CFD Simulation of Jet Cooling and Implementation of Flow Solvers in GPU

M D. L O K M A N H O S A I N

Master of Science Thesis
Stockholm, Sweden 2013
CFD Simulation of Jet Cooling and Implementation of Flow Solvers in GPU

M D . L O K M A N H O S A I N

Master’s Thesis in Scientific Computing (30 ECTS credits)
Master Programme in Scientific Computing 120 credits
Royal Institute of Technology year 2013
Supervisor at ABB was Rebei-Bel Fdhila
Supervisor at KTH was Jesper Oppelstrup
Examiner was Michael Hanke

TRITA-MAT-E 2013:30
ISRN-KTH/MAT/E--13/30--SE

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci
Abstract

In rolling of steel into thin sheets the final step is the cooling of the finished product on the Runout Table. In this thesis, the heat transfer into a water jet impinging on a hot flat steel plate was studied as the key cooling process on the runout table. The temperature of the plate was kept under the boiling point. Heat transfer due to a single axisymmetric jet with different water flow rate was compared to cases of a single jet and two jets in 3D. The RANS model in ANSYS Fluent was used with the $k - \epsilon$ model in transient simulation of the axisymmetric model and steady flow for the 3D cases. Two different boundary conditions, constant temperature and constant heat flux were applied at the surface of the steel plate. The numerical results were consistent between 2D and 3D and compared well to literature data. The time dependent simulations for the 3D model requires very large computational power which motivated an investigation of simpler flow solvers running on a GPU platform. A simple 2D Navier-Stokes solver based on Finite Volume Method was written using OpenCL which can simulate flow and heat convection. A standard CFD problem named "Lid Driven Cavity" in 2D was chosen as validation case and for performance measurement and tuning of the solver.
Referat
CFD-simulering av kylning med vattenstrålar och
GPU-implementering av strömningslösare


Transient simulering av 3D modellerna kräver stora datorresurser vilket motiverar en undersökning om enklare strömningsmodeller som kan köra på GPU-plattform. En enkel 2D Navier-Stokes-lösare baserad på Finita Volym-metoden implementerades i OpenCL för simulering av konvektiv värmetransport. Lid Driven Cavity-problemet i 2D valdes för verifiering och tidtagning.
Acknowledgements

I would like to acknowledge the invaluable guidance, encouragement and advice from my supervisor Prof. Jesper Oppelstrup throughout this thesis work. This thesis would not have been possible without his support. So my first and foremost thanks goes to him.

This dissertation is done at ABB. So, I would like to express my gratitude to my supervisor Rebei Bel-Fdhila at ABB for his constant support and assistance in this master thesis work. I am also grateful to ABB and Anders Daneryd, project manager at ABB for giving me this opportunity.

My special thanks goes to Davoud Saffar Shamshirgar, Ph.D student at NA group, KTH for his advice and discussion about GPU programming. I thank Reza Fakhraie and Bahram Saadatfar from Energy Department at KTH for their constructive discussion on GPU.

Finally, I would like to acknowledge the Computational Fluid Dynamics course at Mechanics department at KTH for the Matlab template of a 2D Navier-Stokes solver which helped me a lot to implement the solver on GPU.
## Contents

### List of Figures

1 Introduction ........................................ 1

2 Physical and Theoretical Background .............. 3
   2.1 Physics of Impinging Jet ............................ 3
   2.2 Governing Equations ................................ 4

3 Literature Review .................................. 9

4 Computational domain and Mesh ..................... 13

5 Results and Discussion .............................. 17
   5.1 2D- Axisymmetric model .......................... 18
   5.2 3D 1-jet model .................................. 24
   5.3 3D 2-jet model .................................. 24

6 GPGPU ................................................. 29
   6.1 Different Methods ................................ 30
   6.2 Why OpenCL? .................................... 31
      6.2.1 OpenCL basics ............................... 31
   6.3 Algorithm and Implementation .................... 34
   6.4 Results and Performance .......................... 35
      6.4.1 Speedup .................................... 36
      6.4.2 Profiling and Debugging ...................... 37
      6.4.3 Optimization techniques ...................... 38

7 Conclusion and Future Work .......................... 45

Bibliography .......................................... 47
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Hot rolling steel mill and the Runout table process [14]</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Typical Axisymmetric jet configuration</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Different zone of an axisymmetric impinging jet</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Visualization of hydraulic jump (picture taken from a sink at home)</td>
<td>6</td>
</tr>
<tr>
<td>4.1</td>
<td>Geometry of the 2D-Axisymmetric Case</td>
<td>13</td>
</tr>
<tr>
<td>4.2</td>
<td>Mesh of the 2D-Axisymmetric Case</td>
<td>14</td>
</tr>
<tr>
<td>4.3</td>
<td>Geometry for the 3D 1-jet model</td>
<td>14</td>
</tr>
<tr>
<td>4.4</td>
<td>Geometry for the 3D 2-jet model</td>
<td>15</td>
</tr>
<tr>
<td>4.5</td>
<td>Full mesh for the 3D 1-jet model</td>
<td>15</td>
</tr>
<tr>
<td>4.6</td>
<td>Zoomed mesh for the 3D 1-jet model</td>
<td>16</td>
</tr>
<tr>
<td>5.1</td>
<td>Velocity vector plot of the stagnation zone</td>
<td>18</td>
</tr>
<tr>
<td>5.2</td>
<td>Interface shape for different water velocity at inlet</td>
<td>19</td>
</tr>
<tr>
<td>5.3</td>
<td>Volume Fraction of water liquid for two different water flow rate</td>
<td>20</td>
</tr>
<tr>
<td>5.4</td>
<td>Water film thickness comparison</td>
<td>21</td>
</tr>
<tr>
<td>5.5</td>
<td>Comparison of nusselt number at the stagnation point with theory</td>
<td>22</td>
</tr>
<tr>
<td>5.6</td>
<td>Comparison of Nusselt number with theory upto 5 diameter from impinging point</td>
<td>23</td>
</tr>
<tr>
<td>5.7</td>
<td>Nusselt Number for different water speed at inlet</td>
<td>24</td>
</tr>
<tr>
<td>5.8</td>
<td>Nusselt number comparison between different boundary conditions</td>
<td>25</td>
</tr>
<tr>
<td>5.9</td>
<td>Nusselt Number comparison between 2D and 3D models</td>
<td>25</td>
</tr>
<tr>
<td>5.10</td>
<td>Interface comparison between 2D and 3D models</td>
<td>26</td>
</tr>
<tr>
<td>5.11</td>
<td>Volume of fluid for the 2-jet simulation in 3D shows the splashing at YZ symmetry plane which occurs due to the collision of two waves from two jets</td>
<td>27</td>
</tr>
<tr>
<td>6.1</td>
<td>A typical GPU platform architecture</td>
<td>32</td>
</tr>
<tr>
<td>6.2</td>
<td>Index space for NDRange [20]</td>
<td>33</td>
</tr>
<tr>
<td>6.3</td>
<td>OpenCL memory hierarchies [20]</td>
<td>34</td>
</tr>
<tr>
<td>6.4</td>
<td>Sample output from GPU program</td>
<td>37</td>
</tr>
<tr>
<td>6.5</td>
<td>Speedup for different problem size</td>
<td>37</td>
</tr>
<tr>
<td>6.6</td>
<td>Time taken by each kernel in one time step</td>
<td>38</td>
</tr>
<tr>
<td>6.7</td>
<td>GPU time summary plot which shows the total running and idle time</td>
<td>39</td>
</tr>
<tr>
<td>6.8</td>
<td>Idle time of GPU device (snap of one iteration of CG algorithm)</td>
<td>40</td>
</tr>
</tbody>
</table>
6.9  Time taken by each kernel in one time step for different local workgroup size  .  41
6.10 Total GPU time taken by the kernels in one time step for different Local workgroup size
6.11 Time taken by each kernel in one time step for different Global workgroup size 42
6.12 Total GPU time taken by the kernels in one time step for different Global workgroup size . 42
Chapter 1

Introduction

The runout table is the part of steel production line where the steel sheet is cooled from about 900°C to 200°C while the sheet is running at a certain speed. Due to the big temperature difference boiling is important but very difficult to model. The cooling process in the runout table is crucial for the quality of the rolled sheet and must be controlled accurately. Detailed understanding of the cooling and how it is related to the microstructure of the steel is very important. Many techniques are used. For example, impinging water at different angles with different type of nozzles, sprays, liquid curtains etc. Among them the liquid impinging jet perpendicular to the plate with circular nozzle (often called liquid bars) are common because of their high mass and heat transfer rate and robust technology.

A number of experimental and theoretical studies have been conducted to understand the physics of the heat transfer due to impinging jets over a flat plate. Impinging air or water jet cooling techniques are very useful for many other applications like cooling of combustion engines, electronic microchips. At ABB, simplified models based on semi-empirical heat transfer correlations are used to model and optimize such complex processes, but more first principles models would allow customization of the process by simulation.

Computational Fluid Dynamics (CFD) has been used to analyze limited portions of the process like for a single liquid curtain hitting a moving strip segment. The cooling problem of the steel strip at Runout table is then reduced to a jet impingement cooling problem of a hot flat surface. We have analyzed 2D axisymmetric single jet model, 3D single and double jet model. We have considered single jet simulation to understand the flow behavior and two jets simulation to investigate the jet interaction. For performing these simulations we have used RANS (Reynolds Averaged Navier-Stokes equations) model in ANSYS FLUENT. It was observed that the water film at the wall was very thin, down to a millimeter. To capture the free surface of the water, to preserve conservation and to resolve the water film on the plate, a very fine mesh is required around the free surface and near the sheet surface.
where boundary layer effects are important. Typically the Runout table is consists of several jets. In our study we have considered upto two jets and it was very time consuming to perform a simulation with the available resources since the mesh size became very large. The collision of the film from two jets is an unsteady process and requires a time dependent solver. Considering several jets and important 3D effects, the CFD simulations become very expensive in terms of CPU time and can not be handled in R&D industrial projects. Then we started to find some way to accelerate the process and end up with the GPU platform. It was a question how much speedup can we gain from this new technology. To find answer to that question and to understand the complexity of the platform, at these investigation level we chose to implement a simple flow solver which can simulate flow and temperature. For this task we implemented a 2D time dependent incompressible Navier-Stokes equation solver using Finite Volume method with pressure projection in a staggered grid. For the pressure projection part we have implemented the Conjugate Gradient method. With our solver we solved the standard CFD test problem named Lid driven cavity. And as a GPU implementation package we chose OpenCL since it is a platform independent module.
Chapter 2

Physical and Theoretical Background

2.1 Physics of Impinging Jet

Liquid jet impingement is an effective way for cooling used in many applications because of its capacity to transfer very high heat fluxes by using the latent heat effect in boiling. When a liquid jet hits a wall surface, a sudden increase in pressure occurs which then forces the liquid to accelerate from the stagnation point outwards in a thin liquid film of the order of a millimeter which covers the whole surface. The friction effect over the plate creates a kinematic boundary layer and the temperature difference between the liquid and the surface creates a thermal boundary layer. The thermal and the momentum boundary layer thicknesses are determined by the underlying flow situations. The thickness of the film or the boundary layer is determined by the Reynolds number of the flow. The liquid film thickness can significantly vary for the laminar and the turbulent flow. The velocity of the liquid decreases with distance from the stagnation point and hence the liquid film and also the boundary layer gets thicker. The most important changes between flow regimes occurs within 5-8 jet diameter radial distance from the impinging point. This region is termed as potential core.

Two types of jets are used for cooling, the free surface liquid jet and the submerged jet. For free surface liquid jets into gas the entrainment of the surrounding fluid is minimal whereas it is significant for the submerged case. The shape of the free surface is determined by the gravitational force, pressure, and surface tension. All these forces are affected by the shape of the jet nozzle and also by the speed of the water at nozzle exit. In the present study we will consider the free surface liquid jet. Figure 2.1 shows a typical free surface axisymmetric liquid jet configuration.

In figure 2.1 (b), we can see the Stagnation zone beneath the jet nozzle and outside this region the Radial or parallel flow zone. The width of the stagnation zone depends on the jet diameter, distance between the nozzle and plate and the Reynolds
number. The measurement of this zone is essential because it is the most important zone where most of the complex physics associated with the cooling happens. The parallel / radial flow zone is divided into different regions depending on the film, momentum and the thermal boundary layer thicknesses. Figure 2.2 describes different regions in the parallel flow zone.

Another interesting and important phenomenon is the hydraulic jump, see Figure 2.3. This occurs because of the deceleration of the liquid. The location of the hydraulic jump was determined experimentally and a correlation has been established in [11] as discussed later.

2.2 Governing Equations

The governing equations for fluid flow are the Navier-Stokes equations describing the conservation of mass and momentum. The general form is [16]:

\[
\begin{align*}
\frac{D\rho}{Dt} + \rho \frac{\partial u_i}{\partial x_i} &= 0 \\
\rho \frac{D u_i}{Dt} &= -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho F_i
\end{align*}
\]  
\tag{2.1}

where, \( \tau_{ij} = \mu \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_i} \delta_{ij} \right) \) is the stress tensor, \( \rho \) density, \( u_i \) denotes the velocity, \( p \) pressure, \( F_i \) is the force term and \( \mu \) is molecular viscosity. The summation convention has been used in the equations.
2.2. GOVERNING EQUATIONS

Figure 2.2: Different zone of an Axisymmetric impinging jet [8], where, Region I: The Stagnation zone, Region II: The laminar boundary layer where the momentum boundary layer $\delta$ is smaller than the liquid film thickness $h(r)$, Region III: The momentum boundary layer reaches the film surface, Region IV: This is the region from transition to turbulent where the momentum and the thermal boundary layer both reach the liquid surface, Region V: The flow is fully turbulent, $T_f$ is fluid temperature, $U_f$ is fluid velocity at inlet, $U_m$ is the free stream velocity, $\delta$ viscous boundary layer thickness, $r_0$ radius at which the viscous boundary layer reaches the free surface, $r_t$ the radius at which turbulent transition begins and $r_h$ is radius at which turbulence is fully developed

In Equation (2.1), the derivative term $\frac{D}{Dt} = \frac{\partial}{\partial t} + u_j \frac{\partial}{\partial x_j}$ called the material derivative.

In some cases the density change is negligible and considering $\rho = constant$ the equation (2.1) reduces to the incompressible form [16]:

\[
\frac{\partial u_i}{\partial x_i} = 0
\]

\[
\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \nu \nabla^2 u_i + F_i
\]
Where $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity. In the incompressible form of Navier-Stokes equations, the conservation of mass equation becomes a divergence free constraint for the momentum equation and using that constraint the stress tensor term $\tau_{ij}$ in Equation (2.1) becomes $\frac{\partial^2 u_i}{\partial x_j \partial x_j} = \nabla^2 u_i$.

**Turbulence model equations**

For turbulent flow, most commercial software uses the Reynolds Averaged Navier-Stokes equations (RANS) model. It takes less computational effort than the time-accurate equations and is robust for a wide range of fluid flows. It is derived from the standard equations by averaging after decomposing the flow variables into mean and fluctuating components like $\phi = \bar{\phi} + \phi'$, where $\bar{\phi}$ is the mean (time averaged) and $\phi'$ is the fluctuating component of variables like velocity, pressure or other scalar quantity. This formulation leads to the continuity and momentum equations as follows [41]:

$$\begin{align*}
\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i} (\rho \bar{u}_i) &= 0 \\
\frac{\partial}{\partial t} (\rho \bar{u}_i) + \frac{\partial}{\partial x_j} (\rho \bar{u}_i \bar{u}_j) &= - \frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} [\mu \frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} - \frac{2}{3} \delta_{ij} \frac{\partial \bar{u}_l}{\partial x_l}] + \frac{\partial}{\partial x_j} (-\rho \bar{u}_i \bar{u}_j) 
\end{align*}$$

(2.3)

Note the Reynolds stress term $(-\rho \bar{u}_i \bar{u}_j)$ which requires additional modeling to close.
2.2. GOVERNING EQUATIONS

the RANS equations. The Boussinesq hypothesis can be used to relate the Reynolds stress to the mean velocity gradients as follows:

\[-\rho u'_i u'_j = \mu_t \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \left( \rho k + \mu_t \frac{\partial \sigma_{kk}}{\partial x_k} \right) \delta_{ij} \]  

(2.4)

Where \( \mu_t \) is the turbulent viscosity and \( k \) is the turbulent kinetic energy.

This modeling is available with \( k - \varepsilon \) and \( k - \omega \) in FLUENT which requires to solve another two transport equations, for example for turbulent kinetic energy \( k \) and \( \varepsilon \) turbulent dissipation rate in the \( k - \varepsilon \) model used for all our simulations as turbulence model. There are also several version of the \( k - \varepsilon \) equations. We used the Realizable \( k - \varepsilon \) model since it predicts the spreading rate for axisymmetric as well as planar jets very well, according to Fluent theory guide, Ver 14.5 : sec:4.3.3.2. For details of these equations and the modeling we refer to the Fluent Theory Guide and User Manual [41].

Convective Heat and mass transfer

Our study requires to solve the energy equation with the \( k - \varepsilon \) model. The energy equation is [41]:

\[ \frac{\partial}{\partial t}(\rho E) + \frac{\partial}{\partial x_i}[u_i(\rho E + p)] = \frac{\partial}{\partial x_j} \left( k_{eff} \frac{\partial T}{\partial x_j} + u_i(\tau_{ij})_{eff} \right) + S_h \]  

(2.5)

Where, \( E \) is the Total Energy, \( T \) is temperature, \( k \) is thermal conductivity, \( k_{eff} = k + \frac{C_p \mu_t}{Pr_l} \) the effective thermal conductivity, \( Pr_l \) - Turbulent Prandtl number, \( (\tau_{ij})_{eff} \) the deviatoric stress tensor and \( S_h \) is source term.

In the convective heat transfer model for the Jet impingement, the analyzed quantities are the heat transfer coefficient \( h \) and the non dimensional heat transfer coefficient, the Nusselt number \( Nu \), at the wall surface. The convective heat transfer coefficient is calculated from the heat flux \( q'' \) at the wall surface by,

\[ q'' = h(T_s - T_{\infty}) \]  

(2.6)

where, \( T_s \) and \( T_{\infty} \) are the surface temperature and the free stream fluid temperature respectively.

The local surface heat flux can be obtained by using the Fourier law of heat conduction at the boundary layer by,
The Nusselt number $Nu_d$ based on diameter of the nozzle can be calculated from the heat transfer coefficient $h$ by,

$$Nu_d = \frac{hd}{k_f}$$

(2.8)

where, $d$ is the diameter of the jet nozzle and $k_f$ is the heat conductivity of fluid.

**Interface Capturing method**

The impingement of water jet is a free surface flow which was modeled by the Volume of Fluid method (VOF). It calculates the volume fraction of liquid in all the control volumes (or, cells) and depending on the value of the volume fraction it finds the cells that contain the interface. If the volume fraction $\alpha_q$ for phase $q$ is 0 then the cell is empty, if 1 then the cell is full but if $0 < \alpha_q < 1$ then the cell contains the interface. Then a method called Geometric Reconstruction can be used to approximate the interface in those cells containing the interface by a piecewise linear interpolation approach. Another option could be to solve the VOF equation with the Level-Set method. The VOF-equation that is solved in VOF model is [41]:

$$\frac{1}{\rho_q} \left[ \frac{\partial}{\partial t} (\alpha_q \rho_q) + \nabla \cdot (\alpha_q \rho_q \vec{v}_q) \right] = S_{\alpha_q} + \sum_{p=1}^{n} (\dot{m}_{pq} - \dot{m}_{qp})$$

(2.9)

Where,

$m_{pq}$ is the mass transfer from phase $p$ to phase $q$.

$m_{qp}$ is the mass transfer from phase $q$ to phase $p$.

$S_{\alpha_q}$ is source term and $\rho_q$ is the density for phase $q$.

For further details of the methods used see the FLUENT theory guide [41].
Chapter 3

Literature Review

Much theoretical and experimental work has been done on the impinging jet on a flat surface but numerical simulations are rare in this field. In this chapter we discuss analytical and experimental work by other authors.

Since it is a direct measure of the heat transfer coefficient, the Nusselt number study is very important. The Nusselt number $Nu_0$ at the stagnation point has been described as a function of Reynolds number $Re$ and the Prandtl number $Pr$ by some authors as

$$ Nu_0 = CPr^mRe^n $$

(3.1)

The values of $C$, $m$ and $n$ were fitted to the analytical solution of the boundary layer equations or experiments. Many correlations for the Nusselt number outside the stagnation zone have been established. Outside the stagnation zone the Nusselt number correlations are thoroughly investigated only for laminar flow whereas for turbulent flow different authors summarized different results. It has been observed that turbulent heat transfer is very different from laminar cases. The experimental results follow the analytical correlations very well for laminar jets but not for turbulent jets.

There are two types of boundary conditions in common use, the constant temperature and the constant heat flux at the wall surface. Boiling is not considered since we assume low wall temperature.

Liu and Lienhard in [9] have used an integral method to solve the energy equation analytically and studied the temperature distribution and the Nusselt number using a constant heat flux surface. They have compared the experimental data and the theoretical data for Reynolds number up to $5.27 \times 10^4$ and reported a good agreement with the theory.
Ma et al. in [5] have used an integral method to study the laminar free surface axisymmetric jet considering an arbitrary heat flux condition at the surface. They have established a Nusselt number correlation.

Stevens and Webb in [11] studied the heat transfer coefficient for an axisymmetric liquid jet for a uniform heat flux surface. They have provided a different correlation for the stagnation point Nusselt number that includes the jet to the plate spacing and also the free stream velocity as in equation 5.2. According to these authors, the conservation of mass near the stagnation zone suggests to scale of the stagnation point Nusselt number by using \((u_f/d)\) where \(u_f\) is the free stream velocity. This correlation gives a slightly higher value according to [1].

Lienhard in [8] has summarized the stagnation zone Nusselt number and has discussed turbulent jets. He mentioned that the Nusselt number at the stagnation zone for uniform wall temperature and uniform heat flux boundary conditions are identical with no relation to the distance from the impinging point. He reported good agreement with the theory and experiment for laminar cases, however it was emphasized that the turbulent jets can increase the heat transfer coefficient by 30 – 50%.

Webb and Ma in [10] report a detailed study of the laminar axisymmetric and planar jet. They have also summarized the correlations for different regions given by other authors. They observed the experimental data to obey the theoretical predictions up to a certain diameter before the transition to the turbulence occurs but it deviates for the turbulent jets. However, the laminar jets behaved according to the theoretical prediction. They have summarized that there were some errors in the stagnation zone prediction. According to them the turbulence yields much higher heat transfer and the laminar models then serve as a lower bound for the local heat transfer.

Liu et al. in [1] has investigated the single phase laminar liquid jet heat transfer of the surface at a constant heat flux. They noted that the Nusselt number shows a peak downstream at the turbulence transition point. The peak corresponds to the point where the turbulence becomes fully developed. If the Reynolds number increases, the peak becomes more pronounced and occurs a short distance from the impinging point as we found in our study (Illustrated in chapter5). Ma in [4] also found this sort of hump from his experiments for different fluids with large Reynolds and Prandtl number.

The Nusselt number is heavily dependent on the nozzle diameter [11] and the Prandtl number. For high Prandtl number the regions divided into Figure 2.2 can be different because the Prandtl number relates the thicknesses of the thermal and viscous boundary layer as \(\delta_T = \delta / \sqrt{Pr}\) where \(\delta\) and \(\delta_T\) is the viscous and thermal boundary layer thickness respectively and \(Pr\) is the Prandtl number. The Prandtl number of water at 20°C is 7.0, and in this case the thermal boundary layer is
thinner than the viscous boundary layer [1], [8]. It means that, for this case the Region IV will not occur.

The literature review in this chapter indicates that the heat transfer behavior for turbulent jets is still not very clear. In our simulation cases we have Reynolds number at the jet from $3 \times 10^4$ to $1.5 \times 10^5$ which are turbulent. In Chapter 5 the simulations are compared with the standard correlations from the literature.
Chapter 4

Computational domain and Mesh

2D-Axisymmetric Model:
The computational domain geometry and mesh were built using Ansys Workbench 14.0. The domain was chosen as small as possible to minimize the number of mesh elements. Figure 4.1 shows the geometry and the boundary conditions and Figure 4.2 shows the mesh for the 2D axisymmetric case.

![Figure 4.1: Geometry of the 2D-Axisymmetric Case](image)

The 2D-axisymmetric mesh is structured with perfect quadrilateral cells. A very fine mesh was generated around the pipe down to the wall to capture the interface and in the boundary layer at the wall surface. The mesh quality is presented in table 4.1.

3D Models:
In the 3D model, symmetry was considered and only half of the domain was modeled. The 3D geometry is shown in Figure 4.3, where the XY-plane is the symmetry plane and the ZX-plane is the wall surface. Except the inlet and pipe (see figure) all the other planes were considered as pressure outlet.
In the 3D mesh the cells were concentrated around the jet and at the wall surface as it was in 2D. Hexahedral elements were used and a smooth transition between the cell layers was maintained for better accuracy. The mesh only for the 1-jet model is presented here because the 2-jet mesh does not differ much from the 1-jet mesh. For the 2-jet case we used 2 symmetry planes (XY plane and the left YZ plane) with a fine mesh along the YZ symmetry plane to capture the collision between the two jets.

The mesh for the 1-jet model is presented in Figure 4.5 and 4.6. The mesh quality is presented in table 4.1. The quality of mesh is decided depending on some proper-
ties of the cells in FLUENT. In table 4.1 different column shows different properties where the aspect ratio is the ratio of the largest and smallest edge of a cell. Orthogonal quality and skewness measures cell shape. \( Y^+ \) value is the dimensionless distance from the first cell center to the wall. The mesh quality is good according to FLUENT user manual.

<table>
<thead>
<tr>
<th>Model</th>
<th>Total elements</th>
<th>Cell type</th>
<th>Max Aspect Ratio</th>
<th>Min Orthogonal quality</th>
<th>Max Skewness</th>
<th>( Y^+ ) value at surface</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D-Axisymmetric</td>
<td>26890</td>
<td>All Quad</td>
<td>37</td>
<td>1</td>
<td>No skewed cell</td>
<td>2~15</td>
</tr>
<tr>
<td>3D 1-jet</td>
<td>1,357,632</td>
<td>All Hex</td>
<td>44</td>
<td>0.74</td>
<td>0.55</td>
<td>4~30</td>
</tr>
<tr>
<td>3D 2-jet</td>
<td>1,621,835</td>
<td>All Hex</td>
<td>51</td>
<td>0.70</td>
<td>0.57</td>
<td>4~33</td>
</tr>
</tbody>
</table>

Table 4.1: Mesh information for all the models

Figure 4.4: 3D 2-jet geometry (a) Real computational domain (b) Computational domain after applying the symmetry

Figure 4.5: Full mesh for the 3D 1-jet model
Figure 4.6: 3D 1-jet mesh (a) Zoomed (beam) (b) Zoomed (pipe top) (c) Zoomed Symmetry plane
Chapter 5

Results and Discussion

In this chapter the results for the 2D-axisymmetric model, 3D 1-jet model and 3D 2-jet model are presented. Results of the 2D-axisymmetric model are compared with the analytical solutions from the literature and the results from the 3D models are compared with that of the 2D-axisymmetric model.

For the 2D Axisymmetric model we have done 5 simulations with a constant temperature and one simulation with constant heat flux boundary condition at the steel surface. For the single jet 3D model a constant temperature of 100°C and a constant heat flux boundary condition were considered with a water speed 5 m/s at the inlet. For the double jet model the mesh size was increased significantly and this is why only one simulation with constant temperature boundary condition was done for this case. All the 2D Axisymmetric simulations and the 3D single jet simulations were done using pseudo transient solver in ANSYS FLUENT. On the other hand, the 3D 2-jet case were simulated in a transient solver because the 2-jet simulation is an unsteady process due the collision of the film from two jets. the pseudo-transient solution method is a form of implicit under-relaxation for steady-state cases. This method uses a pseudo-transient time-stepping approach and it allows us to obtain solutions faster and more robustly.

The sheet temperature was considered low to avoid boiling. Heat transfer properties: the heat transfer coefficient and the Nusselt number were calculated for different water speeds in the range 1 - 5 m/s . The water temperature at inlet was 20°C and the plate was kept at constant temperature of 100°C. Constant heat flux at the strip surface was also considered for a nozzle speed of 5m/s.

In all the simulations the jet diameter \( d = 30mm \) and the distance from nozzle to wall surface \( z = 200mm \) were held constant. The radial distance of the domain outlet from the impinging point was chosen as 600mm. So, in our case the results are presented up to 20 nozzle diameters from the impinging point.
CHAPTER 5. RESULTS AND DISCUSSION

5.1 2D- Axisymmetric model

![Velocity vector plot of the stagnation zone](image)

Figure 5.1: Velocity vector plot of the stagnation zone

The impinging point is directly beneath the jet nozzle center with highest pressure and at the same time the velocity vanishes: a stagnation point. Because of the high pressure the water accelerates parallel to the surface. Most of the physical changes happen in the stagnation zone. The velocity gradient creates shear stress which leads to high heat transfer according to the Reynolds analogy between shear and heat transfer. At the stagnation zone there is a very thin thermal boundary layer which remains constant [10], but when the water accelerates parallel to the surface the boundary layer grows and the flow regime transitions from stagnation to boundary layer [1], where the maximum heat transfer occurs. After that the heat transfer coefficient decreases gradually downstream. The characterization of the regions in figure 2.2 is basically for a laminar jet and the zonal division can be much different for high Reynolds number. The flow behavior at the stagnation zone can be observed from the velocity vector plot of the stagnation zone in figure 5.1.

Interface

The Volume of Fluid (VOF) method was used to simulate the free surface flow with a high resolution interface capturing method named Geometric Reconstruction for all our simulations. The interfaces between the two phases (air-water) for different inlet water velocity ($u_0 = 1ms^{-1} - 5ms^{-1}$) are shown in figure 5.2. The jet diameter decreases downwards from acceleration by gravity,

$$\frac{d}{d_0} = (1 + 2g(z-z_0)/u_0^2)^{-1/2}$$

is clearly seen for lower water speed (See figure 5.3). Here, $d_0$ is the initial diameter and $z_0$ is the distance between the jet and the strip. The narrowing of the jet at impact influences the position of highest heat transfer coefficient as discussed later.

Film Thickness

The correlation (5.1) was given by Liu et al. in [1] for the liquid film thickness $h(r)$ on the surface starting from the radial position $r_0 = 0.177dRe_d^{1/3}$.

$$h(r) = 0.1713 \frac{d^2}{r} + \frac{5.1417}{Re_d} \frac{r^2}{d}$$  \hspace{1cm} (5.1)
5.1. 2D- AXISYMMETRIC MODEL

We have compared the correlation in (5.1) with the simulations data in Figure 5.4 and found a very good agreement with a maximum 3\% deviation. But of course the correlation in equation 5.1 does not consider the hydraulic jump. It can be used as a very good estimator for the liquid film thickness from $r_0$ to the position before the hydraulic jump.

**Nusselt Number**

As mentioned before that, many author have established many correlations for different regions showed in figure 2.2 which produce a smooth curve for the Nusselt number but for high Reynolds numbers the region division does not follow in the same way that is why it was very difficult to get a smooth curve by the existing correlations. Webb and Ma in [10] have considered the stagnation zone as $r < 0.4to0.8d$ where d is the diameter of the jet and r is radial distance from the impinging point whereas Liu et al. in [1] have defined the stagnation zone as $r < 0.787d$.

The Nusselt number $Nu_0$ at the stagnation point from our simulations has been compared with three other existing correlations as a function of Reynolds number in figure 5.5. Since the thermal boundary layer is constant at the stagnation zone, the Nusselt number results are valid for either uniform wall temperature or the uniform heat flux( [10]). Therefore the comparison at the stagnation zone can be made regardless of the boundary conditions.
Steven and Webb (1989) in [11] have given a correlation for the Nusselt number at the stagnation zone $N_u_0$ from their experiments for a turbulent jet,

$$N_u_0 = 2.67Re_d^{0.57} \left( \frac{z}{d} \right)^{-1/30} \left( \frac{u_f}{d} \right)^{-1/4} Pr^{0.4}$$  \hspace{0.5cm} (5.2)

where $z$ is the distance from nozzle to the plate, $u_f$ is free stream velocity and $d$ is the diameter of the jet.

According to Joo, P. H. ([12]), Steven et al. (1992) have given another correlation for stagnation heat transfer which is:

$$N_u_0 = 0.93Re_d^{1/2} Pr^{0.4}$$  \hspace{0.5cm} (5.3)

Lui et al. in [1] established the correlation for stagnation zone from their laminar analysis, as:

$$N_u_0 = 0.787Re_d^{1/2} Pr^{1/3}$$  \hspace{0.5cm} (5.4)
5.1. 2D- AXISYMMETRIC MODEL

![Graph](image1.png)

From figure (5.5) it can be seen that the correlation (5.2) gives the highest Nusselt number since it considers the free stream velocity of the liquid with the correlation, which is much higher than the Nusselt number from our simulations. On the other hand, the other two correlations give lower values compared to our results because these correlations are for laminar flow and according to [8] the turbulent Nusselt number may increase by 30% to 50% from the laminar one. According to all these correlations we have reasonable results.

Liu et al. gave the correlations for different regions of the boundary layer for the laminar cases with a constant heat flux surface in [1], which are presented in table 5.1. They provided correlations to measure the border of different regions too.

Now, in figure 2.2, \( r_0 = 0.1773dRe_d^{1/3} \) is the border of Region II and \( r_t = \frac{1.2 \times 10^3d}{Re_d^{0.422}} \) is the border of Region III. But if \( Re_d > 1.1 \times 10^5 \) then \( r_t < r_0 \) and the region distribution in figure 2.2 becomes invalid ( [1]), as in our case, because we have Reynolds number \( 1.5 \times 10^5 \). Therefore comparisons were made with the correlations up to Region II. Figure 5.6 compares the Nusselt number from the simulation results for Reynolds number \( 1.5 \times 10^5 \) with the correlations in table 5.1 up to 5d from the impinging point. We can see that the patterns of the Nusselt number curves are similar and the maximum value occurs almost at the same place but the levels are higher, as expected, for turbulent flow.
Figure 5.5: Comparison of nusselt number at the stagnation point with theory

<table>
<thead>
<tr>
<th>Region</th>
<th>Range</th>
<th>( N_u_d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stagnation Zone</td>
<td>( 0 \leq r/d &lt; 0.787 ), ( Pr &gt; 3 )</td>
<td>( 0.797Re_d^{4/7}Pr^{1/3} )</td>
</tr>
<tr>
<td>Transition: Stag. to b.l.</td>
<td>( 0.787 &lt; r/d &lt; 2.23 )</td>
<td>( \frac{27}{80}Re_dPr\frac{r}{\delta}^{1/4} )</td>
</tr>
<tr>
<td>b.l. Region II</td>
<td>( 2.23 &lt; r/d &lt; 0.177Re_d^{-0.422} )</td>
<td>( 0.632Re_d^{1/2}Pr^{1/3}\left(\frac{r}{d}\right)^{1/2} )</td>
</tr>
</tbody>
</table>

Table 5.1: Correlation for Nusselt number for different boundary layer regions by [1]

Figure 5.7 shows the Nusselt number for different flow rates in 2D-axisymmetric cases with a constant temperature boundary condition at the plate. We can see that all the curves have similar pattern but the maximum value occurs at different points. As the water velocity decreases, the peak gets closer to the axis. This is because the water beam diameter decreases for lower speed which has an impact on the stagnation zone radius. The definition of the stagnation zone is not very clear from the literature. All the papers consider the radius of the stagnation zone as a constant with respect to the jet diameter. They did not consider the flow rate or the Reynolds number while measuring the stagnation zone except [11]. From our
5.1. 2D- AXISYMMETRIC MODEL

Simulation results it can be summarized that the radius of the stagnation zone has connection with the minimum diameter of the water beam between jet nozzle exit and the surface. The diameter of the water beam is influenced by the velocity at the nozzle exit and the distance between the nozzle and the plate. If this two are considered with the stagnation zone Nusselt number measurement then it is obvious to get very high values as we can see in figure 5.5 and 5.6.

**Hydraulic jump**

In Figure 5.2 we can see a hydraulic jump at 500mm for 1ms$^{-1}$ water velocity and Reynolds number $3 \times 10^4$. A correlation was given as a function of Reynolds number by Stevens and Webb in [11]. The position of the hydraulic jump was measured for $Re$ up to $2 \times 10^4$. The correlation agrees with experiment with an average error of 15%.

$$r_{hj} = 0.0061dRe_d^{0.82}$$ \hspace{1cm} (5.5)

Their correlation was derived for a nozzle diameter up to 8.9mm whereas we have the nozzle diameter 30mm. The nozzle diameter would influence the location of the hydraulic jump. We have 5 different Reynolds numbers and we got hydraulic jump at 500mm only for the smallest $Re$ $3 \times 10^4$. For higher $Re$ the jump is expected to be outside the computational domain as the position is proportional to $Re$ according to the relation 5.5.
5.2 3D 1-jet model

The 3D models are compared with the 2D-Axisymmetric models, considering the 2D cases as reference for the 3D cases. Two simulations were done at a water velocity $5 \text{ m/s}$. One is with constant heat flux and the other with constant temperature boundary condition. In figure 5.8 it can be seen that the 3D models give higher Nusselt number at the stagnation point, otherwise the numbers are almost same downstream. The Nusselt number curves for the 2D and 3D with constant temperature boundary condition are the same outside the stagnation zone and the same holds for the constant heat flux boundary condition, although the constant heat flux boundary condition gives a slightly higher value than the constant temperature downstream for both 2D and 3D cases. However, the Nusselt number at the stagnation zone is found to be the same regardless of the boundary conditions for both 2D and 3D cases as Webb and Ma described in [10]. In figure 5.10 we can see that the film heights are exactly the same for 2D-Axisymmetric, 3D 1-jet and 3D 2-jet model with water velocity $5 \text{ m/s}$.

5.3 3D 2-jet model

The 3D 2-jet model seemed to be very difficult to simulate in Fluent. When the films from two jets collide, there was too much splashing which leads to convergence
5.3. 3D 2-JET MODEL

Figure 5.8: Nusselt number comparison between different boundary conditions

Figure 5.9: Nusselt Number comparison between 2D and 3D models

problem. Because of that problem the jet distance was increased to 400\textit{mm} to reduce the splashing but it was very difficult to have a steady solution. Therefore, the simulation was re-done in transient mode. Finally the convergence was good but simulation is still questionable because of the formed wave due to the collision. The residual for the continuity equation was about $10^{-2}$ and the residual for the other equations were below $10^{-5}$ but they were fluctuating all the way, this might
be because of the wave after collision. In figure 5.9 we can see that the 3D-2 jet models agree with the 3D-1 jet model. Figure 5.11 shows the effect at the corner of the two symmetry plane which tells the flow pattern after collision. Some water was flowing outside the domain along the line of collision through the top surface which was a pressure outlet, which results in back flow into the domain and this is somewhat unphysical for the simulation. Then we increased the domain height but still the water was flowing outside. After all, it is obvious to have overall high heat transfer with two jet compared to the 1-jet model because of the position of the jets and high water flow rate. In Figure 5.9 the Nusselt number curve ends at a location \( \frac{r}{d} = 13.33 \) since the jets are 400 mm apart.

**Convergence and Conservation**

For all the simulations the conservation of different properties were ensured. The residual for the continuity equation was about \( 10^{-3} \) and for rest of the equations it was below \( 10^{-6} \). For the 2D-Axisymmetric and 3D 1-jet models there was almost no fluctuation. However, there were fluctuations in all the residuals for the 3D 2-jet case as mentioned before.
5.3. 3D 2-JET MODEL

Figure 5.11: Volume of fluid for the 2-jet simulation in 3D shows the splashing at YZ symmetry plane which occurs due to the collision of two waves from two jets
Chapter 6

GPGPU

The growth in computational power in the last few years has enabled great strides in the development of computational sciences, and recently, hardware for graphics has attracted attention. Graphic cards are generally designed for high resolution graphics or for video games, but graphic card manufacturers like Nvidia and AMD have developed cards that can be used for general purpose computing. This has created the GPGPU (General Purpose Graphic Processing Unit) parallel programming paradigm. The latest graphic card has more than 2000 GPU cores and a single precision floating point performance of 3.95 Tflops and double precision floating point performance 1.31 Tflops. The card fits into a single workstation, consumes less power than a cluster but provides a great speed up compared to a single CPU. Different programming modules also have been developed to use this new platform efficiently. At present there are two , CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language) which can be used with C, C++, python and many other programming languages. Another package named OpenGL can be used to visualize simulation outputs with very high resolution. A number of simulations have been done in molecular dynamics, fluid dynamics, acoustics and many other branches using GPGPU.

In this study we have implemented a 2D Navier-Stokes solver in GPGPU in the C programming language. In this paper different methods that are implemented by other people on GPGPU have been discussed. Moreover, the achieved speedup, the performance of our solver along with the discussion of various optimization techniques and the difficulties in coding are presented. We have chosen the OpenCL package because it is an open standard supported by several manufacturers whereas CUDA is supported only by NVIDIA.
6.1 Different Methods

This study focuses on methods for Computational Fluid Dynamics. The governing equations for any simulation is the Navier-Stokes equation. There are simplified methods based on grids or particle or hybrid. Methods based on grids are called the Eulerian methods and the methods based on particle are called Lagrangian methods. Both methods have their own advantages and disadvantages but some available hybrid methods combine both to get the benefit from both type of methods.

Grid or Mesh based methods

Methods like Finite Difference, Finite Volume, Finite Element etc. are developed based on grids. In CFD the most widely used method is the Finite Volume method (FVM). It is a purely mesh dependent method and while solving Navier-Stokes equation one solves the continuity, momentum or vorticity and energy equation (if required) in a checker board (staggered grid) fashion. For time dependent flows the common way to solve the discrete equations is the so-called projection or pressure correction method (Sometimes called splitting methods). For detailed derivation and discretization of FVM and Projection method we refer to [16]. In our OpenCL implementation this method was chosen to solve the time dependent incompressible Navier-Stokes equation (2.2) and Conjugate Gradient (CG) Method was implemented to solve the algebraic system in the pressure correction part. For the details of the CG method we refer to [17].

Simplified models like Jos Stam’s 'Stable fluids' in [27] have been developed for faster fluid simulations. This is a very popular method for fluid simulation in Computer graphics and animation. Another popular method for fluid flow in computer graphics is the so called staggered Marker and Cell (MAC) grid for free surface flows. Nowadays, also the level set method is widely used for interface tracking. A two phase flow has been simulated using level set method on GPU in [35].

A different class of method for fluid simulation is the Lattice Boltzmann method. These methods execute cellular automata which emulate the Boltzmann equation and can be averaged to produce solutions to the Navier Stokes equation. Several implementations of these methods have been done using GPGPU, for example in [32].

Particle based methods

Fluid particle simulation comes in two major varieties: Smoothed Particle Hydrodynamics (SPH) and Discrete Vortex Methods (DVM). SPH uses fluid particles to represent flow whereas DVM uses vortex particles called 'vortons' which represent tiny vortex elements. These methods are usually implemented using a tree structure and the Fast Multipole Method (FMM) to calculate interaction between particles, which is a very well known method for particle simulation. [33], [34] have
6.2. WHY OPENCL?

used SPH, [29], [30] used FMM and [36], [38], [39] used vortex particle method to solve different problems on GPU.

Hybrid methods

This method uses the backtracking style called Semi-Lagrangian technique. It transfers the vortex particle into a grid and then solves the Poisson equation on the grid to get the velocity field. These methods are known as Particle in Cell (PIC) or Vortex in Cell (VIC). Details of implementation techniques for these methods are described in [40]. Some details of hybrid methods, Multiresolution method, adaptive meshing techniques, interface capturing particle methods and about different boundary conditions have been discussed in [19].

6.2 Why OpenCL?

CUDA and OpenCL both have their own advantages. CUDA is a bit older than OpenCL which has more libraries and a bit easier to use for the programmers, but it is only supported by Nvidia cards. On the other hand, OpenCL is a platform independent module. It is a programming package which can run on heterogeneous systems that use GPGPUs, GPUs, CPUs, or any other parallel programming device. Once you write your code, you can run it on any parallel device which is really great, this is why we have chosen OpenCL for our implementation. In contrast, writing an optimized code using OpenCL is much more difficult than CUDA. One has to think about the portability which requires extra care while designing the algorithm and may have performance penalties. It is very hard to write a code which can run on different platforms with the same performance. It might be wise to leave the task to the platform it works better and also switching between platforms to get the best out of the available resources.

6.2.1 OpenCL basics

In order to understand the OpenCL, one needs to know some OpenCL terms, programming modules and the memory model. There are two parts of an OpenCL program: Programming for the Host and Programming for the device (GPUs). The first part is writing the host program like a normal C/C++ program that runs on CPU and the other part is writing the kernels that use some special syntax other than normal C/C++ that run on GPUs. The host sets up the environment needed to run kernels on GPUs, create and compile the kernels, transfer the required data from CPU memory to GPU memory back and forth, creates and maintains a command queue to initiate any task to the GPU devices, issues command to GPU devices and finally collects all the results when done.
Figure 6.1 describes a typical GPU platform architecture where a workstation/cluster contains one or more CPU that works as Host and one or more GPUs that works as the parallel device. Each GPU device has many compute units (Streaming multi-processors for Nvidia) and each multiprocessor has many processing elements (PE). Processing element corresponds to Scaler processor (SP) for Nvidia.

**Workitems**: The basic unit of work in an OpenCL device.

**Workgroups**: Workitems are further organized into different workgroups. There are two different work groups that need to be defined when running an OpenCL kernel: A Local and a Global work group. Global work group size must be divisible by the Local work group size in each dimension.

Local workgroup: A Local work group is a group of the workitems that will be executed by a single multiprocessor (Or, Compute unit (OpenCL name)).

Global workgroup: A Global work group is executed concurrently by the available multiprocessors in the device. The local and Global workgroup defines the Index space.

**Index Space**

Index space is a grid of workitems where each of the workitems has a local id and a global id. Any kernel runs according to the workitems ids. Figure 6.2 describes a N-Dimensional Range index space where Lx, Ly are the local ids within a local workgroup in x and y dimension. Gx, Gy are global ids and Wx and Wy are local workgroup ids in x and y direction.
### 6.2. WHY OPENCL?

OpenCL Memory Model

OpenCL memory model is the most crucial part in OpenCL programming. Figure 6.3 shows the OpenCL memory model together with the interaction between the memory model and the platform model.

**Host memory:** This is the CPU memory and only visible by the Host.

**Global memory:** The largest memory in device where all the workitems in all workgroups and the host have read/write permission.

**Constant memory:** This a part of global memory that is allocated and initialized by the host which remains constant throughout the kernel execution. Workitems have read-only permit to this memory.

**Local memory:** Memory of a multiprocessor which is shared among all the workitems in a local workgroup.

**Private memory:** This is a private memory to a single workitem.

For more details of OpenCL programming see [18], [20] and [21].
6.3 Algorithm and Implementation

The incompressible Navier-Stokes equation (2.2) and the Energy equation were discretized using the splitting method described in [16]. This is a straightforward implementation of the staggered grid projection method and not discussed here. The performance of the solver depends totally on the pressure correction part where one has to solve the Poisson equation in each time step (see Figure 6.6). The Conjugate Gradient method was implemented to solve the system of equations. The Conjugate Gradient method algorithm is presented in a table style to link the kernel name with the algorithm in Table 6.1.

In the implementation of CG kernels the main bottleneck was the global reduction in the scalar product and CG is heavily dependent on scalar products. The local memory was used to aid the global reduction in an efficient way (For details see [24]).

Note: There is no global synchronization and no barrier synchronization within the workitems in different workgroups, which forced us to write many kernels to ensure global synchronization and correct data access in different steps of the algorithm. The convergence criterion for CG was checked by the host only in each 50th iteration because global synchronizations are so expensive. Algorithms with fewer synchronizations and barriers perform better; The Jacobi iteration with multi-grid acceleration should be investigated as an alternative to CG.
Conjugate Gradient method

The algorithm in table 6.1 solves a linear system of equations $Ax = b$. But in this case we do not need the matrix because the laplacian in the poisson equation $\Delta P = b$ is just a 5-point stencil operation. In order to solve the system we need only the initial guess $x$ - all zeros - and the right hand side $b$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Name of Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: initial guess $x_0$ and the right side $b$.</td>
<td></td>
</tr>
<tr>
<td>Step 1: $r = b$</td>
<td>kernel_initialize</td>
</tr>
<tr>
<td>Step 2: $p = r$</td>
<td></td>
</tr>
<tr>
<td>Step 3: $rsold = r*r$</td>
<td>kernel_r_dot_r, kernel_reduce_rsold</td>
</tr>
<tr>
<td>Repeat</td>
<td></td>
</tr>
<tr>
<td>Step 4: $Prod = A*p$ (5-point stencil)</td>
<td>kernel_apply_BC2P, kernel_stencil</td>
</tr>
<tr>
<td>Step 5: $\mu = p*Prod$</td>
<td>kernel_p_dot_Ap, kernel_reduce_den</td>
</tr>
<tr>
<td>Step 6: $\alpha = rsold/\mu$</td>
<td>kernel_alpha</td>
</tr>
<tr>
<td>Step 7: $x = x + \alpha*p$</td>
<td></td>
</tr>
<tr>
<td>Step 8: $r = r - \alpha*Prod$</td>
<td></td>
</tr>
<tr>
<td>Step 9: $rsnew = r*r$</td>
<td>kernel_r_dot_r, kernel_reduce_rsnew</td>
</tr>
<tr>
<td>Step10: $\varepsilon = \sqrt{rsnew}$</td>
<td>This part is done by Host in each 50th iteration</td>
</tr>
<tr>
<td>if $\varepsilon &lt; tolerance * |b|$ then STOP</td>
<td></td>
</tr>
<tr>
<td>Step11: $\beta = rsnew/rsold$</td>
<td>kernel_beta</td>
</tr>
<tr>
<td>Step12: $p = r + \beta*p$</td>
<td></td>
</tr>
<tr>
<td><strong>Output</strong>: The solution is $x$.</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: CG algorithm and the corresponding kernels

### 6.4 Results and Performance

A serial version of the Navier-Stokes solver was written in C from a matlab version and then it was implemented for GPU using OpenCL. The performance of the parallel solver was measured on the 'Lid Driven Cavity' standard test problem. Single precision accuracy was considered in the entire GPU code. Table 6.3 shows the time taken by the OpenCL and the serial code. It can be seen that the conjugate gradient method was taking more iteration to converge for the serial version but using -O3 flag it takes the same number of iteration to converge compared to OpenCL.
code. There are two things to note here. GPU has only 32-bit registers whereas a 64-bit computer has 64-bit registers and 80 bits in the implementation of IEEE floating point arithmetic. Moreover, from the basic of error analysis one can see that the parallel environment and serial environment should give slightly different results specially when working with scaler product of two vectors. In our case the CG method is heavily dependent on the scaler products. On the other hand, -O3 flag does some loop reordering and loop unrolling which might have similar issues like parallel environment, this is why it takes same number of iterations as GPU code. But, if we think about the convergence then the number of iterations can be considered as an internal matter and the comparison between the GPU and the serial code is fair.

<table>
<thead>
<tr>
<th>Workstation Configuration</th>
<th>GPU Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel(R) Xeon(R) W3503</td>
<td>Nvidia Tesla C2075</td>
</tr>
<tr>
<td>Clock Speed: 2.4GHz</td>
<td>Computer Unit: 14</td>
</tr>
<tr>
<td>Memory : 9GB</td>
<td>Memory: 6GB</td>
</tr>
<tr>
<td>CPU cores: 2</td>
<td>GPU cores: 448</td>
</tr>
</tbody>
</table>

Table 6.2: Computer configuration

<table>
<thead>
<tr>
<th>Dimension</th>
<th>GPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nx=Ny</td>
<td>Time(sec)</td>
<td>CG iter.</td>
</tr>
<tr>
<td>32X32</td>
<td>0.24</td>
<td>201</td>
</tr>
<tr>
<td>64X64</td>
<td>0.40</td>
<td>351</td>
</tr>
<tr>
<td>128X128</td>
<td>0.72</td>
<td>651</td>
</tr>
<tr>
<td>256X256</td>
<td>1.38</td>
<td>1251</td>
</tr>
<tr>
<td>512X512</td>
<td>6.45</td>
<td>2451</td>
</tr>
<tr>
<td>1024X1024</td>
<td>41.60</td>
<td>4851</td>
</tr>
<tr>
<td>2048X2048</td>
<td>308.70</td>
<td>9651</td>
</tr>
</tbody>
</table>

Table 6.3: Time taken by the GPU and Serial code for different problem size for 10 time steps

Figure 6.4 shows sample output from the GPU solver visualized using Matlab. The initial condition for the velocity was 1m/s at the lid on the top and a linear temperature profile was chosen as an initial condition for the temperature. Since the flow is laminar \((Re = 100)\) the solution becomes steady after about 10sec.

### 6.4.1 Speedup

The speedup of the parallel solver was measured comparing with the unoptimized version of the serial solver and also with the optimized version using -O3 flag. Due to time constraint, the parallel solver was not optimized but the optimization techniques have been discussed in Section 6.4.3. The performance of the solver can
6.4. RESULTS AND PERFORMANCE

Figure 6.4: Output from the GPU program (a) Problem domain (Filled with water) (b) Contour of Velocity profile (c) Contour of Temperature profile at t=20s with Re=100

be optimized as discussed later. One can have an idea about the speedup using a GPU device compared to a serial code on the CPU from Figure 6.5.

![Speedup Graph]

Figure 6.5: Speedup for different problem size

6.4.2 Profiling and Debugging

There are a few debugging and profiling tools for OpenCL program available now. The debugger gDebugger was used to debug the code and the Nvidia Compute Visual Profiler was used to profile our code. The debugger and the profiler was used both in windows and linux platform (Ubuntu 12.04). Some useful profiling output will be presented in section 6.4.3.
6.4.3 Optimization techniques

Before starting the optimization process one needs to profile the code and find the bottleneck of the program. The profiler will give detailed analysis of every kernel that is running on the GPU device.

Kernel optimization

In the Navier-Stokes solver we have 11 kernels for the Conjugate Gradient method and 10 kernels for the main solver. Figure 6.6 shows the time taken by each kernel in one time step. From Figure 6.6 we can see that most of the time was taken by the kernels of the pressure correction part, so the CG kernels are the first targets for optimization.

![Gpu Time Summary Plot](image)

Figure 6.6: Time taken by each kernel in one time step

Now, the question is, how much computational capacity of the GPU device have we used? From the GPU time summary plot in Figure 6.7 one can see that we have used only about 14% of the total time and about 86% of the time the GPU device was idle. It tells us how much room do we have for optimization. Next question is, why the GPU is idle? In Figure 6.8 different colors are showing the running time of different kernels and the white spaces are the idle time between each of the kernel run instances. So the number of kernel should be minimized to reduce the idle time. One can try to launch the next kernel before the previous has finished, to overlap the idle time by computation. But that is possible only if there is no data dependency of the previous kernel. The kernels in CG have data dependencies. It
6.4. RESULTS AND PERFORMANCE

is wise to choose algorithms or methods that are simple and can be written in a way so that they have little data dependency. On the other hand, writing a very big kernel is not recommended because big kernels will require many registers per workitem but a single multiprocessor has only few registers.

![GPU Time Summary Plot](image)

**Figure 6.7:** GPU time summary plot which shows the total running and idle time

**NDRange optimization**

The Global Workgroup size and the Local Workgroup size both have a serious impact on the performance of the program. The better choice of the above parameters are chosen, the better the performance is. There is no universal formula for the optimal size of the Global and Local workgroup. The only way is to experiment with different numbers keeping in mind that the global workgroup size must be divisible by the local workgroup size in each dimension. Before choosing the numbers you have to know the hardware very well that you are going to use.

Local workgroup size has certain relation to the alignment of the memory in the device and the amount of register that is available in a single multiprocessor. One Local work group will run on one multiprocessor and one multiprocessor has a limited amount of register which is divided among all the work items in a local workgroup. If the requirement of the register exceed the available register, then it
fails to launch the kernel.

Nvidia says that, the local workgroup size should be chosen as a multiple of 32 workitems in order to achieve optimal computing efficiency and facilitate coalescing (access consecutive memory addresses). [22]

From Figure 6.9 and 6.10 one can understand the impact of the Local workgroup size on the performance. In this example the Global workgroup size was kept constant (128x64) and the Local workgroup size was chosen 16x8 and 32x4. The later one takes total 12.31% less time than the former one. Which is because the later one is a multiple of 32 and it helps to access the memory in a coalesced (that is, access the consecutive memory addresses) way which gives high memory throughput.

On the other example the Local workgroup size was kept constant 32x4 (Since it worked better in previous example) and the Global workgroup size was taken 128x64 and 64x64. From Figure 6.11 and 6.12 we can see that the case 64x64 took about 40% more time than the case 128X64. Which is because, the actual problem size is much bigger than the maximum global workitems (Total workitems in a Global workgroup) and to cover the whole problem one has to iterate on the data. There are certain limitations on the global workgroup size. We can not just take the global workgroup as big as we wish. Now, for example we want to solve the Navier-Stokes equation on a 512x512 grid. The GPU card (Nvidia Tesla C2075) we used for our purpose supports $2^{16} - 1$ maximum workitems which is much smaller than 512X512. Only possible way to solve this problem is to loop over the data. If we iterate on 512X512 grid with a 64x64 global workgroup size than we have to iterate more compared to 128x64 global workgroup size, this is why global workgroup size 64x64 takes more time.

In order to get the best performance from the available hardware one has to tune
6.4. RESULTS AND PERFORMANCE

and experiment with different Global and Local workgroup size and try to find the optimized value for the used hardware. There is a calculator from Nvidia named occupancy calculator which can be used to estimate a suitable workgroup size.

![Comparison Summary Plot](image)

**Figure 6.9:** Time taken by each kernel in one time step for different local workgroup size

![Total GPU time taken by the kernels in one time step for different Local workgroup size](image)

**Figure 6.10:** Total GPU time taken by the kernels in one time step for different Local workgroup size

**Memory level optimization**

One needs to write the kernels in a way so that the Global memory access pattern is coalesced to gain high memory throughput. The width of the thread (workitems) block and the width of the accessed array must be a multiple of 32 to access the memory in fully coalesced way [23]. Another way of optimizing a kernel is to use low latency and high bandwidth memory whenever possible. This is done by using the local memory which is shared among all the workitems in a local workgroup. This is 100 times faster than the global memory access but only if there is no bank conflict. Bank conflict happens when two or more workitems try to access the
same memory address at the same time. For details of Bank conflict see [20], [23]. In our implementation, local memory was used for all the global reductions in the scalar products. The kernel for the matrix-vector product (which is a 5-point stencil operation) can also be written using local memory.

**Instruction optimization**

To optimize the code at a instruction level, one should minimize the branching as much as possible. Using the single precision instead of double precision is recommended if single precision is enough for the purpose. Instruction can be reduced significantly by using the native functions like divide, sine, cosine, bitwise operator instead of % (The modulo operator) etc. `cl_nvPragma_unroll` extension can also be used to unroll any loop to avoid branching within a warp (32 workitems). Try
6.4. RESULTS AND PERFORMANCE

to use shared memory instead of global memory to reduce memory instructions.
For more optimization techniques see *Nvidia OpenCL Best Practice Guide* [22] and
*OpenCL Programming Guide for the Cuda Architecture* [23].
Chapter 7

Conclusion and Future Work

Comparing the theoretical and simulation results it can be concluded that Heat transfer CFD simulation is very difficult because of the uncertainty in heat flux prediction. It is very hard to estimate the heat transfer coefficient using RANS model, specially when the flow is turbulent. One can try DNS (Direct Numerical Simulation) simulation for better approximation but it is almost impossible to conduct DNS simulation for industrial research. GPU usage can be a good solution in terms of computational power. But it is very hard to write an efficient solver using GPU platform because one has to take care of every hardware details. There are other drawbacks for this platform also, for example the performance is poor for complex algorithms. From our case of CG algorithm it can be seen that the global reduction and applying boundary conditions have great impact on performance. So, it is wise to choose simple algorithm that requires less synchronization to get better performance and speedup.

In this present study we have studied the heat transfer phenomenon of a single phase impinging water jet with a stationary surface. As we mentioned before that, because of very high temperature the boiling phenomenon occurs in the Runout table and the steel surface has a certain speed. So the next task would be to analyze the heat transfer for a moving surface and also introduce high temperature to study the boiling phenomenon. The 2D-Navier Stokes solver written in GPU here can be extended to 3D. Other possible way could be to implement the particle or hybrid methods described in section 6.1 in GPU which may reduce the computational effort significantly.
Bibliography


BIBLIOGRAPHY


[41] FLUENT theory guide and user manual, ANSYS FLUENT 14.5.