Parallel evolutionary optimization with grid computing

Ng, Hee Kiang

2006

Ng, H. K. (2006). Parallel evolutionary optimization with grid computing. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/35751>

<https://doi.org/10.32657/10356/35751>

Parallel Evolutionary Optimization with Grid Computing

Ng Hee Khiang

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Master of Engineering

2006

Table of Contents

Lists of FiguresIV

Lists of TablesVI

ABSTRACT VII

ABSTRACT VII

ACKNOWLEDGEMENT VIII

ACRONYMSIX

Introduction 11

1.1 Grid Computing 11

1.1.1 Economic Benefits of Resource Sharing 12

1.1.2 Service Oriented Grid Computing 13

1.2 Evolutionary Algorithms 14

1.2.1 Complex Engineering Optimization Designs Problems 14

1.3 Research Focus and Scope..... 15

1.4 Layout of the Thesis 17

2 Grid Computing18

2.1 Introduction..... 18

2.2 Enabling Factors 18

2.3 Grid Infrastructure 19

2.4 Globus..... 20

2.4.1 Grid Security Infrastructure (GSI) 20

2.4.2 Grid Resource Allocation and Management (GRAM) 21

2.4.3 Monitor and Discovery Service (MDS)..... 22

2.4.4 Globus Access to Secondary Storage (GASS)..... 23

2.4.5	GridFTP	24
2.5	Grid Middleware.....	26
2.5.1	GridRPC.....	26
2.5.2	NetSolve.....	27
3	Evolutionary Algorithms	29
3.1	Introduction.....	29
3.2	Genetic Algorithms.....	29
3.3	Parallel Genetic Algorithms.....	31
3.4	Parallel Evolutionary Optimization of Computationally Expensive Engineering Design Problems	36
4	Grid Optimization Framework	37
4.1	Introduction.....	37
4.2	Grid Based System Architecture.....	39
4.2.1	Client Layer	40
4.2.2	Grid Resource layer	40
4.3	Single Cluster Based System Architecture	42
4.4	Integration of the NetSolve service onto the Grid Computing Platform	44
4.5	Workflow of the Grid Enabled PHGA	50
4.6	Conclusion	53
5	Empirical Study	54
5.1	Communication Overhead	55
5.2	Benchmark Test Function.....	61
5.3	Real World Problem: Aerodynamic Airfoil Design	64
6	Concluding Remarks	78
6.1	Research Contributions.....	78

6.1.1	Grid Optimization Framework.....	79
6.1.2	Grid Enabling GA using Bundling of jobs	79
6.1.3	Grid Enabling GA using Resized Bundling of jobs based on Grid Resources MFLOPs	80
6.1.4	Grid Enabling GA using Resized Bundling of jobs based on Grid Resources MFLOPs and number of CPUs in the cluster	81
6.2	Future Work	81
6.2.1	HPC LINPACK benchmark.....	82
6.2.2	Nested Layer of Grid Resources	82
6.2.3	Web Services Framework	83
Bibliography		i
<u>Appendix A : Publications.....</u>		viii
<u>Appendix B : Adjoint methods for aerodynamic sensitivity analysis</u>		ix

Lists of Figures

Figure 1: Grid Computing Resource Sharing	12
Figure 2 : Grid computing key Components	20
Figure 3 : Remote Resource Access – GRAM	21
Figure 4 : MDS Components Overview	22
Figure 5 : GRAM and GASS Flow.....	23
Figure 6 : GridFTP Files Transfer Types.....	25
Figure 7 : Overview of the system architecture with NetSolve.....	28
Figure 8 : The Schematic Workflow of a Canonical GA.....	30
Figure 9 : Flow chart for each subpopulation of multi-island model	34
Figure 10 : Two-island HPGA Model.	35
Figure 11 : Architecture of Grid Optimization Framework.....	39
Figure 12 : Meta Scheduling Overview	40
Figure 13 : GOF Submission Model Vs Typical Submission Model	42
Figure 14 : NetSolve-enabled ‘airfoil analysis’ architecture	43
Figure 15 : C wrapper to airfoil FORTRAN code	44
Figure 16 : Grid-enabled PHGA Infrastructure	45
Figure 17 : Grid-enabled PGA.....	46
Figure 18 : Overview of the Client application logic	47
Figure 19 : Workflow of Grid-Enabled PHGA	52
Figure 20 : Communication Overheads of GOF	57
Figure 21 : Comparison of approximate execution time incurred for (a) n subpopulations in 1 cluster, (b) n subpopulations in n clusters.	60
Figure 22 : Execution time of GOF on Rastrigin problem for hierarchical serial GA, hierarchical parallel GA.....	62

Figure 23 :Execution time of GOF on Rastrigin problem for parallel hierarchical GA....63

Figure 24 :Wall-clock time for geographically distant clusters with two subpopulations
deployed.....64

Figure 25 : Simplified aerodynamic airfoil design problem.....65

Figure 26 : AeroFoil Parameterization using 12 Hicks-hennes Functions66

Figure 27: One of the Airfoils Represented by the Hicks-Hennes Functions.....67

Figure 28 :Airfoil geometry characterized using 24 design variables with the NACA 0012
as baseline.68

Figure 29 : Execution time for hierarchical parallel GA with two subpopulations and two
heterogeneous clusters of differing specifications for low-fidelity airfoil design
optimization72

Figure 30 : Execution time for hierarchical parallel GA with two subpopulations and two
heterogeneous clusters of differing specifications for high-fidelity airfoil
design optimization.....73

Figure 31:Execution time for Moderate-fidelity analysis code for 2 subpopulations
deployment with/without CPUs resized bundling on hpc-pdpm and pdcc.....76

Figure 32 : Execution time for Moderate-fidelity analysis code for 2 subpopulations
deployment with/without CPUs resized bundling on surya and pdcc77

Lists of Tables

Table 1: List of computing resources employed for the experiments.....55

Table 2 : Comparison of execution time for different x and y setting and different
subpopulation size.....60

Table 3 : Average wall-clock time to transfer a subpopulation of chromosome62

Table 4 : Specifications and MFLOPS of the three clusters.....69

Table 5 : Execution time for Rastrigin test function on the various clusters.....69

Table 6 : Execution time of variable-fidelity airfoil problem on clusters with differing
specifications70

Table 7 : Resized bundling of chromosomes to heterogeneous cluster.....74

Table 8 : Resized bundling of chromosomes to heterogeneous cluster with CPUs
considerations.75

ABSTRACT

A rising trend in science and engineering is in the utilization of increasingly high-fidelity and accurate analysis codes in the design analysis and optimization process. In many application areas such as photonics, electromagnetics, aerospace, biomedical, micro-electro-mechanical systems and coupled-field multidisciplinary system design processes, simulation procedures involving Computational Structural Mechanics (CSM), Computational Fluid Dynamics (CFD) or Computational Electronics & Electromagnetics (CEE) are an important step in the design process. The time taken for these processes generally varies from many minutes to hours or days of supercomputing time. This often leads to high computing costs in the design optimization process, hence a much longer design cycle time to locate a near optimum design solution. This thesis presents a Grid-enabled scalable parallel evolutionary framework for solving computationally expensive optimization problems under limited time budget. The essential backbone of the approach leverages the de facto standards of Grid computing to facilitate secure access to heterogeneous processing nodes and software resources that may be dispersed across many geographical locations in the evolutionary optimization search. By exploiting the Grid evolution framework and a multi-level parallelization strategy of hierarchical parallel GAs, we present the evolutionary optimization of a realistic 2D aerodynamic airfoil structure in a Grid environment having multiple heterogeneous clusters, *i.e.*, clusters with differing specifications and processing nodes. From our results obtained, it is possible to conclude that the Grid enabled hierarchical parallel evolutionary algorithm is not a mere hype but offers a credible alternative, providing significant time reduction to complex engineering design optimization.

ACKNOWLEDGEMENT

Throughout the project, the author has received plenty of support from a number of different individuals. Hence the author would like to express his gratitude in this thesis.

First and foremost, the author would like to express his sincere gratitude and appreciation to **Asst. Prof. Ong Yew Soon**, for being an enlightening supervisor, and above all, a friendly and helpful mentor.

Secondly, the author would like to express his sincere appreciation to Dudy Lim, for his contribution in the formulation of the dissertation.

Lastly, the author would like to acknowledge the help received from **all the staff of Parallel and Distributed Computing Centre (PDCC)**, for their technical support and providing the computing resources necessary to make this project possible. The contribution from the centre, which ranges from equipping him with the necessary manpower as well as the computing power and resources, has helped in the successful completion of the work.

Thirdly, the author would like to thank Honda Research Institute Europe, Germany for sponsoring this research work. Last but not least, the author would like to express his heartfelt thanks to friends and associates who have contributed much spiritual support.

ACRONYMS

The commonly used acronyms in the thesis are listed below.

CA	Certificate Authority
CoG	Commodity Grid Kits
EA	Evolutionary Algorithm
HPC	High Performance Computing
HPGA	Hierarchical Parallel Genetic Algorithm
GA	Genetic Algorithm
GASS	Globus Access to Secondary Storage
GEODISE	Grid Enabled Optimization and Design Search for Engineering
GGF	Global Grid Forum
GIIS	Grid Information Index Service
GOF	Grid Optimization Framework
GRIS	Grid Resource Information Service
GridRPC	Grid Remote Procedure Call
GSI	Grid Security Infrastructure
HTTP	Hyper Text Transfer Protocol
LDAP	Light Directory Access Protocol
MDS	Monitoring and Discovery Service
MPI	Message Passing Interface
PBS	Portable Batch System
PDF	Problem Description File
PGA	Parallel Genetic Algorithm
PHGA	Parallel Hybrid Genetic Algorithm

PVM	Parallel Virtual Machine
SGE	Sun Grid Engine
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
TLS	Transport Layer Security
VO	Virtual Organization
WSRF	WS-Resources Framework

Chapter 1

Introduction

1.1 Grid Computing

Grid [1-3] Computing refers to the coordinated and secured sharing of computing resources among dynamic collections of individuals, institutions and resources. This terminology is chosen to connote the idea of a ‘power Grid’ where users can plug into the computing infrastructure like plugging into a power Grid. Hence, the term ‘Grid’ is sometimes used synonymously with a networked, high performance computing infrastructure. The current research and investment into computational Grids is motivated by the assumption that coordinated access to diverse and geographically distributed resources can lead to high return of investments (ROI). The ‘Grid’ concept is motivated by the process of coordinating resource sharing and problem solving in a dynamic environment. The mode of sharing is highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of organizations and/or institutions defined by such sharing rules form what is commonly called a virtual organization (VO) [4].

Grid Computing has gained widespread popularity in the research community for distributed and parallel computing because of its tremendous potential in enabling complex applications, especially those requiring huge amounts of computational power and data storage requirements. A motivating example for us is complex design in science

and engineering. The rising trend in the science and engineering world is the utilization of increasingly high-fidelity and accurate analysis codes in the design analysis and optimization process. In this research work, our objective is to engage a collaborative Grid environment where disparate computing resources can be pooled together so that computationally expensive design optimization problems can harness the aggregation of computing resources to improve the search performances. This scenario where organizations share their computing resources to create a pool of computing is depicted by the diagram in Figure 1.

Figure 1 : Grid Computing Resource Sharing

1.1.1 Economic Benefits of Resource Sharing

Taking full ownership of high performance computing (HPC) systems such as supercomputers for tackling computationally expensive problems may be extremely costly for most organizations to take up. Intra-organizational activities capitalizing on

HPC would also typically experience high and low resource utilization periodic trends determined by product delivery deadlines. There are times when these dedicated HPC systems are inadequately utilized. As such, progress in commercial and research activities involving the use of specialized, sophisticated and expensive computing systems are greatly impeded by the opportunity costs of operation.

The introduction of Grid computing serves to provide possible economic benefits in resource sharing as collaboration lies at the heart of Grid computing. In addition, it provides the necessary infrastructure and tools to encourage seamless collaborations between international universities, research institutions and the industries.

1.1.2 Service Oriented Grid Computing

Service-oriented computing (SOC), an emerging paradigm for distributed computing, represents a major shift in the way that software systems are designed, developed, and consumed. Services are platform and language independent software elements that can be described, published, orchestrated using XML or other feasible methods. This concept of service-oriented computing is seeing wide acceptance and becoming a standard approach for tackling distributed, heterogeneous computing problems. Web services/Grid services are fast evolving, and the proposed WSRF (WS-Resources Framework) [5] is seen as a joint effort between industry and academia. It is a sign of the convergence of Web services and Grid Services. The principle motivation for adopting Web services is interoperability, where a service consumer can access Web services using standard protocols such as SOAP (Simple Object Access Protocol) [6], HTTP, irrespective of the operating systems and programming languages that either parties use. Therefore, it increases the accessibility of existing capabilities by presenting existing codes as Web services and enables collaborations across geographical and institutional boundaries. For

example, a Grid application may be initiated via a Web service call irrespective of its geographic location.

1.2 Evolutionary Algorithms

Evolutionary algorithms [7], unlike conventional gradient-based numerical optimization methods produce new design points that do not use information about the local slope of the search function and are thus not prone to stalling at local optima. They have shown considerable success in locating the global optimum solution of many optimization problems characterized by non-convex and disjoint solution spaces. However, this increase in capacity to locate the global optimum design is often achieved at the expense a longer design cycle time.

In recent years, Evolutionary Algorithms (EAs) have been successfully applied to many complex engineering design optimization problems. However, high computational costs associated with the use of high-fidelity simulation models pose a serious impediment to the successful application of Evolutionary Algorithms (EAs) to complex design in science and engineering. This is primarily because a single function evaluation (involving the analysis of a complex engineering system) often consumes many minutes to hours of computer time and EAs typically require thousands of fitness function evaluations to locate a near optimal solution.

1.2.1 Complex Engineering Optimization Designs Problems

Complex engineering optimization problems commonly have very large design spaces. In such design spaces, typically many thousands of function evaluations are required to locate the near optimal value that minimize or maximize the value of the objective function under given constraints. Often in aerospace design processes, variable-fidelity analysis codes are employed to strike a balance between design cost, time and estimation

accuracy. Nevertheless, in design optimization process where high-fidelity analysis codes are used, each function evaluation may require a finite element or computational fluid dynamics simulation costing hours of supercomputer time. Thus, the overwhelming part of the total run time in complex engineering design optimization is usually taken up by the evaluations of the analysis code. In this thesis, we considered the Rastrigin benchmark test function and a realistic aerodynamic airfoil shape design optimization problem in our study.

Benchmark Test Function – Rastrigin

The Rastrigin problem is one of the commonly used benchmark test function for studying the convergence properties of evolutionary algorithm. The Rastrigin problem is a highly multimodal function with many local minima and a global minimum located at $(0, \dots, 0)$. There are many other local minima surrounding the global minimum. It has a complexity of $O(n \ln(n))$, where n is the number of the function parameters. [8]

Aerodynamic Shape Design of Airfoils

Aerodynamic design represents one of the most frequently tackled computationally expensive and intensive complex engineering design problem in aeronautics. Due to the large design spaces, usually stochastic optimization algorithms such as Evolutionary Algorithms (EAs) are employed in the aerodynamic search in order to arrive at a near optimum design.

1.3 Research Focus and Scope

Recent advances in technologies has sparked the research and development of Grid Computing as a form of parallelism in distributed computing. Although some form of distributed computing technologies have been exploited for quite a while, they are not

amenable to the scale of integration and complexity that are concerned in complex engineering design. The issue of standards is one major challenge, as many existing technologies do not have the common interfaces and methods of doing things. As such, they remain disparate and difficult to integrate. It is for these reasons that Grid computing has gained widespread attention, as it sets about the motion of establishing a set of open standards for distributed resources and ubiquitous services, for representation, discovery and control. Such standard initiatives pave the way for new technologies to be amalgamated; delivering more powerful and more complex services, which has led to the intensity of research in Grid, particularly Grid-related applications worldwide.

Finally, as Grid computing move into the mainstream of parallel computing, the challenge to researchers is to identify and demonstrate computationally expensive engineering optimization problems that can fully exploit the potential of this unlimited pool of Grid computing resources. The model of optimization parallelism pursued in this thesis has the great potential for scaling to take advantage of unlimited Grid computing resources.

In this thesis, we present a Grid optimization framework (GOF) for solving computationally expensive optimization problems under limited time budget. For the sake of brevity, we refer the Grid optimization framework as GOF in short. In particular, we consider a Hierarchical Parallel Genetic Algorithm deployed on the proposed Grid optimization framework with geographically disparate heterogeneous clusters containing different processing node sizes. We consider a multi-population coarse-grained GA at the first level where subpopulations are distributed across the available clusters on the Grid environment. The individuals of each subpopulation are farmed onto the processing nodes using a master-slave model. Experimental study are conducted to analyze the effectiveness of the proposed GOF using a benchmark test function and a realistic airfoil

shape design problem. In particular, we investigated two GOF frameworks, i.e., parallel executions on both single and multiple clusters. The effect of cluster heterogeneity on the performances of GOF for evolutionary design optimization is also studied using a the realistic airfoil aerodynamic design problem.

1.4 Layout of the Thesis

This thesis is organized as follows:

Chapter 2 introduces some Grid computing technologies necessary for composing an environment that facilitates seamless consumption of resources available on the Grid. Technologies useful for making new and existing applications join the Grid is also discussed in the chapter.

Chapter 3 gives a brief overview on Evolutionary Algorithms (EA). In particular we focus on Genetic Algorithm (GA) and its parallel implementations.

Chapter 4 describes the proposed Grid optimization framework for parallel evolutionary optimization of complex engineering design problems having computationally expensive analysis codes.

Chapter 5 presents the results of experimental studies conducted on the Grid optimization framework introduced in chapter 4 using the Rastrigin benchmark test problem and an aerodynamic airfoil shape optimization problem.

Chapter 6 concludes this thesis by summarizing the core contributions made in the present program of research, and outlines some potential directions for future research.

Chapter 2

Grid Computing

In this chapter, a brief description on the core technologies used for creating a Grid computing environment is presented. In particular, the roles of various Grids computing technologies considered in this dissertation are outlined. We hope it would assist the reader to better comprehend the Grid Optimization Framework (GOF) reported in the subsequent chapters of the dissertation.

2.1 Introduction

Grid computing may be considered as distributed computing taken to the next evolutionary level and had been coined as tomorrow's supercomputer. A Grid may be defined as both a hardware and software infrastructure for providing flexible, secure, and coordinated resource sharing among dynamic collection of virtual organizations. The World Wide Web has provided an infrastructure on how information can be shared globally. The objective of the Grid is to extend this functionality for ubiquitous computation and storage functionality. Just as the Web changes the way we communicate with each other, the Grid will change the way we access to the computation, storage resources as well as the way we conduct computations. [9]

2.2 Enabling Factors

Advancement in communication devices such as the satellites and the Internet enable remote access to regional and international databases and sensor networks. Collaborative

infrastructures like the Access Grid [10] have moved exchange of information beyond the desktop. All these advances have and will profoundly influence the way people work with each other. It is worth noting that computing power has steadily increased over the years, as shown by Moore's Law [11], and this progress is expected to continue for the next decade. In addition, the improvements in network bandwidth and decreasing hardware and network cost over the years have promoted the migration trend from a centralized computing center to one that is geographically dispersed and possibly heterogeneous computing resources. This leads into the shaping of what we now term as the Grid approach or concept.

2.3 Grid Infrastructure

Security, resource management, information services, and data management form some of the key components of Grid infrastructure, see Figure 2.

Each Grid resource may be equipped with different levels of security policy. Single sign-on authentication methods and negotiating authorizations are also necessary to facilitate secure execution over the Grid. After a job is submitted, the Grid resource manager proceeds to assign suitable resources to the job, monitors the status of job execution, and then marshals the results back to the correct client program.

For the Grid resource manager to make informed decisions on resource assignments, it is necessary for the Grid resource manager to maintain updated knowledge about all available Grid resources, their specifications, capacities and online workload. This knowledge about the Grid resources is maintained and provided using the Grid Information Service (GIS) [12], sometimes also known as the Monitoring and Discovery Service (MDS) [13]. Data management on the other hand, is concerned with how data and shared storage are transferred and accessed, in the Grid environment respectively.

Figure 2 : Grid computing key Components

2.4 Globus

Globus [14,15] is the *de facto* standard for Grid computing infrastructure over the recent years. The Globus toolkits is an open-source, open architecture Grid middleware (collections of APIs, protocols, software) that provides the necessary implementations to facilitate security, resource management, information services, and data management in the Grid environment.

2.4.1 Grid Security Infrastructure (GSI)

The Globus GSI [16,17] provides functionalities for possible secure Grid computing. GSI defines a credential format based on X.509 identity certificates [18] and a protocol based on transport layer security (TLS) [19]. An X.509 certificate is issued to a Grid entity, such as a user, and together with the user's private key of the certificate, creates a credential that other Grid entities, such as the resources, authenticate the client for usage. The TLS-based protocol is used to perform authentication and provide message protection (encryption, integrity, checking) on data streams. GSI certificates are issued by trusted parties known as the certificate authority (CA). All the Grid entities within the Grid must trust the CA that issued the certificate. In addition, the use of credential management

system such as the MyProxy [20] reduces the hassle for user to deal with the GSI credential issues.

2.4.2 Grid Resource Allocation and Management (GRAM)

GRAM [21] is the fundamental Globus service that serves to enable instantiations, management and Grid jobs monitoring of remote resources. To invoke a job using GRAM, a client specifies the job to be executed, i.e., by specifying details such as program name, working directory, input and output locations, and the queue in which the program should execute in. To begin, the specification file is sent to the remote resources to create and instantiate new instances of the job on the remote resources. Subsequently control and monitoring of the job through Globus service mechanisms may also be attained. The architecture [22] of GRAM is depicted in Figure 3:

Figure 3 : Remote Resource Access – GRAM

2.4.3 Monitor and Discovery Service (MDS)

Resource information on the Grid environment is obtained from the MDS. MDS provides the mechanisms for publishing and discovering the resource status and configuration information of the Grid resources. It has a hierarchical structure consisting of Grid Information Index Service (GIIS) and Grid Resource Information Service (GRIS) as information providers. The information providers collect information on the Grid resources and send these online data to the GRIS. GRIS communicates using LDAP [23] and provides information about the resources and registers them to the GIIS. Hence the GIIS provides the functions for collective-level indexing/searching. Figure 4 presents a conceptual view of the MDS components.

Figure 4 : MDS Components Overview

2.4.4 Globus Access to Secondary Storage (GASS)

GASS [24] is a remote file management mechanism for Globus jobs. The idea behind this is to provide a mechanism to allow Globus jobs to stage files to or from a job (i.e. input and output files). It was however not optimized for transferring large files, hence the GridFTP was introduced in [25]. The flow integration of the GRAM and GASS is illustrated in Figure 5. When a user submits a job, the GASS server is initiated, and the request is remotely executed at the resource site. Once the job execution ceases, the results are sent back to the GASS server as standard out or error.

Figure 5 : GRAM and GASS Flow

2.4.5 GridFTP

GridFTP is the common data transfer and access protocol which provide secure, efficient data movement over the Grid. The client uses GridFTP to move input and output files between the resources. The GridFTP protocol is based on FTP, the highly-popular Internet file transfer protocol. It provides the GridFTP server and client, which are implemented using the in.ftpd daemon and globus-url-copy command (and related APIs), respectively. It supports the standard file transfer between server and client, as well as third party file transfer as illustrated in Figure 6. Figure 6a, which depicts a standard file transfer, show the file being transferred from the client to the server machine.

(a) Standard File Transfer

Alternatively, a client can also initiate a file transfer request between GridFTP server 1 and GridFTP server 2. Figure 6b illustrates a third party file transfer event.

(b) Third Party File Transfer

Figure 6 : GridFTP Files Transfer Types

2.5 Grid Middleware

The collection of APIs, protocols, and software that allow creation and use of a distributed system, such as a Grid, is known as middleware. Over the recent years, a variety of middleware packages have been developed for Grid computing, two of which are presented here in this section.

2.5.1 GridRPC

GridRPC [26] represents a remote procedure call (RPC) mechanism in Grid computing. The GridRPC API is an ongoing initiative for standardizing and implementing a portable and simple remote procedure call (RPC) mechanism for Grid computing and is pursued by the Global Grid Forum Research Group. The GridRPC API is designed to address one of the factors that has hindered widespread acceptance of Grid computing – the lack of a standardized, portable and simple programming interface. GridRPC provide high-level, easy-to-use task-parallel programming abstraction for the Grid. It hides the complexities of Grid security, resource discovery, and scheduling from the user. Furthermore, it provides capabilities on the parallel tasking as well as synchronization of large computational tasks.

2.5.2 NetSolve

NetSolve [27-30] is a parallel/distributed computing platform, which provides suitable functionalities and features for easy and rapid system integration. It is designed to permit usage of network resources for numerical computing by providing convenient access to problem functions in addition to managing computational resources. The basic objective is to provide a uniform, portable and efficient means to access computational resources over a cluster network. It provides remote access to computational resources, both hardware and software registered in the system and serves as a middleware with easy to use general purpose scientific computing environment (e.g. Matlab and Mathematica), which are essential tools for the majority of computer scientists and engineers.

In essence, NetSolve is a middleware, developed by University of Tennessee. It consists of two major components, namely the Agent and the Server. The Agent plays the role of resources discovery and allocation, whilst at the same time maintaining the load balance and fault tolerance. The Server is the number of clusters, nodes or CPUs available in the system. These two components are then linked to the Client. This is where the users will send out the problems to be analyzed. This can be in the form of codes or analyses files from software such as ABAQUS, FLUENT, SYSNOISE, *etc.* These codes or files are then sent out to the Servers via the Agent, who controls the direction of the files to be sent out in sequence or in parallel. In summary, it serves as a middleware, with the capability of employing general purpose scientific computing environments such as Matlab and Mathematica, which are tools commonly used by the majority of computer scientists and engineers. Its ability to be pre-programmed in a way that best utilizes the available computing resources is another benefit and justification of its employment.

The flexible framework of NetSolve makes it a useful tool to support the future implementation of a Grid-based architecture described in the next chapter. It facilitates

the implementation of Grid based architecture efficiently and conveniently by extension of the single cluster deployment. Parallelism is achieved by first deploying a software application as a NetSolve service. Hence, using the farming client application programming interface, the application routine can be readily executed in parallel on all the remote servers that exists on the Grid. An overview of this arrangement is shown in Figure 7.

Figure 7 : Overview of the system architecture with NetSolve

Chapter 3

Evolutionary Algorithms

3.1 Introduction

In many real-world problems, Evolutionary Algorithms (EAs) have been applied with a great degree of success [31-34]. Their popularity in complex design lies in their ease of implementation and as a vital tool for locating globally optimum and creative designs. Evolutionary Algorithms (EAs) are modern stochastic search techniques inspired by the Neo-Darwinian theory of natural selection and evolution . There are numerous variants of EAs, including evolutionary strategies (ES), evolutionary programming (EP), genetic algorithms (GA), genetic programming (GP) and many others. In this thesis, GA is used and will be discussed.

3.2 Genetic Algorithms

The term Genetic Algorithms or GA was first introduced by Holland in his seminal book referring to the probabilistic search techniques inspired by the ‘survival of the fittest’ principle of the Darwin theory of natural selection. Artificial creatures are created, which will then compete in a struggle for life [35]. The fittest will then survive and are then allowed to reproduce. New creatures will then be created again using some operators such as crossover and mutation. The schematic workflow of this repetitive process is shown in Figure 8. First, an initial population of binary coded chromosomes representing random candidate solutions is formed. An objective function that tests the fitness of a candidate

solution is used to rate the chromosome so that fittest solutions have greater chances of surviving to the next generations. Those that survive then undergo standard genetic operators such as mutation and crossover to form a new population of child chromosomes that inherits from their parents. The evolution cycle repeats until some stopping criteria are satisfied.

```

Procedure GA
Begin
  for  $i:=1$  to  $n$  do begin
     $initialise(x_i)$            // with random values
     $f(x_i):=evaluate(x_i)$     // determine fitness
  end
  while termination condition not satisfied do begin
     $p_{1...n}:=$  select from  $x_{1...n}$  in random order
    for  $i:=1$  to  $n$  step 2 do begin
       $x'_i, x'_{i+1}:=crossover(p_i, p_{i+1})$ 
       $mutate(x'_i)$ 
       $mutate(x'_{i+1})$ 
    end
     $x_{1...n}:=x'_{1...n}$            // copy new population
    for  $i:=1$  to  $n$  do begin
       $f(x_i):=evaluate(x_i)$ 
    end
  end {while}
end

```

Figure 8 : The Schematic Workflow of a Canonical GA

Ever since its introduction, many variations of the standard GA have appeared. Steady State GA, Incremental GA and Parallel GA are some of the more prominent ones [36]. The GA employed in this research work is the standard GA or otherwise known as the Canonical GA.

3.3 Parallel Genetic Algorithms

GAs have often been shown to converge to good quality solutions under tractable computational budget. However, since the canonical GA commonly takes many thousands of calls to the objective function before possible convergence to near global optimum solutions, it may be computationally prohibitive if the optimization problem is computationally very expensive. Hence, various efforts have focused on making GAs faster. One of the most promising methods is to use the parallel variations of GAs, considered to be a desirable feature of any framework for optimization of computationally expensive problems.

A well-known strength of EAs is the ease of extensions to incorporate parallelism [37]. For instance, Parallel Genetic Algorithms (PGAs) [38] are extensions of the standard Genetic Algorithm. Instead of evaluating a single individual sequentially, the basic concept of PGA is a simple division of the tasks in a standard GA across many processors. The other advantage of PGA, such as those working with multiple populations, is facilitating speciation. It is a process by which different populations evolve in different directions simultaneously. They have been shown to speed up the search process as well as to obtain higher quality solutions when dealing with complex optimization problems.

In general, PGAs may be categorized as global single-population master-slave GAs, single-population fine-grained GAs, or multiple-population coarse-grained GAs.

Master-slave PGA

In master-slave PGAs, it is assumed that there is only a single panmictic population, i.e., a simple GA. Like the simple GA, each individual compete and reproduce with any other in the master-slave PGAs. However, unlike the simple GA, evaluation of individuals is

distributed by scheduling fractions of the population among the processing slave nodes. In addition, master-slave PGAs use parallel computing to speed up the operation of the simple GA without changing the basic operations of the sequential GA. Such a model has the advantage of ease of implementations and does not alter the search behavior of the canonical GA, i.e., existing theory of simple GA still applies. Further, it poses as an efficient method of parallelization when evaluation of the fitness functions is computationally expensive. A motivating example for us is aerodynamic wing design, where one function evaluation involving the solution of the Navier–Stokes equations can take many hours of computer time.

Fine-grained PGA

Fine-grained parallel GAs [39] consist of only a single population, which is spatially structured. They are designed to run on closely-linked massively parallel processing systems, i.e., a machine consisting of a large number of processing elements and connected in a specific high-speed topology. For instance, the population of individuals in a fine-grained PGA may be organized as a 2-Dimensional Grid, since many massively parallel computers have processing elements that are connected using this topology. Consequently, selection and mating in fine-grained parallel GAs are restricted to a small neighborhood. Nevertheless, neighborhoods overlap to permit some interactions among all the individuals so that good solutions may disseminate across the entire populations. Sometimes, fine-grained parallel GAs are also termed as cellular models. [40]

Multi-population PGA

Multiple-population (or multiple-deme) GAs are more sophisticated, as they consist of several subpopulations which exchange individuals occasionally. This exchange of

individuals is called migration and is controlled by several parameters. Hence, the important characteristics of multiple-deme parallel GAs are in the use of multiple subpopulations and migration. Multiple-deme parallel GAs are known by different names. Sometimes they are known as “distributed” GAs, because they are usually implemented in distributed-memory MIMD (Multiple Instruction/Multiple Data) computers. Since the computation to communication ratio is usually high, they are occasionally known as coarse-grained GAs. Besides, since multiple-deme parallel GAs resemble the “island model” in population genetics that considers relatively isolated demes, it is often also known as “island” parallel GAs.

In multiple-deme parallel GAs, the arrival of individuals from other populations can trigger evolutionary changes. It is also noticed that there is relatively little change between migrations, but new solutions are found shortly after individuals are exchanged. Multiple-deme GAs, although popular, is also a class of parallel GAs that is probably the most difficult to understand. This is because the effects of migration are not fully understood. Migration is usually controlled by several parameters [41]. The following lists these control parameters:

- Migration rate determines how many individuals migrate from a population;
- Migration frequency (migration interval) determines how often migration occurs;
- Migration topology determines the destination of the migrants;
- Migration policy determines which individuals migrate and which are replaced at the receiving deme.

The island model strategy eliminates the global synchronization required in the panmictic approach. However, in the majority of multi-deme parallel GAs, certain level of synchronization is usually required. For this reason, migration should occur at predetermined constant intervals. The algorithm flow chart for each subpopulation is illustrated in Figure 9.

Figure 9 : Flow chart for each subpopulation of multi-island model

Hierarchical PGA

The different parallel GA models may also be hybridized to produce new hierarchical PGA models. One may form a hierarchical PGA that combines a multi-deme GA (at the upper level) and a fine-grained GA or master-slave GA (at the lower level).

On the other hand, multi-deme parallel GAs may also be designed with multiple levels in a manner that the migration rate is faster at the lower level and possessing a communications topology much denser than the upper level. Generally, hierarchical PGAs are considered to be more effective in generating significant wall-clock time saving than any of the standalone PGA models. Figure 10 illustrates specifically a two-island model. It shows the specific independent sequential GA operations running on each subpopulation as well as the periodic migration between the two subpopulations. The migration characteristic is controlled by several migration parameters.

Figure 10 : Two-island HPGA Model.

In spite of extensive studies on parallel EAs, existing frameworks for handling computationally expensive problems have met with limited success in applications to real-world complex problems in science and engineering. Some of the key factors responsible for this are as follows:

1. The inability to acquire sufficient processing elements to unleash the full potential of PGAs in providing optimum design solutions efficiently. This fundamental

difficulty arises from the reality that often limited computational resources are available to the design team.

2. In addition, the PGA is often constrained by the limited licenses of commercial analysis codes available to each design team for the parallel evaluations of chromosomes. In [42] for instance, given that only eight site licenses of the VSAERO CFD analysis code are available for use in the Computational Engineering and Design Center, University of Southampton, maximum parallelism in the evolutionary search is limited to eight each time. This is primarily due to the high costs associated with the site licenses of these commonly employed commercial packages, which also include, ANSYS, ABACUS, FLUENT, SYSNOISE and DYTRAN.

An intuitive way to mitigate the problem is to pool together the heterogeneous clusters and software licenses belonging to the many design teams that may be located at dispersed geographical locations.

3.4 Parallel Evolutionary Optimization of Computationally Expensive Engineering Design Problems

In complex engineering design problems, the standard evolutionary operations of general PGAs (mutation, crossover, mating, selection, etc.) consume significantly low computational efforts. Often it is the fitness calculation of the chromosomes or potential designs that accounts for most of the computational efforts, considering that high-fidelity analysis codes are commonly used in complex design in science and engineering. Hence, parallelism offers an excellent solution to mitigate the problem of high computational expense in design evolutionary optimization process.

Chapter 4

Grid Optimization Framework

4.1 Introduction

Today, computing is becoming an essential technology for complex design in science and engineering. In many application areas such as photonics, electromagnetics, aerospace, biomedical, micro-electro-mechanical systems and coupled-field multidisciplinary system design processes, the design process generally requires a Computational Structural Mechanics (CSM), a Computational Fluid Dynamics (CFD) or a Computational Electronics & Electromagnetics (CEE) simulation procedure. Here, a single analysis of the design involving CFD, CSM or CEE could take up many minutes to hours or even days of supercomputing time [43, 44]. This often leads to high computing costs in the design optimization process, hence a much longer design cycle time to locate near optimum design solution.

Here, the benefits of Grid computing in the context of evolutionary computation, especially on computationally expensive design optimization problem are numerous, in particular, the ability to tap on vast computing power. The adoption of Grid computing in the design process would help facilitate collaborations among remote teams. Hence, specialized high-fidelity analysis codes, commercial libraries, processing nodes, data and information owned by different design teams that span across geographically distributed locations may be shared and better utilized. Multiple components administered and designed by different teams may also be assembled via remote collaborations.

Presently, solving large-scale design optimization problems is generally regarded as impractical since this often requires a huge amount of computational power that are extremely costly for any single organization to take full ownership of. Hence, the use of Grid computing presents a viable and cost effective option to large-scale design optimization problems. Related works on optimization using Grid computing can be found in [45-51]. The Grid Enabled Optimization and Design Search for Engineering (GEODISE) of the e-science group in UK that represents one of the recent initiatives for engineering design search and optimization is most similar to our present work. However, in contrast to GEODISE which only supports only system calls via Globus API, we propose the use of high-level interfaces like GridRPC and Grid services to access the Grid resources. Further, GEODISE may not be designed for multi-clusters in a distributed Grid environment. Besides, like most existing systems, it lacks suitable meta-scheduling mechanism for scheduling jobs to multiple remote clusters of compute nodes. Instead GEODISE requires users to perform manual resource selection. This is clearly impractical in a Grid environment since all form of resources joins and leaves the environment constantly.

4.2 Grid Based System Architecture

The architecture of the GOF is illustrated in Figure 11 and is made up of several layers. The client layer of the GOF represents the optimization module that will access Grid resources. In this dissertation, a Parallel Hierarchical Genetic Algorithm is considered. The Grid application layer of the GOF is made up of the Globus toolkits, Middleware and other grid computing technologies discussed previously in Chapter 3. Last but not least, the Grid resource layer contains the pool of resource that exist on the various Grid environment. In the context of design optimization, this represent the grid enabled analysis codes or evaluation functions.

Figure 11 : Architecture of Grid Optimization Framework

4.2.1 Client Layer

Matching of resources to requests and efficient scheduling of these requests forms a critical part of any grid application. In the present work, we incorporate suitable meta-scheduling mechanism into the existing GridRPC API. In particular, each client has its own metascheduler mechanism [52] by employing the extended GridRPC API. Upon initialization, the client program retrieves a list of resources available on the Grid environment and their status using the extended GridRPC API. The metascheduler locates for suitable grid service and then submits the requests to the appropriate computing cluster/nodes that provides the requested service. This is illustrated in Figure 12.

Figure 12 : Meta Scheduling Overview

4.2.2 Grid Resource layer

Here we focus on the Grid Resource layer of the GOF that is illustrated in Figure 13. It contains the local scheduler for distribution of jobs or requests onto the Grid computing

cluster. Our GOF differs from the traditional way of executing the parallel jobs on a computational Grid, such that the jobs are bundled before submission. Figure 13a illustrates the common way of dispatching task onto each grid resource. In Figure 13b, jobs are sent in bundle to the clusters, where they are executed in parallel. Thus, it reduces the overhead incurred in having to perform multiple job submission to the resources.

(a) GOF Submission Model

Figure 13 : GOF Submission Model Vs Typical Submission Model

4.3 Single Cluster Based System Architecture

In this dissertation, NetSolve is employed as the local scheduler. It serves to provide a uniform, portable and effective means of accessing computing resources over a cluster of network servers. This gives users the ability to access remote resources registered in the system. NetSolve employs a client-server architecture in the parallelism of tasks. The client program will contact the NetSolve agent, which will distribute the task to the appropriate servers registered to the NetSolve agent.

Here, the development of single-cluster NetSolve-enabled parallel GA for aerodynamic airfoil design is described. In this particular setup, a “airfoil analysis” NetSolve service is implemented. The “airfoil analysis” Netsolve service is deployed across all the nodes in the cluster and at the same time they are registered to the Netsolve Agent. Figure 14 shows the interaction between the NetSolve Agent and the ‘airfoil analysis’ services deployed on the cluster.

Figure 14 : NetSolve-enabled ‘airfoil analysis’ architecture

To deploy NetSolve Services on the nodes, a problem description file (PDF) is created to specify the NetSolve system, such as the interface and prototype for access by remote clients. The NetSolve system defines data structures known as NetSolve *objects*. A PDF then contains the NetSolve objects that define a problem specification for a given service. Each NetSolve object describes the *object type* and *data type*, which are high-level objects that are comprised of integers, and arrays of characters and floats. Take for example, the aerodynamic airfoil analysis code which was originally written in FORTRAN codes is wrapped using C programming language. The prototype is illustrated in Figure 15.

```

void airfoilana(float *mach, float *ago, float *alphIn, float *pdOut,
               float *gOut, float * rsdEu, float *rsdAdj, int ND)
{
    int i;
    pdgrad_(mach,aoa,alphIn,pdOut,gOut,rsdEu,rsdAdj);
}

```

Figure 15 : C wrapper to airfoil FORTRAN code

The source code in C language, as shown in Figure 15, provided the prototype access to the NetSolve enabled ‘airfoil analysis’ service code, when deployed on the cluster. The airfoil PDF file will contain a function, ‘AIRFOIL’, to solve the airfoil analysis problem according to the above prototype. The variables in the above prototype specifies the input and output argument to the airfoil analysis service. The NetSolve agent contains the list of services that is available in the cluster.

4.4 Integration of the NetSolve service onto the Grid

Computing Platform

Here, the design optimization problem is extended onto the Grid. From which, this will enable the problem to be executed across the Grid computing platform, unleashing and utilizing the full power of Grid computing technology. Figure 16 illustrates the infrastructure of the Grid-enabled PHGA considered.

Figure 16 : Grid-enabled PHGA Infrastructure

First, the reproduction operation of the adopted GA was Globus-enabled. This implies that the reproduction of each subpopulations evaluation is executed remotely on the Grid resources. For such an arrangement, the procedure begins with the Client GA main program farming the subpopulations to the various Grid resources, whose remote implementation is implicitly consuming the analysis service. This arrangement, thus, enables the GA subpopulations to be farmed out in parallel across the cluster nodes at the first level as shown in Figure 17. At the second level, the potential evaluation analyses in each subpopulation are subsequently farmed out in parallel across the nodes at the cluster level. This reduces the total computational cost in comparison to the conventional process based upon a sequential execution of each GA subpopulation requests.

Figure 17 : Grid-enabled PGA

1st Level: Farming of subpopulations

2nd Level: Farming of chromosomes or potential airfoil designs for analysis

From the perspective of the Client, it requires the users of the client application to input the values of the 3 variables. These variables are namely the number of subpopulation, the number of chromosomes (B) and the number of generations (C) associated with the adopted GA formulation. The entering of the 3 variables will then start off the solution process. Upon completion of this particular analysis, the results obtained will then be sent back to the Client for collation and assessment. This will then start off another execution run, for new combinations of (A,B,C) . This Client programming logic is illustrated in Figure 18.

Figure 18 : Overview of the Client application logic

Parallel EAs are well known for their effectiveness in convergence to near global optimum solutions within tractable time, especially in terms of diversity of the solution and the ability to escape from local optima as the result of the co-evolving subpopulations [53]. The use of larger subpopulations generally leads to higher probabilities that the entire search space is better explored. Hence better quality of the solution can generally be expected. However, more subpopulations imply the need for greater computing resources which is not easily available and often too costly for any organization to own. Therefore, the number of computing nodes available in any organization often hampers the degree of parallelism than may be attained. This bottleneck results in serious impediment to the practical application of embarrassingly parallelism in EA. Motivated

by these reasons, we consider a Grid enabled Hierarchical Parallel Genetic Algorithm by leveraging and extending from Grid computing technologies. Using the GOF, it is our objective to improve the search efficiency of the design optimization process. In particular, we deploy the subpopulations in the GOF across multiple clusters in parallel.

To use the Grid computing platform, a GSI proxy credential is required to authenticate the execution of the airfoil analysis function on the Grid resources. At the start of the optimization problem execution, the user's proxy certificate is used to initiate the user's proxy credential in order to allow the user to submit jobs to the Grid resources. Issues that are related to the credential management are not the focus of this thesis, and hence will not be further discussed.

The benefits of having a Grid-enabled GA, in the context of evolutionary design optimization process, are numerous. The main attraction is its ability to tap on the vast computing power and to achieve better utilization of the available libraries during the design search process. An example would be in an organization where specialized analysis codes, possessed by different design teams, often spans across geographically distributed locations. Using the Grid computing platform, these codes and data could be easily assessed, shared and utilized in a manner, which best improves the efficiency and effectiveness of the establishment.

In the implementation of the proposed Grid computing technology in this thesis, two essential features are highlighted. The first is that an extended GridRPC API is used to 'Gridify' the applications. This is done or 'Gridified' using Globus API built into our extension of the GridRPC API. This choice is down to its relative simplicity in implementation and its ability to offer high-level abstraction, *i.e.* the concealing of complex Grid environment from the users.

The second feature of the proposed procedure is the implementation of a meta-scheduler. From a survey of the literature, it was identified that the current state-of-the-art GridRPC implementations lack the feature for automatic resource discovery and selection on the Grid environment. As a result, users have to perform look-up and select resources that suit their needs manually before they can assign computing tasks to the respective resources. This leads to the inefficient use of resources since large amount of resources exist on Grid and these are often dynamic. Furthermore, most optimization problems generally have a large number of evaluation tasks that requires many identical computations, with different analysis parameter sets. Hence it is extremely inefficient if the interactions between the Clients and resources are repeated many times for the same remote procedure call. This is overcome here with the use of a meta-scheduler which schedules GridRPC requests and balances the workload across multiple clusters on the Grid.

Another characteristic of the proposed Grid computing system is the employment of a Lightweight Directory Access Protocol (LDAP), to query the MDS database for available resources and workload information. To the GridRPC Client, this acts as a centralized directory service to scout for the available resources and identify their location within the Grid environment. In its initialization process, the GridRPC function requires a simple entry from the MDS server and a virtual organization name to retrieve the available resources and check whether or not these resources are capable of servicing further GridRPC requests. These checks were performed for software, *i.e.* remote procedure, specialized libraries, *etc*, hardware, *i.e.* platform, operating system, *etc* and the workload. In this regard, the Ganglia [54,55] monitor toolkit is employed, to monitor and provide information relating to the available clusters in the computing systems.

4.5 Workflow of the Grid Enabled PHGA

The workflow of the Grid-enabled PHGA procedure in Figure 19 can be outlined as nine crucial stages. The nine stages are described as follows:

- 1) Prior to the start of the evolutionary search, the Grid enabled PHGA contacts the meta-scheduler, requesting for services and resources necessary required for conducting the evolution of the GA subpopulations.
- 2) The metascheduler then obtains a list of the available resources together with their status of availability. Such status information is acquired from services provided by the Globus Monitoring and Discovery Service (MDS) and Ganglia monitoring toolkit. The populations of design analysis requests may then be farmed out onto the available Grid resources accordingly to the workload and resource information provided by the monitoring services. It is worth noting that whenever a new computing resource or software service, for instance an 'airfoil-analysis' service, is added, they get reflected in the monitoring services automatically, since mechanisms are provided to ensure any new resources are registered to the Globus MDS before they are allowed to join the Grid environment.
- 3) These resource information and services are then provided to allow the PHGA to proceed with the parallel evolutionary search.
- 4) Upon obtaining the information in relation to the resources and services, the Grid Security Infrastructure (GSI) credentials are subsequently generated. This forms the authentication or authority to use the computing resources available in the system.
- 5) The GridFTP mechanism provided in the GOF then transfers the subpopulations in the form of ASCII data files to the computing clusters that have the correct services to perform genetic evolution and fitness evaluation.

- 6) Parallel evolution of the multiple subpopulations is then started at the remote computing clusters using the Globus job submission protocol.
- 7) Whenever the Globus Resource Allocation Manager (GRAM) gatekeeper of a cluster receives a request to start the 'subpopulation-evolution' service, an instance of this service gets instantiated on the master node of the cluster. Subsequently, the nested set of analysis service requests within the 'subpopulation-evolution' service can then be farmed across the processing nodes within the cluster using any cluster level local scheduler, for example, NetSolve, Sun Grid Engine [56], Condor, Portable Batch System (PBS) [57], Loadlever [58] or others. The responsibility of the local agent in each cluster is to perform local scheduling and resource discovery across the processing nodes within a cluster.
- 8) When the 'analysis' services completed execution, the fitness values of the chromosomes are conveyed back to the 'subpopulation-evolution' service so that standard GA operations such as mutation, crossover and selection can take place.
- 9) Similarly when the 'subpopulation-evolution' services deployed across the remote clusters completes, the resultant evolved subpopulations are then marshalled back to the PHGA client using the Global Access to Secondary Storage (GASS). The migration operation of the PHGA then proceeds. The process repeats until the termination condition is met.

Figure 19 : Workflow of Grid-Enabled PHGA

4.6 Conclusion

In this chapter, a general framework that leverages on the de facto standards of Grid computing to facilitate secure access to heterogeneous processing elements and software resources that may be dispersed across many geographical locations in the evolutionary search is proposed and presented. The Grid optimization framework, which employs Grid computing technologies, in solving computationally expensive and intractable design problems using parallel optimization algorithms had been successfully deployed the benchmark test problem and the airfoil design optimization problem, onto the Nanyang Campus Grid [59], from which the optimization problem is executed across the Grid computing platform, unleashing and utilizing the full power of Grid computing technology.

Chapter 5

Empirical Study

In this chapter, we present an analysis on GOF using the PHGA for complex engineering design optimization. We are interested in how the PHGA fares on a Grid environment with heterogeneous clusters, clusters of different number of computing nodes, and geographically disparate clusters. In our experiments, to study the efficiency of PHGA, the termination condition of the GA search is configured based on maximum number of search generations rather than based on search convergence. Further we consider a commonly used benchmark optimization problem and a realistic aerodynamic airfoil shape optimization problem with varying degree of fidelity to emulate both computationally inexpensive and expensive design problems. We employ four heterogeneous computing clusters in our empirical study. One of the clusters is located at Honda Research Institute (Honda RI) Europe, Germany, and the rest in Parallel and Distributed Computing Centre (PDCC) situated physically at Nanyang Technological University, Singapore. They are chosen to represent the existence of heterogeneous clusters commonly experienced in real world environment where one resource might differ from others in terms of processing speed, number of processors, or average intensity of workload. It is worth noting that our decision to use clusters that are geographically distributed across continents further emulates the realistic case where resources of design teams are often located in disparate manner. The complete specification of the computing resources selected is as listed in Table 1.

Cluster Name	Location	No. of CPUs	CPU Type
pdcc	PDCC, Singapore	30	<ul style="list-style-type: none">• PIV Xeon 3.06GHz
hpc-pdpm	PDCC, Singapore	22	<ul style="list-style-type: none">• PIV Xeon 2.6GHz
surya	PDCC, Singapore	29	<ul style="list-style-type: none">• PIII 450MHz• PIII 550MHz• PIII 733MHz
etl	Honda RI, Germany	12	<ul style="list-style-type: none">• PIII 650MHz

Table 1: List of computing resources employed for the experiments

In the rest of this chapter, we present case studies on the communication overhead incurred by a hierarchical serial GA and a hierarchical parallel GA on a GOF with single and multiple clusters. Subsequently, empirical study on the realistic aerodynamic airfoil design is then presented.

5.1 Communication Overhead

To start, an investigation on the possible communication overhead incurred in the proposed GOF is considered. Generally, two forms of communication overheads are incurred in the GOF, i.e. the inter-cluster and intra-cluster overheads as depicted in Figure 20. In our study, the inter-cluster communication overhead is the time to execute a Globus remote job execution transferring the input design file to the resource cluster via GridFTP. The intra-cluster communication overhead is attributed to the farming of chromosomes across a cluster of processing nodes in each cluster. Here, NetSolve is used

as the local scheduler. The total execution time for the PHGA using GOF can be estimated as follows:

Case 1. n -Subpopulations in a Single Cluster. In this case, the PHGA is executed in such a way that each subpopulation is evolved in a cluster containing multiple CPUs sequentially. This implies that only the chromosomes are evaluated in parallel. Hence, the total execution time for such a GOF arrangement can be formulated as:

$$T = Gen \times SubPop \times (Pop \times O_{intra} + \alpha \times F) \quad (1)$$

where T = Total execution time.

Gen = Number of generations the GA will evolve.

$SubPop$ = Number of subpopulations.

Pop = Number of chromosomes in one subpopulation.

α = The parallelism factor for a cluster, which is a factor of the population size and CPU specification.

O_{intra} = Intra-cluster communication overhead among all CPUs.

F = Wall-clock time taken for one fitness function evaluation, i.e. the problem size.

The $Pop \times O_{intra}$ term accounts for the farming of chromosomes within a cluster and is defined as a function of the population size. F is the wall-clock time taken to evaluate a single chromosome, hence $\alpha \times F$ represents the total time taken to evaluate an entire population of chromosomes in parallel on a single cluster.

In addition, if the GOF is started on a processing node external to the clusters where the subpopulations evolve, there is an additional overhead incurred, O_{inter} per subpopulation in every generation. This is a result of Globus communication at the Grid layer and is defined by equation (2).

$$T = Gen \times SubPop \times (O_{inter} + Pop \times O_{intra} + \alpha \times F) \quad (2)$$

where O_{inter} = Inter-cluster communication overhead, which includes job submission and file transfer. Here, we assume each subpopulation evolves sequentially.

Figure 20 : Communication Overheads of GOF

Case 2. n -Subpopulations across n -Clusters. Next, we consider the case where all n -subpopulations of the PHGA evolved across a n -clusters GOF in parallel, i.e., the subpopulation size equals cluster size. The total execution time becomes:

$$T_{\max} = Gen \times \left(\left(\sum_{i=1}^{SubPop} O_{inter-i} \right) + (SubPop-1) \times O_{inter} + Pop \times O_{intra} + \alpha \times F \right) \quad (3)$$

The first term in equation (3), $\sum_{i=1}^{SubPop} O_{inter-i}$, is a result of the Globus inter cluster communication overhead to farm out the GA populations onto the pool of computing clusters on the Grid. Here we consider the worst-case condition where farming and execution of chromosomes on computing nodes starts only after the last subpopulation has been farmed onto a cluster. This leads to the extra overhead of $(SubPop-1) \times O_{inter}$. The other overheads are the same as described in equation (1). From (1) and (3), and assuming O_{inter} are similar for $i \in \{1, 2, \dots, SubPop\}$, it is possible to infer that in order for a GOF to be effective in providing speed-up to the PHGA, the total execution time must be less than the sequential counterpart for the same number of subpopulations. This leads to

$$Gen \times (Subpop \times O_{inter} + (SubPop-1) \times O_{inter} + Pop \times O_{intra} + \alpha \times F) < Gen \times SubPop \times (Pop \times O_{intra} + \alpha \times F) \quad (4)$$

$$\Leftrightarrow (2 \times SubPop - 1) \times O_{inter} + Pop \times O_{intra} - SubPop \times Pop \times O_{intra} < (SubPop-1) \times \alpha \times F \quad (5)$$

$$\Leftrightarrow \frac{2 \times SubPop - 1}{(SubPop-1) \times \alpha} \times O_{inter} + \frac{Pop \times O_{intra} \times (1 - SubPop)}{(SubPop-1) \times \alpha} < F \quad (6)$$

$$\Leftrightarrow \frac{2 \times SubPop - 1}{(SubPop-1) \times \alpha} \times O_{inter} - \frac{O_{intra}}{\alpha} \times Pop < F \quad (7)$$

Consider SubPop=2,

$$\frac{2 \times 2 - 1}{(2-1) \times \alpha} \times O_{inter} - \frac{O_{intra}}{\alpha} \times 80 < F$$

$$\Leftrightarrow \frac{3}{\alpha} \times O_{inter} - \frac{O_{intra}}{\alpha} \times 80 < F$$

$$\Leftrightarrow 3 \times O_{inter} - O_{intra} \times 80 < \alpha \times F \quad (8)$$

As mentioned previously, F is the time to evaluate one single evaluation, thus $\alpha \times F$ is the time to evaluate one single subpopulation on the cluster. From equation 8, one can easily observed that GOF is effective in providing speed-up. Similarly, considering SubPop=3 or 4, the effectiveness of the GOF remains to hold.

Alternatively, from equation 4 one may arrive at equation 9 such that

$$2 \times O_{inter} < Pop \times O_{intra} + \alpha \times F \quad (9)$$

From equations 7 and 8, this implies that GOF is effective in contributing speed-up if the evaluation function is computationally expensive or the total time taken to evolve a single subpopulation is greater than twice the Globus inter cluster communication overhead. Further, we illustrate an example of the latter using Figure 21. In the figure, x denotes the execution time for a single subpopulation, i.e., $x = Pop \times O_{intra} + \alpha \times F$, y denotes the inter cluster communication overhead for an additional subpopulation, i.e., $y = 2 \times O_{inter}$. It can be easily derived for a sequential execution of n subpopulations using a single cluster, a total execution time of nx is incurred. In the case of using a GOF, the total execution time is found to be $x+ny$.

Figure 21 : Comparison of approximate execution time incurred for (a) n subpopulations in 1 cluster, (b) n subpopulations in n clusters.

In Table 2, we tabulate the execution time of GOF for 2, 3, 4, and 5 subpopulations. Note that this implies y is fixed at 4 while x varies according to the number of subpopulations. The table shows that as long as equation 8 is satisfied, GOF shall remain as effective, i.e. the total execution time using GOF is less than a hierarchical serial GA for the same number of subpopulations. When $y=4$, the breakpoints are found at $x=9, 7, 6, 6$ for 2, 3, 4, 5 subpopulations respectively.

No. of Subpopulations	$y=4$													
	$x=3$		$x=4$		$x=5$		$x=6$		$x=7$		$x=8$		$x=9$	
	SC	NC	SC	NC	SC	NC	SC	NC	SC	NC	SC	NC	SC	NC
2	6	11	8	12	10	13	12	14	14	15	16	16	18	17
3	9	15	12	16	15	17	18	18	21	19	24	20	27	21
4	12	19	16	20	20	21	24	22	28	23	32	24	36	25
5	15	23	20	24	25	25	30	26	35	27	40	28	45	29

SC = Single Cluster, NC = n clusters for n subpopulations

Table 2 : Comparison of execution time for different x and y setting and different subpopulation size.

It is worth noting that the formulation derived in equation 3 for the total execution time remains valid for the case where n subpopulations are evolved using a single cluster. The only core difference lies in the inter-cluster communication overhead incurred. In the later case, O_{inter} would generally be expected to be lower since the subpopulations are farmed in parallel within itself.

5.2 Benchmark Test Function

To emulate a computationally inexpensive problem, we harnessed one of the commonly used benchmark function in GA. The Rastrigin test function is defined as:

$$Ras(x) = 20 + \sum_{i=1}^j x_i^2 - 10 \sum_{i=1}^j \cos 2\pi x_i$$

where j denotes the number of dimensions used for the function. A 10 dimensional Rastrigin function is used as the model signifying inexpensive design problems.

Here, we conduct experiments for varying number of subpopulations on a single cluster and across n clusters. The maximum number of generations before search termination is set to 100 and each subpopulation size N is configured as 80 chromosomes. In addition, we employed a uniform crossover and mutation operators at probabilities 0.9 and 0.1, respectively. A linear ranking algorithm is used for selection with elitism enforced. However, the core parameters crucial to our analysis are subpopulations size N and termination generations. Two communication protocols with differing inter-cluster communication overheads are considered here. The inter-cluster communication overheads with or without the incorporation of firewall security in the farming of subpopulations to disparate clusters are tabulated in Table 3.

Communication Protocol Setting	Inter-cluster Communication Overheads, O_{inter} (seconds)
High Security (high communication overhead)	30
Low security (low communication overhead)	0.01

Table 3 : Average wall-clock time to transfer a subpopulation of chromosome in different communication protocol.

The averaged wall-clock time (in seconds) incurred by the Grid-enabled, i.e., based on the proposed GOF, when searching on the PHGA Rastrigin problem as a function of the number of subpopulations is shown in Figure 22. The results reported are average of ten runs for each case. It is found that for using both communication protocols, the Grid-enabled PHGA displays significant speedup over the hierarchical serial GA on the Rastrigin function.

Figure 22 : Execution time of GOF on Rastrigin problem for hierarchical serial GA, hierarchical parallel GA

Next, we also consider the study of two additional cases, i.e. n parallel subpopulations on a single cluster and n parallel subpopulations on n clusters for both high and low communication overhead. From the results reported in Figure 23, it appears that the GOF is effective only for the low communication overhead protocol. Note that the non-linearity existing in the execution time for increasing subpopulation and cluster size in Figure 23 is most likely attributed to the heterogeneous clusters employed in the experiment.

Figure 23 : Execution time of GOF on Rastrigin problem for parallel hierarchical GA

In what follows, we investigate the effect of employing computing resources in a geographical distributed scenarios which is common in a Grid environment. In our study, we consider a 2-subpopulation PHGA, each deployed across two clusters that are physically located across two geographical far continents, i.e., one cluster is located in Singapore while the other in Germany. We compare the result between running two subpopulations in a single cluster (*et1*) and across two clusters (*et1* and the other on *hpc-pdpm* or *surya*). The execution time of five independent Grid enabled PHGA runs using low communication overhead protocol are summarised in Figure 24 for the Rastrigin function.

Figure 24 : Wall-clock time for geographically distant clusters with two subpopulations deployed.

From the results obtained, it appears that even though the clusters may be geographically disparate, the GOF remains to provide significant reduction in execution time than a hierarchical serial GA. The extra communication overheads incurred due to long distance communication appear to be insignificant compared to the significant efficiency contributed via parallelizing the subpopulations across the abundance of available Grid cluster resources.

5.3 Real World Problem: Aerodynamic Airfoil Design

One of the most important parts of an aircraft is during its takeoff to get the body off the ground and subsequently to maintain it there. This usually comes from the efficient process of lifting the body up. By far the most common means of lifting, widely used in today’s industry, is the wing. It is a concept ergonomically designed and embodied in current modern machinery. In actual fact, wings of all shapes and sizes can be used to lift these bodies up, but special care must be taken to ensure that these wings are of the proper geometry. This is to ensure stability, efficiency and performance excellence.

Due to the short time span of this project, simplifications were made to the original design of the aerodynamic airfoil design problem. As the main emphasis of the

project is to show the possibility of using EA, using the proposed alternative procedure, to solve such problems, the assumptions and changes made does not affect the gist of the problem. As such, the details of the problem are not discussed here, but can be found in literatures published elsewhere [60, 61].

The simplified problem, considered in this project, is to achieve the best 24 design variables with an optimized drag profile, *i.e.* a minimized drag and a maximized lift, at a Mach number value of $M_\infty \in [0.45, 0.5]$, with the best midpoint value at 0.5 at an angle of attack of 0.2° . These are schematically illustrated in Figure 25.

Figure 25 : Simplified aerodynamic airfoil design problem

In simplifying further the complex aerodynamic airfoil design problem, the problem is represented using mathematical expressions or parameterization. Here, it is parameterized using the weighted sum of a number of basis functions. The Hicks and Hennes [62] expressions are employed, which give the upper and lower thickness, y , of the airfoil as the sum of the basis functions, $f_j(x)$, as shown in equation (10),

$$y = y_{basic} + \sum_{j=1}^N z_j f_j(x) \quad (10)$$

where x is the position along the chord and z_j are design variables. From (10), it can be seen that the greater the number of parameters considered, the larger the set of shapes represented in the equation and thus, the larger the design search space. In this work, 12 such functions have been employed, as shown in Figure 26. The shape of one of the airfoil represented by the Hicks-Hennes series functions is shown in Figure 27.

Figure 26 : AeroFoil Parameterization using 12 Hicks-hennes Functions

Figure 27 : One of the Airfoils Represented by the Hicks-Hennes Functions

In this chapter, we study the Grid-Enabled PHGA for complex real world problem, specifically the efficient optimization of aerodynamic airfoil shape design. In particular, we consider the parametric design optimization of 2D airfoil structure using a subsonic inverse pressure design problem. The target pressure profile is generated from the NACA 0012 airfoil, which itself is also the baseline shape. The airfoil geometry is characterised using 24 design variables with the NACA 0012 airfoils as the baseline, as shown in Figure 28. Hence, there exists for this problem a global solution corresponding to $z_1 = \dots = z_{24} = 0$. The free-stream conditions in this problem are subsonic speed of Mach 0.5, and zero angle of attack (AOA), corresponding to symmetric pressure profiles on the upper and lower walls. It is worth noting that the inverse problem constitutes a good test problem for facilitating our study on complex engineering design optimization problems of variable-fidelity. Secondly, the inverse design problem also has a practical purpose, as the designer generally has an idea of the desired pressure profile that yields good aerodynamic performance. For example, in transonic design, a shock front on the upper surface generally leads to undesirably high pressure drag that degrades the efficiency of

the airfoil. A typical approach to inverse pressure design is to ‘smoothen’ the pressure distribution on the upper-surface in a way that maintains the area under the curve, so as to maintain the lift force generated by the airfoil. Thus, the inverse pressure design problem can be formulated as a minimization problem of the form

$$I(w, S) = \frac{1}{2} \int_{\text{wall}} (p - p_d)^2 d\sigma \quad \text{subject to constraints.} \tag{11}$$

Sigma represents the curvilinear coordinates for integration along the wall boundary (closed integral).

Figure 28 : Airfoil geometry characterized using 24 design variables with the NACA 0012 as baseline.

A common way to measure the processing power of a computing cluster is the use of benchmark problems provided in LINPACK. Table 4 shows the Million Floating Point Operations (MFLOPs) for three clusters. The table indicates that the three clusters used to perform the experiments are clearly heterogeneous. The *pdcc* cluster has a higher MFLOPs compared to *surya*. This implies that it is a more powerful cluster than the latter, *i.e.*, greater than 6 times more powerful.

Cluster Name	No. of CPUs	CPU Type	MFLOPs (average)
Pdcc	30	PIV Xeon 3.06GHz	920
hpc-pdpm	22	PIV Xeon 2.6GHz	800
surya	29	<ul style="list-style-type: none">PIII 450MHzPIII 550MHzPIII 733MHz	150

Table 4 : Specifications and MFLOPS of the three clusters.

The computationally inexpensive Rastrigin test function consumes relatively insignificant CPU time and memory on the heterogeneous clusters of computing nodes. Thus, even though the heterogeneous clusters differ in specifications, this is clearly not visible for the Rastrigin test function as illustrated in Table 5. A single evaluation of the Rastrigin function takes merely 4 seconds on all the three clusters of different specifications.

	Rastrigin Function		
Cluster	<i>surya</i>	<i>hpc-pdpm</i>	<i>pdcc</i>
Wall-Clock Time (sec)	4	4	4

Table 5 : Execution time for Rastrigin test function on the various clusters.

Using 24 design variables and the cost function in equation 11, we conduct an empirical study of Grid-Enabled PHGA for aerodynamic airfoil shape design using variable-fidelity analysis code on the three clusters. Here, a single moderate-fidelity analysis of the airfoil geometry using an Euler CFD solver takes around 110 seconds on a Pentium III processor, while a low-fidelity takes around 10 seconds. Table 6 summarises the

execution time for a single airfoil analysis on three clusters with differing specifications defined in Table 5. From Table 7, the effects of differing specifications of clusters on the airfoil analysis code are evident, especially on the moderate-fidelity analysis code. Hence, the variable-fidelity analysis code we consider here provides a more practical and realistic study on using Grid-Enabled PHGA for complex engineering design.

Variable Fidelity	Low-fidelity Analysis code			Moderate-fidelity Analysis code		
Cluster	surya	hpc-pdpm	Pdcc	surya	hpc-pdpm	pdcc
Wall-Clock Time	10 s	9 s	8 s	110 s	54 s	37 s

Table 6 : Execution time of variable-fidelity airfoil problem on clusters with differing specifications

Next, we applied GOF to search on the variable-fidelity airfoil design problems. In particular, we consider a hierarchical parallel GA with two subpopulations and two heterogeneous clusters of differing specifications for airfoil design optimization. The maximum number of generations before termination is configured as 100 generations and each subpopulation size N is 80 chromosomes. The execution time in seconds is the averaged of 10 runs for the low and high fidelity airfoil design problems are summarized in Figure 29 and 30, respectively.

From the results in figures 29 and 30, one may conclude that by limiting the execution of one subpopulation per cluster could led to poor search efficiency in the Grid-Enabled PHGA due to the differences in the specifications of the clusters that exists on the Grid. Note that we are able to achieve the best performance in efficiency if both subpopulations are executed in parallel on the *pdcc* high-end cluster alone. This is

observed for both the low or moderate fidelity analysis codes. Using two clusters, i.e., *pdcc* + *hpc-pdpm* or *pdcc* + *surya*, clearly results in a poorer search efficiency than the *pdcc* high-end cluster alone. Further, on the low-fidelity airfoil problem, the search efficiency for *pdcc* + *hpc-pdpm* or *pdcc* + *surya* or *hpc-pdpm* + *surya* are found to be better than using the *hpc-pdpm* or *surya* alone. In contrast, moderate-fidelity search efficiency of *pdcc* + *surya* or *hpc-pdpm* + *surya* does fare better than executing the two parallel subpopulations on *hpc-pdpm* alone. The increase in computational cost for each analysis of the chromosome in the case of using moderate-fidelity airfoil problem greatly amplifies the effects of differing specifications between the clusters. As *surya* is very much lower in computing specifications as compared to the other clusters, it has the effect of slowing the evolution since all subpopulations have to wait for it to complete execution of its subpopulation before migrations between the subpopulations may proceed.

(a)

(b)

(c)

Figure 29 : Execution time for hierarchical parallel GA with two subpopulations and two heterogeneous clusters of differing specifications for low-fidelity airfoil design optimization

(a)

(b)

(c)

Figure 30 : Execution time for hierarchical parallel GA with two subpopulations and two heterogeneous clusters of differing specifications for high-fidelity airfoil design optimization

Clearly, by enforcing each cluster to evolve only a single subpopulation each time generally limits the potential of the GOF to generate high performance in efficiency. Hence, to fully utilise the Grid computing cluster resources for complex engineering design of computationally expensive optimization problems, resized bundling of chromosomes is proposed. Here we pool all chromosomes in the subpopulations together and submit chromosomes to cluster according to their specifications, for instance the MFLOPS. This way more chromosomes are sent to the high-end clusters than its lower-end counterparts.

Using the MFLOPS of the three heterogeneous clusters defined in Table 4, resized bundling may be performed as follows:

$$R_i = \frac{C_i}{\sum_{i=1}^n C_i} \times Pop \tag{12}$$

where R_i = Ratio of chromosomes sent to the i -th cluster.

C_i = MFLOPS of the cluster.

n = Total number of clusters to be used.

Table 7 tabulates the percentage of chromosomes that will be sent to three clusters of differing specification for execution using equation 12.

Cluster Name	MFLOPs (average)	Ratio (%)
pdcc	920	49.2
hpc-pdpm	800	42.8
surya	150	8

Table 7 : Resized bundling of chromosomes to heterogeneous cluster.

Further to the above resizing of bundle in term of the MFLOPs of the cluster, the heterogeneity of the clusters based on the number of CPUs in each cluster can be considered as follows:

$$R_i = \frac{U_i C_i}{\sum_{i=1}^n U_i C_i} \times Pop \tag{13}$$

- where R_i = Ratio of chromosomes sent to the i -th cluster.
- C_i = MFLOPS of the cluster.
- U_i = Number of CPUs in the cluster.
- n = Total number of clusters to be used.

Taking into account the number of CPUs in each of the cluster, Table 8 tabulates the percentage of chromosomes that will be sent to three clusters of differing specification for execution using equation 13.

Cluster Name	Number of CPUs	MFLOPs (average)	Ratio (%)
pdcc	28	920	57.4
hpc-pdpm	20	800	35.6
surya	21	150	7

Table 8 : Resized bundling of chromosomes to heterogeneous cluster with CPUs considerations.

With the new bundling ratio as tabulated above, we further investigated the impact of the resized bundling of the chromosomes not only based on the MFLOPs of the computing clusters, but take into considerations of the number of CPUs in the clusters. An average of 5 runs was collected and plotted for a 2 subpopulations using the two equations, one with

only the MFLOPs of the cluster, while the other includes the number of CPUs in the cluster. From the result shown in Figure 31 and Figure 32, it is observed that in general, the resizing the bundle of chromosomes to be sent to each cluster based on the MFLOPs of the cluster and number of CPUs taken in account will be more efficient compared to a simple resized of the bundling of chromosomes based on the MFLOPs of the cluster.

Figure 31: Execution time for Moderate-fidelity analysis code for 2 subpopulations deployment with/without CPUs resized bundling on hpc-pdpm and pdcc

Figure 32 : Execution time for Moderate-fidelity analysis code for 2 subpopulations deployment with/without CPUs resized bundling on surya and pdcc

Chapter 6

Concluding Remarks

This chapter concludes the thesis with a brief synopsis of the contributions of the present research. Areas for future work and a program of future research are also outlined.

6.1 Research Contributions

The primary focus of this research is to develop an optimization framework for evolutionary computationally expensive complex engineering design problems that tap on vast compute power of the emerging Grid computing technologies. This proposed GOF demonstrate the utility of Grid computing in parallel evolutionary optimization. Further, based on this proposed GOF, we mitigate the problems of complex engineering optimization problems and perform experimental studies. As we know, deriving efficient optimum solutions on computationally intensive problems has remained a challenging research issue in scientific computing. Fortunately, it is well known that a sub-linear improvement in search efficiency may be achieved via parallelism of EAs. For such applications, it is often desirable to harness great deals of computational resources and use their aggregate power for tackling the computationally expensive design process, which can be overcome with Grid computing technologies.

The primary contributions of the present program of research are summarized below.

6.1.1 Grid Optimization Framework

In this work, we present a scalable parallel framework, which employs Grid computing technologies, for solving computationally expensive and intractable design problems. Using an aerodynamic airfoil design optimization problem as an example, the application of Grid computing strategies is discussed. The investigations revealed that the employment of these high-fidelity analysis codes, in conjunction with the proposed computing technologies, has considerably reduces the design time.

In solving the realistic aerodynamic airfoil design optimization problem, large search spaces are often required, implying that the computational budget based on a single HPC cluster may be limited and insufficient. Thus, an alternative would be to ‘migrate’ the most time consuming part of the design cycle process onto the Grid environment, which often has unlimited computational resources. This will undoubtedly help to shorten the entire optimization design cycle time. As such, Grid computing can be viewed upon as a medium offering very powerful and inexpensive computational resources, pooled together to solve large complex problems. It will be especially useful in reducing the bottleneck problems in optimization phases, with the use of clusters of computers for analyses and simulations and performing alternative designs in parallel.

6.1.2 Grid Enabling GA using Bundling of jobs

In this research, we look into the use of cluster based architecture integrated with the Grid using the bundling techniques to harness a multi-cluster searching of the optimal solution. First, let consider the transfer of input design file from the client to the various clusters is done in GridFTP, where the average time to deliver the file to the destination took only a very insignificant 0.01s. For a very simple blasting of the evaluations to the different Grid nodes using the simple submissions, either using the flocking of the condor pool of

resources or simply job submission to the Grid resources job manager, the total transfer time will become significantly large and result in performance degradation. Take the example of the number of generation count for termination is set to 100 generations, with a total of subpopulations sending to 5 different clusters and each subpopulation size of 80 chromosomes, that will add up to a total of $80 * 0.01 * 5 = 4s$, and a total of 400s delay in the total execution. Next, each individual submission of jobs to the resource will also incur a job execution delay. Consider a single submission of a job to the Grid nodes took an estimate of 1s, a total of 80 jobs (80 chromosomes) will add up to 80s, and with 5 subpopulations, and 100 generations of iterations, the total figure become significant large. Furthermore, blasting of 80 jobs will create a messy 80 input files to be transferred. Thus, bundling of subpopulation as a single job each cluster, will hence significantly reduce this issues.

6.1.3 Grid Enabling GA using Resized Bundling of jobs based on Grid Resources MFLOPs

Based on the basis on GOF bundling, we further this techniques using resize bundling. Here, the PHGA subpopulations bundle of chromosomes used for optimising the airfoil design problem are resized based on the MFLOP of the various clusters. From the result, it is shown that in general, the resizing the bundle of chromosomes to be sent to each cluster based on the MFLOPs of the cluster will be more efficient compared to simply bundling of chromosomes in fixed to each individual selected cluster. Without proper resized chromosome bundle, the performance of the GOF is affected by the slower cluster in the execution. However, using resized bundle of chromosomes, with more chromosomes being sent to the higher end cluster, there is improvement

6.1.4 Grid Enabling GA using Resized Bundling of jobs based on Grid Resources MFLOPs and number of CPUs in the cluster

We further investigated the impact of the resized bundling of the chromosomes not only based on the MFLOPs of the computing clusters, but also on the number of CPUs in the clusters were also taken into consideration. An average of 5 runs was collected and plotted for a 2 subpopulations using the two equations, one with only the MFLOPs of the cluster, while the other includes the number of CPUs in the cluster. From the result gathered, it is observed that in general, the resizing the bundle of chromosomes to be sent to each cluster based on the MFLOPs of the cluster and number of CPUs taken in account will be more efficient compared to a simple resized of the bundling of chromosomes based on the MFLOPs of the cluster.

6.2 Future Work

The aim of this thesis is to present a study on Grid Enabled Optimization (GOF), which employs Grid computing technologies, for complex engineering design problems using parallel evolutionary optimization algorithms. We have successfully deployed the airfoil design optimization problem, onto the Nanyang Campus Grid, from which the optimization problem is executed across the Grid computing platform, unleashing and utilising the full power of Grid computing technology. Studies was made to determine how the Grid enabled Hierarchical Parallel Genetic Algorithm fares on a Grid environment with heterogeneous clusters, clusters of different number of computing nodes, and geographically disparate clusters. From the results present and analyzed, it is possible to conclude that Grid enabled Hierarchical Parallel Evolutionary Algorithm is not a mere hype but offers a credible alternative for providing significant speed-up to

complex engineering design optimization. Further, dynamic resized bundling based on the specifications of a cluster further helps unleash the performance efficiency of the GOF.

The Grid platform will become more advanced, powerful, dynamic and complex in the coming future, with more number of components and geographically distributed. This means that major research is needed on this topic to understand how effectively and efficiently to deploy and implement a Grid aware optimization applications, and this project research represent the small step in that direction. In summary, the GOF framework has the potential to be of great value to the EA community, and at the same time representing an important class of Grid application architecture.

In the future, investigation on some other issue such as different communication overhead involved in the execution of GOF from different environmental settings, such as geographical distant or different communication protocols had on the impact of the GOF, and the use of distributed benchmark to determine the resized bundling of chromosomes, and multi-level bundling farming algorithms. A summary of potential future work is outlined below.

6.2.1 HPC LINPACK benchmark

In the current work, standard LINPACK was used to determine the computing power of the Grid resources, and obtained the bundling decision. The benchmark was conducted on each sub nodes of the Grid cluster, and an average value was obtained as the benchmark for each of the computing cluster. For better evaluation of the computing cluster, HPC LINPACK can be used instead to determine the MFLOPs of the Grid resources.

6.2.2 Nested Layer of Grid Resources

In the existing GOF architecture, only a single level of Grid resources was utilized to perform the parallelism at the chromosomes layers. However, it is possible to consider the

further utilizing of Grid resources at the chromosomes layers, by farming the chromosomes bundle at each Grid cluster layer to another layer of Grid resources. Take an example of chromosome bundle of 200 evaluation sent to a Grid cluster of 20 processing nodes, where each evaluation can take up to 30 minutes of computing time. With this scenarios, further farming on the chromosome at the first layer into another layer of parallelism will speed up the time to complete the entire evaluations.

6.2.3 Web Services Framework

“The Only Thing Constant Is Change” – Looking back on the last few decades, this seems certainly to be true for the computing industry. We see the emergence of the World Wide Web (WWW), the evolution of service oriented computing of Web and today, the Grid technologies, are experiencing a similar trajectory. The Grid technologies is evolving and converging with the Web service communities, to establish a service oriented infrastructure, known as Grid service, which simply means Web service that provides a set of well-defined interfaces and that adapt specific conventions for the Grid.

GOF was proposed and built in the early phase of the movement of Grid into this service oriented infrastructure. However, the architecture of the GOF allows the possibilities to adapt to the growing service semantics. The meta-scheduling, the credential verification and most importantly, the optimization evaluation can be deployed as Web services.

Appendix A

Publications

- **Hee-Khiang Ng**, Dudy Lim, Yew-Soon Ong, Lars Freund, Shuja Parvez, Bu-Sung Lee, Bernhard Sendhoff “*A Multi-Cluster Grid Enabled Evolution Framework For Aerodynamic Airfoil Design Optimization*”, First International Conference on Natural Computation/Second International Conference on Fuzzy Systems and Knowledge Discovery, 2005.
- **Hee-Khiang Ng**, Yew-Soon Ong, Terence Hung, Bu-Sung Lee, “*Grid Enabled Optimization*”, Advances in Grid Computing: European Grid Conference, 2005.
- W. B. Song, Y. S. Ong, **H. K. Ng**, A. J. Keane, S. Cox and B. S. Lee, “*A Service-Oriented Approach for Aerodynamic Shape Optimization across Institutional Boundaries*”, Eighth International Conference on Control, Automation, Robotics and Vision, Special Session on Computational Intelligence on the Grid, 2004.
- Q. T. Ho, Y. S. Ong, W. T. Cai, **H. K. Ng**, and B. S. Lee, “*GAD Kit - A Toolkit for Gridifying Applications*”, Fifth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2004.
- **H. K. Ng**, Y. S. Ong, T. Hung and B. S. Lee, “*Complex Engineering Design Problem Solving Environment Grid Portal*”, 2nd International Conference on Scientific and Engineering Computation, 2004.

Appendix B

Adjoint methods for aerodynamic sensitivity analysis

An overview of the Euler adjoint solver, for the aerodynamic shape sensitivity analysis, is presented. The adjoint approach was first applied by Jameson to the design optimization of transonic airfoils. Based on an optimal control theory applied to systems of partial differential equations, it treats the problem of shape optimization as a constrained optimization problem. The constraint appears in the form of an Euler or a Navier-Stokes flow equation. For differentiable problems, this method yields the gradient of the cost function at any given design point.

In comparison with traditional finite difference approximations, the computational cost of the adjoint method is only one solution cycle of the flow equation, plus one solution cycle of the adjoint equation which has roughly the same computational cost as the flow equation. Furthermore, the cost of the sensitivity analysis using the adjoint method is independent of the number of design variables. For a typical steepest descent method, the gradient information can, therefore, be employed to efficiently search for the local minima. Recently, this method has also successfully been extended to the wing and wing body design optimization.

The adjoint method can be also applied to inverse design problems, where the cost function is the quadratic sum of the difference between a design's surface pressure and a target pressure distribution. In direct design problems, the cost functions are usually the aerodynamic performance indices such as the drag coefficient, the drag-to-lift ratio, *etc.* In this project, the efficiency of the adjoint method is exploited in conjunction with the proposed EA optimization procedure, which is based on the Euler equations. This is done by firstly using an inverse pressure design problem which is then followed by the direct optimization process. Details of these formulations are described below.

The adjoint method

In illustrating the adopted adjoint approach for sensitivity analysis, the following constrained minimization problem is considered,

$$\min_{w,S} I(w,S) \quad \text{subject to} \quad R(w,S) = 0 \quad (\text{B.1})$$

where w and S are the design variables. The scalar cost function I and the constraint function R is assumed to be differentiable. The associated *Lagrangian* is thus given by,

$$L = I(W,S) + \langle \psi, R(w,S) \rangle \quad (\text{B.2})$$

where ψ is the *Lagrange multiplier*, otherwise also known as the *adjoint* or *auxiliary variable*, and has the same dimension as R . The main idea in introducing ψ is to reduce the problem to that of an unconstrained one, with an additional set of equations to solve, *i.e.* the *adjoint equations*. This can be obtained from the first-order necessary condition for optimality, which states that at the optimum, either local or global, the variation of δL is zero for all admissible and independent variations of δw and δS ,

$$\delta L = I_w(w,S)\delta w + I_s(w,S)\delta S + \langle \psi, R_w(w,S) \rangle \delta w + \langle \psi, R_s(w,S) \rangle \delta S = 0 \quad (\text{B.3})$$

The variations in w , independent of S , now simply implies that an adjoint equation that governs the *Lagrange multiplier*,

$$\psi : \langle \psi, R_w(w,s) \rangle + I_w(w,S) = 0 \quad (\text{B.4})$$

Thus, it can be seen that the variations of the Lagrangian depends only on the variations in S , *i.e.*,

$$\delta L = (I_s(w,S) + \langle \psi, R_s(w,S) \rangle) \delta S \quad (\text{B.5})$$

Thus, equation (B.5) shows the sensitivity of the Lagrangian with respect to shape changes. As it is an algebraic expression that depends on the analytical values of ψ , one can compute the sensitivities by solving (B.4). This will be shown below, whereby the above general theory is applied to the aerodynamic airfoil design problem.

Variational calculus and adjoint equations

In the aerodynamic airfoil shape design optimization problem, the following terms is defined, *i.e.* w is the flow variable, S is the shape design variables and R is the governing equations. Here, an inviscid flow over an airfoil, is considered, such that it can be modelled by the two-dimensional Euler equations in the physical domain, as given by,

$$\frac{\partial w}{\partial t} + \text{div}g(w) = 0 \quad (\text{B.6})$$

The coordinates in the physical domain is denoted by x_i , where $i = (1,2)$. The following transformation to a computational domain, is then considered, with the coordinates ξ_i , where,

$$K_{ij} = \frac{\partial x_i}{\partial \xi_j}, \quad J = |K|, \quad S = JK^{-1}, \quad S_{ij} = J \frac{\partial \xi_i}{\partial x_j} \quad \text{where } (i, j = 1, 2) \quad (\text{B.7})$$

Transformation of (B.7) yields the following contra-variant equation,

$$\frac{\partial W}{\partial t} + \text{div}G(w, S) = 0 \quad (\text{B.8})$$

where $W = Jw$ and $G_i(w, S) = S_{ij}g_j(w)$. As the computational domain is defined by a fixed grid, the transformation matrix S is, thus, directly related to the shape of the airfoil, and for the moment can be considered as the shape design variables. Hence, transforming to the computational domain elicits the relation between the flow variables and the shape, *i.e.* they must satisfy the following constraint at steady-state,

$$\text{div}G(w, S) = 0 \quad (\text{B.9})$$

Consequently, the variations of G takes the following form,

$$\delta G_i = S_{ij} \frac{\partial g_j}{\partial w} \delta w + g_j \delta S_{ij} \stackrel{\Delta}{=} C_i \delta w + g_j \delta S_{ij} \quad (\text{B.10})$$

Adjoint of the Euler equations for inverse pressure design problem

As evident in (B.5), the gradient expression depends on the cost-function Jacobian, I_S . As a first stab, an inverse design problem, which involves minimizing the difference between the surface pressure, p , of a given airfoil and a desired pressure profile, p_d , is considered. This problem may be observed in two different aspects. Firstly, it may be viewed as a test problem, where the desired pressure distribution, p_d , is computed from a known shape. Convergence to this known shape will, thus, serve as a validation of the proposed algorithm. Secondly, the inverse design problem also has a practical purpose, as the designer generally has an idea of the desired pressure profile that yields good aerodynamic performance. For example, in transonic design, a shock front or sharp pressure gradient on the upper surface generally leads to undesirably high pressure drag that degrades the efficiency of the airfoil. A typical approach to inverse pressure design is to ‘smoothen’ the pressure distribution on the upper-surface in a way that maintains the area under the curve, so as to maintain the lift force generated by the airfoil. Thus, for such inverse pressure design problem, a minimization formulation can be formed, as given by,

$$I(w, S) = \frac{1}{2} \int_{wall} (p - p_d)^2 d\sigma \quad \text{subject to (B.9)} \quad (\text{B.11})$$

Consequently, the *Lagrangian*, in the problem is, thus, given by,

$$L = \frac{1}{2} \int_{wall} (p - p_d)^2 d\sigma + \int_{\Omega} \psi^T \text{div} F d\Omega \quad (\text{B.12})$$

Therefore, following on from the previously discussed arguments, the adjoint equation in (B.4) now comprises of a differential part: the differential adjoint equation of (B.13) and a set of adjoint boundary conditions of equation (B.14),

$$\frac{\partial \psi}{\partial t} - C_i^T \frac{\partial \psi}{\partial \xi_i} = 0 \quad \text{over } \Omega \quad (\text{B.13})$$

$$J(\psi_2 \frac{\partial \xi_2}{\partial x_i} + \psi_3 \frac{\partial \xi_2}{\partial x_2}) = (p - p_d) \quad \text{on the wall boundary} \quad (\text{B.14})$$

This leads to the following sensitivity expression for the Lagrangian formulation,

$$\delta L = - \int_{\Omega} \frac{\partial \psi^T}{\partial \xi_i} g_j \delta S_{ij} d\Omega - \int_{wall} (\psi_2 \delta S_{21} + \psi_3 \delta S_{22}) p_d \zeta_1 \quad (B.15)$$

It is now obvious that in order to obtain the sensitivity information, one only needs to solve (B.13) with (B.14), which is a linear differential equation of the same dimension as the Euler equations. Hence, the overall cost of the gradient calculation is only one CFD simulation, in addition to the Euler equations.