathbf{e} = \mathbf{r} \tag{2.10}$$

The last equation above is called the *residual equation*. It can be interpreted as follows

Relaxation of the original equation $A\mathbf{u} = \mathbf{f}$ with an arbitrary initial guess $\mathbf{v}$ is equivalent to relaxing on the residual equation $A\mathbf{e} = \mathbf{r}$ with the specific initial guess $\mathbf{e} = 0$

2.4.2 Intergrid transfer operators

The process of transferring approximation from the coarser grid to the finer grid is called *interpolation* or *prolongation*. The linear interpolation operator, denoted by I^h_{2h} takes the coarse-grid vectors and produces the fine grid vectors according to the rule $I^h_{2h}v^{2h} = v^h$; is given by

$$v^h_{2j} = v^{2h}_j,$$

$$v^h_{2j+1} = \frac{1}{2}\left(v^{2h}_j + v^{2h}_{j+1}\right), \qquad 0 \leq j \leq \frac{n}{2} - 1$$

For 2 dimensional problems I_{2h}^{h} is given by

$$\begin{aligned} v_{2i,2j}^{h} &= v_{i,j}^{2h}, \\ v_{2i+1,2j}^{h} &= \frac{1}{2}\left(v_{i,j}^{2h} + v_{i+1,j}^{2h}\right), \\ v_{2i,2j+1}^{h} &= \frac{1}{2}\left(v_{i,j}^{2h} + v_{i,j+1}^{2h}\right), \\ v_{2i+1,2j+1}^{h} &= \frac{1}{4}\left(v_{i,j}^{2h} + v_{i+1,j}^{2h} + v_{i,j+1}^{2h} + v_{i+1,j+1}^{2h}\right), \\ 0 \le 1, j &\le \frac{n}{2} - 1 \end{aligned}$$

The other intergrid transfer operator is the restriction operator denoted by I_{h}^{2h}. It moves vectors from the fine grid to a coarse grid. It is given by the full weighted restriction operator

$$v_{j}^{2h} = \frac{1}{4}\left(v_{2j-1}^{h} + v_{2j}^{h} + v_{2j+1}^{h}\right), \qquad 0 \le j \le \frac{n}{2} - 1 \tag{2.11}$$

In 2 dimension the same is given by

$$\begin{aligned} v_{ij}^{2h} = \frac{1}{16}[&v_{2i-1,2j-1}^{h} + v_{2i+1,2j-1}^{h} + v_{2i-1,2j+1}^{h} + v_{2i+1,2j+1}^{h} \\ &+ 2(v_{2i,2j-1}^{h} + v_{2i,2j+1}^{h} + v_{2i-1,2j}^{h} + v_{2i+1,2j}^{h}) \\ &+ v_{2i,2j}^{h}], \qquad 1 \le i, j \le \frac{n}{2} - 1 \end{aligned}$$

2.4.3 Two grid Correction Scheme

The two grid correction scheme can be defined by the following algorithm.

$$\mathbf{v}^{h} \leftarrow \mathrm{TG}(\mathbf{v}^{h}, \mathbf{f}^{h})$$

- Relax ν_1 times on $A^{h}\mathbf{u}^{h} = \mathbf{f}^{h}$ on Ω^{h} with initial guess $\mathbf{v}$
- Compute the fine-grid residual $\mathbf{r}^{h} = \mathbf{f}^{h} - A\mathbf{v}^{h}$ and restrict it to the coarse grid by $\mathbf{r}^{2h} = I_{h}^{2h}\mathbf{r}^{h}$.
- Solve $A^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ on Ω^{2h}
- Interpolate the coarse-grid error to the fine grid by $\mathbf{e}^{h} = I_{2h}^{h}\mathbf{e}^{2h}$ and correct the fine grid approximation by $\mathbf{v}^{h} \leftarrow \mathbf{v}^{h} + \mathbf{e}^{h}$
- Relax ν_2 times on $A^{h}\mathbf{u}^{h} = \mathbf{f}^{h}$ with initial guess $\mathbf{v}$

2.4.4 Multigrid Scheme (V Cycle)

In the two grid correction scheme, the coarse grid problem can be solved using the two grid correction scheme itself. This can be recursively applied to successive coarse grid problems. This is the basis for *V cycle scheme.*

To simplify notations, let us call the right side of the residual equation $\mathbf{f}^{2h}$, rather than $\mathbf{r}^{2h}$, the solution of the residual equation $\mathbf{e}^{2h}$ as $\mathbf{u}^{2h}$, and $\mathbf{v}^{2h}$ denotes the approximation to the solution $\mathbf{u}^{2h}$.

Now the V cycle scheme can be defined as follows

V Cycle Scheme

$$\mathbf{v}^h \leftarrow V^h(\mathbf{v}^h, \mathbf{f}^h)$$

1. Relax ν_1 times on $A^h\mathbf{u}^h = \mathbf{f}^h$ with a given initial guess $\mathbf{v}^h$.
2. If Ω^h = coarsest grid, then go to step **4**.
   Else
   $$\mathbf{f}^{2h} \leftarrow I_h^{2h}(\mathbf{f}^h - A^h\mathbf{v}^h),$$
   $$\mathbf{v}^{2h} \leftarrow 0,$$
   $$\mathbf{v}^{2h} \leftarrow V^{2h}(\mathbf{v}^{2h}, \mathbf{f}^{2h}).$$
3. Correct $\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h\mathbf{v}^{2h}$.
4. Relax ν_2 times on $A^h\mathbf{u}^h = \mathbf{f}$ with initial guess $\mathbf{v}^h$

2.4.5 μ Cycle

Now the μ cycle scheme can be defined as follows

μ Cycle Scheme

$$\mathbf{v}^h \leftarrow \mu^h(\mathbf{v}^h, \mathbf{f}^h)$$

1. Relax ν_1 times on $A^h\mathbf{u}^h = \mathbf{f}^h$ with a given initial guess $\mathbf{v}^h$.
2. If Ω^h = coarsest grid, then go to step **4**.
   Else
   $$\mathbf{f}^{2h} \leftarrow I_h^{2h}(\mathbf{f}^h - A^h\mathbf{v}^h),$$
   $$\mathbf{v}^{2h} \leftarrow 0,$$
   $$\mathbf{v}^{2h} \leftarrow \mu^{2h}(\mathbf{v}^{2h}, \mathbf{f}^{2h}) \ \mu \text{ times}.$$
3. Correct $\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h\mathbf{v}^{2h}$.
4. Relax ν_2 times on $A^h\mathbf{u}^h = \mathbf{f}$ with initial guess $\mathbf{v}^h$

When $\mu = 1$, it degenerates to V cycle scheme. $\mu = 2$ is called W cycle.

2.4.6 Full Multigrid (FMG)

The above methods can be optimised more by providing a good initial guess on every grid. This is done using the Full Multigrid (FMG) V cycle

Full Multigrid V-Cycle

$$\mathbf{v}^h \leftarrow \mathrm{FMG}^h(\mathbf{f}^h)$$

1. If Ω^h = coarsest grid, set $\mathbf{v}^h = 0$, go to step **3**.
   Else
   $$\mathbf{f}^{2h} \leftarrow I_h^{2h}(\mathbf{f}^h),$$
   $$\mathbf{v}^{2h} \leftarrow \mathrm{FMG}^{2h}(\mathbf{f}^{2h}) \ \mu \text{ times}.$$

2. Correct $\mathbf{v}^h \leftarrow I_{2h}^h \mathbf{v}^{2h}$.

3. $\mathbf{v}^h \leftarrow V^h(\mathbf{v}^h, \mathbf{f}^h)$ ν_0 times

2.5 Application

2.5.1 Problem

The multigrid algorithms discussed above is applied to the following problem-

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial x^2} = 0 \tag{2.12}$$

subject to boundary conditions

$$u(x, 0) = x^2, \tag{2.13}$$
$$u(0, y) = y^2, \tag{2.14}$$
$$u(x, 1) = 1 - x^2, \tag{2.15}$$
$$u(1, y) = 1 - y^2. \tag{2.16}$$

2.5.2 Results

Solution

For obtaining solution a Red Black Gauss Seidel iteration has been used. The solver was run on a grid of size 4097×4097 (16 million grid points). The solution is plotted in figures 2.2 and 2.3

Comparison of multigrid schemes

For comparing different multigrid schemes, let us define a work unit (WU) as the one iteration on the finest grid. Hence one iteration on the next coarsest grid is equivalent to 1/4 WU and so on.

The comparisons are presented in figures 2.5 and 2.4.

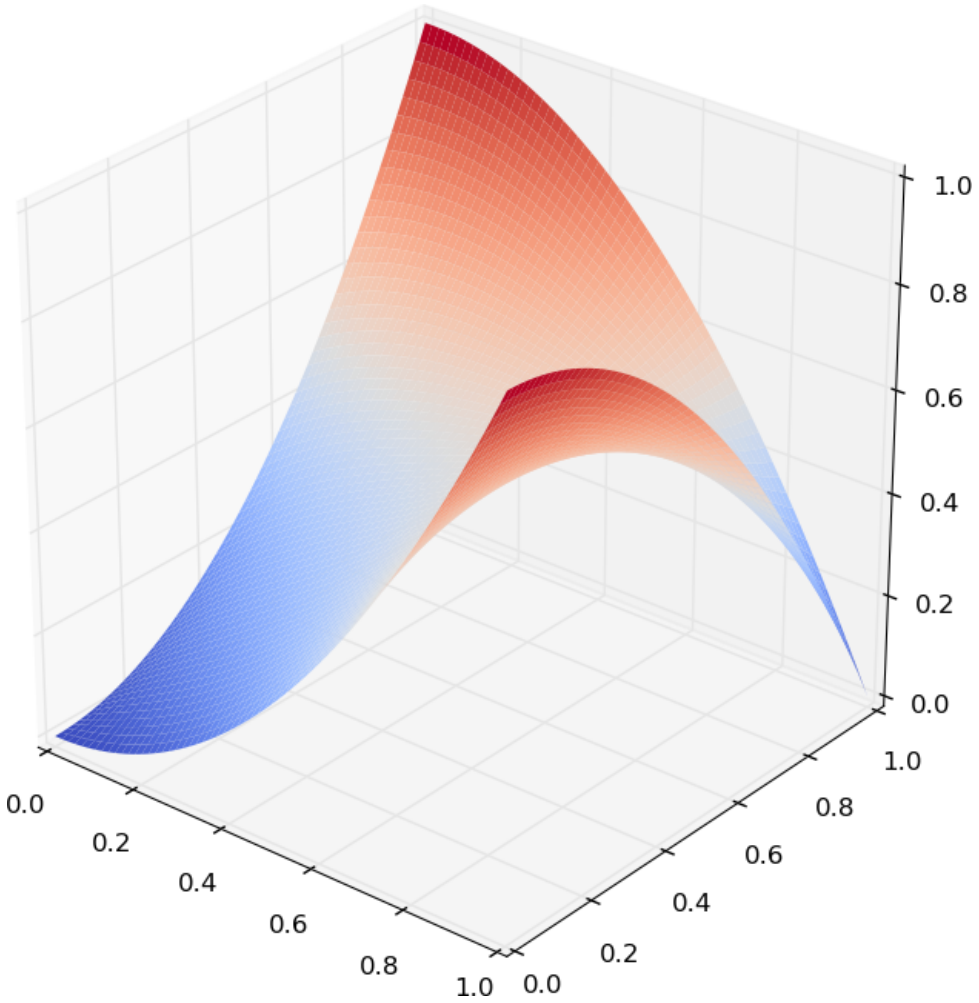

Figure 2.2: Solution of the boundary value problem on a 4097×4097 grid

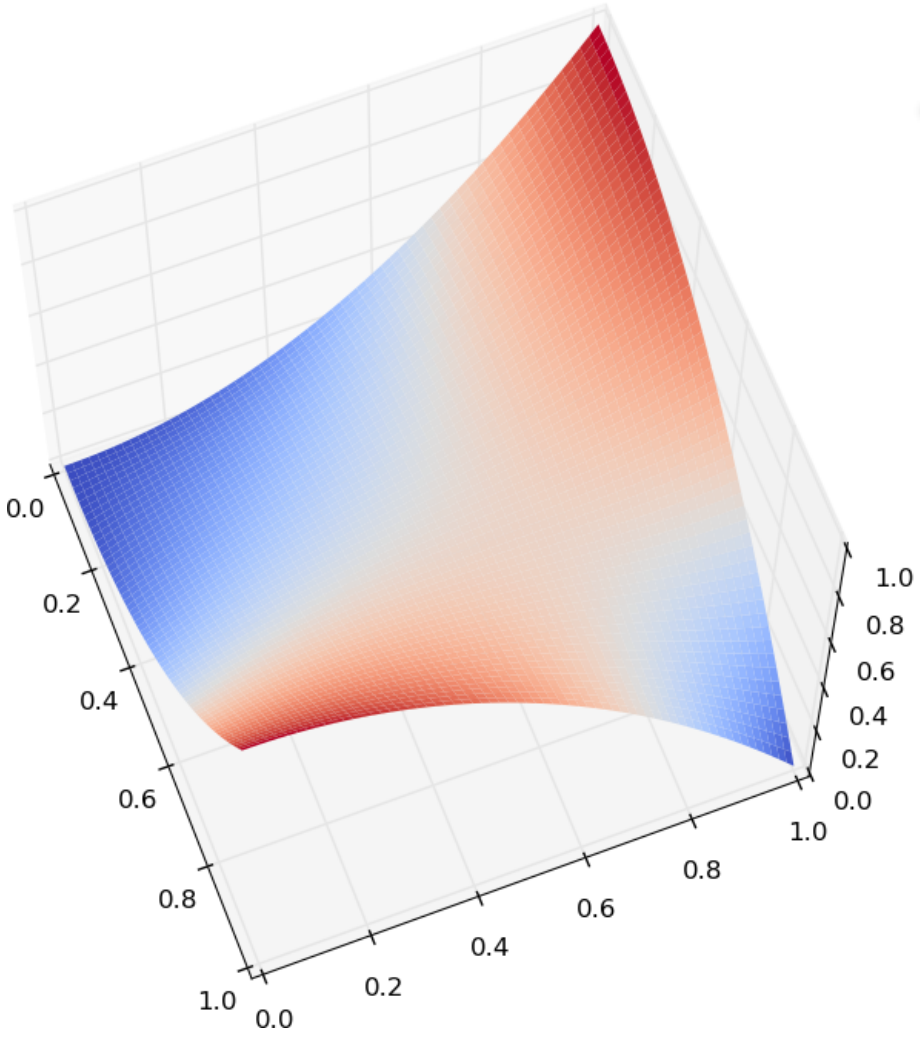

Figure 2.3: Solution of the boundary value problem on a 4097×4097 grid

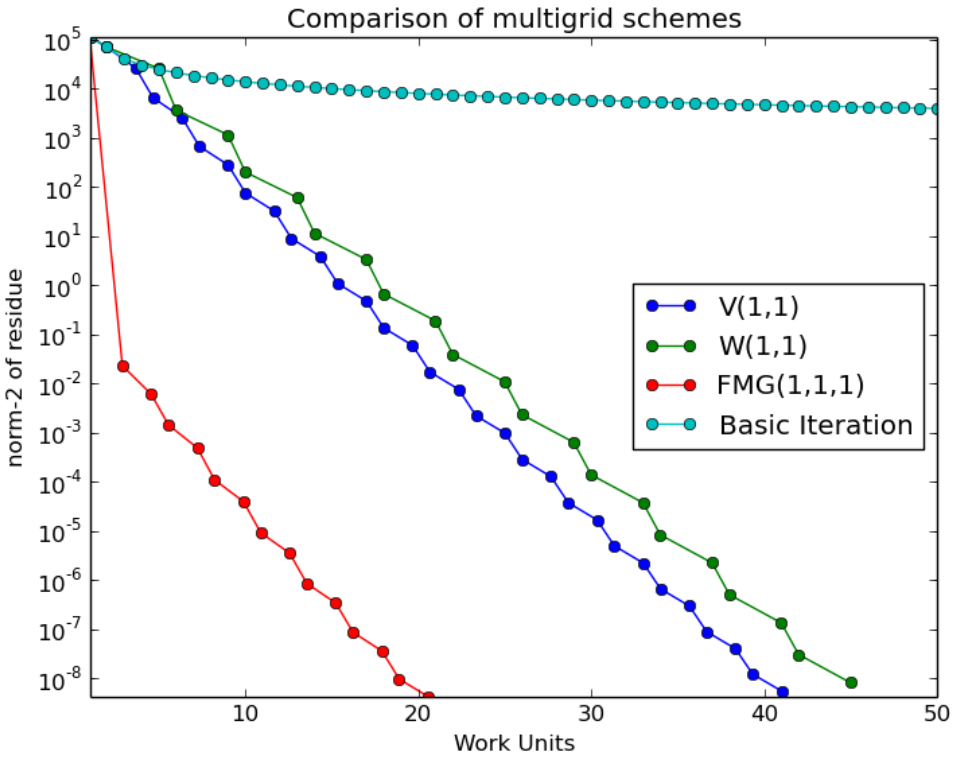

Figure 2.4: Convergence plot different multigrid schemes (In V(ν_1, ν_2), W(ν_1, ν_2), FMG(ν_0, ν_1, ν_2), ν_0, ν_1 and ν_2 represents the same quantities defined in the algorithm)

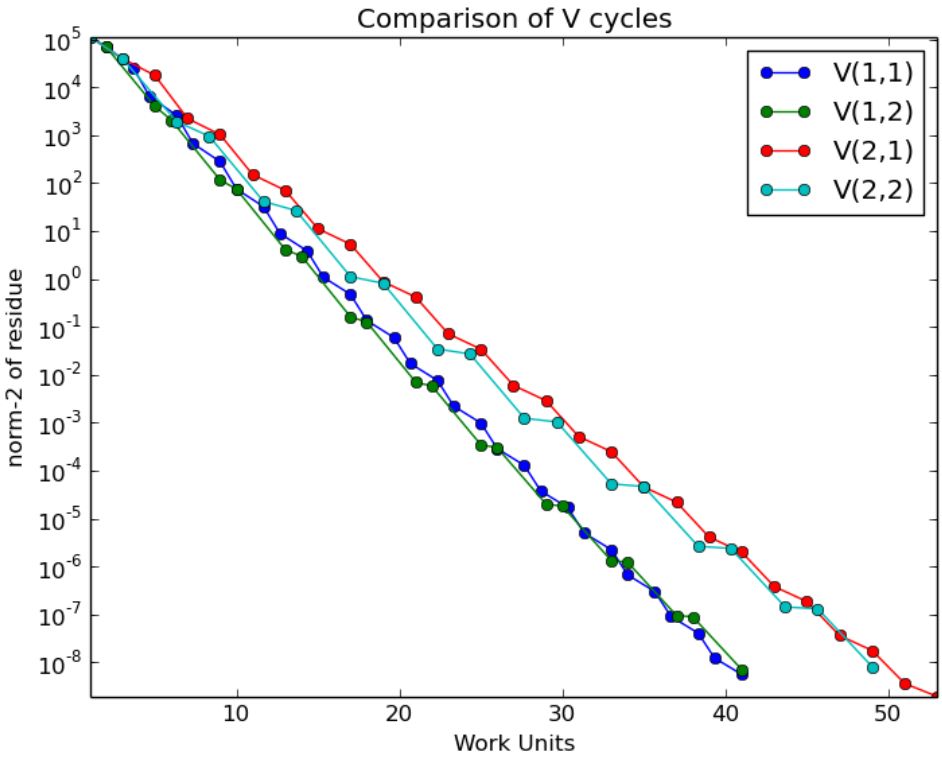

Figure 2.5: Convergence plot of V-cycle scheme for different no. of pre relaxation (ν_1) and post relaxation (ν_2) sweeps (V(ν_1, ν_2))

Table 2.1: Time elapsed for convergence for different grid sizes for V-Cycle scheme

Grid Size	No. of grid points	Time (s)	Ratio
257 × 257	66,049	0.03	-
513 × 513	263,169	0.11	3.67
1025 × 1025	1,050,625	0.48	4.36
2049 × 2049	4,198,401	2.07	4.31
4097 × 4097	16,785,409	8.43	4.07
8193 × 8193	67,125,249	33.8	4.01

Table 2.2: Time elapsed for convergence for different grid sizes for FMG V-Cycle

Grid Size	No. of grid points	Time (s)	Ratio
257 × 257	66,049	0.03	-
513 × 513	263,169	0.16	5.33
1025 × 1025	1,050,625	0.79	4.94
2049 × 2049	4,198,401	3.67	4.65
4097 × 4097	16,785,409	6.33	4.45
8193 × 8193	67,125,249	69.78	4.27

2.5.3 Inference

- From figure 2.4, it is clear that multigrid methods perform better than conventional iteration.
- The Full Multigrid V-cycle was the fastest algorithm to produce the solution.
- Iterations without using any multigrid methods shows a time complexity of $O(n^2)$, as seen from table 2.3. As the no. of grid points become 4 times, the time taken increases to approximately 16 times.
- The time elapsed for FMG scheme is found to vary linearly with grid size as seen in table 2.2. Hence it has a time complexity close to $O(n)$

Table 2.3: Time elapsed for convergence for different grid sizes for iteration without using multigrid

Grid Size	No. of grid points	Time (s)	Ratio
33 × 33	1,089	0.02	-
65 × 65	4,225	0.34	17
129 × 129	16,641	5.43	15.97
257 × 257	66,049	90.93	16.75

- The time elapsed for V-cycle scheme is found to scale in between the above two schemes (Ref table 2.1). This is in close match with the theoretical prediction of $O(n \log n)$ [2]

2.5.4 Conclusion

Various multigrid algorithms were explored in this section and there performances were computed. The solver was tested on Laplace equation on a unit square and was run on 67 million grid points.

Chapter 3

Parallel Programming with OpenMP and CUDA

3.1 Introduction

This section discusses about application of parallel programming in numerical simulations. The report is on shared memory architecture of parallel programming using OpenMP and CUDA C/C++. OpenMP is an API for shared memory parallel programming in C/C++ and Fortran. It is ideal for running on multi core CPUs. CUDA is a parallel computing model developed by Nvidia and is implemented by the graphics processing unit(GPU) the produce. The GPU is a special-purpose CPU, designed with over 100+ processors so that a single instruction works over a large block of data (SIMD/Single Instruction Multiple Data), all of them applying the same operation.

3.2 Basics of Parallel Programming

3.2.1 Data Parallelism

Data Parallelism refers to mode of parallelism where the data is distributed across many processors and all the processors execute the same set of instructions on the data. In the iterative methods discussed in section 2.3, the same calculation is carried out on each grid point independently. Hence this can be easily parallelised.

3.2.2 Data Race

A *data race* is said to occur when two or more threads access the same memory location at the same time with atleast one thread using it for writing. This may lead to ambiguous results and hence should be avoided. This can happen if the wrong algorithm has been chosen to do the parallelism. Taking the case of the iterative methods discussed in section 2.3, the Gauss Seidel iteration will lead a data race conditions. Whereas the Jacobi and Red Black Gauss Seidel methods are suited for parallel computing.

3.2.3 Reduction

Reduction refers to given a associative binary operator $\odot$ and an ordered array $s = [a_1 a_2 \dots a_n]$, reduce $(\odot, s)$ returns $a_1 \odot a_2 \odot \cdots \odot a_n$. This is commonly used when taking sum or finding maximum of residuals at every grid point. The common procedure of summing one-by-one takes $O(n)$ operations. But with parallel processors reduction can be done using map reduction algorithm which takes only $O(n \log(n)$ operations and each operation can be performed independently.

3.3 OpenMP

OpenMP is fairly easy to implement. It comes along with standard gcc compiler suite. It can be invoked using *-fopenmp* flag during compiling. Some important aspects of OpenMP are discussed below.

3.3.1 Parallelising for loops

In OpenMP parallelising a for loop involves using the following directive before a for loop.

```
# pragma omp parallel for
```

for example

```
# pragma omp parallel for
for( int i = 0; i < 1000; ++i )
{
    ...
}
```

It splits the for loop counter into sections and distributed it across the processors (threads) to work on independently.

3.3.2 Reduction

This is implemented in OpenMP using the *reduction* clause. For example, to find the sum of an array

```
# pragma omp parallel for reduction(+:sum)
for( int i = 0; i < 1000; ++i )
{
    sum += array[i];
}
```

and to find the maximum of an array

```
# pragma omp parallel for reduction(max:m)
for( int i = 0; i < 1000; ++i )
{
    m = max(m, array[i]);
}
```

3.4 CUDA

3.4.1 Programming Model

A typical program has the following two parts-

Kernel or Device Code

A kernel is a function callable from the host and executed on the CUDA device simultaneously by many threads in parallel.

Host Code

The host code is the code run on the CPU. The typical routine of a host code is as follows

1. Initialise the device (GPU)
2. Allocate memory on GPU
3. Copy data from the host memory to GPU memory
4. Execute the kernel code on GPU
5. Copy the processed data from GPU memory to host memory
6. Free memory on GPU

3.4.2 Implementing data parallel tasks

Here is sample kernel for performing Jacobi iterations

```
__global__ void Jacobi( float* input,  float* output,
    unsigned int Ni, unsigned int Nj, float h)
{
    unsigned int i = threadIdx.x + blockIdx.x * blockDim.x; // Y - ID
    unsigned int j = threadIdx.y + blockIdx.y * blockDim.y; // X - ID

    unsigned int iPrev = (i-1)  * Nj + j; // Previous Y element
    unsigned int iNext = (i+1)  * Nj + j; // Next Y element

    unsigned int jPrev = i * Nj + j-1; //Previous X element
    unsigned int jNext = i * Nj + j+1; // Next X element

    unsigned int index = i * Nj + j; // Current element

    if( i > 0 && j > 0 && i < (Ni-1) && j <(Nj-1))
        output[index] = 0.25f * (input[iPrev] + input[iNext]
        + input[jPrev] + input[jNext] - 4*h*h);
}
```

The parts of the code is explained below

Keyword *__global__* declares the function as a kernel to be called from the host and executed on the GPU

```
__global__ void Jacobi( float* input,  float* output,
    unsigned int Ni, unsigned int Nj, float h)
```

Each kernel get access to the variables *threadIdx*, *blockIdx* and *blockDim*. The below code calculated i and j indices from these variables.

```
{
    unsigned int i = threadIdx.x + blockIdx.x * blockDim.x; // Y - ID
    unsigned int j = threadIdx.y + blockIdx.y * blockDim.y; // X - ID
```

The kernel code can access only one dimensional array. Hence i and j has to converted to its 1D equivalent as given below

```
    unsigned int iPrev = (i-1)  * Nj + j; // Previous Y element
    unsigned int iNext = (i+1)  * Nj + j; // Next Y element

    unsigned int jPrev = i * Nj + j-1; //Previous X element
    unsigned int jNext = i * Nj + j+1; // Next X element

    unsigned int index = i * Nj + j; // Current element
```

Check boundary nodes

```
    if( i > 0 && j > 0 && i < (Ni-1) && j <(Nj-1))
```

Perform the Jacobi iteration

```
        output[index] = 0.25f * (input[iPrev] + input[iNext]
        + input[jPrev] + input[jNext] - 4*h*h);
}
```

This code is called from the host as follows.

```
        Jacobi<<<dimGrid,dimBlock>>>(in, out, Ni, Nj, h);
```

3.4.3 Reduction

Reduction in CUDA can be done using routines in 'Thrust' library that come with the *cuda-toolkit*.

3.5 Results

The multigrid algorithms implemented in the previous section was modified with CUDA to run on the GPU. The GPU used was Nvidia GeForce GT 630M. The code was also extended to use OpenMP and was tested on a multicore CPU (Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz, 2 cores)

3.5.1 Benchmarks

3.5.2 Conclusion and Remarks

- OpenMP and CUDA has increased computation performance significantly.
- OpenMP code ran 1.29× faster on 2 cores.

Table 3.1: Time elapsed (in s) for convergence for different grid sizes for FMG V-Cycle with CUDA and OpenMP implemented

Grid Size	Grid points	Serial Code	OpenMP (1 core)	OpenMP (2 cres)	CUDA
65×65	4225	0.085	0.095	0.075	0.079
129×129	16,641	1.255	1.301	0.744	0.703
257×257	66,049	6.261	6.282	3.489	4.492
513×513	263,169	36.558	36.688	28.377	26.355

- CUDA code ran 1.39× faster.

Chapter 4

Application of Variational Principles for Plane Compressible flow Calculations

4.1 Introduction

The classical problem of steady inviscid subsonic flow past an aerofoil can be formulated in two ways. The usual formulation is as a boundary value problem consisting of a set non linear partial differential equations and a set of boundary conditions. However, it can also be formulated in terms of complementary variational principles. This report describes a variational method for obtaining numerical solutions. The method consists of replacing the infinitely dimensional variational problem by a finitely dimensional problem by means of finite differences, and an approximate maximizing function is then found by standard methods.

4.2 Formulation

The boundary value problem for plane subsonic flow past an aerofoil can be formulated as follows.

Let $\vec{u} = (u_1, u_2)$ be the velocity vector in the Cartesian coordinate system. Far from the aerofoil, C $\vec{u} = (U, 0)$ where U is a constant.

For an irrotational flow, a velocity potential can be defined as

$$\vec{u} = \nabla\phi$$

The pressure and density are defined by p and ρ respectively. The speed of sound is defined by

$$c^2 = \frac{dp}{d\rho}$$

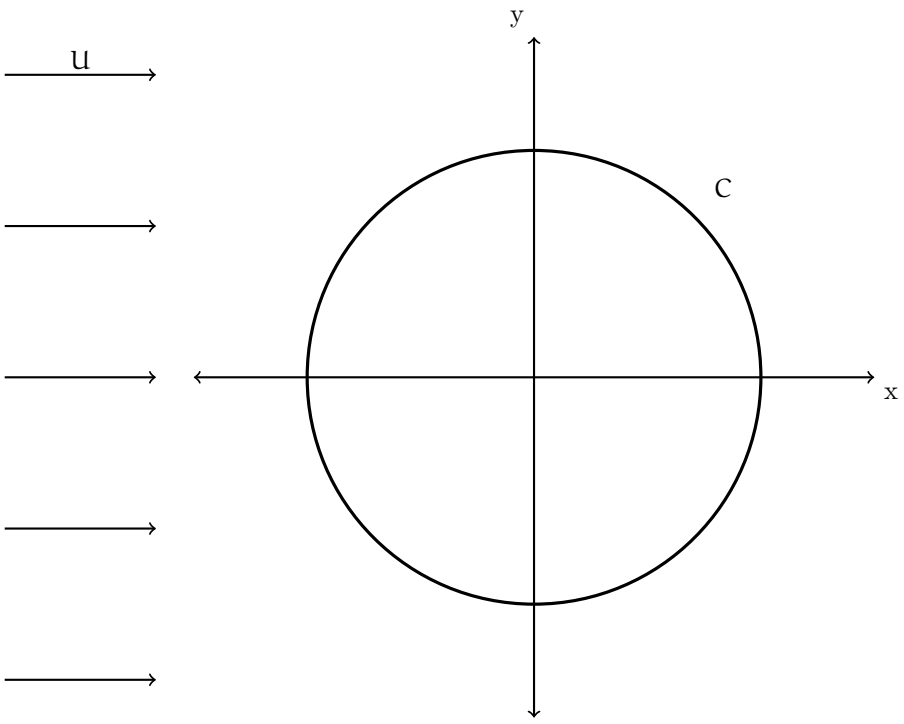

Figure 4.1: Sketch of the flow field

$$p = p_0 \left(1 - \frac{q^2}{2\beta c_0^2}\right)^{\alpha}$$

$$\rho = \rho_0 \left(1 - \frac{q^2}{2\beta c_0^2}\right)^{\beta}$$

where

$$q^2 = \vec{u}.\vec{u}, \qquad \alpha = \frac{\gamma}{\gamma - 1}, \qquad \beta = \frac{1}{\gamma - 1}$$

the suffix 0 indicates stagnation values.

The boundary value problem for ϕ is equivalent to a variational problem of maximizing the integral

$$J[\phi] = \int_{R_1} p\,dV + \int_B \phi h\,dA \tag{4.1}$$

(dV = area element, dA = arc length element of $B = C$). Then $J[\phi]$ is a maximum if $\nabla.(\rho\vec{u} = 0$ and $\rho\vec{u}.\hat{n} = h$ on B. Here the normal mass flu h is prescribes on B such that

$$\text{outflow} = \int_B h\,dA = 0$$

If the flow region R_1 becomes infinite, the variational integral (4.1) becomes unbounded. To remove this difficulty, the integral can be formulated as follows

$$J[\phi] = \int\int_{\infty} [p - p_\infty + p_\infty.\nabla(\phi - \phi_\infty)]dxdy \tag{4.2}$$

where

p_∞ = pressure at infinity,
ρ_∞ = density at infinity,
ϕ_∞ = potential for a uniform stream
ϕ_0 = potential for incompressible flow past C.

To extend the method to a general class of aerofoils, a conformal mapping from the aerofoil C onto a unit circle can be used. Let $z = x + iy$ and $\sigma = r(\cos\theta + i\sin\theta)$, then,, the transform modulus is given by

$$T = \left|\frac{dz}{d\sigma}\right| = (x_r^2 + y_r^2)^{\frac{1}{2}}$$

The Jacobian of the transformation is

$$J = \frac{\partial(x, y)}{\partial(r, \theta)} = x_r y_\theta - x_\theta y_r$$

Since the transformation is conformal

$$y_\theta = r x_r \qquad \text{and} \qquad y_r = -\frac{1}{r}x_\theta$$

so

$$T^2 = x_r^2 + \frac{1}{r^2}x_t heta^2$$

and

$$J = r\left(x_r^2 + \frac{1}{r^2}x_\theta^2\right)$$

Hence

$$J = rT^2 \tag{4.3}$$

The coordinates r, θ are orthogonal so the element of length $ds = |dz|$ is given by

$$ds^2 = h_1^2 dr^2 + h_2^2 d\theta^2$$

Also

$$ds^2 = |dz|^2 = \left|\frac{dz}{d\sigma}\right|^2 |d\sigma|^2$$

$$= T^2(dr^2 + r^2 d\theta^2)$$

Therefore

$$h1 = T \qquad \text{and} \qquad h_2 = rT$$

so

$$\nabla\phi = \frac{1}{T}\left(\hat{r}\phi_r + \frac{\hat{\theta}}{r}\phi_\theta\right)$$

Since

$$q^2 = (\nabla\phi)^2$$

and

$$\phi = U(r\cos\theta + \chi)$$

we have

$$q^2 = \frac{U^2}{T^2}\left[1 + 2\cos\theta.\chi_r - \frac{2}{r}\sin\theta.\chi_\theta + \chi_r^2 + \frac{1}{r^2}\chi_\theta^2\right] \tag{4.4}$$

Also

$$p = p_0\left(1 - \frac{q^2}{2\beta c_0^2}\right)^\alpha$$

and by using the relation

$$p_0 = \left(\frac{\rho_0}{\rho_\infty}\right)^\gamma p_\infty$$

we get after some manipulation

$$p = p_\infty\left[1 + \frac{(\gamma-1)M_\infty^2}{2T^2}\left(T^2 - 1 - 2\cos\theta.\chi_r + \frac{2}{r}\sin\theta.\chi_\theta - \chi_r^2 - \frac{1}{r^2}\chi_\theta^2\right)\right]^\alpha \tag{4.5}$$

where the free stream Mach number M_∞ is defined by

$$M_\infty^2 = \frac{2\beta U^2}{2\beta c_0^2 - U^2}$$

Now

$$\frac{\gamma p_\infty}{\rho_\infty} = c_0^2 - \frac{U^2}{2\beta}$$

Thus since

$$\phi_o = U\left(r + \frac{1}{r}\right)\cos\theta,$$

$$\rho_\infty\nabla\phi_0.\nabla(\phi - \phi_\infty) = p_\infty\frac{\gamma M_\infty^2}{T^2}\left[\frac{r^2-1}{r^2}\cos_t heta.\chi_r - \frac{r^2-1}{r^3}\sin\theta.\chi_\theta\right] \tag{4.6}$$

When the expressions (4.3), (4.5) and (4.6) are used in (4.2), the variational integral becomes

$$J[\chi] = p_\infty\int_0^{2\pi}\int_1^\infty\left\{\left[1 + \frac{(\gamma-1)M_\infty^2}{2T^2}\left(T^2 - 1 - 2\cos\theta.\chi_r + \frac{2}{r}\sin\theta.\chi_\theta - \chi_r^2 - \frac{1}{r^2}\chi_\theta^2\right)\right]^\alpha - 1 + \frac{\gamma M_\infty^2}{T^2}\left(\frac{r^2-1}{r^2}\cos\theta.\chi_r - \frac{r^2+1}{r^3}\sin\theta.\chi_\theta\right)\right\} rT^2\,dr \tag{4.7}$$

The boundary conditions on χ are

$$\begin{aligned} &\frac{\partial\chi}{\partial r} = -\cos\theta \qquad \text{at} \qquad r = 1, \\ &\chi = 0\left(\frac{1}{r}\right) \qquad \text{as} \qquad r \to \infty \end{aligned} \tag{4.8}$$

The local Mach number$M$ and the local non dimensional pressure p_L are given by

$$M = M_\infty \frac{q}{U}\left[1 + \frac{1}{2}(\gamma - 1)M_\infty^2\left(1 - \left(\frac{q}{U}\right)^2\right)\right]^{\frac{1}{2}} \tag{4.9}$$

$$p_L = \frac{p}{p_\infty} = \left[1 + \frac{1}{2}(\gamma - 1)M_\infty^2\left(1 - \left(\frac{q}{U}\right)^2\right)\right]^{\frac{\gamma}{\gamma-1}} \tag{4.10}$$

where q is given by equation (4.4)

The transform modulus for an ellipse is given by

$$T^2 = \frac{1}{r^4}\left[\left(r^2 + \lambda^2\right)^2 - 4\lambda^2 r^2 \cos^2\theta\right] \tag{4.11}$$

where λ^2 is the following function of τ, the thickness ratio of the ellipse,

$$\lambda^2 = \frac{1-\tau}{1+\tau} \tag{4.12}$$

4.3 Numerical Method

The objective of the calculation is to find for given M_∞ and aerofoil shape a function χ which maximizes $J[\chi]$ as given by (4.7) subject to boundary conditions (4.8).

4.3.1 Domain

For non lifting symmetric bodies (about $y = 0$), it is only necessary to treat the interval $0 \le \theta \le \pi$.

Since the derivatives in both directions are approximated by finite differences, it is necessary to have a finite computation region. This is obtained by replacing the infinite integration limit on r by a finite limit R and insisting that the reduced potential χ satisfies an appropriate condition at $r = R$. The simplest condition to impose is that χ equals the reduced potential for incompressible flow at $r = R$.

Thus the variational integral reduces to

$$J[\chi] = p_\infty \int_0^\pi \int_1^R F(r, \theta, \chi_r, \chi_\theta)\, dr \tag{4.13}$$

where

$$F = \left\{\left[1 + \frac{(\gamma-1)M_\infty^2}{2T^2}\left(T^2 - 1 - 2\cos\theta.\chi_r + \frac{2}{r}\sin\theta.\chi_\theta - \chi_r^2 - \frac{1}{r^2}\chi_\theta^2\right)\right]^\alpha \right.$$

$$\left. -1 + \frac{\gamma M_\infty^2}{T^2}\left(\frac{r^2-1}{r^2}\cos\theta.\chi_r - \frac{r^2+1}{r^3}\sin\theta.\chi_\theta\right)\right\} r T^2$$

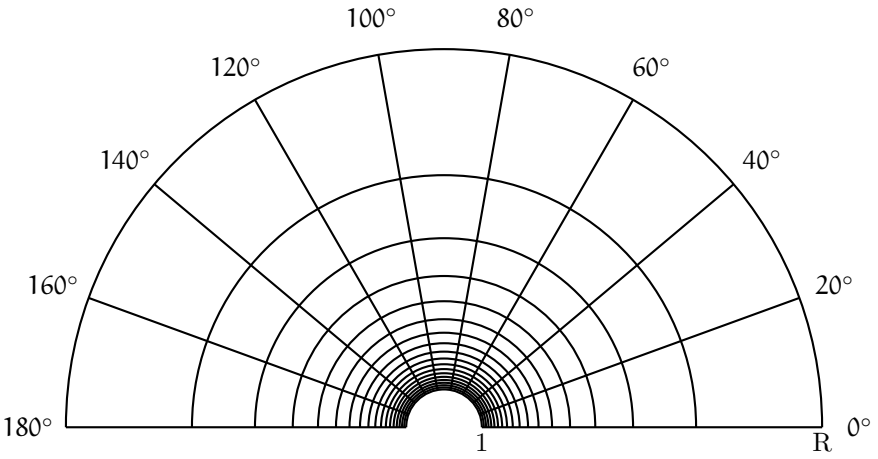

Figure 4.2: Mesh around the body

The boundary conditions are now

$$\begin{aligned} \frac{\partial \chi}{\partial \theta} &= 0 && \text{at} \quad \theta = 0, \pi \\ \frac{\partial \chi}{\partial r} &= -\cos\theta && \text{at} \quad r = 1 \\ \chi &= \frac{1}{R}\cos\theta && \text{at} \quad r = R \end{aligned} \tag{4.14}$$

4.3.2 Discretisation

$$\begin{aligned} r = 1, \ldots, R; \qquad & k_j = r_{j+1} - r_j \\ \theta = 0, \ldots, \pi; \qquad & h_i = \theta_{i+1} - \theta_i \end{aligned}$$

θ varies linearly from 0 to π. r is mapped as $r = 1/\sigma$, where σ varies linearly in the range [1/R, 1].

Thus, the infinitely dimensional variational problem is now replaced by a finite-dimensional problem. Consider four neighbouring points as shown in figure (4.3). The derivatives of χ in the rectangle (i, j) can be approximated as

$$\frac{\partial \chi}{\partial \theta} = \frac{\chi_{i+1,j} + \chi_{i+1,j+1} - \chi_{i,j} - \chi i, j+1}{2h_i}$$

$$\frac{\partial \chi}{\partial \theta} = \frac{\chi_{i,j+1} + \chi_{i+1,j+1} - \chi_{i,j} - \chi i+1, j}{2k_j}$$

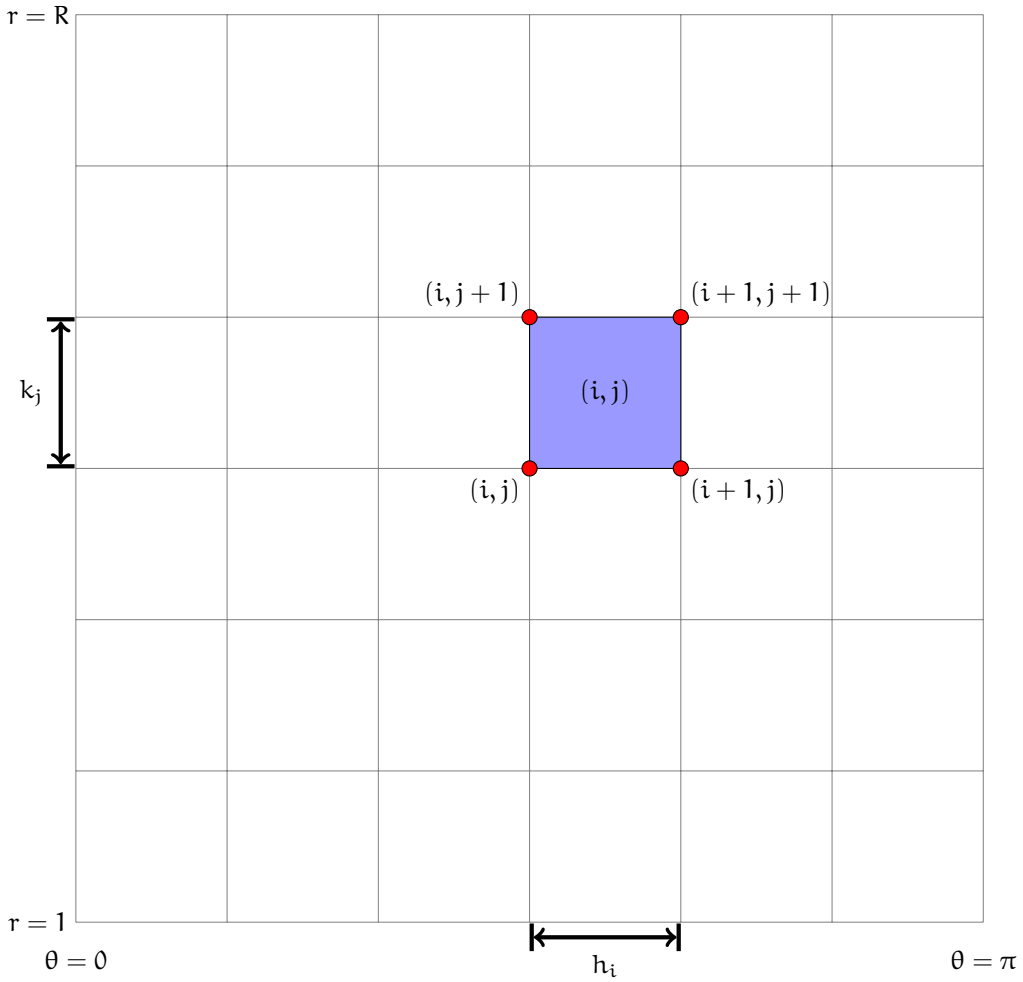

Figure 4.3: Computational domain

For a rectangular region i, j, $J[\chi]$ can be approximated as

$$J_{i,j} = p_\infty \Bigg\{ \Bigg[1 + \frac{(\gamma-1)M_\infty^2}{2T_{i,j}^2} \Bigg(T^2 - 1 - 2\cos\theta_i' \frac{\chi_{i,j+1} + \chi_{i+1,j+1} - \chi_{i,j} - \chi_{i+1,j}}{2k_j} + \frac{2}{r_j'} \sin\theta_i' \frac{\chi_{i+1,j} + \chi_{i+1,j+1} - \chi_{i,j} - \chi_{i,j+1}}{2h_i} - \left(\frac{\chi_{i,j+1} + \chi_{i+1,j+1} - \chi_{i,j} - \chi_{i+1,j}}{2k_j} \right)^2 - \frac{1}{r^2} \left(\frac{\chi_{i+1,j} + \chi_{i+1,j+1} - \chi_{i,j} - \chi_{i,j+1}}{2h_i} \right)^2 \Bigg) \Bigg]^\alpha - 1 + \frac{\gamma M_\infty^2}{T_{i,j}^2} \Bigg(\left(\frac{r_j'^2 - 1}{r_j'^2} \right) \cos\theta_i' \frac{\chi_{i,j+1} + \chi_{i+1,j+1} - \chi_{i,j} - \chi_{i+1,j}}{2k_j} - \left(\frac{r_j'^2 - 1}{r_j'^3} \right) \sin\theta_i' \frac{\chi_{i+1,j} + \chi_{i+1,j+1} - \chi_{i,j} - \chi_{i,j+1}}{2h_i} \Bigg) \Bigg\} \quad (4.15)$$

$$\theta_i' = \theta_i + 0.5h_i; \qquad r_j' = r_j + 0.5k_j$$

$$T_{i,j}^2 = \frac{1}{r_j'^4} \left[(r_j'^2 + \lambda^2)^2 - 4\lambda^2 r_j'^2 \cos^2\theta_i' \right]$$

$$\lambda^2 = \frac{1-\tau}{1+\tau}$$

Boundary conditions for χ-

1.
$$\frac{\partial\chi}{\partial r} = -\cos\theta; \text{ at } r = 1$$
$$\chi_{i,1} = \frac{1}{k_2 + 2k_1} \left[\frac{1}{k_2} ((k_1 + k_2)^2 \chi_{1,2} - k_1^2 \chi_{i,3}) + k_1 (k_1 + k_2) \cos\theta \right] \quad (4.16)$$

2.
$$\frac{\partial\chi}{\partial\theta} = 0 \text{ at } \theta = 0, \pi$$
$$\chi_{m+1,j} = \chi_{m-1,j} \qquad \chi_{0,j} = \chi_{2,j} \quad (4.17)$$

3.
$$\chi = \frac{1}{R}\cos\theta \text{ as } r \to \infty$$
$$\chi_{i,n} = \frac{1}{R}\cos\theta_i \quad (4.18)$$

Summing the contributions for each rectangle, $J[\chi]$ can be approximated as

$$J[\chi] = \bar{J} = \sum_{i=1}^{n-1} \sum_{j=1}^{m-1} J_{i,j} \quad (4.19)$$

Therefore, the values of $\chi_{i,j}$ which maximizes this expression is given by the solutions of the equations

$$\frac{\partial \bar{J}}{\partial \chi_{i,j}} = 0 \tag{4.20}$$
$$i = 1, \ldots, n$$
$$j = 2, \ldots, m-1$$

for a given i, j, the equation to maximize can be written in the form (by summing the contributions from the four surrounding rectangles)

$$g(\chi_{i,j}) = \frac{\partial \bar{J}}{\partial \chi_{i,j}} \sum_{s=(i,j),(i-1,j),(i,j-1),(i-1,j-1)} \Big[\alpha \left(A_s \chi_{i,j}^2 + B_s \chi i, j + C_s \right)^{\alpha-1} \left(2A_s \chi_{i,j} + B_s \right) + D_s \Big] H_s = 0 \tag{4.21}$$

Using Newton Raphson method, an improved estimate is given by

$$\chi_{i,j}^{(q+1)} = \chi_{i,j}^{(q)} - \frac{g(\chi_{i,j}^{(q)})}{g'(\chi_{i,j}^{(q)})} \tag{4.22}$$

$$g' = \frac{\partial g_{i,j}}{\partial \chi_{i,j}} \tag{4.23}$$

4.3.3 Coefficients A, B, C, D, H

The coefficients A, B, C, D, H are given by -

$$\begin{aligned}
A_{i,j} &= -\frac{(\gamma-1)M_\infty^2}{2T_{i,j}^2} \left(\frac{1}{4k_j^2} + \frac{1}{4r'^2_j h_i} \right) \\
A_{i,j-1} &= -\frac{(\gamma-1)M_\infty^2}{2T_{i,j-1}^2} \left(\frac{1}{4k_{j-1}^2} + \frac{1}{4r'^2_{j-1} h_i} \right) \\
A_{i-1,j} &= -\frac{(\gamma-1)M_\infty^2}{2T_{i-1,j}^2} \left(\frac{1}{4k_j^2} + \frac{1}{4r'^2_j h_{i-1}} \right) \\
A_{i-1,j-1} &= -\frac{(\gamma-1)M_\infty^2}{2T_{i-1,j-1}^2} \left(\frac{1}{4k_{j-1}^2} + \frac{1}{4r'^2_{j-1} h_{i-1}} \right)
\end{aligned} \tag{4.24}$$

$$
\begin{aligned}
B_{i,j} = \frac{(\gamma-1)M_\infty^2}{2T_{i,j}^2}\Bigg(&2\cos\theta_i'\frac{1}{2k_j} - \frac{2}{r_j'}\sin\theta_i'\frac{1}{2h_i} \\
&+\frac{1}{2k_j^2}(\chi_{i,j+1}+\chi_{i+1,j+1}-\chi_{i+1,j}) \\
&+\frac{1}{2r_j'^2h_i^2}(\chi_{i+1,j}+\chi_{i+1,j+1}-\chi_{i,j+1})\Bigg) \\
B_{i,j-1} = \frac{(\gamma-1)M_\infty^2}{2T_{i,j-1}^2}\Bigg(&-2\cos\theta_i'\frac{1}{2k_{j-1}} - \frac{2}{r_{j-1}'}\sin\theta_i'\frac{1}{2h_i} \\
&-\frac{1}{2k_{j-1}^2}(\chi_{i+1,j}-\chi_{i,j-1}-\chi_{i+1,j-1}) \\
&+\frac{1}{2r_{j-1}'^2h_i^2}(\chi_{i+1,j-1}+\chi_{i+1,j}-\chi_{i,j-1})\Bigg) \\
B_{i-1,j} = \frac{(\gamma-1)M_\infty^2}{2T_{i-1,j}^2}\Bigg(&2\cos\theta_{i-1}'\frac{1}{2k_j} + \frac{2}{r_j'}\sin\theta_{i-1}'\frac{1}{2h_{i-1}} \\
&+\frac{1}{2k_j^2}(\chi_{i-1,j+1}+\chi_{i,j+1}-\chi_{i-1,j}) \\
&-\frac{1}{2r_j'^2h_{i-1}^2}(\chi_{i,j+1}-\chi_{i-1,j+1}-\chi_{i-1,j+1})\Bigg) \\
B_{i-1,j-1} = \frac{(\gamma-1)M_\infty^2}{2T_{i-1,j-1}^2}\Bigg(&-2\cos\theta_{i-1}'\frac{1}{2k_{j-1}} + \frac{2}{r_{j-1}'}\sin\theta_{i-1}'\frac{1}{2h_{i-1}} \\
&-\frac{1}{2k_{j-1}^2}(\chi_{i-1,j}-\chi_{i-1,j-1}-\chi_{i,j-1}) \\
&-\frac{1}{2r_{j-1}'^2h_{i-1}^2}(\chi_{i,j-1}-\chi_{i-1,j-1}-\chi_{i-1,j})\Bigg)
\end{aligned}
\tag{4.25}
$$

$$
\begin{aligned}
C_{i,j} = 1 + \frac{(\gamma-1)M_\infty^2}{2T_{i,j}^2}\Bigg(T_{i,j}^2 - 1 &- \frac{\cos\theta_i'}{k_j}(\chi_{i,j+1} + \chi_{i+1,j+1} - \chi_{i+1,j}) \\
&+ \frac{\sin\theta_i'}{r_j' h_i}(\chi_{i+1,j} + \chi_{i+1,j+1} - \chi_{i,j+1}) \\
&- \frac{1}{4k_j^2}(\chi_{i,j+1} + \chi_{i+1,j+1} - \chi_{i+1,j})^2 \\
&- \frac{1}{4r_j'^2 h_i^2}(\chi_{i+1,j} + \chi_{i+1,j+1} - \chi_{i,j+1})^2 \Bigg) \\
C_{i,j-1} = 1 + \frac{(\gamma-1)M_\infty^2}{2T_{i,j-1}^2}\Bigg(T_{i,j-1}^2 - 1 &- \frac{\cos\theta_i'}{k_{j-1}}(\chi_{i+1,j} - \chi_{i,j-1} - \chi_{i+1,j-1}) \\
&+ \frac{\sin\theta_i'}{r_{j-1}' h_i}(\chi_{i+1,j-1} + \chi_{i+1,j} - \chi_{i,j-1}) \\
&- \frac{1}{4k_{j-1}^2}(\chi_{i+1,j} - \chi_{i,j-1} - \chi_{i+1,j-1})^2 \\
&- \frac{1}{4r_{j-1}'^2 h_i^2}(\chi_{i+1,j-1} + \chi_{i+1,j} - \chi_{i,j-1})^2 \Bigg) \\
C_{i-1,j} = 1 + \frac{(\gamma-1)M_\infty^2}{2T_{i-1,j}^2}\Bigg(T_{i-1,j}^2 - 1 &- \frac{\cos\theta_{i-1}'}{k_j}(\chi_{i-1,j+1} + \chi_{i,j+1} - \chi_{i-1,j}) \\
&+ \frac{\sin\theta_{i-1}'}{r_j' h_i}(\chi_{i,j+1} - \chi_{i-1,j+1} - \chi_{i-1,j+1}) \\
&- \frac{1}{4k_j^2}(\chi_{i-1,j+1} + \chi_{i,j+1} - \chi_{i-1,j})^2 \\
&- \frac{1}{4r_j'^2 h_{i-1}^2}(\chi_{i,j+1} - \chi_{i-1,j+1} - \chi_{i-1,j+1})^2 \Bigg) \\
C_{i-1,j-1} = 1 + \frac{(\gamma-1)M_\infty^2}{2T_{i-1,j-1}^2}\Bigg(T_{i-1,j-1}^2 - 1 &- \frac{\cos\theta_{i-1}'}{k_{j-1}}(\chi_{i-1,j} - \chi_{i-1,j-1} - \chi_{i,j-1}) \\
&+ \frac{\sin\theta_{i-1}'}{r_{j-1}' h_{i-1}}(\chi_{i,j-1} - \chi_{i-1,j-1} - \chi_{i-1,j}) \\
&- \frac{1}{4k_{j-1}^2}(\chi_{i-1,j} - \chi_{i-1,j-1} - \chi_{i,j-1})^2 \\
&- \frac{1}{4r_{j-1}'^2 h_{i-1}^2}(\chi_{i,j-1} - \chi_{i-1,j-1} - \chi_{i-1,j})^2 \Bigg)
\end{aligned}
\tag{4.26}
$$

$$
\begin{aligned}
D_{i,j} &= -1 + \frac{\gamma M_\infty^2}{T_{i,j}^2}\left[-\left(\frac{r_j'^2-1}{r_j'^2}\right)\frac{\cos\theta_i'}{2k_j} + \left(\frac{r_j'^2-1}{r_j'^3}\right)\frac{\sin\theta_i'}{2h_i}\right] \\
D_{i,j-1} &= -1 + \frac{\gamma M_\infty^2}{T_{i,j-1}^2}\left[\left(\frac{r_{j-1}'^2-1}{r_{j-1}'^2}\right)\frac{\cos\theta_i'}{2k_{j-1}} + \left(\frac{r_{j-1}'^2-1}{r_{j-1}'^3}\right)\frac{\sin\theta_i'}{2h_i}\right] \\
D_{i-1,j} &= -1 + \frac{\gamma M_\infty^2}{T_{i-1,j}^2}\left[-\left(\frac{r_j'^2-1}{r_j'^2}\right)\frac{\cos\theta_{i-1}'}{2k_j} - \left(\frac{r_j'^2-1}{r_j'^3}\right)\frac{\sin\theta_{i-1}'}{2h_{i-1}}\right] \\
D_{i-1,j-1} &= -1 + \frac{\gamma M_\infty^2}{T_{i-1,j-1}^2}\left[\left(\frac{r_{j-1}'^2-1}{r_{j-1}'^2}\right)\frac{\cos\theta_{i-1}'}{2k_{j-1}} - \left(\frac{r_{j-1}'^2-1}{r_{j-1}'^3}\right)\frac{\sin\theta_{i-1}'}{2h_{i-1}}\right]
\end{aligned}
\tag{4.27}
$$

$$
\begin{aligned}
H_{i,j} &= r_j' T_{i,j}^2 h_i k_j \\
H_{i,j-1} &= r_{j-1}' T_{i,j-1}^2 h_i k_{j-1} \\
H_{i-1,j} &= r_j' T_{i-1,j}^2 h_{i-1} k_j \\
H_{i-1,j-1} &= r_{j-1}' T_{i-1,j-1}^2 h_{i-1} k_{j-1}
\end{aligned}
\tag{4.28}
$$

Chapter 5

Conclusion

5.1 Summary

Bibliography

[1] H. Rasmussen, and N. Heys, *Application of a Variational Method in Plane Compressible Flow Calculation*, Aeronautical Research Council Current Papers 1974.

[2] William L. Briggs, Van Emden Henson, and Steve F. McCormick, *A Multigrid Tutorial*, SIAM 2nd Edition 2000.

[3] NVIDIA CUDA Zone, *https://developer.nvidia.com/how-to-cuda-c-cpp*

[4] James Gain, Michelle Kuttel, Sebastian Wyngaard, Simon Perkins and Jason Brownbridge, *Parallel Algorithms*, *http://people.cs.uct.ac.za/ jgain/lectures/Algorithms.pdf*