CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Performance Optimization in Computational Fluid Dynamics with Parallelism for Linear System Solvers

A thesis submitted in partial fulfilment of the requirements

For the degree of Master of Science in Mechanical Engineering

By

Matt Blomquist

May 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 in Computational Fluid Dynamics with Parallelism for Linear System Solvers

By

Matt Blomquist

Master of Science in Mechanical Engineering

Abstract text here…

Chapter 1: 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 [Versteeg 2007]. The aerospace industry was an early adopter of CFD, primarily using the technology to study the effects of drag and lift on aerodynamic bodies. Aircraft manufacturers like Airbus, Boeing, and Lockheed Martin, typically spend millions of dollars creating prototype aircraft to study the effects of fluid flow around the designed aircraft body [need a ref here]. By introducing a tool like CFD, aircraft manufacturers are able to filter out potential design flaws prior to actually manufacturing any form of aerodynamic body. As CFD has matured as a standard technology, it (CFD) has even replaced the need to physically test the aerodynamic effect of new aircraft designs [reference Boeing 787 design].

In performance racing, the use of CFD has become so integrated in the design process that every F1 car manufacturer / performance team has a dedicated CFD department. Furthermore, the use of CFD has expanded past simply modeling aerodynamic effects it now encompasses energy efficiency predictions in the combustion engines of.

Chapter 2: Objective

The scope of this study is to provide a framework for improving the computational performance of legacy CFD code, perform an optimization study of the current solutions to CFD problems, and determine the costs associated with implementing new technologies in legacy CFD code. The framework for improving the computational performance of legacy CFD code considers two classes of examples. The first class of examples, considers simple diffusion problems solved using the finite volume method and the centralized differencing scheme. This class of example is primarily used due to its simplicity as implementing parallelism on a simple problem provide a higher adherence of understanding when compared to a problem with increased complexity. The second class of examples focused on a two-phase flow boiling problem that incorporates a solution to the full, incompressible Navier-Stokes equations. This example is used to demonstrate the effectiveness of implementing parallelism in a complex, CFD problem.

The optimization study considers the rate of convergence for the solutions to sparse linear systems, the computational resource requirement for different solutions to sparse linear systems, and the duration of computation for given solutions on a variety of hardware platforms. The study also considers different means of implementation for each solution algorithm and how that implementation affects the performance metrics of the optimization study. The optimization study considers the same two classes of CFD problems as mentioned previously and considers cases for a 1-dimensional, 2-dimensional, and 3-dimensional environment.

Finally, the determination of cost associated with implementing each particular CFD solution is examined for both the simple diffusion problem and the two-phase boiling problem. The performance indications of cost are considered for the 1-dimensional, 2-dimensional, and 3-dimensional environments and assess the required revision and restructuring of legacy code necessary to implement new solution types.

Chapter 3: Mathematics

Computational Fluid Dynamics

Governing Equations

Discretization and Finite Volume Method

Application to Diffusion Problems

Application to Convection Problems

Solution Procedure for the Flow Field

Test Scenarios and Computational Domains

Heat Transfer in 1-Dimension

Heat Transfer in 2-Dimensions

Heat Transfer in 3-Dimensions

Microchannel Flow Boiling in 3-Dimensions

Solution Methods

Serial Methods for Numerical Computation

Parallel Methodologies for Numerical Computation

Array Vectorization

Array vectorization consists of performing identical functions on each element of an array in parallel.

Parallel Processing

Parallel processing, as opposed to array vectorization, involves distributing the computational steps in the solution algorithm into separate steps that can be performed in parallel. The typical protocol for defining these functions is OpenMP.

Parallel Processing with Clusters

Clusters offer additional computational resources for parallel processing with minor drawbacks.

Parallel Processing with GPUs

GPUs offer a massively parallel architecture for solving stuff.

Solution Algorithms

Direct Solution Methods

Iterative Solution Methods

Jacobi Iteration

Gauss-Seidel Iteration

Successive Over-Relaxation

Alternating Direction Methods

Conjugate Gradients

Generalized Minimum Residual

Bi-Conjugate Gradients & BiCGStab

Preconditioning for Improved Convergence

Precondition is a standard technique used in the solution of large sparse matrices to improve the rate of convergence and reduce the overall computational time. There are several standard preconditioning methods, but they all share a basic ideology to create a similar, simplified system matrix that can be easily inverted to solve a system in the form Ax = b.

Jacobi Preconditioning

Jacobi preconditioning is a basic form of preconditioning that, for some systems, will drastically improve the rate of convergence without a significant increase to the computational overhead. Jacobi preconditioning creates a single, diagonally dominant matrix with values that correspond to the inverse of the original matrix diagonals.

Successive Over-Relaxation

LU & ILU Factorization

Multigrid Methods

Parallel Implementations

Multi-coloring

Multi-elimination

Memory Considerations and Sparse Matrix Formats

As the mesh used in the solution of computational fluid dynamics problem get finer, a large portion of the system coefficient matrix, A, has a value of zero. These zero values, are stored in the same data format as the non-zero values and result in allocations in the system memory. For example, a 100x100 matrix would have 460 non-zero values and 9,540 zero values. This equates to 3.68E-3 MB of RAM for the non-zero values and 6.11E-1 MB of zero values for 64-bit floating point numbers. While this amount of memory may seem trivial, the allocation requirements for larger matrices becomes intractable. For a three-dimensional diffusion problem with a grid size of 100x100x100 nodes, the temperature solution matrix has 6.94 million non-zero values and 9.99 trillion zero values. This would equate to 55 MB for the non-zero values and 8 TB for the zero values.

< figure of sparse matrix (MATLAB spy) >

To create a more tractable matrix for large systems, several sparse matrix formats have been designed to only store the non-zero values. The compressed sparse row (CSR) format utilizes three arrays to store the non-zero floating-point values, the column indices, and the row index. This is a standard format that can be used in several basic linear algebraic packages such as LAPACK and PLASMA. This is also the primary format used for the Intel MKL Pardiso algorithm. An example of this storage format can be seen below.

< figure of CSR format >

Another sparse matrix format is the compressed sparse column (CSC) format, which is primarily used in the MATLAB based sparse matrices. The CSC format consists of three arrays that carry the non-zero values, the column index, and the row index. An example of this storage format can be seen below.

< figure of CSC format >

Chapter 4: Numerical Model

Diffusion

Two-phase flow boiling

Chapter 5: 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.

Chapter 6: Conclusion

References