Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Performance Improvement of Multi-scale Modeling of Concrete Performance Solver Using Hybrid CPU-GPU System

A Thesis Submitted to the School of Electrical and Computer Engineering In Partial Fulfillment of the Requirements for the Degree of Master of Science In Computer Engineering

By: Lina Felleke

Advisor: Fitsum Assamnew

Co-Advisor: Dr. Esayas G/Yohannes

April 2015

Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Performance Improvement of Multi-scale Modeling of Concrete Performance Solver Using Hybrid CPU-GPU System

By: Lina Felleke

Advisor: Fitsum Assamnew

Co-Advisor: Dr. Esayas G/Yohannes

April 2015

Addis Ababa University
Addis Ababa Institute of Technology
School of Electrical and Computer Engineering

Performance Improvement of Multi-scale Modeling of Concrete Performance Solver Using Hybrid CPU-GPU System

By Lina Felleke

Approval by Board of Examiners

Dr.Yalemzewd Negash	_____
Dean, School of Electrical	Signature
And Computer Engineering	
Fitsum Assamnew	_____
Advisor	Signature
Dr. Esayas_G/Yohannes	_____
Co-Advisor	Signature
Dr. Sirinavas Nune	_____
External Examiner	Signature
Menore Tekeba	_____
Internal Examiner	Signature

Declaration

I, the undersigned, declare that this thesis work, to the best of my knowledge and belief, is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been fully acknowledged.

Name: LinaFelleke Signature: _____

Place: Addis Ababa

Date of submission: _____

This thesis has been submitted for examination with my approval as a university advisor.

Fitsum Assamnew Signature: _____
Advisor Name

Dr. Esayas_G/Yohannes Signature: _____

Co-Advisor Name

Abstract

The multi-scale modeling of concrete performance solver is a tool developed at University of Tokyo. It is used to help engineers predict the material and structural properties of concrete structures under coupled mechanistic and environmental actions. The current implementation of multi-scale modeling of concrete performance solver is developed for Central Processing Unit (CPU) and CPU clusters and has longer runtimes. In this work, we present a CPU- Graphics Processing Unit (GPU) hybrid implementation of the tool. Changes were made to the LU factorization module by moving the computations to the GPU after profiling revealed it to be compute intensive. Tests conducted on the modified tool using real world sample concrete structure show that using the hybrid CPU-GPU implementation improved the performance by 35.29% on single node and 37.5% on cluster.

Key Words:, Hybrid CPU-GPU system, General Purpose Computation on Graphics Processing Units, Multi frontal Massively Parallel Solver, Matrix Algebra on GPU and Multicore Architectures

Acknowledgment

First and foremost I thank God. This thesis wouldn't have been done without his blessing.

I also offer my sincerest gratitude to my advisor, Mr. Fitsum Assamnew, who has supported me throughout my thesis with his continuous encouragement and technical expertise in the area of parallel computing. I also thank him for reading the drafts of this work carefully and giving me valuable feedback for modification.

I would like to pass my heart felt gratitude to Dr. Esayas G/Yohannes, who has been in great help by offering much structural advice and insight throughout my work.

I would like to pass my sincere gratitude to AAU for the scholarship provided by the University for funding my studies.

Special Thanks goes to SECE for the provided support and equipment I have needed to produce and complete my thesis.

I would like to pass my sincere gratitude to AAU for the scholarship provided by the University for funding my studies.

I would like to thank Henok Gebru, Iyasu Daniel, Mesay G/hiwot and Meron Demrew for their continuous support and love.

Finally, I thank my family for their continuous support, encouragement and love.

Table of Content

List of Figures

List of Tables

List of Acronyms

DuCOM-COM3	Durability models of concrete constitutive models of reinforced concrete
BLACS	Basic linear algebra communication subprograms
BLAS	Basic linear algebra subprograms
COM3	Constitutive models of reinforced concrete
CUDA	Compute Unified Device Architecture
DuCOM	Durability Models of Concrete
DDR2	Double data rate synchronous dynamic
DDR3	Double data rate types three synchronous dynamic
DRAM	Dynamic random access memory
FEA	Finite Element Analysis
FFT	Fast Fourier transforms
FLOPS	Floating-point operations per second
GCC	Gnu c compiler
GPGPU	General Purpose Computations on Graphics Processing Units
Gprof	GNU Profiling Tool
GPU	Graphics Processing Unit
HIPS	Hierarchical iterative parallel solver
HPC	High-performance computing
HYPRE	High performance preconditioners
IFORT	Intel Fortran compiler
LA	Linear Algebra
LAPACK	Linear algebra package
LTS	Long term support
MAGMA	Matrix Algebra on GPU and Multicore Architectures
MATLAB	Matrix laboratory
MIC	Many Integrated Core
MKL	Math-kernel Library
MPI	Message Passing Interface
MPICH	High performance portable MPI
MUMPS	Multi frontal Massively Parallel Solver
NFS	Network File System
NIC	Network interface card
OpenACC	Open Accelerators
OpenCL	Open Computing Language
OOC	Out-of-core
PARDISO	Parallel Sparse Direct Solver
PBLAS	Parallel Basic Linear Algebra Subprograms
PCIe	PCI express
PDE	Partial Differential Equations
PGI	Portland Group
PHREEQC	PH-Redox Equilibrium
PLASMA	Parallel Linear Algebra for Scalable Multi-core Architectures
RAM	Random Access Memory

RC-PC	Reinforced concrete pipe culvert
RNG	Random number generators
ScaLAPACK	Scalable Linear Algebra Package
SMP	Shared memory programming
SSH	Secure shell
SVD	Singular value decomposition
VML	Vector Math Library
VSL	Vector Statistical Library

Chapter 1 Introduction

1.1 Background

The use of parallelization in boosting applications' performance has been the area of interest for many researchers. One of the areas of research is utilization of general-purpose graphics processing units (GPGPUs) for acceleration of applications. Graphics processing units (GPUs) are specialized processors with dedicated memory that conventionally perform floating-point operations required for rendering graphics. GPUs have been around for years working alongside the computer's CPU to speed up graphics operations. Only recently have GPUs been specifically developed with the computational precision as well as the computational power to effectively complement the performance of the latest CPUs. With hundreds of low power cores on a single socket, GPUs have the potential to dramatically increase computing capacity [1], provided that the compute workload will fit in the available memory of the GPU.

Modern GPUs consist of fully programmable floating-point pipelines with notable computational power and memory bandwidth. Current GPUs exceed current CPUs by about a factor of 10 in peak floating-point performance and by about a factor of 5 in Memory bandwidth [2], which makes them an interesting target for general purpose computing [3, 4, 5]. As a result, GPGPUs has seen successful utilization in different field of areas such as Molecular Dynamics, Quantum Chemistry, Materials Science, and Computational Structural Mechanics [6]. In general, GPUs provide performance boost when the execution of applications spends most of its time on numerical analysis than branching and memory operations.

Computational structural mechanics is a well-established methodology for the design and analysis of concrete, cement, and structures like bridges, roads etc. found in civil and mechanical engineering. Mathematical models are used to ideally represent these physical systems. Modern computer simulation tools that are based on finite-element methods play a major role in computational structural mechanics. Finite Element Methods are a technique of discretization of mathematical models in to discrete model. Partial differential equations (PDE), such as, Poisson, Laplace, Navier-Stokes,

and so on can be solved using finite element methods. In every finite element analysis (FEA) there are three main stages; Pre-processor, Processor, Post-processor. The pre-processor handles geometric modeling such as geometric primitives and other modeling tools, followed by mathematical modeling such as mesh nodes and finite elements. It also handles material behavior, boundary conditions. The processor computes the solution to the input problem. It can use direct or iterative methods to arrive at the solution. The post-processor stage is where the results are presented.

Multi-scale modeling can be used in computational and structural mechanics for effective performance assessment of structural and integrated material systems. Multi-scale modeling is based on the view point that any system of interest can always be described by a hierarchy of models of different complexity. The system of interest can be described adequately by a coarse-grained model, except in some small cases where more detailed models are needed. Multi-scale materials modeling combines existing and emerging methods from diverse scientific disciplines to bridge the wide range of simulation time and material length scales including nanometer to meter that are inherent in a number of essential phenomena and processes in materials science and engineering.

Multi-scale modeling of concrete performance solver called DuCOM-COM3 is a finite-element based computational program to evaluate various durability aspects of concrete. It is software used for performance assessment of concrete structures under coupled mechanistic and environmental actions during design. The current implementation of DuCOM-COM3 uses Message Passing Interface (MPI) enabled direct solver called multi-frontal massively parallel solver (MUMPS) library to solve a series of non-linear equations. It is capable of running on a single computer or on a CPU cluster of computers. The current implementation takes long running time to come up with results. In this thesis, a heterogeneous CPU-GPU system implementation to enhance performance of multi-scale modeling of concrete performance solver has been studied.

1.2 Statement of the Problem

Material engineers are using multi-scale modeling to predict the best combination of materials for durable concrete. The multi-scale modeling of concrete performance solver implemented by University of Tokyo, DuCOM-COM3 uses many third party non-linear equation solvers such as MUMPS [7]. The runtime of the current implementation is drastically long on single computer systems. Even, the MPI based parallel implementation did not perform well on our test cluster. The inadequacy of the current system is that it has highly data parallel operations whose implementation on CPUs is inefficient [8]. Though the MPI parallelization technique improves the computational time from the previous versions, it would benefit further from implementation on GPUs, which are more suited for data parallel operations.

1.3 Objectives

General Objective

The general objective of this thesis is to improve the runtime of multi-scale modeling of concrete performance solver called DuCOM-COM3 by utilizing general-purpose GPUs.

Specific Objective

- Designing and building hybrid CPU-GPU cluster for experiment.
- Implement hybrid CPU-GPU multi-scale modeling of concrete performance solver.
- Evaluate the performance of the our hybrid CPU-GPU implementation on single-node and hybrid CPU-GPU cluster

1.4 Scope

The main focus of this work is performance improvement of DuCOM-COM3 using a heterogeneous CPU-GPU system. We modified MUMPS to support computation on GP-GPUs using the Matrix Algebra on GPU and Multicore Architectures (MAGMA) library. The factorization routines in MAGMA are used to port the LU factorization module from CPU to GPU. The hybrid CPU-GPU DuCOM-COM3 solver is tested both on single computer with a GPU and our hybrid CPU-GPU cluster built for this work.

1.5 Organization of the Thesis

The second chapter surveys related work in performance improvement of DuCOM-COM3 and LU factorization performance improvement techniques are also discussed. Chapters 3, covers brief overview of DuCOM-COM3 and direct and iterative solvers that are used by it, like MUMPS, scalable linear algebra package (ScaLAPACK), Intel Math-kernel Library (MKL). Also, a multi-core and GPU based library of linear algebra computations called MAGMA is described here. Chapter four describes in detail the implementation of our CPU-GPU hybrid DuCOM-COM3 solver. The design and implementation of a hybrid CPU-GPU cluster used as a test bed for our experiments is described in this section. Experiment results are discussed in chapter five. Finally, chapter six presents our conclusions from this work and suggestions for future work.

Chapter 2 Literature Review

The project of constitutive models of reinforced concrete (COM3) was initiated in 1980 for structural behavioral simulation and assessment of safety and seismic performance [9]. Afterwards, the DuCOM project started in 1991 based on multi-component cement hydration model and COMT3-1982 [9] version for thermal stress analysis. Different modifications have been done on DuCOM-COM3 ever since the project start time in 1980 as COM3. But, most of the changes were focused on techniques of analyzing material and structural properties of concrete.

One of the basic library used by DuCOM-COM3 is MPI enabled MUMPS solver, which is used for solving a series of non-linear system of equations. It solves the system of non-linear equations using three phases. These are analysis, factorization and solution phases. All the three phases can be executed both in parallel and sequentially. From profiling results of MUMPS solver via gnu profiling tool called gprof, 60% of the total computation time of the solver is spent in the factorization step. Hence, improving performance of this phase opens a way for performance improvement of the solver. As a result better performance of DuCOM-COM3 is expected. This review is organized in to three based on implementation techniques of the factorization phase. These are CPU implementation, GPU implementation and hybrid CPU-GPU implementation.

A lot of techniques have been used to improve performance of LU factorization in CPU. Because of their robustness and performance, direct methods can be preferred to iterative methods, especially in an industrial context. However, they require a large amount of memory because the factorization of a sparse matrix leads to factors with an amount of non-zeros much larger than in the original matrix. Even if the memory of modern computers keeps increasing, applications always require solving larger and larger problems, which still need more memory than available. Furthermore, some types of architectures have a limited amount of memory per processor or per core. The use of out-of-core (OOC) solvers, where the disk is used to extend the physical memory, is then compulsory. The work in [10] designed and implemented OOC direct methods from both a theoretical and a practical point of view. Two classes of direct methods, which are multi-frontal and super-nodal algorithms, were studied in regards

to OOC performance. Their work showed that OOC techniques allow solving large sparse linear systems efficiently. This is an already integrated feature in MUMPS. In [11], an OOC sparse non-symmetric LU factorization algorithm with partial pivoting was implemented. Based on experiments conducted, this work showed that it can easily factor matrices whose factors are larger than main memory at rates comparable to those of an in-core solver.

The ongoing hardware evolution exhibits an escalation in the number, as well as in the heterogeneity, of computing resources. The pressure to maintain reasonable levels of performance and portability forces application developers to leave the traditional programming paradigms and explores alternative solutions such as GPGPU techniques [12-16].

In [12], GPU-based sparse LU factorization intended for circuit simulation is implemented. The Sparse matrixes are preprocessed and their zeroes are eliminated and as a result dense arrays are produced. Dense arrays produced at preprocessing stage are used for intermediate vector format due to their convenience for indexed accesses. In order to improve data locality, they put sorting procedure in the preprocessing stage. Their experiments demonstrate that the GPU implementation outperforms the CPU on matrices which are increasingly denser.

Sparse Matrix Solvers on the GPU are investigated and implemented in [13]. They port two widely applicable solvers, a sparse matrix conjugate gradient solver and a regular-grid multi-grid solver, to the GPU. Their tests have shown that both the CPU and GPU implementations are bandwidth limited.

In work [14], the development of Linear Algebra Package reconfigurable computing (LAPACKrc) is studied. Reconfigurable computing is a radically new computing paradigm that is changing the way high performance computing is done today. The LAPACKrc library represents the first industrial-quality family of linear algebra solvers intended for heterogeneous Field Programmable Gate Array (FPGA)-enhanced High Performance Computing (HPC) architectures, capable of speedups exceeding 100x per FPGA processor. Their latest direct sparse solver prototype demonstrated a speedup of up to 125x (compared against state-of-the-art CPU direct

sparse solvers), which provides support for broad-based science/engineering breakthrough.

In paper [15], a sparse iterative solvers package, a communication-avoiding (CA) sparse iterative solver (CA- GMRES), preconditioners based on dense linear algebra (DLA) operations including batched GEMM, GEMV and batched LU, QR, and Cholesky for the parallel factorization of many small matrices, and a mixed-precision orthogonallization with application to sparse linear and Eigen problem solvers were described. Sparse linear algebra computations comprise a fundamental building block for many scientific computing applications, ranging from national security to medical advances, highlighting their importance and potential for broad impact. The new developments in this work harness expertise in Dense Linear Algebra (DLA) namely, the MAGMA libraries, providing LAPACK for GPUs and auto-tuned BLAS to develop high-performance sparse solvers, and building blocks for sparse computations in general.

The NVIDIA CUDA Sparse Matrix library (cuSPARSE) provides a collection of basic linear algebra subroutines used for sparse matrices that delivers up to 8x faster performance than the latest MKL. The cuSPARSE library is designed to be called from C or C++, and the latest release includes a sparse triangular solver [16]. Key Features of cuSPARSE are listed below.

- Level 1 routines for sparse vector x dense vector operations
- Level 2 routines for sparse matrix x dense vector operations
- Level 3 routines for sparse matrix x multiple dense vectors (tall matrix)
- Routines for sparse matrix by sparse matrix addition and multiplication
- Conversion routines that allow conversion between different matrix formats
- Sparse Triangular Solve
- Tri-diagonal solver

The work in [17, 18], effectiveness of heterogeneous architectures for sparse direct solvers is studied using QR-MUMPS as a sample sparse direct solver. They used CPU kernels from previous work and presented new kernels of factorization for GPUs, which enhanced the performance of QR factorization.

It is clear that utilizing general purpose GPUs for improvement of the sparse matrix solvers has improved the runtime. In this thesis, we studied the effect of GPU based sparse matrix solvers on the MPI enabled MUMPS library used in DuCOM-COM3. This is achieved by replacing the ScaLAPACK library used by MUMPS with the GPU enabled MAGMA library.

Chapter 3 DuCOM-COM3

In this chapter brief overview of the computerized implementation of multi-scale modeling of concrete performance solver called DuCOM-COM3 and linear algebra libraries and other direct solvers used in this thesis are described.

3.1 DuCOM-COM3

Increases in computer power have now enabled engineers to combine materials science with structural mechanics in the design and the assessment of concrete structures. The techniques developed have become especially useful for the performance assessment of such structures under coupled mechanistic and environmental actions. This allows effective management of infrastructure over a much longer lifecycle, thus satisfying the requirements for durability and sustainability.

The implementation of multi-scale modeling of concrete performance solver, DuCOM-COM3, is capable of predicting overall structural behaviors as well as constituent materials of concrete. It is used in simulating both constituent materials and structural responses under external loading and under ambient conditions called events. These events are discretized in space into sub-events. This computerized scheme of multi-scale modeling of structural concrete performance has many series of non-linear equations to be solved. To simplify the computerized scheme an approximation technique called Galarkin's method is used. Each sub-event is discretized in space by applying Galarkin's method of weighted residual function on finite element scheme [7].

DuCOM-COM3 is composed of two components; the combination of constitutive models of reinforced concrete (COM3) and durability models of concrete (DuCOM). It enables to simulate reinforced concrete pipe culvert (RC-PC**)** structures, soils under coupled ambient and mechanistic actions.

DuCOM is a finite element based computational program to evaluate various durability aspects of concrete. It traces the development of concrete hardening (hydration), structure formation and several associated phenomenon, from casting of

concrete to a period of several months or even years. As such this tool can be utilized to study the effect of ingredient materials, environmental conditions as well as the size and shape of structure on the durability of concrete [19]. The term durability considered here takes in to account both the green concrete stage problems as well as matured concrete exposed to environment. This tool can be used to analytically trace the evolution of microstructure, strength and temperature with time for any arbitrary initial and boundary conditions. Since the main simulation program is based up on finite-element methods, it could be applied to analyze real life concrete structures of any shape, size or configuration. Furthermore, dynamic coupling of several phenomena ensures that the effect of changing environmental conditions is easily integrated in to the overall simulation scheme. COM3 is concrete structural analysis program which uses the material analysis results of DuCOM as an input.

The integrated DuCOM-COM3 has the following features [9]:

- Hydration heat of cement in concrete and thermal conduction
- Micro-structure formation, phase-transformation, moisture balance and migration
- Chloride ion penetration and chemical fixation
- Carbon dioxide diffusion and carbonation and ion balance of pore solution
- Oxygen diffusion and micro-cell corrosion of dispersed steel in concretes
- Calcium ion transport and leaching from CSH gel structures
- Electric field and currency
- Aging concrete mechanics, autogenously and drying shrinkage, freezing expansion, damage and stresses
- Short/long-term time-dependent high nonlinear mechanics under static and dynamic loads
- Hydroxyl-ion migration is computed in iterative procedure of nonlinear computation
- High cycle fatigue of material and structures
- Soil foundation and underground water system

- Multi-ions-association aqueous balance of sodium, potassium, calcium, sulfate and hydroxyl

3.2 LAPACK Package

Linear Algebra Package (LAPACK) provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigen value problems, and singular value problems [20]. It relies on basic linear algebra subprograms (BLAS). Matrix factorizations procedures such as LU, Cholesky, QR, singular value decomposition (SVD), Schur, and generalized Schur are also provided. It handles dense and banded matrices but general sparse matrices are not handled. Also, real and complex matrices, in both single and double precision are supported.

3.3 ScaLAPACK Package

ScaLAPACK is an improved library that solves dense and banded linear systems; least squares problems, eigenvalue problems, and singular value problems [21, 22]. It supports the use of a block cyclic data distribution for dense matrices and a block data distribution for banded matrices, parametrizable at runtime; block-partitioned algorithms to ensure high levels of data reuse; well-designed low-level modular components that simplify the task of parallelizing the high level routines. The library is currently written in FORTRAN. The most recent version of ScaLAPACK is 2.0.0, released in November 11, 2011.
ScaLAPACK has two matrix factorization subroutines.

Subroutine PDGETRF

PDGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

Subroutine PDPOTRF

PDPOTRF computes the Cholesky factorization of a real symmetric positive definite matrix A.

3.4 MUMPS Library

MUMPS is a package for solving systems of linear equations of the form Ax = b, where A is a square sparse matrix that can be either asymmetric, symmetric positive definite, or general symmetric. It implements a direct method based on a multi-frontal approach which performs a direct factorization matrix A

$$A = LU \qquad (1)$$

where L is a lower triangular matrix and U is an upper triangular matrix.

If the matrix is symmetric, then the factorization as shown in equation (2) is performed.

$$A = LDL^T \qquad (2)$$

where D is block diagonal matrix with blocks of order 1 or 2 on the diagonal.

The system Ax = b is solved in three main steps in MUMPS. These are analysis, factorization and solution phases. Analysis phase also called preprocessing and is a stage where the input matrix is adjusted in a way suitable for later processing in factorization and solution phases. Preprocessing highly influences the performance (memory and time) of the factorization and solution steps. During analysis, the host performs reordering only on the information on the matrix structure without taking account of numerical values. The goal of the preordering is to find a permutation matrix P so that the subsequent factorization has the least of matrix entries which change from an initial zero to a non-zero called fill-in. The host also computes a mapping of the nodes of the assembly tree to the processors. The mapping is such that it keeps communication cost during factorization and solution to a minimum and balances the memory and computation required by processes. Both parallel and sequential

implementation of the analysis phase is available. In addition to the symmetric orderings, the package offers pre-processing facilities like permuting to zero-free diagonal and pre scaling. A mapping of the multi-frontal computational graph is then computed and used to estimate the number of operations and memory necessary for factorization and solution. The output of the analysis phase is Matrix A_{pre}.

During factorization the first task is to identify the symmetry of the preprocessed matrix A_{pre}. Based on the symmetry of A_{pre}, a direct factorization A_{pre}= LU or A_{pre}= LDL^T is computed. The numerical factorization on each frontal matrix is performed by a process determined by the analysis phase and potentially one or more other processes that are determined dynamically. After the factorization, the factor matrices are kept distributed (in core memory or on disk). They will be used at the solution phase.

During solution phase, the right-hand side vector b is distributed from the host to the other processes. They compute the solution vector x using the distributed factor computed during the factorization phase. The solution vector is then collected and assembled on the host.

Compared to other sparse direct solvers, MUMPS's uniqueness is due to [23]:

- Classical partial numerical pivoting during numerical factorization require the use of dynamic data structures
- The ability to automatically adopt to computer load variation during numerical phase
- High performance, by exploiting the independence of computations due to sparcity and that available for dense matrices
- The capability for solving wide range of problems, including symmetric, unsymmetrical, and rank-deficient systems using either LU or cholesky factorization.

MUMPS supports real, complex, single, and double precision arithmetic. A parallel analysis and OOC functionality are also available. Most recent experimental functionalities involve the computation of the determinant, the computation of the

entries in the inverse of matrix A and exploiting sparcity of the right-hand sides to reduce the amount of floating-point operations and accesses to the factor matrices.

The software is mainly written in Fortran 90 although a C interface is available. The parallel version of MUMPS requires MPI [24, 25] for message passing and makes use of BLAS [236, 27], basic linear algebra communication subprograms (BLACS), and ScaLAPACK [21] libraries. The sequential version relies on BLAS only. The parallel version of MUMPS distributes the tasks among available processors. A host/head processor is required to perform most of the analysis phase, to distribute the incoming matrix to the other processors in the case where the matrix is centralized, and to collect the solution.

3.5 The MAGMA Library

MAGMA is a new generation of linear algebra (LA) libraries that achieve the fastest possible time to an accurate solution on hybrid/heterogeneous architectures by fully exploiting the power that each of the hybrid components offers [28]. Complex challenges arise from these systems' heterogeneity, massive parallelism, and gap in compute power vs. CPU-GPU communication speeds. Addressing such challenges is based on the idea that optimal software solutions have to hybridize, combining the strengths of different algorithms within a single framework.

MAGMA uses hybridization methodology where algorithms of interest are split in to tasks of varying granularity and their execution scheduled over the available hardware components. Scheduling can be static or dynamic. In either case, small non parallelizable tasks, often on the critical path are scheduled on CPU and large parallelizable ones are scheduled on GPU.

MAGMA uses column major data layout as a standard. It is composed of dense linear algebra routines, a successor of LAPACK and ScaLAPACK, specially developed for heterogeneous GPU-based architectures. As shown in Figure 3-1, MAGMA is built on CPU-centric libraries (BLAS and LAPACK) and GPU- centric libraries CUDA and Open Computing Language (OpenCL)). It also took advantage of Many Integrated Core (MIC) architecture of Intel. As shown in the MAGMA software stack

figure below, single CPU and single GPU, multicore CPUs and multiple GPUs and also distributed architectures are supported.

Designed to be similar to LAPACK in functionality, data storage, and interface, it allows scientists to effortlessly port their LAPACK relying software components and to take advantage of the new hybrid architectures.

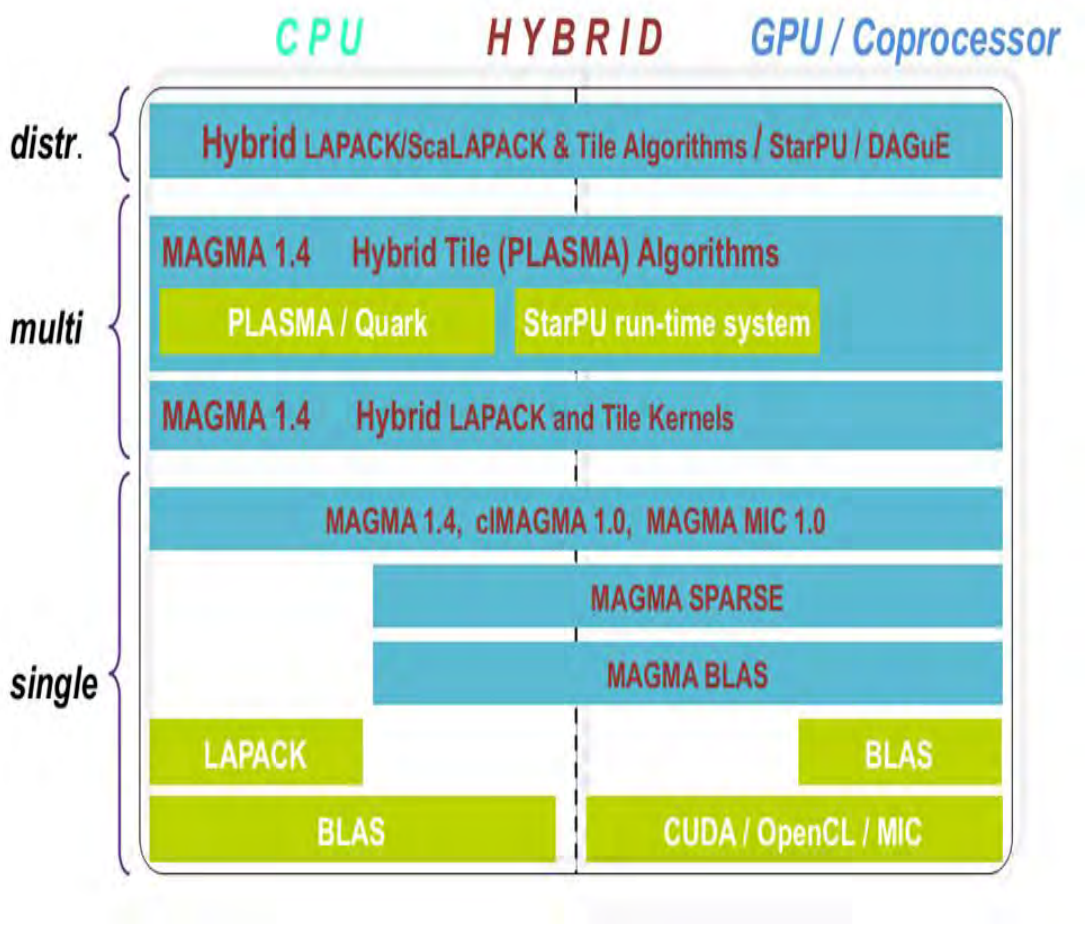

Figure 3-1 MAGMA software stack [29]

MAGMA 1.5.0 is intended for CUDA enabled NVIDIA GPUs such as the GeForce 8 Series, the Tesla GPUs, and some Quadro GPUs [1]. MAGMA's double precision routines can be used on CUDA enabled GPUs that support double precision arithmetic. These are for example the GeForce 200 Series and the Tesla solutions. It supports Tesla (compute capability 1.x), Fermi (compute capability 2.x), and Kepler (compute capability 3.x) GPU architectures.

MAGMA includes the following functions [28]:

- LU, QR and Cholesky factorization.
- Hessenberg, bidiagonal, and tridiagonal reductions
- Linear solvers based on LU, QR and Cholesky decompositions.
- Eigenvalue and singular value problem solvers.
- Generalized Hermitian-definite Eigen problem solver.

- Mixed-precision iterative refinement solvers based on LU, QR and Cholesky factorizations.
- MAGMA BLAS including gemm, gemv, symv, and trsm

All subprograms have four versions corresponding to four data types.

- s – float(real single-precision)
- d – double(real double-precision)
- c – magmaFloatComplex (complex single-precision)
- z – magmaDoubleComplex (complex double-precision)

For each function there are two LAPACK-style interfaces. The first one, referred to as CPU interface, takes the input and produces the result in the CPU's memory. The second, referred to as GPU interface, takes the input and produces the result in the GPU's memory. The algorithm names are derived by the corresponding LAPACK names, prefixed by magma, and for the case of the GPU interface suffixed by GPU.

Key Features of MAGMA 1.5.0 are high performance, multiple precision support (Z, C, D, S, and MP), Hybrid algorithms, Out-of-GPU memory algorithms and Multiple GPU support. It has Support on Linux, Windows, Mac OS X operating systems and C/C++, Fortran programming languages and scripting languages such as Matrix laboratory (MATLAB) and Python.

Depending on the type of matrix to be factorized, magma has two main factorization subroutines each with CPU interface, single GPU interface and multiple GPUs interface. MAGMA_DGETRF_GPU and MAGMA_DPOTRF_GPU are magma matrix factorization function with single GPU interface.

3.6 Intel MKL

Intel MKL is a computing math library of highly optimized, extensively threaded routines for applications that require maximum performance. Intel MKL provides comprehensive functionality support in these major areas of computation [30]:

- BLAS (level 1, 2, and 3) and LAPACK linear algebra routines

- The Parallel Sparse Direct Solver (PARDISO) direct sparse solver, an iterative sparse solver
- ScaLAPACK distributed processing linear algebra routines for Linux and Windows operating systems, as well as the BLACS and the Parallel Basic Linear Algebra Subprograms (PBLAS).
- Fast Fourier transforms (FFT) functions in one, two, or three dimensions with support for mixed radices, as well as distributed versions of these functions provided for use on clusters of the Linux and Windows operating systems.
- Vector Math Library (VML) routines for optimized mathematical operations on vectors.
- Vector Statistical Library (VSL) routines, which offer high-performance vectorized random number generators (RNG) for several probability distributions, convolution and correlation routines, and summary statistics functions.
- Data Fitting Library, which provides capabilities for spline-based approximation of functions, derivatives and integrals of functions, and search.
- Extended Eigen solver, a shared memory programming (SMP) version of an Eigen solver based on the Feast Eigenvalue Solver.

Intel MKL library version 11.1 is used in DuCOM-COM3 since it includes most of the libraries necessary for its linear algebra algorithms and solvers such as BLAS, PARDISO,andScaLAPACK.

Chapter 4 Design and Methodology

There are different methods to improve performance of existing library. One of these is optimizing the library functions. Another recent trend to boost performance of applications is porting to GPUs. This technique can be done either by porting the functions to GPU and optimize it or replacing the CPU libraries used by respective GPU libraries if there are already built in libraries. This work demonstrates DuCOM-COM3 performance improvement by modifying MUMPS factorization through replacement of ScaLAPACK routines for factorization by MAGMA's, respective routines.

4.1 Implementing Hybrid CPU-GPU multi-scale modeling concrete performance Solver

With today's multi-core CPUs and GPUs it is also of importance to use algorithms and methods which use the potential of the hardware to its maximum. Design of hybrid CPU-GPU system deals with how to use the CPUs and GPUs as efficiently as possible in order to achieve the shortest possible simulation wall-clock time.

There are three techniques in porting applications to GPU. The first one is through CUDA kernels. In this scenario CUDA kernel is written for the given parallel application by making some modifications and rearrangements in order to fit the application in GPU. The second one is by using CUDA libraries. In this technique any existing CUDA library with the similar functionality of the CPU library used are searched and used instead of CPU library. The use of already existing GPU libraries will allow application developers to port their applications to GPU and enhance performance with a very low effort. The third technique is through high level programming directives of GPU programming models such as Portland Group (PGI) compiler and Open Accelerators (OpenACC). Among The above mentioned techniques, using existing libraries to port applications to GPU is used in this work.

Among solvers DuCOM-COM3 uses, MUMPS is one of them. MUMPS is a direct multi-frontal solver. All the three stages of MUMPS, preprocessing, factorization and

solution, to solve serious of equations can be performed in parallel. Based on MUMPS profiling results of gprof, factorization is the most intensive one and it took 60% of total computation time of the solver. The above two natures of mumps solver fits the solver for factorization in GPGPU implementation. Since most of computation times are spent with in factorization step, porting the factorization to GPU while the remaining steps are computed in CPU will result hybrid CPU-GPU solver which outperforms CPU based implementation. In order to accomplish this, the MUMPS factorization module was modified to support both single GPU and hybrid CPU-GPU cluster. As such MAGMA subroutines were used to replace corresponding subroutines from MUMPS library.

Figure 4-1 depicts CPU based MUMPS implementation. As shown in appendix B, ScaLAPACK subroutines PDGETRF and PDPOTRF are used for LU factorization and cholesky factorization respectively in CPU based MUMPS implementation.

Figure 4-2 depicts our hybrid CPU-GPU MUMPS implementation for single node with a GPU and hybrid CPU-GPU cluster. As shown in appendix C, MAGMA subroutines MAGMAF_DGETRF_GPU and MAGMAF_DPOTRF_GPU are used for LU factorization and cholesky factorization respectively in this implementation.

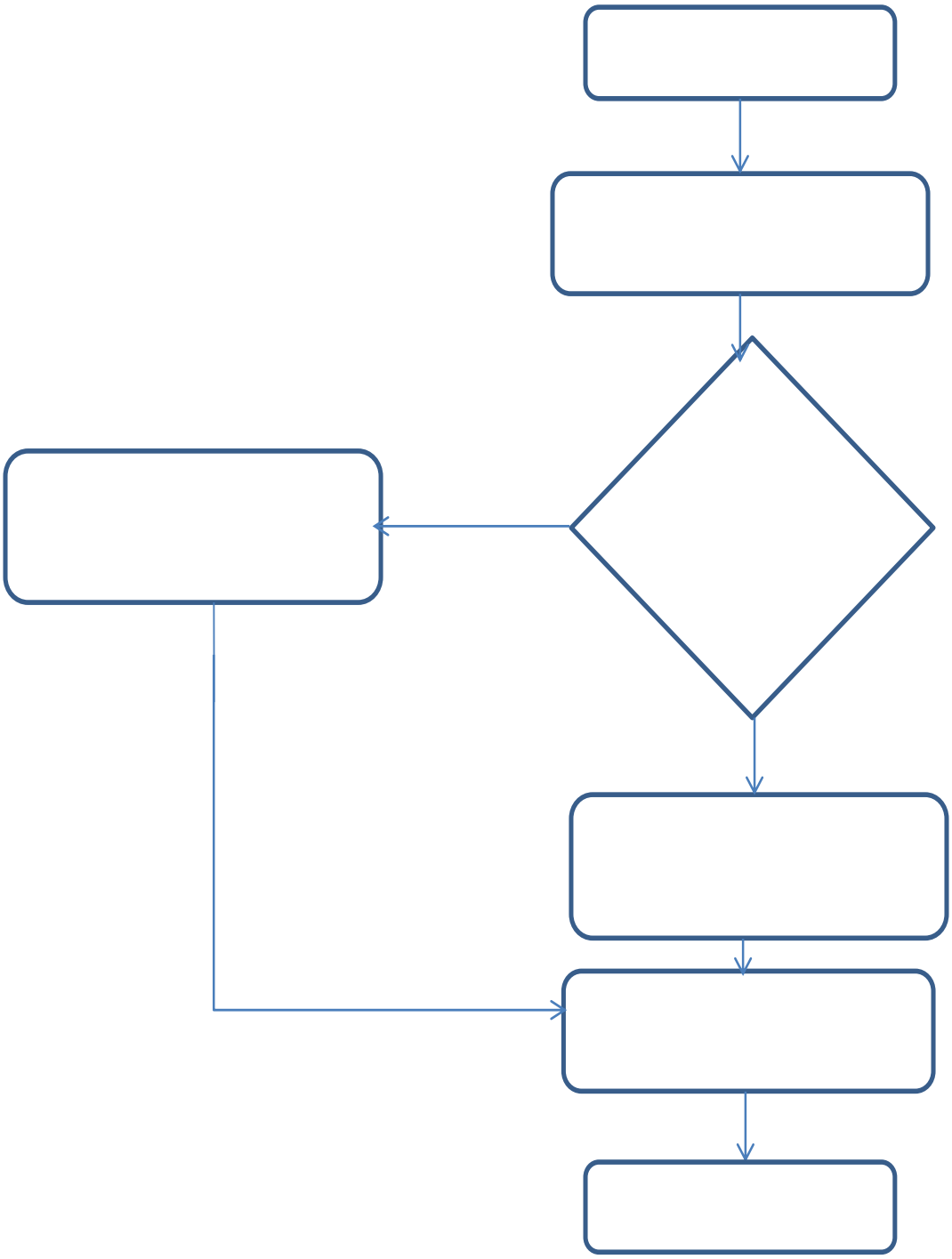

Figure 4-1 Flowchart of CPU Based MUMPS Solver

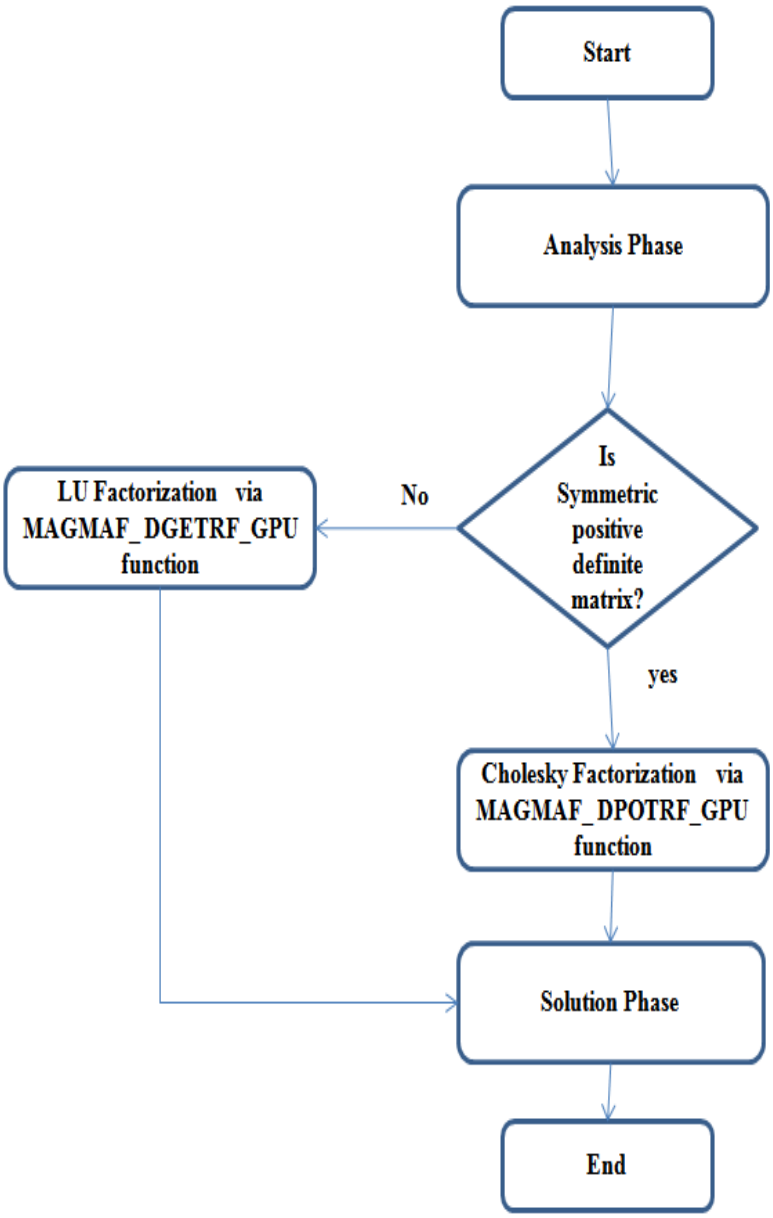

Figure 4-2 Flowchart of Hybrid CPU-GPU MUMPS for single GPU

4.2 Design and Implementation of CPU-GPU Cluster

High-performance computing (HPC) is the use of parallel processing for running advanced application programs efficiently, reliably and quickly. The term applies especially to systems that function above a teraflop or 10^{12} floating-point operations per second (FLOPs). The term HPC is occasionally used as a synonym for supercomputing, although technically a supercomputer is a system that performs at or near the currently highest operational rate for computers. Some supercomputers work at more than a petaflop or 10^{15} FLOPs. In order to conduct our experiment, a CPU-GPU hybrid compute cluster had to be built, as we didn't have access to any. This section describes the procedures we followed to build cluster.

4.2.1 Choosing Cluster Type

There are at least three distinct computational system types that can be built basing on GPUs. The possibility of the utilization of a concrete graphics card for each of the schemes is determined by the supported programming features expressed as compute capability version of the GPU [31]. These are GPU only scheme, hybrid CPU-GPU scheme and GPU-only clusters.

In GPU only scheme, calculations are run on GPU only and CPU stands idle. In such a scheme usually only one GPU per node is used. If it is necessary to scale system beyond one GPU the speed of exchange of data between GPUs through CPU/Random Access Memory (RAM) becomes a bottleneck unless the GPUs incorporate GPU direct technology which allow direct data transfer between GPUs. GPUs with any compute capability version can be used for this scheme. GPUs with compute capability less than 2.0 can be efficiently used only in this way and allow GPU calculations only of simple highly parallel tasks.

In hybrid CPU-GPU scheme, CPU generates tasks for GPU and does the balancing. GPUs with compute capability greater than 2.0 can be used for hybrid CPU-GPU calculations. Load balancing is made between CPU-GPU partitions. Thus, such scheme allows efficient parallelization on one node or between different nodes. But still, for optimal performance with multiple GPUs, especially with parallelization on

multiple nodes, it is preferable to use identical hardware. Depending on the hardware and tasks specificity, one GPU per several cores of CPU may be installed in such systems.

In GPU-only clusters, GPUs generate tasks for GPUs and do the load balancing. Such scheme becomes possible with compute capability 3.5 which introduces “dynamic parallelism” for GPUs. GPUs can be located on one or different nodes. Since during the computations the CPUs are mostly idle and to avoid data exchange delays it is necessary to minimize the number of nodes by maximizing the number of GPUs per node. These systems can be built on special platforms that support multiple GPUs – 4 and more. There are no extra requirements for CPUs and RAM size because GPUs and video memory are mainly used. GPU clusters are intended for tasks that require unprecedented performance.

Currently, for all spheres of scientific computing most in demand are nodes capable of running hybrid CPU-GPU calculations. Such systems can be built both on the basis of new nodes as well as on the basis of already existing nodes and workstations by equipping them with graphics cards of a corresponding productivity. Equipment of the older workstations with high end graphics cards of the previous generations which support compute capability 2.0 and higher can boost the performance of these nodes in hybrid calculations by 60% and more at a low cost [31]. Based on the available NVidia graphics cards of the institute and hybrid nature of the algorithm to be computed in the cluster, the best cluster system which can be build is hybrid CPU-GPU cluster. In this case both the CPU and GPU will be efficiently utilized.

4.2.2 Choose Hardware

In this step necessary hardware are chosen carefully based on the cluster type to be built.

1. CPUs

Choice of CPUs depend on how much of the programs are actually using CPU power. The CPU on head node should be as good as or better than the compute nodes if a lot of developing is achieved on the head, but if the head node is just for remote login

then a cheap 2 core should suffice. Since DuCOM-COM3 has a lot of codes that are executed in CPU, Intel core i7 processor is chosen.

2. GPUs

The available NVidia GeForce GTX260 GPUs with compute capability of 1.3 are used. These GPUs have 192 CUDA cores, 576MHZ graphics clock, 1242MHZ processor clock and 36.9 billion/sec of texture fill rate. Its memory specs are 999MHZ memory clock, 896MB standard memory configuration, 448 bit memory interface width and 111.9 GB/sec memory bandwidth.

Figure 4-3 NVidia GeForce GTX 260[32]

3.Memory

Choice of memory size and speed may be limited by the choice of motherboards, but for a serious yet affordable machine using 4 GB of double data rate synchronous dynamic (DDR2) on a cheap motherboard is a good point. In this work, a machine with 16 GB double data rate types three synchronous dynamic (DDR3) RAM is used.

4. Motherboard

The two most important factors in motherboard choice are the CPU socket and number of GPUs it can support. In this work the available Asus SABERTHOOTH X58 motherboard is used.

5. Hard Drives

Head node with 1 TB or more big hard drive is a must. This is due to the computation results of all compute nodes will be stored in the head nodes. Compute nodes don't need as much larger hard drive. This is due to the use of hard drive in compute node is limited in storing the operating system and any software needed, things like data files should all be written to the head node via a universal mount file system. In this work, the available head node and compute nodes each with 1TB hard drives are used.

6. Power Supplies

Finding power supply that can supply many more watts than your nodes requires at peak is a better idea. This can determine by checking the power specs on the CPU and GPU which are typically about 100 and 200 watts respectively. On top of this other factors like memory, hard drives, fans, etc. should also be considered. Doubling the combined CPU and GPU wattage should give a good low number for acquiring a power supply. Checking the minimum recommended power supply wattage from the GPU manufacturer and adding 200 is also a good way to do it. The power supply used in this cluster is capable for our purpose.

7. Cases

Since we didn't test the system for multiple GPU, the available cases are used in this work.

8. Network Interface Cards

Installation of a gigabit (10/100/1000) network interface card (NIC) is recommended. The motherboard will most likely have one (or a slower 10/100) but this one is probably controlled by the CPU and could slow down your program during communication. These could be considered a cheap investment that will keep your cluster performance optimal. PCI express (PCIe) X8 wide NICs are used in this work.

9. Gigabit Switch

If the cluster to be built has more than two total nodes, a good gigabit switch is needed. We used two TP-Link gigabit Ethernet switches as interconnect between one head node and two compute nodes.

Table 4-1 depicts the hardware chosen to build the hybrid CPU-GPU cluster based on the above mentioned nine steps.

Table 4-1 Chosen hardware for the cluster to be built

Hardware	Type	Quantity
CPU processor	Intel core i7	3
GPU processor	NVidia GeForce GTX260	3
Motherboard	SABERTHOOTH X58	3
power-supply	SILENCER MK2	3
Switch	24 port TP-Link gigabit Ethernet switch	2
Network Card	PCIe x8 wide	3
Internal HDD	1TB for head node	1

4.2.3 Allocate Space, Power and Cooling

The goal for this step is to assess the physical infrastructure, including space, power and cooling needs, network considerations and storage requirements to ensure optimal system choices with room to grow the cluster in the future. Clusters are mainly rack mounted, with multiple machines installed in a vertical rack. Vendors offer many server solutions that minimize the use of rack space.

4.2.4 Assembly and Physical Deployment

In this step a head node and two compute nodes are connected via TP-Link gigabit Ethernet switches. Our hybrid CPU-GPU cluster physical deployment is shown in Figure 4-4.

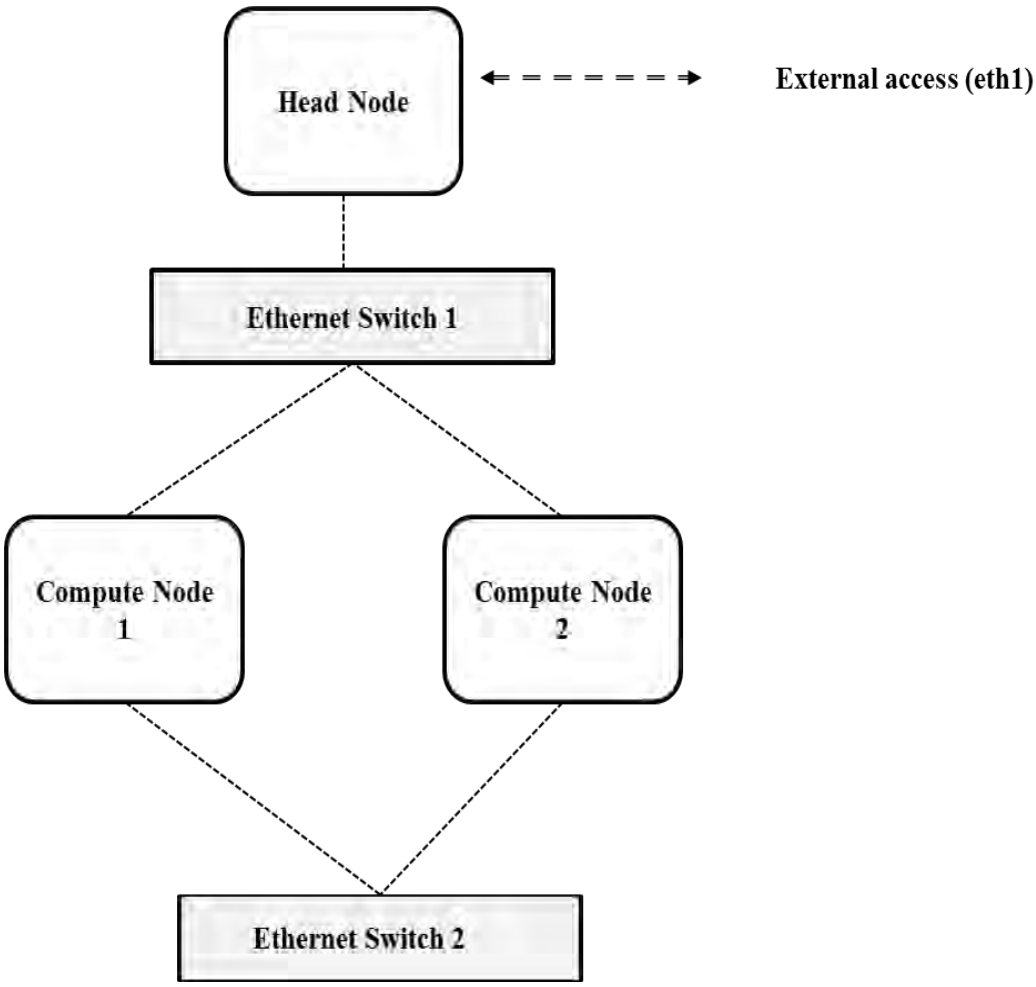

Figure 4-4 Connection between head node and compute nodes

4.2.5 Building High Performance Portable MPI (MPICH) Cluster on Nodes

Hybrid CPU-GPU cluster is built by installing MPICH cluster in three nodes running Ubuntu 14.04 long term support (LTS) server with host names ubuntu01, ubuntu02 and ubuntu03.

The following procedures were followed to build MPICH cluster [33].

- Defining hostnames
- Installing Network File System (NFS)

  NFS allows us to create a folder on the master node and have it synced on all the other nodes. This folder can be used to store programs.

- Sharing Master Folder
- Mounting /master in nodes
- Defining a user for running MPI programs
- Installing secure shell (SSH) Server
- Setting up password less SSH for communication between nodes
- Installing gnu c compiler (GCC) and other compilers
- Installing MPICH2
- Setting up a file called "machinefile" and it is created in user's home directory with node names followed by a colon and a number of processes to spawn

4.3 Porting DuCOM-COM3 to Linux

DuCOM-COM3 source code is made up of 23 FORTRAN 2010 source files. It uses Intel MKL, Iphreeqc library, MUMPS direct solver, high performance precoditioners (HYPRE) iterative solver and hierarchical iterative parallel solver (HIPS) solver. It also uses MPI for parallel implementations. To compile and link the source code for Linux environment, respective Linux libraries of the above mentioned libraries were installed. Besides that Intel Fortran compiler (IFORT) version 14.1.0 was also used. As described in appendix A, make file for DuCOM-COM3 compilation was prepared.

4.4 Experimental Setup

4.4.1 Single GPU Setup

Ubuntu 14.04 LTS Desktop is installed on the machine used for our experiments. This machine has Intel(R) Core(TM) i7 950 @ 3.07GHz CPU and NVIDIA GeForce GTX260 GPU.

All the required libraries listed below are installed:

- Intel composer 2015
- MAGMA 1.5.0
- ScaLAPACK 2.0.0
- MUMPS 4.10.0
- CUDA 6
- PHREEQC 3.16
- HIPS 1.2
- METIS 5.10

4.4.2 Building Hybrid CPU-GPU Cluster

A hybrid CPU-GPU cluster which is built in section 4.2 is used. All the necessary libraries mentioned above are installed.

Chapter 5 Experiments and Results

This chapter presents experiments conducted and results obtained for our hybrid CPU-GPU MUMPS solver implemented for performance improvement of DuCOM-COM3 in the previous chapter. In this chapter, our hybrid CPU-GPU MUMPS implementation is evaluated and results are presented. Performance comparisons are done using sparse matrix collection of University of Florida and also on DUCOM-COM3.

This chapter is organized as follows. The first section reviews the input matrices used during evaluations described in this report. In order to see the influence of GPU porting of the factorization of MUMPS solver, our hybrid CPU-GPU MUMPS implemented in chapter 4 and MUMPS 4.10.0 are tested using symmetric positive definite, general symmetric and unsymmetrical matrices. The tests are done both on single node and hybrid CPU-GPU cluster built on section 4.2. Finally, performance of DUCOM-COM3 was tested using our hybrid implementation on single node and hybrid CPU-GPU cluster.

5.1 Test Data

Sparse matrix collection from University Of Florida is used as a test data for the first experiment. This collection has a large and actively growing set of sparse matrices that arise in real applications. The Collection is widely used by the numerical linear algebra developer community for the development and performance evaluation of sparse matrix algorithms. The matrices are available in three formats: MATLAB mat-file, Rutherford-Boeing, and Matrix Market. In this work, the Rutherford-Boeing format was used. The Matrices used in the first experiment are shown in Table 5-1 and Table 5-2.

Table 5-1 Matrices used for LU Decomposition

Name	lock1074	Almedar	Poc20stif	NotreDame_www
Matrix size	1074*1074	6245 X 6245	52329 x 52329	325,729 x325,729
number of nonzero	51,588	42,581	2,698,463	929,849
Sparcity (%)	95.6	99.9	99.81	99.9992

Table 5-2 Matrices used for Cholesky LU Decomposition

Name	Andrianov/ex3sta1	Andrews	Gearbox
Matrix size	17443 x 17443	60000 x 60000	1.534,769 x 1,534,769
Number of nonzero	678,998	760,154	9,080,404
sparcity (%)	99.78	99.98	99.99962

For the second experiment we used two concrete sample inputs called Itmp and Rtmp as an input for DuCOM-COM3. Itmp is a concrete sample which is used for analyzing concrete behavior before load is applied to it whereas Rtmp is used for analyzing concrete behavior after that concrete is exposed to environmental effects and loads. Each concrete samples have three data files that include matrices used as an input for HEAT-model, HYGR-model and MECH-model which are described in Table 3-1. The rest of the models won't have inputs during execution of DuCOM-COM3.

5.2 Experimental Results

5.2.1 Effect of Porting Factorization Phase of MUMPS to GPU

The first experiment investigates the performance of MUMPS 4.10.0 solver and hybrid CPU-GPU MUMPS in symmetric positive definite, general symmetric and asymmetric matrices. Since the type of factorization of MUMPS solver depends on the type of matrix used, input matrix were grouped in to matrices for LU factorization and matrices for Cholesky factorization as shown in Table 5-1 and Table 5-2

respectively. Figure 5-1 and Figure 5-2 depict the performance of factorization of MUMPS 4.10.0 and hybrid CPU-GPU MUMPS in terms of computation time in single node and cluster across a range of sparse matrix types.

The symmetric matrix is first partitioned into blocks. Massively parallel tasks like matrix multiplication are performed on GPUs and other serial tasks with heavy inner data dependence such as triangular solvers are executed on CPUs. The data communication cost between the host CPU and GPU is covered because most of the matrices have sparcity of 99.9% and above and Hence, the dense Multifrontal matrix ready for factorization is very small.

Figure 5-1 depicts that our hybrid CPU-GPU MUMPS implementation on a single node reduced runtime requirement of LU factorization by 89.74% for matrix lock1074, 88.8% for matrix Almedar. However, there is no change in running time for pct20stif and NotreDame_www matrices. This is due to as the sparcity increases, MUMPS computation time decreases for both CPU MUMPS and our hybrid CPU-GPU MUMPS. But due to the large matrix sizes, our implementation faced data communication overhead between CPU and GPU. Hence, though the matrices factorization benefited due to the hybrid implementation, the above two mentioned reasons covered the performance improvement. A maximum runtime reduction of 89.74% was achieved when matrix lock1074 was used as an input. This showed us MUMPS exploits parallelism from the dense nature of the input matrix.

Figure 5-1 depicts that our hybrid CPU-GPU MUMPS implementation on a hybrid CPU-GPU cluster reduced runtime requirement of LU factorization by 84.61% for matrix lock1074, 94.4% for matrix Almedar and 50% for pct20stif and NotreDame_www matrices. Hence, a maximum runtime reduction of 94.4% was achieved when matrix Almedar was used as an input.

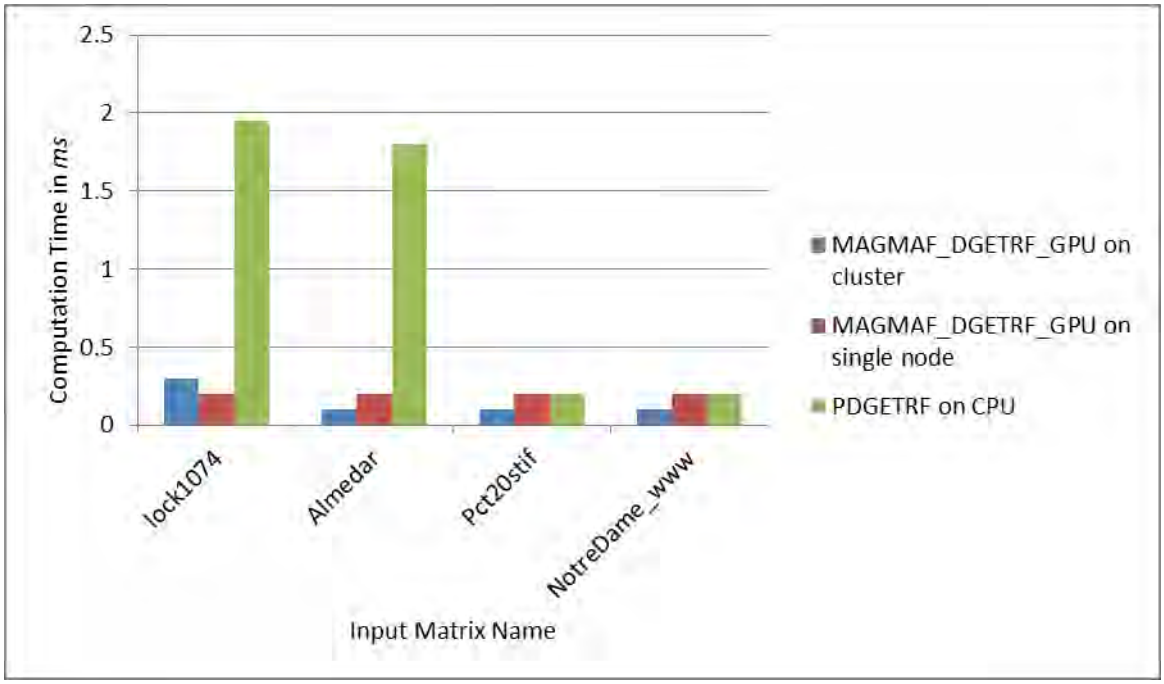

Figure 5-1 LU Factorization

Figure 5-2 depicts that our hybrid CPU-GPU MUMPS implementation on a single node reduced runtime requirement of cholesky factorization by 50% for Ex3stal and Gearbox matrices and 88.8% for matrix Andrews. A maximum runtime reduction of 88.8% was achieved when matrix Andrews was used as an input. This showed us MUMPS exploits parallelism from the dense nature of the input matrix.

Figure 5-2 depicts that our hybrid CPU-GPU MUMPS implementation on a hybrid CPU-GPU cluster reduced runtime requirement of cholesky factorization by 55% for matrix Ex3stal, 90% for Andrews matrix and 57.5% for Gearbox matrix reduction was achieved. Hence, a maximum runtime reduction of 90% was achieved when matrix Andrews was used as an input.

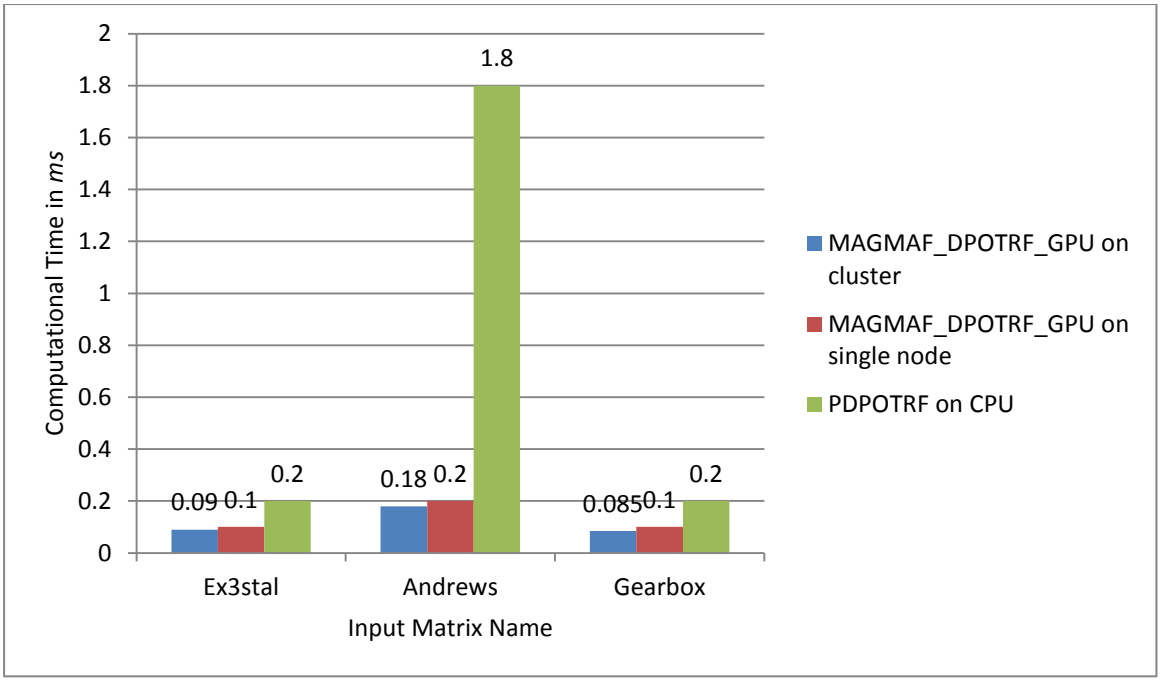

Figure 5-2 Cholesky Factorization

From Figure 5-1 and Figure 5-2, it can be seen that the MAGMA LU factorization subroutines used in this application outperform ScaLAPACK factorization routines respectively. This is due to utilization of heterogeneous GPUs and CPUs by the MAGMA library.

5.2.2 Effect of Hybrid CPU-GPU MUMPS in DuCOM-COM3 Performance

Following the above set of experiments, effect of hybrid CPU-GPU MUMPS in DuCOM-COM3 performance was investigated. Two real world concrete samples called Itmp and Rtmp are used inputs for testing the Modified DuCOM-COM3. Figure 5-3 depicts that our hybrid CPU-GPU MUMPS implementation on a single node reduced runtime requirement of DuCOM-COM3 by 35.29% for Itmp input data while for Rtmp 20% reduction was achieved. Hence utilization of both CPU and GPU by tasks that best fit in them considerably enhances performance.

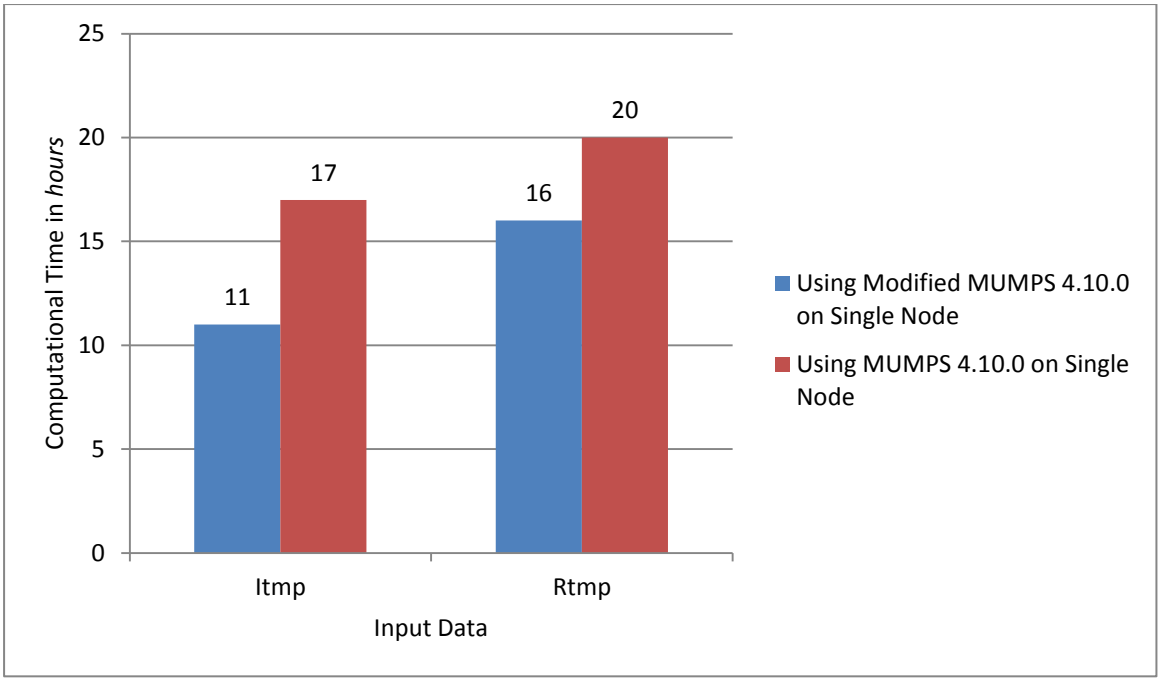

Figure 5-3 Performance Comparison of DuCOM-COM3 on Single Node

The results for the experiments done on the CPU-GPU cluster are shown in Figure 5-4 Our hybrid CPU-GPU MUMPS implementation reduced runtime requirement of DuCOM-COM3 by 37.5% as compared to CPU only cluster for the Itmp input data. For the Rtmp input data runtime was reduced by 22.2% from 18 hours to 14 hours to complete computation. However, we had anticipated more performance on the cluster as there are many compute resources that could be utilized. It is believed that the reason behind this lower than expected improvement on the hybrid CPU-GPU cluster arose from data communication overhead. That is data has to be exchanged between the nodes and that same data had to be copied to the GPU for computation.

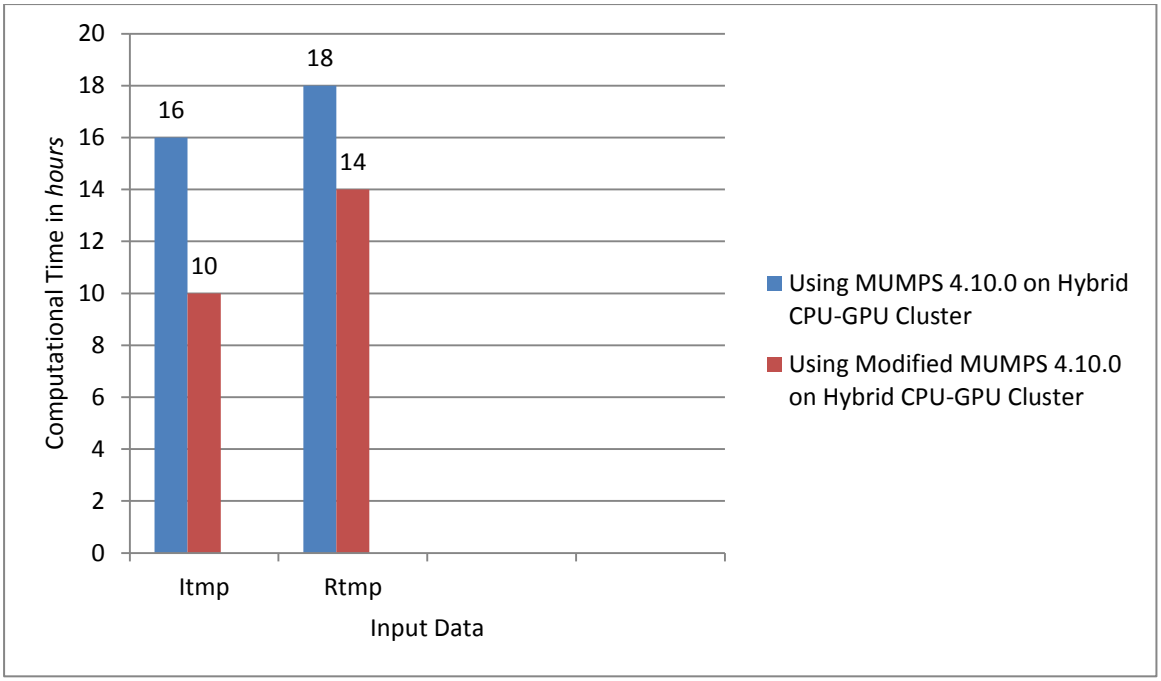

Figure 5-4 Performance Comparison of DuCOM-COM3 on Hybrid CPU-GPU Cluster

Chapter 6 Conclusion and Recommendation

This thesis has investigated the effect of porting LU factorization module of DuCOM-COM3 to GPU. The MUMPS library used in DuCOM-COM3 was modified to support a CPU-GPU heterogeneous system. This chapter summarizes the findings based on experiments conducted. Then, it reviews the major contributions and presents suggestions for future work.

6.1 Conclusion

In order to identify the bottleneck in the MUMPS solver of DuCOM-COM3, profiling was carried out. Based on the profiling results, the factorization phase was identified as the most time consuming one among the three phases of MUMPS solver. Hence, the factorization is ported to GPU while the other stay in CPU and as a result a hybrid CPU-GPU MUMPS solver was produced.

Experiments were conducted on a single machine with GPU and a CPU-GPU cluster. In the experiments, symmetric positive definite, general symmetric and unsymmetrical sparse matrices from University of Florida sparse matrix collection were considered. From the results obtained, it was observed that the hybrid CPU-GPU implementation resulted a maximum runtime reduction of LU factorization89.74% for the single machine and 94.4% for the cluster.

The effectiveness of hybrid CPU-GPU implementation was further investigated by using it in DUCOM-COM3. In this case, concrete material and structural analysis samples were used as input. From the results obtained, the use of hybrid CPU-GPU MUMPS in DuCOM-COM3 resulted a maximum runtime reduction of 35.29% and 37.5% in single node and cluster setups respectively as compared to the base DuCOM-COM3 implementation.

In this thesis, the implementation of hybrid CPU-GPU MUMPS by replacing original SCALAPACK subroutines used for factorization in MUMPS by respective MAGMA subroutines. In addition, the use of clustering computers to produce a high

performance computer was studied. Hence, a hybrid CPU-GPU cluster using MPICH and Ubuntu server was build. As a result DUCOM-COM3 solver benefited both from the hybrid CPU-GPU MUMPS and the MPICH cluster built and its performance is improved. As the results of our experiments depicted, the exploitation of heterogeneous systems by hybrid CPU-GPU algorithms enhances performance of applications by low cost and effort.

6.2 Recommendation

There are many possible extensions to the performance improvement of DuCOM-COM3. The following are recommendation for further improvement.

- The investigations carried out in this thesis used NVidia GeForce GTX260 GPUs. These are Tesla GPUs with compute capability 1.3. The use of Kepler GPUs with computes capability 3.X and higher would benefit the performance. This is because Kepler GPU has a feature called GPU Direct which allows a direct memory transfer between CPU and GPU.
- There are large data read operations in DuCOM-COM3. So, investigating better way of representing input data would be another approach for performance improvement.
- There are also other solvers that are used in DuCOM-COM3 and Investigating whether these solvers are suitable to GPGPU porting and if so finding a way to port them to GPU will enhance performance.

Bibliography

[1] D. Pierce *et al.,* "Sparse Multifrontal Performance Gains via NVIDIA GPU," Jan. 16, 2009.

[2] NVIDIA. (Aug. 2014).*CUDA Compute Unified Device Architecture Programming Guide 6.0*[online].Available:http://www.nvidia.com/cuda. Accessed On: Aug. 15, 2014.
[3]M.Macedonia, "The GPU enters computing's mainstream.Computer," *IEEE Comput. Society Press*, Los Alamitos, CA,2003, pp. *106–108.*

[4] J. D. Owens *et al*., "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum 26(1),* Davis, CA, 2007, pp 80-113.

[5] *General Purpose Computations on Graph. Processing Units* [online]. Available: http://www.gpgpu.org. Accessed On: Nov. 10, 2013.
[6]. "GPU-accelerated applications for HPC industries," *NVIDIA,* Santa Clara, CA, Nov. 2013.

[7].K. Maekawa*e. al.* , "Multi-scale Modeling of Concrete Performance," Dept. of Civil Eng. ,Univ. of Tokyo, Japan, May 20 , 2003.

[8] R. Duarte, "Improving Performance of Data-Parallel Applications on CPU-GPU Heterogeneous Systems," Ph.D. Dissertation, Dept. of Elect., Univ. of Rhode Island, Kingston, RI, 2013.

[9]K. Maekewa and T. Ishida, "Multi-scale Concrete Model," ConcreteLaboratory, Dept. of Civil Eng., Univ. of Tokyo, Tokyo, Japan, DuCOM-COM3 Version 5.15, Oct. 31, 2013.
[10] E. Agullo ,"On the Out-Of-Core Factorization of Large Sparse Matrices," Feb. 2011.

[11] J. R. Gilbert *and* S. Toledo *,"*High-Performance Out-of-Core Sparse LU Factorization," *9th SIAM Conf. on Parallel processing for Sci. Comput., Mar.* 1999.

[12]L. Ren*et al.,* "LU Factorization for Parallel Circuit Simulation on GPU," *Tsinghua Nat. Laboratory for Inform. Sci. and Technology*, Dept. of Electron. Eng., Tsinghua Univ., Beijing, China, June, 2012.

[13]J. Bolz*et al.,* "Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid," 2008.

[14] J. Gonzalez1 and Rafael C. Nunez, "Extreme-Speed Scalable Direct Sparse Solvers for Heterogeneous Supercomputing –An Enhancement to the LAPACKrc Library," *Accelogic,* Weston, FL, 2013.

[15] "Breakthroughs in Sparse Solvers for GPUs," *CCOE,* Univ. of Tennessee, ,Knoxville, 2014.

[16] *cuSPARSE* [online]. Available: *https://developer.nvidia.com/cuSPARSE*. Accessed On: Dec. 15, 2014.

[17] E. Agullo *et al.*, "QR Factorization on a Multicore Node Enhanced with Multiple GPU Accelerators," *INRIA*, Univ. of Bordeaux, Bordeaux, France, Dec 16, 2010.

[18] F. Lopez Joint work with IRIT Toulouse, "Towards a multifrontal QR factorization for hetrogenousarchetectures over runtime system," *MUMPS Users Group Meeting*, Bordeaux, France, 2013.

[19] "DuCOM," *Concrete Laboratory, Dept. of Civil Eng., School of Eng., Univ. of Tokyo, Tokyo,Japan*,2008.

[20] (2011, Nov 16). *LAPACK-Linear Algebra Package* [online]. Available: http://www.netlib.org/lapack/.Accessed on: 20 February 2014 .

[21] L. S. Blackford*et al.*, "*ScaLAPACK Users'Guide," SIAM Press*, Auckland, New Zealand, 1997.

[22] J. Schulze, "Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods," *Proc. 11th AnnuBIT Conf.*, Manchester, UK, 2001.

[23] P.R. Amestoy*et al.*, "a fully asynchronous multifrontal solver using distributed dynamic scheduling," Aug. 9, 1999.

[24] G. Karypis and V. Kumar, "METIS – A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0," Univ. of Minnesota, Minneapolis, MN, Sept. 1998.

[25] M. Snir*et al.*, "MPI: The Complete Reference",, The MIT Press, Cambridge, MA, 1996.

[26] J. J. Dongarra*et al.*, "Algorithm 679. A set of Level 3 Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, Salt Lake City, UT, Sec. 16:1–17, 1990.

[27] J. J. Dongarra *et al.*, "Algorithm 679. A set of Level 3 Basic Linear Algebra Subprograms: model implementation and test programs," *ACM Transactions on Mathematical Software*, Salt Lake City, UT, Sec. 16, pp. 18–28, 1990.

[28] (2013,Jan. 10).*Matrix Algebra on Multicore Architectures* [online]. Available: http://icl.cs.utk.edu/magma/, (Accessed: Jan. 23, 2014).

[29] K. J. Dongarra*et al.*, "MAGMA: a New Generation of Linear algebra Libraries for GPU and Multicore Architectures," Univ. of Tennessee, Konxville, TN, MAGMA 1.4 Release, Aug. 14, 2013.

[30]Intel developer team, "Intel Math Kernel Library for Linux OS User's Guide,"Silicon Valley, CA, Version 007, Aug. 25, 2008.

[31] P. Nikolay*et al.*, "Hybrid CPU-GPU calculations – a promising future for computational biology,"*Third International Conf. High Performance Computing HPC-UA 2013,*Kyiv ,Ukraine, 2013.

[32] (2008, May). *NVIDIA GeForce GTX 200 GPU Architecture Overview* [online]. Available:http://www.nvidia.com/docs/IO/55506/GeForce_GTX_200_GPU_Technical_Brief.pdf.Accessed on: 20 Jan. 2013.

[33] (2014, Feb.16).*MpichCluster* [online]. Available:https://help.ubuntu.com/community/MpichCluster, (Accessed On: 12December2014.

[34].B. Sothern. (2005). *Math.Libraries* [online]. Available: http://www.westgrid.ca/Math libraries/. Accessed on: 23 November 2013 .

[35]. E. Agullo ,"The MUMPS library," 1st *Joint Laboratory for Petascale Computing Workshop* , INRIA and Univ. of Illunois, Illunois, June 10-12, 2009 .

[36]. J. Kraus. (2013, Mar 13). *An Introduction to CUDA-Aware MPI* [online]. Available:http://devblogs.nvidia.com/parallelforall/introduction-cuda-aware-mpi/. Accessed on: 26 November 2013 .

[37] S. Boggan and D.M. Pressel,"GPUs: An Emerging Platform for General-Purpose Computation," *Army Research Laboratory*, Aug. 2007.

[38] F. Lopez, "Towards a multifrontal QR factorization for heterogeneous architectures over runtime systems," *Toulouse Inst. Of Comput. Sci. Research*, Univ. Toulouse, Toulouse, France

[39] (2012, May 01). SCA*LAPACK-Scalible Linear Algebra Package* [online]. Available:http://www.netlib.org/scalapack/, (Accessed On : 20 February 2014).
[40] (2014, May 02). *PLASMA* [online]. Available: http://icl.cs.utk.edu/plasma, Accessed on: 10 May 2014 .

[41] "NVIDIA CUDA Programming Guide," NVIDIA, Santa Clara, CA, Version 2, June 07, 2008.

[42] P. Gupta. (2013, Apr. 30). *How to Build a GPU-Accelerated Research Cluster*[online]. Available:http://devblogs.nvidia.com/parallelforall/How- to- Build-a-GPU-Accelerated-Research-Cluster/.Accessed on: Dec. 20, 2013.

Appendix A

Subroutine DMUMPS_146 of MUMPS_4.10.0

```
SUBROUTINE DMUMPS_146(MYID,root,N,IROOT,
&COMM,IW,LIW,IFREE,
&A,LA,PTRAST,PTLUST_S,PTRFAC,
&STEP,INFO,LDLT,QR,
&WK,LWK,KEEP,KEEP8,DKEEP)
IMPLICITNONE
INCLUDE'dmumps_root.h'
INCLUDE'mpif.h'
TYPE(DMUMPS_ROOT_STRUC)::root
INTEGERN,IROOT,COMM,LIW,MYID,IFREE
INTEGER(8)::LA
INTEGER(8)::LWK
DOUBLEPRECISIONWK(LWK)
INTEGERKEEP(500)
DOUBLEPRECISIONDKEEP(30)
INTEGER(8)KEEP8(150)
INTEGER(8)::PTRAST(KEEP(28))
INTEGER(8)::PTRFAC(KEEP(28))
INTEGERPTLUST_S(KEEP(28)),STEP(N),IW(LIW)
INTEGERINFO(2),LDLT,QR
DOUBLEPRECISIONA(LA)
INTEGERIOLDPS
INTEGER(8)::IAPOS
INTEGERLOCAL_M,LOCAL_N,LPIV,IERR,allocok
INTEGERFWD_LOCAL_N_RHS,FWD_MTYPE
INCLUDE'mumps_headers.h'
EXTERNALnumroc
INTEGERnumroc
IF(.NOT.root%yes)RETURN
IF(KEEP(60).NE.0)THEN
IF((LDLT==1.OR.LDLT==2).AND.KEEP(60)==3)THEN
```

```
CALLDMUMPS_320(WK,root%MBLOCK,
&root%MYROW,root%MYCOL,root%NPROW,root%NPCOL,
&root%SCHUR_POINTER(1),
&root%SCHUR_LLD,root%SCHUR_NLOC,
&root%TOT_ROOT_SIZE,MYID,COMM)
ENDIF
RETURN
ENDIF
IOLDPS=PTLUST_S(STEP(IROOT))+KEEP(IXSZ)
IAPOS=PTRAST(STEP(IROOT))
LOCAL_M=IW(IOLDPS+2)
LOCAL_N=IW(IOLDPS+1)
IAPOS=PTRFAC(IW(IOLDPS+4))
IF(LDLT.EQ.0.OR.LDLT.EQ.2.OR.QR.ne.0)THEN
LPIV=LOCAL_M+root%MBLOCK
ELSE
LPIV=1
ENDIF
IF(associated(root%IPIV))DEALLOCATE(root%IPIV)
root%LPIV=LPIV
ALLOCATE(root%IPIV(LPIV),stat=allocok)
IF(allocok.GT.0)THEN
INFO(1)=-13
INFO(2)=LPIV
WRITE(*,*)MYID,':problemallocatingIPIV(',LPIV,')inroot'
CALLMUMPS_ABORT()
ENDIF
CALLDESCINIT(root%DESCRIPTOR(1),root%TOT_ROOT_SIZE,
&root%TOT_ROOT_SIZE,root%MBLOCK,root%NBLOCK,
&0,0,root%CNTXT_BLACS,LOCAL_M,IERR)
IF(LDLT.EQ.2)THEN
IF(root%MBLOCK.NE.root%NBLOCK)THEN
WRITE(*,*)'Error:symmetrizationonlyworksfor'
WRITE(*,*)'squareblocksizes,MBLOCK/NBLOCK=',
```

```
&root%MBLOCK,root%NBLOCK
CALLMUMPS_ABORT()
ENDIF
IF(LWK.LT.min(
&int(root%MBLOCK,8)*int(root%NBLOCK,8),
&int(root%TOT_ROOT_SIZE,8)*int(root%TOT_ROOT_SIZE,8)
&))THEN
WRITE(*,*)'Notenoughworkspaceforsymmetrization.'
CALLMUMPS_ABORT()
ENDIF
CALLDMUMPS_320(WK,root%MBLOCK,
&root%MYROW,root%MYCOL,root%NPROW,root%NPCOL,
&A(IAPOS),LOCAL_M,LOCAL_N,
&root%TOT_ROOT_SIZE,MYID,COMM)
ENDIF
IF(LDLT.EQ.0.OR.LDLT.EQ.2)THEN
CALLpdgetrf(root%TOT_ROOT_SIZE,root%TOT_ROOT_SIZE,
&A(IAPOS),
&1,1,root%DESCRIPTOR(1),root%IPIV(1),IERR)
IF(IERR.GT.0)THEN
INFO(1)=-10
INFO(2)=IERR-1
ENDIF
ELSE
CALLpdpotrf('L',root%TOT_ROOT_SIZE,A(IAPOS),
&1,1,root%DESCRIPTOR(1),IERR)
IF(IERR.GT.0)THEN
INFO(1)=-40
INFO(2)=IERR-1
ENDIF
ENDIF
IF(KEEP(258).NE.0)THEN
IF(root%MBLOCK.NE.root%NBLOCK)THEN
write(*,*)"InternalerrorinDMUMPS_146:",
```

```
&"Blocksizedifferentforrowsandcolumns",
&root%MBLOCK,root%NBLOCK
CALLMUMPS_ABORT()
ENDIF
CALLDMUMPS_763(root%MBLOCK,root%IPIV(1),root%MYROW,
&root%MYCOL,root%NPROW,root%NPCOL,A(IAPOS),LOCAL_M,
&LOCAL_N,root%TOT_ROOT_SIZE,MYID,DKEEP(6),KEEP(259),
&LDLT)
ENDIF
IF(KEEP(252).NE.0)THEN
FWD_LOCAL_N_RHS=numroc(KEEP(253),root%NBLOCK,
&root%MYCOL,0,root%NPCOL)
FWD_LOCAL_N_RHS=max(1,FWD_LOCAL_N_RHS)
FWD_MTYPE=1
CALLDMUMPS_768(
&root%TOT_ROOT_SIZE,
&KEEP(253),
&FWD_MTYPE,
&A(IAPOS),
&root%DESCRIPTOR(1),
&LOCAL_M,LOCAL_N,FWD_LOCAL_N_RHS,
&root%IPIV(1),LPIV,
&root%RHS_ROOT(1,1),LDLT,
&root%MBLOCK,root%NBLOCK,
&root%CNTXT_BLACS,IERR)
ENDIF
RETURN
ENDSUBROUTINEDMUMPS_146
```

Appendix B

Subroutine DMUMPS_146 of MUMPS_4.10.0 for single GPU

```
SUBROUTINEDMUMPS_146(MYID,root,N,IROOT,COMM,IW,LIW,IFREE,
A,LA,PTRAST,PTLUST_S,PTRFAC,STEP,INFO,LDLT,QR,WK,LWK,KEEP
,KEEP8,DKEEP)
USEmagma
IMPLICITNONE
INCLUDE'dmumps_root.h'
INCLUDE'mpif.h'
TYPE(DMUMPS_ROOT_STRUC)::root
INTEGERN,IROOT,COMM,LIW,MYID,IFREE
INTEGER(8)::LA
INTEGER(8)::LWK
DOUBLEPRECISIONWK(LWK)
INTEGERKEEP(500)
DOUBLEPRECISIONDKEEP(30)
INTEGER(8)KEEP8(150)
INTEGER(8)::PTRAST(KEEP(28))
INTEGER(8)::PTRFAC(KEEP(28))
INTEGERPTLUST_S(KEEP(28)),STEP(N),IW(LIW)
INTEGERINFO(2),LDLT,QR
DOUBLEPRECISIONA(LA)
INTEGERIOLDPS
INTEGER(8)::IAPOS
INTEGERLOCAL_M,LOCAL_N,LPIV,IERR,allocok
INTEGERFWD_LOCAL_N_RHS,FWD_MTYPE
INTEGER*8::d_A
INCLUDE'mumps_headers.h'
EXTERNALnumroc
INTEGERnumroc
IF(.NOT.root%yes)RETURN
IF(KEEP(60).NE.0)THEN
IF((LDLT==1.OR.LDLT==2).AND.KEEP(60)==3)THEN
```

```
CALLDMUMPS_320(WK,root%MBLOCK,
&root%MYROW,root%MYCOL,root%NPROW,root%NPCOL,
&root%SCHUR_POINTER(1),
&root%SCHUR_LLD,root%SCHUR_NLOC,
&root%TOT_ROOT_SIZE,MYID,COMM)
ENDIF
RETURN
ENDIF
IOLDPS=PTLUST_S(STEP(IROOT))+KEEP(IXSZ)
IAPOS=PTRAST(STEP(IROOT))
LOCAL_M=IW(IOLDPS+2)
LOCAL_N=IW(IOLDPS+1)
IAPOS=PTRFAC(IW(IOLDPS+4))
IF(LDLT.EQ.0.OR.LDLT.EQ.2.OR.QR.ne.0)THEN
LPIV=LOCAL_M+root%MBLOCK
ELSE
LPIV=1
ENDIF
IF(associated(root%IPIV))DEALLOCATE(root%IPIV)
root%LPIV=LPIV
ALLOCATE(root%IPIV(LPIV),stat=allocok)
IF(allocok.GT.0)THEN
INFO(1)=-13
INFO(2)=LPIV
WRITE(*,*)MYID,':problemallocatingIPIV(',LPIV,')inroot'
CALLMUMPS_ABORT()
ENDIF
CALLDESCINIT(root%DESCRIPTOR(1),root%TOT_ROOT_SIZE,
&root%TOT_ROOT_SIZE,root%MBLOCK,root%NBLOCK,
&0,0,root%CNTXT_BLACS,LOCAL_M,IERR)
IF(LDLT.EQ.2)THEN
IF(root%MBLOCK.NE.root%NBLOCK)THEN
WRITE(*,*)'Error:symmetrizationonlyworksfor'
WRITE(*,*)'squareblocksizes,MBLOCK/NBLOCK=',
```

```
&root%MBLOCK,root%NBLOCK
CALLMUMPS_ABORT()
ENDIF
IF(LWK.LT.min(
&int(root%MBLOCK,8)*int(root%NBLOCK,8),
&int(root%TOT_ROOT_SIZE,8)*int(root%TOT_ROOT_SIZE,8)
&))THEN
WRITE(*,*)'Notenoughworkspaceforsymmetrization.'
CALLMUMPS_ABORT()
ENDIF
CALLDMUMPS_320(WK,root%MBLOCK,
&root%MYROW,root%MYCOL,root%NPROW,root%NPCOL,
&A(IAPOS),LOCAL_M,LOCAL_N,
&root%TOT_ROOT_SIZE,MYID,COMM)
ENDIF
     CALLCUBLAS_INIT()
     CALLMAGMAF_INIT()
     CALLCUBLAS_ALLOC(max(1,root%MBLOCK)*root%NBLOCK,size
of_double,d_A)
     CALLCUBLAS_SET_MATRIX(root%MBLOCK,root%NBLOCK,sizeof
_double,A(IAPOS),max(1,root%TOT_ROOT_SIZE),d_A,max(1,root
%TOT_ROOT_SIZE))
IF(LDLT.EQ.0.OR.LDLT.EQ.2)THEN
CALLMAGMAF_DGETRF_GPU(root%MBLOCK,root%NBLOCK,d_A,max(1,r
oot%MBLOCK),root%IPIV,IERR)
IF(IERR.GT.0)THEN
INFO(1)=-10
INFO(2)=IERR-1
ENDIF
ELSE
CALLMAGMAF_DPOTRF_GPU('L',root%TOT_ROOT_SIZE,A(IAPOS),max
(1,root%TOT_ROOT_SIZE),IERR)
IF(IERR.GT.0)THEN
INFO(1)=-40
```

```
INFO(2)=IERR-1
ENDIF
ENDIF
CALLCUBLAS_GET_MATRIX(root%MBLOCK,N,root%NBLOCK,d_A,max(1
,root%TOT_ROOT_SIZE),A(IAPOS),max(1,root%TOT_ROOT_SIZE))
CALLCUBLAS_FREE(d_A)
CALLMAGMAF_FINALIZE()
CALLCUBLAS_SHUTDOWN()
IF(KEEP(258).NE.0)THEN
IF(root%MBLOCK.NE.root%NBLOCK)THEN
write(*,*)"InternalerrorinDMUMPS_146:",
&"Blocksizedifferentforrowsandcolumns",
&root%MBLOCK,root%NBLOCK
CALLMUMPS_ABORT()
ENDIF
CALLDMUMPS_763(root%MBLOCK,root%IPIV(1),root%MYROW,
&root%MYCOL,root%NPROW,root%NPCOL,A(IAPOS),LOCAL_M,
&LOCAL_N,root%TOT_ROOT_SIZE,MYID,DKEEP(6),KEEP(259),
&LDLT)
ENDIF
IF(KEEP(252).NE.0)THEN
FWD_LOCAL_N_RHS=numroc(KEEP(253),root%NBLOCK,
&root%MYCOL,0,root%NPCOL)
FWD_LOCAL_N_RHS=max(1,FWD_LOCAL_N_RHS)
FWD_MTYPE=1
CALLDMUMPS_768(
&root%TOT_ROOT_SIZE,
&KEEP(253),
&FWD_MTYPE,
&A(IAPOS),
&root%DESCRIPTOR(1),
&LOCAL_M,LOCAL_N,FWD_LOCAL_N_RHS,
&root%IPIV(1),LPIV,
&root%RHS_ROOT(1,1),LDLT,
```

```
&root%MBLOCK,root%NBLOCK,
&root%CNTXT_BLACS,IERR)
ENDIF
RETURN
ENDSUBROUTINEDMUMPS_146
```