CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Performance Optimization of Numerical Solution Algorithm for Microchannel Flow Boiling

A thesis submitted in partial fulfilment of the requirements

For the degree of Master of Science in Mechanical Engineering

By

Matt Blomquist

December 2017

(Official end of the semester)

Copyright by Matt Blomquist 2017

California State University, Northridge

The thesis of Matt Blomquist is approved:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Dr. Abhijit Mukherjee |  | Date |
|  |  |  |
| Dr. <NAME> |  | Date |
|  |  |  |
| Dr. <NAME> |  | Date |

Table of Contents

Signature Page

Acknowledgements

Dedication

List of Tables

List of Figures

Abstract

Chapter 1: Introduction

Chapter 2: Objective

Chapter 3: Mathematics

Chapter 4: Numerical Model

Chapter 5: Results

Chapter 6: Conclusion

References

Abstract

Performance Optimization of Numerical Solution Algorithm for Microchannel Flow Boiling

By

Matt Blomquist

Master of Science in Mechanical Engineering

In recent years, there has been significant advancements computation hardware that allows for massive parallelism in computational fluid dynamics algorithms. However, as academia and industry increase the size and complexity of simulations computational performance needs to be enhanced through algorithmic parallelism to avoid significant bottlenecks in solution runtime. In this study, a microchannel flow boiling algorithm is investigated for potential improvements in runtime via algorithmic parallelism. A parametric study has been performed to quantify the impact of multiple Krylov-Subspace solution methods for linear systems using a variety of preconditioning methods chosen to improve speed of convergence. These solutions methods are then compared with solutions from state of the art software packages such as ANSYS, NASTRAN, and COMSOL. The highest impact solution algorithms are then implemented into the microchannel flow boiling algorithm and compared with the serial, iterative solution methodology. The results show a significant decrease in computational duration while resulting in improved accuracy and stability. Additionally, the improved microchannel flow boiling algorithm is tested on a variety of parallel hardware architectures to identify potential scalability.

# Introduction

Since the 1960s, Computational Fluid Dynamics (CFD) has been a tool implemented by scientists and engineers to solve a variety of complex problems related to fluid flow, heat and mass transfer, chemical reactions, and related phenomena. In general, CFD is a numerical solution methodology that is primarily associated with making predictions of heat transfer and fluid-flow processes that do not have exact solutions.

The aerospace industry was an early adopter of CFD, primarily using the technology to study the effects of drag and lift on aerodynamic bodies. For aircraft manufacturers like Boeing, Lockheed Martin, and Northrup Grumman, CFD provided a tool to significantly reduce costs associated with running experimental investigations. Simple mistakes during the design of aircraft were found prior to developing prototype which reduced the lead time for new designs and reduced development program costs by approximately X%.

Typical difficulties in the early days of CFD… cost of computer time, etc.

In recent years, CFD has been a tool to study microscale phenomena that is difficult to experiment on and requires significant investment on specialized equipment. Microchannel flow boiling is one of the aforementioned phenomena that heavily use CFD. Microchannel flow boiling explores two-phase fluid flow in channels with a diameter the size of a human hair.

Microchannel flow boiling and microchannel heat transfer is of general research importance today because the electronic industry is pursuing miniaturization. As the devices get smaller, the requirements to dissipate heat increase by a square of the area reduction. Microchannel heat sinks offer a possible solution to that issue. People have seen huge improvements with both experiments and numerical work, but more needs to be understood about the dynamics of bubbles in these heat sinks to create sustainable and reliable solutions.

My research team has been hard at work exploring these phenomena, but we are significantly limited by the arcane nature of our code. For most scenarios, our microchannel boiling numerical algorithm will run for approximately 4 days before generating enough information for the group to get the data we need. Changing and upgrading our hardware has only been able to provide slight improvements to the overall run time. The overall accuracy of the algorithm is very good as seen in multiple papers that have been published by our group, but the sluggish performance (total duration of runtime) limits our ability to perform analysis and research.

Extremely long runtimes are typical for many CFD applications where accuracy is paramount. NASA is most notable for creating these extremely intensive CFD simulations that can take months to complete. However, NASA has also been very proactive in creating solutions for these runtimes. The typical ideas they come up with are changing hardware, changing algorithms, and using parallel processing.

Improving hardware or using better hardware to solve computationally expensive simulations is a very common approach. However, in most hardcore research areas this solution doesn’t typically yield significantly beneficial results. In industry, where less computationally intensive workloads can be readily found, this can be a very good option.

In early years, using external servers were the only way to solve some of these problems. Places like NASA AMES and Lawrence Livermore Labs were homes to massive supercomputers used specifically for these types of problems. However, with the capability expansion of personal computers, a lot of smaller organizations were able to achieve acceptable results for purchased machines. Only within the last few years has server offloading come back into style in the form of cloud computing. New cloud computing services offer huge benefits for reasonable prices and provide normal people with access to ridiculously powerful machines. Still, a computationally expensive algorithm ends up wasting time which is money, so we will explore improvements to the algorithms themselves.

Pantankar revolutionized the numerical computation world when he developed an algorithm for solving the Navier-Stokes equations. This method was called the SIMPLE method or Semi-Implicit Method for Pressure Linked Equations. This algorithm combined the momentum and energy equations of Navier-Stokes with the continuity equation. Therefore, it was magic and allowed for numerical solution to complex fluid dynamic problems.

Pantankar developed a method using the Thomas algorithm for solving a system of linear equations that he deemed the line-by-line TDMA (tri-diagonal matrix algorithm) method. For uni-dimensional problems, this algorithm provides direct results. However, when solving 2- and 3-dimensional problems, this algorithm becomes an iterative method that provides approximate results. To solve the 2- and 3-dimensional equations, sweeps need to be made line-by-line through the higher dimension grids. This method is the time consuming part of CFD.

In recent years, however, the world of linear algebra has come up with some very elegant methods of solving these types of systems. Krylov developed a number of really awesome sub-space methods for solving large, sparse linear systems that look almost identical to the equations that arise from partial differential equations. PDEs are, in essence, what the Navier-Stokes equations are, so these methods can be readily applied to these problems. In some year, the GMRES method came out and was quickly integrated into a number of commercial software packages for CFD. This method provided a huge improvement and people were very happy. In some other year, the Bi-Conjugate Method was created based on the success of the Conjugate Gradient method that was used for other types of PDEs. There were even fancy projects that came up with software tools that would choose the best solution method so the researchers didn’t have too. Some of these are PARDISO and Mathworks MLDIVIDE algorithm.

In the industrial community, it is often preferable to use very general tools that can be applied to multiple different problems with ease. For this reason, tools like ANSYS, COMSOL, and NASTRAN use algorithms like PARDISO and others for ease of use. In research groups, however, a lot of software will be created to solve very specific problems with very high levels of accuracy. So these groups end up choosing algorithms like GMRES or BiCGStab as they fit the particular problem. In the end, it doesn’t even matter that much. All of these algorithms can be further improved by incorporating parallelism.

Parallel solution algorithms have been around since the 60s and CFD people have been really interested in them since the start of CFD. Parallel solutions can scale with hardware which creates really exciting possibilities for improving speed. The improvements in the form of parallel processing come from two main avenues: improvements in hardware and improvements to algorithms.

The Intel Corporation has been really diligent working on HPC solutions for many years and nearly dominates the market for high-performance CPUs. One of their main developments in the last decade has been vectorization for multicore machined. Vectorization is a simplistic form of parallelization where data gets lined up on the chip and processed at the same time. For independent-type algorithms, this can work great. It’s like using an FPGA in electronics. However, for algorithms with serial dependencies it doesn’t offer much benefit.

One really cool thing about this is that a number of basic libraries for linear algebra programming (linear system solutions) offer intrinsic vectorization. The Intel MKL library is made primarily for this reason and requires little to no change in existing code to generate huge improvements. Additionally, things like BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra Package) have been modified over the years to take advantage of parallelism.

Additionally, new developments in hardware have created graphics processing units that computer entirely in parallel. Year-over-year, these GPUs increase in capability and created a huge area of research with potential benefit to numerical performance.

Nvidia is the world’s largest supplier of GPU technology and have been actively working with researches in the area of high performance computing. CFD is part of HPC and a huge amount of effort has been involved with creating CFD code that works primarily on GPUs. In the typical SIMPLE method, the linear system solvers will be off-loaded from the CPU to the GPU where parallel computations can take place. In recent studies, people have seen up to 16x speed up from highly optimize code. When compared to the line-by-line TDMA, data doesn’t exist. We will provide the comparison here though.

Finally, there have been a number of improvements in linear system solution algorithms that provide greater portability to parallel processing. GMRES and BiCGStab have been modified to be run in parallel. The BLAS and LAPACK libraries have also been updated to be run on massively parallel machines. New algorithms like SPIKE have also been created specifically for parallel processing. On top of all that, pre-conditioners for linear systems have been parallelized and new methods have been created to make super iterative solvers.

# Objective

The objective of this study is to optimize the performance of a numerical solution algorithm for microchannel flow boiling problems using parallelism. The particular microchannel flow boiling algorithm that will be investigated has previously been demonstrated in literature [ref previous publications here] and is a primary candidate as it almost entirely written as a serial algorithm. Previous publications utilizing this microchannel flow boiling algorithm demonstrate a high-accuracy in predicting experimental results [ref] and, being mostly serial in nature, provides a solid foundation for comparison with a parallel alternative. The performance indicators for this study are (1) solution accuracy, (2) computational duration, and (3) algorithm stability.

As mentioned in the previous chapter, increasing the complexity of a CFD solution can significantly increase the computational runtime. In previous uses of the microchannel flow boiling algorithm that is studied herein and in referenced literature, the primary bottleneck of CFD algorithms is the solution of massive sparse linear systems. Following the SIMPLER method, the solution of these systems of equations occurs, at minimum, twice. The first instance is to solve the XXXX block. The second instance is to solve the XXXX block. Additional solution routines are also used if the energy or species equation of Navier-Stokes is required. Since these solution methodologies appear multiple times in the algorithm and dominate computational runtime, the linear solvers are the most significant bottleneck.

In order to optimize the runtime and accuracy of the microchannel flow boiling problem, the following linear equation solution methodologies will be parametrically studied:

* Line-By-Line TDMA
* Bi-Conjugate Gradients (BiCG)
* Bi-Conjugate Gradients Stabilized (BiCGStab)
* General Minimum Residual (GMRES)
* PARDISO

These solution algorithms have all been extensively studied in linear algebra, but there has not been a significant review of their abilities in the scope of computational fluid dynamics. Furthermore, their applicability to microchannel flow boiling has not been determined in an extensive review of literature. The PARDISO algorithm is a proprietary algorithm that is typically used in commercial software packages such as ANSYS, COMSOL, and NASTRAN. The PARDISO algorithm, therefore, provides a reference point for the solution speed of commercial software and will be utilized as such.

In addition to a variety of linear equation solution algorithms, a number of preconditioning methodologies will be investigated. Pre-conditioning algorithms determine starting points for the Krylov methods referenced above. A good pre-conditioning algorithm has been seen to improve convergence for these algorithms by up to XX%. However, when pre-conditioning algorithms are ill-fitted, algorithm stability can be compromised. The particular preconditioning algorithms that will be explored are:

* Jacobi Precondition
* Successive Over-Relaxation
* Multi-grid

Finally, each of these algorithms and preconditioners will be tested in serial, parallel, and massively parallel (GPU specific) configurations. The serial method will be completed as it provides a reference point for the serial nature of the existing microchannel solution. The parallel method will be optimized to use vectorization on a single CPU and the massively parallel methodology will be optimized for use on GPU hardware. The comparison between vectorization and GPU computation will provide a reference point as hardware configurations can significantly differ.

The algorithm with the most significant improvement will then be implemented into the microchannel flow boiling algorithm and compared with the current line-by-line TDMA in terms of accuracy, runtime, and stability. Two different hardware configurations will be used for testing, one of which has GPU compute capabilities. The comparison of these test cases will serve as the optimization of the microchannel flow boiling algorithm.

# Mathematics

# Computational fluid dynamics is a numerical simulation approach to solving systems that primarily involve fluid flow and heat transfer. Simulations are generated from the discretization of partial differential equations and a number of high-level algorithms have been developed to iteratively solve the Navier-Stokes equations. For the microchannel flow boiling simulations discussed herein, the SIMPLER algorithm has been implemented.

# The SIMPLER algorithm starts by utilizing an initial guess for the pressure field, velocity field, and energy field. From these guesses, pseudo-velocities are computed and fed into the pressure equation. The pressure equation is solved and the initial guess pressure field is replaced by the pressure field. Next, the discretized momentum equations are solved and the pressure correction equation is solved. Next, the velocities are corrected with the pressure correction factor and all other discretized transport equations are solved. At this point, the solution checks for convergence. If convergence is not achieved, the process starts over with the pressure, velocity, and transport variables set as the initial guess. If convergence is achieved, the process stops. A figure of this layout can be seen below.

# < figure 6.7 versteeg >

# The SIMPLER algorithm implements a linear solver at Steps 2, 3, 4, and 6. In the case of microchannel flow boiling, the linear solver is called 6 times (for U, V, and W velocities). Therefore, making an improvement to this area of the algorithm can generate up to a 6x benefit for each time-step.

# The baseline linear solver used in the microchannel flow boiling simulation is the line-by-line TDMA. The line-by-line TDMA (3D-TDMA for shorthand) solves a three-dimensional system of equations by iteratively solving one-dimensional systems. These one-dimensional systems represent a line of nodes through the computational domain. For one-dimensional systems, the TDMA is able to directly solve for unknown values in a serial manner. A number of parallel versions of the TDMA have been implemented in recent years, but their improvement in speed is typically low for small numbers of variables.

# In recent studies, parallel implementations of the TDMA show significant increases in computational performance for large systems of equations (order of E10). However, the performance cost of the parallel implementation out weights the serial algorithm for smaller systems. The computational domain of the microchannel flow boiling simulation has a maximum dimensional length of 400 nodes. This translates to a linear system with 398 unknowns, with is far less than the performance inflection point of the parallel TDMA implementations. Therefore, by analysis there would be no benefit to using a parallel implementation of the TDMA for the microchannel flow boiling simulation.

# In order to use the TDMA for a three-dimensional problem, each one-dimensional line of the computational grid is solved and the results are diffused into the equations of neighboring lines. This line-by-line method was originally utilized in CFD by Pantankar, but the method results in an extremely slow process of convergence. In order to improve the results of this method, the sweeping direction of the lines can be varied based on the physics of the simulation. For example, if fluid is moving from the x=0 position to the x=400 direction, it would be ideal to sweep along the axis. This sweeping methodology increases the diffusion of unknown terms by leveraging the TDMA in the primary gradient direction. Additionally, there are a number of parallel methods that can be implemented to further increase the rate of convergence.

# The line-by-line TDMA can be improved with a simple instance of parallelism. By solving lines in parallel, the computation time can be significantly reduced. However, the diffusive properties of the algorithm need to be examined to ensure that the number of cycles for convergence does not exponentially expand. This exponential expansion in convergence steps is due to isolation of the individual line solutions. Simply put, solving the lines in parallel can adversely affect the diffusive properties of the algorithm. It is important to conserve the diffusive property of the algorithm while distributing computational load with parallel processing. This can be achieved by properly selecting the directionality of the parallel solutions. For example, starting a single thread in the top, left, north corner and a second thread in the bottom, south, right corner would reduce the restriction on diffusion.

# This type of algorithm has been explored in the last decade and has shown various runtime improvements on massively parallel clusters of computers. The Thomas algorithm has a linear complexity therefore it does not require a significant amount of memory overhead. Additionally, each line solution has a small amount of data required for transfer between compute nodes. This tends to yield some improvements to the base code. However, the number of convergence steps does increase which reduces the impact of parallelism.

# A more significant improvement in runtime can be achieved by completely replacing the line-by-line TDMA method with a standard linear algebra method. In recent years, the Conjugate Gradient algorithm has been an intense algorithm of study and optimization for positive semi-definite matrix equations. Unfortunately, most CFD equations do not yield these types of differential equations. In XXXX date, Krylov (or some other dude) came up with a conjugate method for non-positive semi-definite matrix equations. This technique can be applied to CFD problems.

# The Bi-Conjugate Gradient method is similar to the CJ, with the exception that a few steps are added to convert the matrix into positive form. Talk some more about bi-conjugate gradient methods. Additionally, improvements have been made to this algorithm to improve the stability which has yielded the Bi-Conjugate Gradients Stabilized method.

# The BiCGStab method follows the same processes as the BiCG method but adds some other stuff at the end. This stuff at the end significantly improves algorithm stability and results in even faster speed up for particular types of problems. When considering the best replacement of the line-by-line TDMA, stability is important. Therefore, it will be one of the algorithms to consider.

# The Generalized Minimum Residual method has also been used in many commercial software packages for its improvements in runtime. The GMRES algorithm works like this… In many tests the GMRES algorithm performs really good and demonstrates a lot of stability.

# Commercial software packages like ANSYS and COMSOL tend to use a proprietary algorithm called PARDISO. This algorithm will examine the matrices that are input and come up with the best potential for solution. Many of the internal solutions have similar structures as GMRES and BiCGStab, but they are proprietary. We don’t exactly know what goes on inside the code. However, since these are widely available and utilized in many commercial packages, this algorithm provides a baseline for the state-of-the-art.

# In order to determine the best potential solution algorithm the following algorithms were developed and tested in a steady state, three-dimensional heat transfer simulation. This type of simulation was chosen for its simplicity to create and implement. It is also important to note that solving the three-dimensional heat transfer problem is identical to solving the three-dimensional convection problem. This is because the coefficient matrix for the linear system is identical in shape between these two cases. The shape is derived from the differencing scheme used and both problems use central differencing scheme.

# The test setup for this parametric study utilized a cubic geometry

# Numerical Model

Algorithm Study

Diffusion

Two-phase flow boiling

# Results

**One-Dimensional Diffusion**

A one-dimensional diffusion problem was solved with a heat flux boundary condition applied to the west side and a constant temperature boundary condition applied to the east side.

**Two-Dimensional Diffusion**

A two-dimensional diffusion problem was solved with a constant temperature boundary condition and a constant heat flux boundary condition. The constant temperature was applied to the north side of a two-dimensional grid and given a temperature of 100 °C. The heat flux boundary condition was applied to the west wall and given a heat flux of 500 W/m2. The nodal mesh was generated as a square and the number of nodes was varied from 10 to 100 in increments of 10 (i.e. 10, 20, 30, …100).

< figure of BC step up >

The following linear system solution algorithms were each used to solve the system of equations using the MATLAB scripting environment as well as in compiled FORTRAN:

* Tri-diagonal matrix algorithm (TDMA)
* Biconjugate Gradients (BiCG)
* Biconjugate Gradients Stabilized (BiCGStab)
* General Minimum Residual (GMRES)
* Pardiso (Intel MKL)
* MLDIVIDE (MATLAB)

For small grid sizes, the TDMA performed nearly as well as the Pardiso and MLDIVIDE algorithms. As the grid sizes increased, however, the TDMA convergence time significantly increased. For all solution algorithms, the compiled FORTRAN environment achieved the lowest time of convergence, with the Intel MKL Pardiso algorithm outperforming the others be a factor of 1.4 times.

< figure results >

With Jacobi-preconditioning applied to each of the algorithms (aside from the TDMA), the results are slightly improved for both the MATLAB and FORTRAN environments.

< figure results with jacobi >

With Incomplete LU-Decomposition (ILU), the convergence rates for BiCGStab and GMRES are nearly identical to the rates of convergence for the Intel MKL Pardiso and MATALB MLDIVIDE algorithms.

< figure results with ilu >

From the results of the two-dimensional case, it appears that the MLDIVIDE and Intel MKL Pardiso algorithms are optimized for near-symmetric matrcies. This agrees with the wide-spread uses of these algorithms in typical commercial software packages such as ANSYS, NASTRAN, and COMSOL.

**Three-Dimensional Diffusion**

A three-dimensional diffusion problem was solved in a similar configuration to the three-dimensional problem. A constant temperature condition was applied to the north wall and a heat flux boundary condition was applied to the west wall. The values for these boundary conditions were the same as in the two-dimensional problem, 100 °C and 500 W/m2. Additionally, the nodal mesh was kept in a cubic configuration with m, n, and l all having the same number of nodes. These values were varied from 10 to 400 with intervals of 10 until 100 nodes per side, then in intervals of 100 to 400 nodes per side.

< figure of BC setup >

The following linear system solution algorithms were each used to solve the system of equations using the MATLAB scripting environment as well as in compiled FORTRAN:

* Tri-diagonal matrix algorithm (TDMA)
* Biconjugate Gradients (BiCG)
* Biconjugate Gradients Stabilized (BiCGStab)
* General Minimum Residual (GMRES)
* Pardiso (Intel MKL)
* MLDIVIDE (MATLAB)

Each of the tested algorithms performed significantly faster than the TDMA for all cases when considering the number of iterations and computational run time. The Pardiso and MLDIVIDE algorithms were also significantly slower than the biconjugate gradient stabilized method as the number of nodes increased.

< figure results >

With Jacobi-preconditioning applied to each of the algorithms (aside from the TDMA), the results are slightly improved for BiCGStab and GMRES.

< figure results with jacobi >

With ILU preconditioning, the convergence rates for BiCGStab and GMRES perform significantly better than the Intel Pardiso and MLDIVIDE algorithm.

< figure results with ILU >

From the results of the three-dimensional case, it appears that the MLDIVIDE and Intel MKL Pardiso algorithms not optimized for matrices that are not symmetric.

# Conclusion

References