Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations

Graduate Studies

5-2015

Advancement of Computing on Large Datasets via Parallel Computing and Cyberinfrastructure

Ahmet Artu Yildirim
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd

Part of the Computer Sciences Commons

Recommended Citation

Yildirim, Ahmet Artu, "Advancement of Computing on Large Datasets via Parallel Computing and Cyberinfrastructure" (2015). *All Graduate Theses and Dissertations*. 4312.
https://digitalcommons.usu.edu/etd/4312

This Dissertation is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

UtahStateUniversity
MERRILL-CAZIER LIBRARY

ADVANCEMENT OF COMPUTING ON LARGE DATASETS VIA PARALLEL COMPUTING AND CYBERINFRASTRUCTURE

by

Ahmet Artu Yıldırım

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

Dr. Daniel Watson Major Professor	Dr. David Tarboton Committee Member
Dr. Vladimir Kulyukin Committee Member	Dr. Minghui Jiang Committee Member
Dr. Stephen Clyde Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2015

Copyright © Ahmet Artu Yıldırım 2015

All Rights Reserved

ABSTRACT

Advancement of Computing on Large Datasets via Parallel Computing and Cyberinfrastructure

by

Ahmet Artu Yıldırım, Doctor of Philosophy
Utah State University, 2015

Major Professor: Dr. Daniel Watson
Department: Computer Science

Large datasets require efficient processing, storage and management to efficiently extract useful information for innovation and decision-making. This dissertation demonstrates novel approaches and algorithms using virtual memory approach, parallel computing and cyberinfrastructure. First, we introduce a tailored user-level virtual memory system for parallel algorithms that can process large raster data files in a desktop computer environment with limited memory. The application area for this portion of the study is to develop parallel terrain analysis algorithms that use multi-threading to take advantage of common multi-core processors for greater efficiency. Second, we present two novel parallel WaveCluster algorithms that perform cluster analysis by taking advantage of discrete wavelet transform to reduce large data to coarser representations so data is smaller and more easily managed than the original data in size and complexity. Finally, this dissertation demonstrates an HPC gateway service that abstracts away many details and complexities involved in the use of HPC systems including authentication, authorization, and data and job management.

(133 pages)

PUBLIC ABSTRACT

Advancement of Computing on Large Datasets via Parallel Computing and Cyberinfrastructure

by

Ahmet Artu Yıldırım, Doctor of Philosophy
Utah State University, 2015

Major Professor: Dr. Daniel Watson
Department: Computer Science

Large datasets require high processing power to compute, high-speed network connections to transmit, or high storage capacity to archive. With the advent of the internet, many in the science community and the public at large are faced with a need to manage, store, transmit and process large datasets in an efficient fashion to create value for all concerned. By example, large environmental researchers analyze large map data to extract hydrologic information from topography. However, processing these data and other tasks is hard – sometimes impossible – in minimal resource environments such as desktop systems.

This dissertation demonstrates novel approaches and programming algorithms that address several issues associated with large datasets. We present a novel virtual memory system that can process large map data which are too big to fit in memory in raster-based calculations. Then, we introduce parallel computer algorithms that reduce large data to smaller, more easily managed forms; and finding the patterns inside of this smaller representation of large data efficiently in large computer systems. We need also software services on the internet to access large remote computer systems. We present a software service named *HydroGate* that abstracts away many details and complexities involved in

the use of large remote computer systems, including authentication, authorization, and data and job management.

ACKNOWLEDGMENTS

Foremost, I would like to thank my advisors Prof. Dan Watson, and Prof. David Tarboton for providing me with the opportunity to complete my Ph.D. thesis at Utah State University. I especially want to thank my advisor, Prof. Dan Watson, whose guidance and support helped me in the research and writing of this thesis; and Prof. David Tarboton who actively supported and provided direction in my work. They have always been available to advise me. I am very grateful for their patience, motivation, enthusiasm and immense knowledge.

I would also like to thank my committee members, Prof. Vladimir Kulyukin, Prof. Minghui Jiang and Prof. Stephen Clyde, for serving as my committee members, supporting me to pursue various projects and giving me insightful discussions about my research.

I gratefully acknowledge the funding sources that made my Ph.D. work possible. I was funded by the Utah Water Research Laboratory, the US Army Engineer Research and Development Center System (under contract number W912HZ-11-P-0338), National Science Foundation (under contract number EPS 1135482) and the Computer Science department of Utah State University.

Special thanks to my family. Words cannot express how grateful I am to my wife, my sister, my mother, and father for all of the sacrifices that you have made on my behalf. Your prayer for me was what sustained me thus far. At the end I would like to express appreciation to my beloved wife, Sule, who was always there to support me towards my goal, and my little daughter Ayse Bilge for her love and for making my life beatiful.

Ahmet Artu Yıldırım

CONTENTS

Page

ABSTRACT . . . iii

PUBLIC ABSTRACT . . . iv

ACKNOWLEDGMENTS . . . vi

LIST OF TABLES . . . ix

LIST OF FIGURES . . . x

ACRONYMS . . . xiii

CHAPTER

1 Introduction . . . 1

REFERENCES . . . 4

2 A Virtual Tile Approach to Raster-based Calculations of Large Digital Elevation Models in a Shared-Memory System . . . 5
- 2.1 Abstract . . . 5
- 2.2 Introduction . . . 5
- 2.3 Background . . . 7
- 2.4 Modified Planchon and Darboux Algorithm . . . 10
- 2.5 Virtual tile approach to the parallel raster-based algorithm . . . 11
- 2.6 Experimental Results . . . 20
- 2.7 Conclusion and Discussion . . . 28

REFERENCES . . . 29

3 Parallel Data Reduction Techniques for Big Datasets . . . 34
- 3.1 Abstract . . . 34
- 3.2 Introduction . . . 34
- 3.3 General Reduction Techniques . . . 36
- 3.4 Parallel Data Reduction Algorithms . . . 47
- 3.5 Parallel WaveCluster Algorithm . . . 51
- 3.6 Conclusion . . . 58

REFERENCES . . . 60

4 A comparative study of the parallel wavelet-based clustering algorithm on 3 dimensional dataset 65
4.1 Abstract 65
4.2 Introduction 65
4.3 Wavelet Transforms 66
4.4 Wavelet-based Clustering Algorithm 69
4.5 Parallel Wavelet-based Clustering Algorithms 73
4.6 Performance Evaluation 77
4.7 Conclusion 82

REFERENCES 82

5 HydroGate: A Service-oriented Gateway to HPC Resources to Facilitate Environmental Research 84
5.1 Abstract 84
5.2 Introduction 84
5.3 Related Works 88
5.4 Science Grid Gateway to HPC Centers 89
5.5 Architecture 90
5.6 Case Study 98
5.7 Future Research 103
5.8 Conclusion 103

REFERENCES 103

6 Conclusions 108

APPENDIX 110

CURRICULUM VITAE 118

LIST OF TABLES

Table Page

2.1 Properties of DEM datasets used in the experiments 23

2.2 Memory profile result of DEM3 in which expected maximum memory limit is 2000 MB 24

2.3 Average ($\overline{HRD}$) and standard deviation ($\sigma_{\overline{HRD}}$) of the difference of hit ratio between the VM with prefetching and without prefetching in percentage with respect to a number of compute threads for varying memory limits 27

2.4 DEM datasets for which Pitremove was successfully run using TauDEM 5.1, the parallel PD algorithm with Operating System's built-in virtual memory system (WVM) and tiled virtual memory (TVM) approach 27

3.1 Execution times and speed-up ratios for varying dataset sizes and number of processors (np) 55

3.2 Performance results of CUDA and CPU versions of Low-Frequency Component Extraction for scale level 1 on datasets (DS1 & DS2); times in microseconds 57

4.1 Datasets used in the experiments 78

4.2 Execution times (in seconds) of the parallel WaveCluster (PWC) algorithms using priority queue (PQ) and merge table (MT) approaches for wavelet levels (ρ) 1 and 2, and the parallel K-means algorithm with a varying number of processors (np) on Dataset1 80

5.1 Template variables with their definition 95

5.2 Variable naming conventions in the listings 99

LIST OF FIGURES

Figure Page

2.1 Partition of the DEM into tiles with one-cell border. Grid cells contain elevation values with nondraining cells (pits) in the tile detail depicted in gray. 13

2.2 Shared virtual memory management strategy 16

2.3 Tile distribution and sharing among the compute threads in pit-remove algorithm. Tiles are numbered with the owner compute thread number 16

2.4 (a)Tile page table maps each page to a region in a DEM dataset, data block in a physical memory or data stored in a swap file on the hard disk (b) Tile state diagram 17

2.5 Illustration of load balancing among the compute threads; In this example, $\sum |S| = 1000$, $\overline{|S|} = 250$ for 4 compute threads and 10 tiles 21

2.6 Time results in log10 seconds for DEM3 as a function of tile size (a) Total execution time (b) Waiting time (c) Algorithm time. (for varying number of compute threads up to 4 and prefetch count $\alpha = 1$) 21

2.7 Speedup Ratio of the parallel PD algorithm for DEM1, DEM2, DEM3 and DEM4 24

2.8 Hit rate results in percentage for virtual memory manager without prefetching (blue bars) and with prefetching (red bars) with respect to memory capacity and number of compute thread for DEM1 25

2.9 Compute thread loads with respect to a number of processed points (a) Without load balancing (b) With load balancing 26

3.1 Sampling illustrations on 2D domain (a) Simple random sampling (b) Systematic sampling 38

3.2 Search space of exhaustive search where $n = 5$ and $d = 3$ 40

3.3 Search space of branch and bound algorithm where $n = 5$ and $d = 2$. Sample traversed path is shown by dashed lines 41

3.4 Flow of the filter model 43

3.5 Flow of the wrapper model with cross validation 44

3.6 Illustration of Low-frequency Component in Art (a) Original image (b) Low-Frequency Component (scale level = 3) (c) Low-Frequency Component (scale level = 4) (d) Low-frequency Component in Art, mosaic painting of Abraham Lincoln by Leon Harmon, which is small in resolution and a coarser representation of the original image 46

3.7 Illustration of map-reduce model where the number of pieces, $M = 6$ . . . 49

3.8 WaveCluster algorithm multi-resolution property. (a) Original source dataset. (b) $\rho = 2$ and 6 clusters are detected. (c) $\rho = 3$ and 3 clusters are detected. (where ρ is the scale level) 53

3.9 Application of WaveCluster approach on fog datasets. (a) DS1 dataset. (b) $\rho = 3$ and K = 35 for DS1. (c) $\rho = 4$ and K = 15 and for DS1. (d) DS2 dataset. (e) $\rho = 3$ and K = 21 for DS2. (f) $\rho = 4$ and K = 6 for DS2. (where ρ is the scale level and K is the number of clusters detected) 58

4.1 Some mother wavelet functions: Haar, Daubechies 4, C3 Coiflet, S8 Symlet 68

4.2 Wavelet decomposition of a sample non-stationary signal. (a) Original signal (A_0). (b) Approximated coefficients at level 1. (A_1). (c) Approximated coefficients at level 2 (A_2). (d) Approximated coefficients at level 3 (A_3). (e) Approximated coefficients at level 4 (A_4). (f) Detailed coefficients at level 1 (D_1). (g) Detailed coefficients at level 2 (D_2). (h) Detailed coefficients at level 3 (D_3). (i) Detailed coefficients at level 4 (D_4). 70

4.3 Multiresolution property of the WaveCluster algorithm at level 2 on 1 dimensional data 71

4.4 Parallel WaveCluster Algorithm with ghost data illustration using 2 processors 75

4.5 Parallel WaveCluster Algorithms with varying input dataset files and wavelet levels 79

4.6 Speed-up Comparison between Parallel WaveCluster Algorithms and Parallel K-Means algorithm; where wavelet levels 1, 2 on Dataset1, $K = 20$ 81

5.1 A Cyberinfrastructure architecture in submitting job to HPC centers using CI server, CI storage system, HydroGate and multiple HPC centers; (1) CI user requests token from HydroGate and submit a job (2) CI server uploads the input package to CI storage server or creates the input package using existing files on the storage (3) HydroGate transfers the input package to a specified HPC center (4) HydroGate creates job script on the HPC center (5) After the job is completed, HydroGate creates the output package on the HPC center and transfers it to the CI storage 91

5.2 Security protocols among the software components in the architecture of HydroGate 92

5.3 HydroGate Job Description Language (HJDL) script and the corresponding batch scripts of PBS and SLURM 95

5.4 HydroGate Pooling Mechanism 97

ACRONYMS

CCL Connected Component Labeling algorithm

CI Cyberinfrastructure

DEM Grid digital elevation model

DFT Discrete Fourier Transform

DMS Distributed memory system

DWT Discrete Wavelet Transform

GPU Graphical processing unit

HPC High Performance Computing

JSON JavaScript Object Notation

MPI Message passing interface

OpenCL Open Computing Language

PDA Planchon and Darboux algorithm

PWC Parallel WaveCluster Algorithm

SCP Secure copy

SIMD Single instruction, multiple data

SMS Shared memory system

SSH Secure shell

CHAPTER 1

Introduction

There are inherent challenges to handle large datasets in storing, transmitting, processing and managing. This phenomenon plays important role in today's information technologies to extract useful information for innovation and decision-making in corporation and scientific research.

Environmental researchers, modelers and water managers commonly use grid digital elevation models (DEMs) to derive information related to topographically driven flow. Advances in technology for creating DEMs has increased their resolution and data size with the result that algorithms for processing them are frequently memory limited. One of the possible solutions to overcome this issue is virtual memory system that is the standard approach to working with data that is too big to fit in memory.

Data reduction techniques are used to save time and bandwidth in enabling the user to deal with larger datasets even in minimal resource environments, such as in desktop or small cluster systems. The key point of this process is to reduce the data without making it statistically indistinguishable from the original data, or at least to preserve the characteristics of the original dataset in the reduced representation at a desired level of accuracy. Because of the huge amounts of data involved, data reduction processes become the critical element of the data mining process on the quest to retrieve meaningful information from those datasets.

Researchers commonly employ clustering algorithms to map a set of objects into groups (i.e., clusters) based on their similar properties, such as spatial and temporal similarity. Sheikholeslami et al. [1] introduced a novel unsupervised clustering approach, called WaveCluster, utilizing a discrete wavelet transform (DWT) that enables data analysts to perform clustering in a multi-level fashion. This method has the ability to discover clusters with arbitrary shapes and can deal with outliers effectively.

A parallel computer is seen as a computer or a set of computers with multiple processors that can work together on solving a problem [2]. Parallel processing techniques are extensively used to divide the task into smaller subtasks and then executing them simultaneously to cope with memory limits and to decrease the execution time of that task.

Parallel computer architectures can be classified according to the memory sharing models. One class of parallel architectures is distributed memory systems (DMS) containing processors or nodes that are connected with each other by means of high-speed network infrastructure. In this model, each processor has its own memory. Traditionally, data in these architectures are shared among processors using message passing via the Message Passing Interface (MPI) [3] library, typically used to construct the communication among the processors by sending/receiving messages.

Another class of parallel architectures are shared memory systems (SMS). As the name implies, each processor communicates each other via dedicated shared memory over a data bus. Advantages of SMS over DMS the absence of the need to store duplicate memory and much faster communication for the processors. On the other hand, distributed memory architectures fully utilize the larger aggregate memory that allows us to solve larger problems which typically imply large memory needs.

Cyberinfrastructure provides scientists a set of middleware services to access software tools and data for collaborative scientific research over the high-speed network [4]. CI-Water project provides cyberinfrastructure to enhance access to data and high performance computing (HPC) for water resources modeling through data-driven simulation.

As the number of heterogeneous computing and storage systems increase, it necessitates the development of gateway services to access these resources in an unified way. We often need access to high performance computing (HPC) resources for running data and computationally intensive models without being an HPC expert. To remedy this, science web portals have been utilized that integrate scientific models, data analysis, and tools to visualize results via web browsers.

In Chapter 2, we present a user-level virtual memory system which is the standard approach for working with data that is too big to fit in memory in raster-based calculations. We see that there is a need to process large DEMs on desktop computers that are often limited in total memory. This is the primary problem addressed in this work. The application area for this portion of the study is the development of parallel terrain analysis algorithms that use multi-threading to take advantage of common multi-core processors [5] for greater efficiency.

In Chapter 3, we consider the problem of big data in terms of data reduction. The goal of this work is to reduce big data in size and complexity, so that the reduced form might be more manageable and computable using the resources in hand. This question leads us to the concept of data reduction techniques. In this chapter, we survey the general-purpose data reduction algorithms. A general discussion is provided on data reduction techniques that are utilized to analyze data efficiently using reduced representations of big data. Special emphasis is given to parallel wavelet-based multi-resolution data reduction techniques on distributed memory systems using MPI, and shared memory architectures on GPUs along with a demonstration of the improvement of performance and scalability.

Chapter 4 extends our parallel WaveCluster algorithm and introduces two new parallel wavelet-based clustering algorithms that address inefficient I/O operations over 2-dimensional datasets and performance degradation by benefiting collective MPI [3] I/O capabilities and efficient usage of data structures. Additionally, we cluster 3-dimensional datasets with the new algorithms.

Chapter 5 presents a science gateway service named HydroGate to access to heterogeneous HPC storage and computational resources. HydroGate abstracts away many details and complexities involved in the use of HPC systems including authentication, authorization, data and job management, and encourages the sharing of scientific applications and promotes collaboration among scientists. HydroGate has been developed as part of the CI-WATER project which aims to broaden the application of cyberinfrastructure (CI) and HPC techniques into the domain of integrated water resources modeling and is capable of dealing with various batch job schedulers and CI storage systems through a JSON-based language.

REFERENCES

[1] G. Sheikholeslami, S. Chatterjee, and A. Zhang, “Wavecluster: a wavelet-based clustering approach for spatial data in very large databases,” *The VLDB J.*, vol. 8, no. 3-4, pp. 289–304, 2000.

[2] P. S. Pacheco, *Parallel Programming with MPI.* Morgan Kaufmann, 1997.

[3] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, 1994.

[4] T. Hey and A. E. Trefethen, “Cyberinfrastructure for e-science,” *Science*, vol. 308, no. 5723, pp. 817–821, 2005. [Online]. Available: http://www.sciencemag.org/content/308/5723/817.abstract

[5] A. Marowka, “Back to thin-core massively parallel processors,” *Computer*, vol. 44, no. 12, pp. 49 –54, December 2011.

CHAPTER 2

A Virtual Tile Approach to Raster-based Calculations of Large Digital Elevation Models in a Shared-Memory System

2.1 Abstract

Grid digital elevation models (DEMs) are commonly used in hydrology to derive information related to topographically driven flow. Advances in technology for creating DEMs has increased their resolution and data size with the result that algorithms for processing them are frequently memory limited. This chapter presents a new approach to the management of memory in the parallel solution of hydrologic terrain processing using a user-level virtual memory system for shared-memory multithreaded systems. The method includes tailored virtual memory management of raster-based calculations for data sets that are larger than available memory and a novel order-of-calculations approach to parallel hydrologic terrain analysis applications. The method is illustrated for the pit filling algorithm used first in most hydrologic terrain analysis workflows.

2.2 Introduction

Grid digital elevation models (DEMs) are commonly used in hydrology to derive information related to topographically driven flow [1–3]. When dealing with large digital elevation model (DEM), datasets, computational efficiency and memory capacity become important considerations. Prior work in TauDEM [4] has advanced parallel methods for terrain processing using a message passing (MPI) approach that allows memory to be distributed across multiple processors in medium-sized cluster computers [5–7].

In desktop computers, virtual memory systems are the standard approach to working with data that is too big to fit in memory. Operating systems typically implement virtual memory using page files that hold on disk contents of memory. However repeated swapping (thrashing) occurs when these get large because the system has limited general capability to anticipate the pages needed next. Most operating systems implement the virtual machine in kernel that makes it difficult, sometimes impossible, to change its functionality and page replacement policy. This necessities the implementation of a user-level tailored virtual memory system to handle a programs' locality better for fine-grain control [8].

There is a need to process large DEMs on desktop computers that are often limited in total memory. This is the primary problem addressed in this work. It is also desirable to have parallel terrain analysis algorithms that use multi-threading to take advantage of common multi-core processors [9] for greater efficiency. This is a secondary consideration in this work.

This chapter presents a new approach to the management of memory and the parallel solution of the raster-based computations for shared-memory multithreaded systems, such as desktop computers. The contributions of this method in the context of parallelism are a tailored user-level tile based virtual memory manager for raster-based calculations for data sets that are larger than available memory and a novel order-of-calculations approach to parallel hydrology analysis applications.

We implemented a modified version of the Planchon and Darboux pit filling algorithm [10] as implemented in [4, 6] as an application of our tiled virtual memory manager and evaluated its effectiveness for pit removal in DEMs of varying size, with a varying number of operating system threads and memory capacity. The results demonstrate several benefits over the use of standard virtual memory approaches.

Furthermore, this study examines a load-balancing technique to minimize the idle times which might occur in case of uneven load between compute threads due to data variability. We used the GDAL library [11] to enable a wide range of raster file formats within the implemented memory manager. We implemented our memory manager using the Microsoft Windows 7 operating system as it is widely used for desktop Geographic Information System terrain analysis and enabling the processing of large DEMs on Windows systems was a goal of this work. The source code of the virtual memory project and the pit-filling algorithm can be found at `https://bitbucket.org/ahmetartu/hydrovtmm`.

The chapter is organized as follows. Section 2.3 provides background and literature review. Section 2.4 gives specifics of the modified Planchon and Darboux Algorithm used here. Section 2.5 describes the design of the multi-threaded tiled virtual memory manager for raster-based calculations that is contributed here. Section 2.6 gives performance results. Finally, we discuss conclusions based on the obtained results in Section 2.7.

2.3 Background

This review addresses three subjects that are needed to set the context for this work. First we review existing DEM pit filling approaches focusing on the Planchon and Darboux Algorithm [10] modified and used in this work. We then examine other efforts that use parallel methods for hydrologic terrain analysis and general methods for virtual memory management to enable the processing of large data sets that do not fit in physically available computer memory.

Pits, defined as grid cells or sets of grid cells completely surrounded by higher grid cells often occur during DEM production and are generally considered to be artifacts of the data collection process [12]. Drainage conditioning to remove pits is an important preprocessing step in the hydrologic analysis of DEMs, and is representative of a broad class of raster-based algorithms (e.g. [1, 13]) designed to determine topographically driven flow. Once drainage conditioning has been performed, a DEM that has no pits is referred to as being hydrologically conditioned. The most common approach to drainage conditioning is pit filling, whereby pit grid cells are raised to a level where they are at a minimum equal to the lowest surrounding grid cell and can drain. A well-known effective pit filling algorithm is described by Planchon and Darboux [10]. Pit removal using the Planchon and Darboux (PD) algorithm is one of the multiple hydrologic terrain analysis functions in the TauDEM package. Time-complexity of the direct implementation of the PD algorithm is reported to grow with the number of cells N in DEM as $O(N^{1.5})$ [10]. Planchon et al. also provided an improved implementation that embedded a recursive dry upward cell function that was reported to achieve a time-complexity of $O(N^{1.2})$ [10]. Alternative pit filling algorithms, some claiming better efficiency have been presented by others [14–16].

The Planchon and Darboux (PD) approach fills pits by covering the whole surface with a layer of "water" up to a level greater than or equal to the highest point in the DEM, then removes the excess water in an iterative manner. Doing so, the algorithm naturally leaves the water in the depressions at the height of their outlet. Let $Z \in \mathbb{R}^2$ be the set of input elevation points (i.e., the input DEM with size m, where each member is an elevation point x_i, $1 \leq i \leq m$) and let $W \in \mathbb{R}^2$ be the output DEM consisting of "filled" elevation points y_i. The goal of the PD algorithm is to increase each elevation point x_i with a minimal difference of elevation. The PD algorithm initializes all grid cells to a large value (greater than the highest point in the domain). It then uses iterative scans across the domain to lower elevation values to the lowest elevation greater than or equal to the original terrain hydrologically conditioned so that they drain to one of their neighbors.

Tarboton et al. [4,6] implemented a parallel version of the PD algorithm. This adopted a distributed memory domain partitioning approach to parallelism and divided the domain into horizontal stripes, one stripe per parallel process. The PD algorithm was applied separately to each stripe in parallel, with a step to exchange information across stripe boundaries at the end of each iteration so as to ensure convergence to the same global solution as obtained by a serial implementation. The original PD algorithm and the Wallis et al. parallel implementation visit each grid cell on each iteration. Scans of the grid cycle through all eight possible combinations of row and column scan orders. PD also offered an improved implementation that used a recursive dry upward tree search each time a cell was set to the original elevation to enhance efficiency. However each iterative pass across the DEM still examined each grid cell.

Subsequent to the Wallis et al. [6] work, the TauDEM team [4] identified the visiting of each grid cell on each iteration as an inefficiency and developed a stack based approach whereby unresolved grid cells are placed on a stack on the first scan, then removed from the first stack on each subsequent scan and placed on a second stack. Stacks are then switched.

This limits the scanning to two directions rather than eight, but was found to result in a speedup of a factor of 2 for small datasets and 4.3 for a modestly large 1.5 GB dataset in comparison to the eight combination full grid scanning. The benefits of the stack thus seem to outweigh the inefficiency of fewer scan directions. The TauDEM team did not evaluate the recursive dry upward approach of the improved PD algorithm. Recursive methods use the system stack to expand system memory, posing a challenge for memory management in large data computations. They also pose a challenge for a domain partitioned parallel approach as cross partition calls are less predictable. They are also hard to implement on a stack as they would require additional code to track the stack position of each grid cell and to handle changing the order in which grid cells on the stack are processed. The two direction stack based modified PD algorithm was incorporated into the publicly released version of TauDEM [4] that was the starting point for this work. The focus of this chapter is on virtual memory management for large DEM data, and the modified PD algorithm as used by TauDEM [4, 6] is used as an example to illustrate a general approach.

Beyond pit filling several parallel computing technologies have been employed in the implementation of the hydrologic algorithms to achieve higher performance by effectively utilizing available processors in the system including MPI for distributed memory architectures [5, 6, 17], OpenMP [18] or low-level standard threading library for multi-threaded shared memory architectures, and NVIDIA CUDA for general-purpose computing on graphics processing units (GPUs) [19, 20]. Xu et al. [18] present a grid-associated algorithm to improve the performance of a D8 algorithm [21] via OpenMP technology which is a thread-level standard of parallel programming. CUDA algorithms for drainage network determination are presented by Ortega and Rueda [20], achieving up to 8x speed-up improvement with respect to corresponding CPU implementation. Do et al. introduced a parallel algorithm to compute the global flow accumulation in a DEM using MPI [17] where hierarchical catchment basins are computed by means of a parallel spanning tree algorithm to model the flow of water. Each processor computes a local flow direction graph using the *D8* flow routing model.

There have been a number of general approaches to better manage virtual memory. The Tempest [22] interface was proposed for customizing the memory policies of a given application on parallel computers. Translation lookaside buffer (TLB) is a limited memory cache that is utilized for efficient memory address translation. However, TLB misses are also common in memory-hungry applications such as databases and in-memory caches. To eliminate these, Basu et al. [23] proposed mapping part of a process' linear virtual address space with a direct segment. Huang et al. [24] introduced a power-aware virtual memory system to reduce the energy consumed by the memory for data-centric applications.

Software programs may implement their own level of memory management to overcome the inefficiencies of system-level virtual memory. For example, ArcGIS reportedly uses quad-tree tiling to organize on disk file storage in a way that facilitated better virtual memory management [25]. However full details of their implementation are not available.

2.4 Modified Planchon and Darboux Algorithm

We employed a modified version of the PD algorithm, as used in TauDEM [4], in this study. The PD algorithm has two phases, initialization (Algorithm 1) and filling (Algorithm 2). Initialization starts by allocating memory space for elevation data, W. Then, if the cell is located at the edge, the algorithm assigns the value of cell in Z to each cell of W with the same cell location, otherwise a maximum number is assigned. Here edge is defined to include cells at the edge of the DEM or internal cells adjacent to cells with no data values, or cells marked as internally draining and not to be filled in a separate input file. Thus, it is assumed that water may drain from the edges of the input DEM or into internally draining or no data cells (because data may not fit neatly into the rectangular DEM domain). We employ stacks to hold indexes of the cells not yet solved. The indexes of the cells having excess "water" in W are pushed onto the first stack, (S_1), which is used later in the filling phase. An upper bound on the number of elements the stack needs to hold is the number of internal (non-edge) grid cells in the DEM.

In the filling phase (Algorithm 2), the cell values of W are decreased in an iterative manner until no cell value is changed. The procedure pulls an unresolved grid cell from

Algorithm 1 Initialisation Phase of Stack-based Planchon and Darboux Algorithm

```
Require: Input DEM $Z$
Ensure: Elevation data $W$, stack $S_1$
 1: procedure initializePlanchon($Z$)
 2:     for Cell number $i \leftarrow 1$ to $|Z|$ do
 3:         if $c_i \in Z$ is on the edge then
 4:             $W(i) \leftarrow Z(i)$
 5:         else
 6:             $W(i) \leftarrow +\infty$
 7:             $S_1 \leftarrow$ push($S_1$, $i$);
 8:         end if
 9:     end for
10:     return $W$, $S_1$
11: end procedure
```

stack, S_1. The operation of $\min N_8(W(i))$ denotes finding the minimum value among 8 neighbours of the cell $W(i)$. In Line 6, the algorithm checks whether the value of $Z(i)$ is greater than or equal to the minimum elevation of its 8 neighbours plus a small parameter ϵ to enforce strictly positive slopes. This parameter may be 0, and is defaulted to 0 where minimally altered hydrologically conditioned surfaces are desired. If *true*, this implies that the cell drains, so the value of $Z(i)$ is assigned to the cell $W(i)$. Otherwise the value $N_{min} + \epsilon$ is assigned to the cell, lowering its value to the elevation at which it drains (line 10). Cells that are set to their original elevation value (line 7) do not need to be revisited so are not placed on the stack S_2. Cells lowered to a neighbor value (plus ϵ) may need to be revisited in a later iteration so are pushed to the stack S_2. The stacks decrease in size progressively as cells are processed. After each iteration, in Line 16, the indexes pointing to the stacks are swapped.

The original PD algorithm did not employ a stack as described above. Rather at each iteration it scanned the entire DEM examining for each cell whether $W(i) > Z(i)$, and if true examined neighbor elevations and applied the excess water removal logic.

2.5 Virtual tile approach to the parallel raster-based algorithm

We implemented a parallel modified PD algorithm for a multi-threaded shared memory

Algorithm 2 Filling Phase of Planchon and Darboux Algorithm

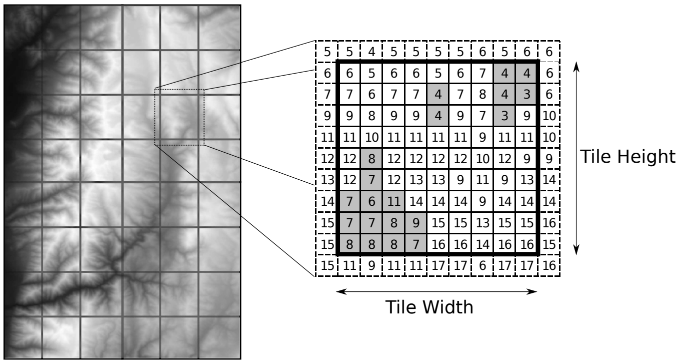
Require: Input DEM Z, elevation data W, stacks S_1 and S_2, ϵ-descending path parameter, i denotes to the visited cell index, $\min N_8(W(i))$ denotes to minimum elevation of the neighbouring cells of cell i

Ensure: Elevation data W

```
procedure fillPlanchon($Z$, $W$, $S_1$, $S_2, \epsilon$)
    do
        for all cell number $i$ in $S_1$ do
            if $W(i) > Z(i)$ then
                $N_{min} \leftarrow \min N_8(W(i))$
                if $Z(i) \geq N_{min} + \epsilon$ then
                    $W(i) \leftarrow Z(i)$
                else
                    if $W(i) > N_{min} + \epsilon$ then
                        $W(i) \leftarrow N_{min} + \epsilon$
                    end if
                    $S_2 \leftarrow push(S_2, i)$
                end if
            end if
        end for
        call swapStacks($S_1$, $S_2$)
    while any cell $c \in W$ is changed
    return $W$
end procedure
```

system with limited memory to process large DEMs. The input DEM is divided into a number of generally square $q \times q$ tiles where q is tile size that can be considered a generalized domain decomposition approach when compared to the TauDEM stripe approach. Tiles may be rectangular along the edges to accommodate domain sizes that are not multiples of q. Each tile is processed individually by one thread. Tiles are distributed among the threads with respect to number of cells in the stacks using a load balancing mechanism. In order to process big DEMs, which might be larger than available physical memory, we adopt a tile-based user-level virtual memory management system. The memory system swaps out tiles and their related data to hard-disk to free the memory space automatically when the predefined memory limit is reached. Tiles are chosen for swapping out by the memory manager according to a least recently used rule.

In our algorithm, we use a single input/output (I/O) thread and multiple compute



5	5	4	5	5	5	5	6	5	6	6
6	6	5	6	6	5	6	7	4	4	6
7	7	6	7	7	4	7	8	4	3	6
9	9	8	9	9	4	9	7	3	9	10
11	11	10	11	11	11	11	9	11	11	10
12	12	8	12	12	12	12	10	12	9	9
13	12	7	12	13	13	9	11	9	13	14
14	7	6	11	14	14	14	9	14	14	14
15	7	7	8	9	15	15	13	15	15	16
15	8	8	8	7	16	16	14	16	16	15
15	11	9	11	11	17	17	6	17	17	16

Figure 2.1: Partition of the DEM into tiles with one-cell border. Grid cells contain elevation values with nondraining cells (pits) in the tile detail depicted in gray.

threads. *Compute threads* are responsible for the execution of the PD algorithm while the *I/O thread* is used to avoid performance bottlenecks due to overlapping disk I/O and compute operations. The I/O thread services all compute threads. The main thread is regarded as a compute thread and is responsible for spawning other compute threads and the I/O thread. Memory address space is shared among all threads, alleviating the overhead associated with interprocess communication in shared memory systems.

We adopt data parallelism among compute threads where each thread works on one tile at a time and executes the same algorithm (Algorithm 3). Each tile is a rectangular subset of the DEM that consists of primary data and a one-cell border that overlaps into the primary data from adjacent tiles as illustrated in Figure 2.5, that is used to facilitate transfer of information between tiles. The primary data of all tiles completely covers the domain without overlap. A tile page, T_k, has the corresponding subset of elevation data, W_k, input DEM data, Z_k and two stacks. Tile pages are indexed by tile page number k that define the page uniquely in the address space. The *initializePlanchon* and *fillPlanchon* functions change values only within the primary data, but refer to values from adjacent grid cells using the edge data held locally as part of each tile page.

After the initialization pass (Lines 3-5) and each filling pass (Lines 9-13) the resulting elevation values are hydrologically conditioned within the local context of each tile, but

Algorithm 3 Multithreaded PD algorithm

```
Require: Input DEM $Z$, tile size $TS$, memory limit $ML$, number of compute threads $NC$,
    each with start index, $SI_i$ and end index, $EI_i$, referencing the tiles it processes.
Ensure: Output DEM $W$ (final elevation data)
procedure PARALLELPLANCHON($Z$, $TS$, $ML$, $NC$)
    for all Compute Threads $i$ in parallel do
        for $k \leftarrow SI_i, EI_i$ do
            call initializePlanchon($T_k$)
        end for
        call exchangeBorders() with barrier()
        call performLoadBalancing() with barrier()
        do
            for $k \leftarrow SI_i, EI_i$ do
                if $T_k$ is not finished then
                    call fillPlanchon($T_k$)
                end if
            end for
            call exchangeBorders() with barrier()
            call performLoadBalancing() with barrier()
        while any cell elevation in $W$ is changed
    end for
    call barrier()
    call writeOutputDEM()
end procedure
```

may need to be modified as each tile is juxtaposed with its surrounding tiles. Hence, local elevation data on the edges must be updated from surrounding tiles. Neighboring threads exchange the border of the tiles. Thus, we implement a barrier to synchronize all threads at this point to make sure all the threads are at the same position. Barriers are shown in Algorithm 3. Although barriers are desired to be avoided to fully exploit the parallelism in algorithm design, in order to circumvent race conditions it is required. Then the *exchangeBorders* function is called for each tile by each owner thread. This accesses the data from each bordering tile and updates its edge data with the corresponding primary data from the adjacent tile.

Remaining cells to be processed at the next iteration can be determined by the number of cells in the stack. This determines the require processing time for each tile. After locally filling the pits of the tiles, imbalance might occur in which some threads finish the local filling before other threads. To avoid this inefficiency, we implement load balancing functionality that distributes the tiles to the threads evenly based on the number of cells in the stacks. Before distributing the tiles, all threads must wait for the main thread that executes the *loadBalancing* function. *loadBalancing* examines the size of the unresolved stack in each tile and adjusts the tiles assigned to each thread by setting SI_i and EI_i prior to the next iteration. More discussion about the load balancing can be found in *Load balancing* section.

A virtual memory management strategy (Figure 2.5) was developed to support implementation of this algorithm in systems where the physical memory is limited. A memory manager is responsible for performing swapping operations as memory reaches its maximum capacity. In system-level virtual memory, each program on the operating system has its own address space that is consisting of memory chunks (pages) [26]. In our approach, each thread is provided a set of tiles on the DEM file and its associated data whose memory is managed by the virtual memory through a well-defined software interface.

All memory management functionality is performed by the I/O thread, using the GDAL library [11]) so that processing in the compute threads may proceed in parallel with I/O swapping of tiles to disk. One of the reasons for a single I/O thread is that GDAL did not support parallel I/O. However even with a parallel I/O library on the PC platforms we are targeting, the parallel I/O capability is limited and a single I/O thread avoids cross thread I/O bottlenecks.

The main thread initializes the memory manager and creates the tile page table. Tile page table is in shared memory accessible to the compute threads and I/O thread. The tile page table contains tile data structures mainly including tile state, memory addresses of original and conditioned DEM data and stack data. We implemented a tailored stack data structure where the data are kept in memory in a contiguous fashion in order to serialize the writing and reading of stack memory content efficiently. The stack data structure contains a dynamically increasing heap array (using the *realloc* utility), and size and capacity variables.

Figure 2.5 illustrates how tiles are assigned to compute threads. Note that the *exchangeBorders* function results in tiles assigned to one thread being accessed by adjacent

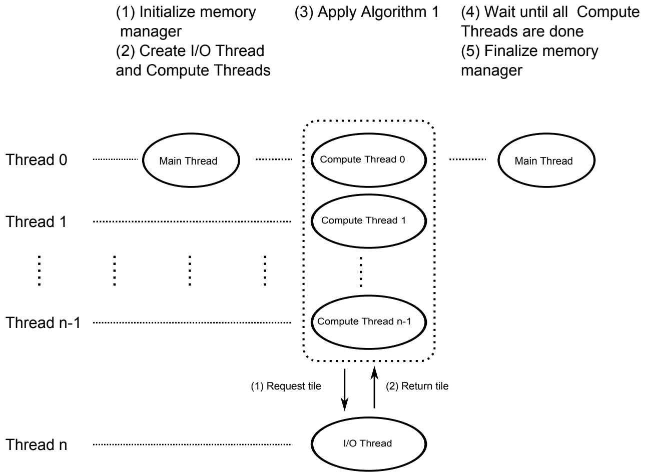


Figure 2.2: Shared virtual memory management strategy

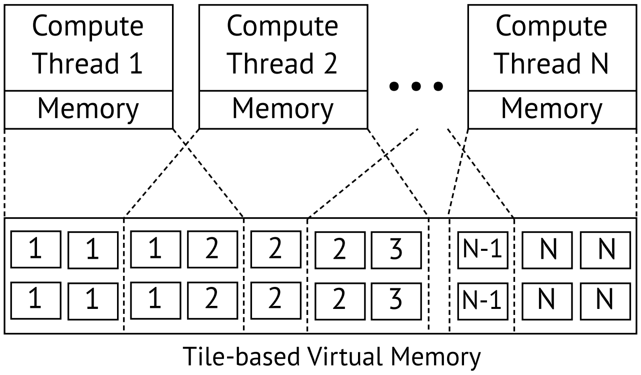
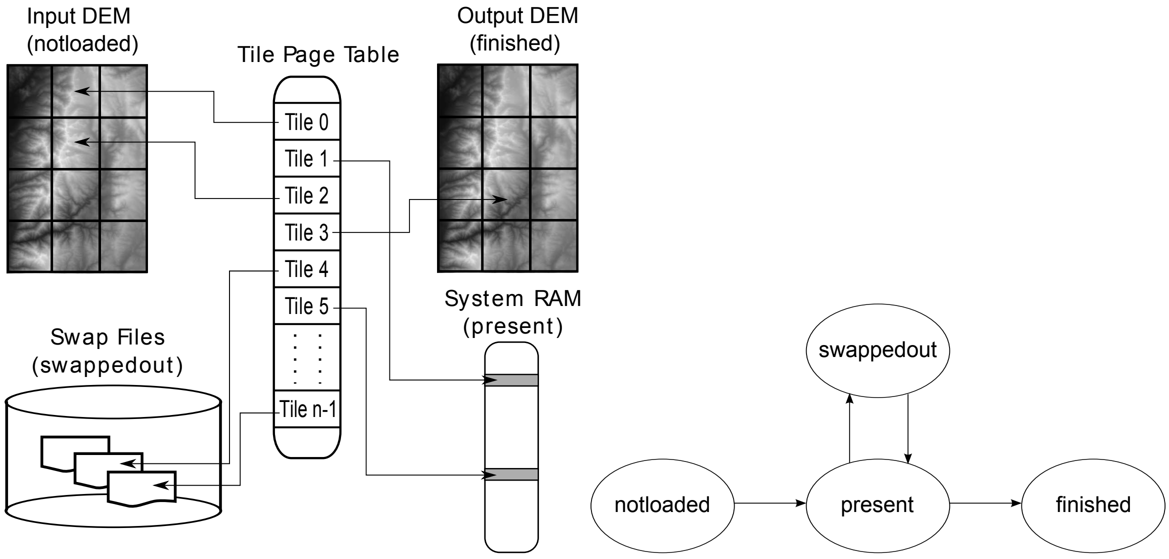


Figure 2.3: Tile distribution and sharing among the compute threads in pit-remove algorithm. Tiles are numbered with the owner compute thread number

threads. This is a non-locking read access so that contention that may occur when a thread attempts to lock a memory block that is owned by another thread [27] can be avoided. Data sharing in shared memory systems is one source of performance degradation and should be avoided where possible, thus the *exchangeBorders* function is called only at infrequent intervals.

Each tile page has state, μ_k where $\mu_k = \{notloaded, present, swappedout, finished\}$: *notloaded*, indicating that the page has not yet been requested by any compute thread;



(a) (b)

Figure 2.4: (a)Tile page table maps each page to a region in a DEM dataset, data block in a physical memory or data stored in a swap file on the hard disk (b) Tile state diagram

present, the page is resident in the memory; *swappedout*, the page is swapped out to the disk; or *finished*, all elevation points in the tile have been hydrologically conditioned and no more pit-filling is required. The condition *finished* is set on a tile when its stack size becomes 0. The algorithm iterates until all the tiles have achieved the state of *finished*, or until all stacks on all processes remain unchanged. The page table is illustrated in Figure 2.4(a) with tile states illustrated in Figure 2.4(b).

A distinguishing property in our model, as compared to most GIS software, e.g., ArgGIS, is that a tile page does not consist only of raster data, but also contains algorithm-specific data such as the input, output and associated data that comprise the memory space of the program.

A *message queue* is used for interprocess communication between compute threads and the I/O thread. The I/O thread constantly retrieves the messages in the message loop in a First In, First Out (FIFO) manner. The message passing between the compute threads and I/O thread is performed using *PostThreadMessage* and *PeekMessage* of Win32 functions. If a compute thread requests a tile retrieval, it sends a message to the I/O thread and waits until the page is returned. If the page is present in memory, the operation is completed without waiting. Each tile has a time stamp variable to keep track of the last request time. The time stamp is updated by the I/O thread when requested by any compute thread. Using this time stamp, the memory manager swaps out the oldest tiles according to the Least Recently Used (LRU) replacement algorithm.

The granularity of the virtual memory manager is an important parameter that affects the system's performance. A small page/tile size can result in larger page tables, which might cause the system to spend its CPU time mostly in I/O operations. On the other hand, too large a tile size can hinder the benefit of using memory management and memory usage can exceed the memory limit by a factor of tile page size.

Thrashing is a well-known phenomenon encountered when a process idles excessively waiting for the referenced page to be loaded by the operating system. In this algorithm, the effect of thrashing is alleviated by pre-fetching tiles using a pre-specified pattern based on knowledge of the sequence in which the algorithm accesses tiles. The prefetching technique has been applied for sequential programs in which predicting patterns of program execution and data access in the near future improves the system efficiency [28]. In the implemented virtual memory system, when a tile is requested, the memory manager caches the next α tiles (from the $flist$ array) in this pattern asynchronously. The goal of this prefetching technique is to increase the *hit rate* of tiles found in present memory when requested. We define the hit rate as the number of tile references whose memory present in RAM divided by the total number of tile references.

2.5.1 API to access to the virtual memory manager

The user-level virtual memory is designed to be used as a general-purpose virtual memory for raster-based computations to execute on a single machine with limited memory through a well-defined interface. The raster-based algorithm accesses the VM through a well-defined interface so as to enable the potential for application with other raster-based algorithms. The API is explained below.

- $initializeMemoryManager(S)$: The function accepts array S that contains the setting values of the virtual memory manager including maximum allowable memory, and the DEM file accessed to determine extent and tile sizes. Using the parameter S, the virtual memory is initialized and memory allocated for resources such as the messaging system and the data structures used by the main thread.

- $getTile(tid, flist[])$: A compute thread requests the tile with uniquely-defined tile id *tid* from VM. A tile can be owned by only one compute thread for writing at a time to avoid race condition error. During the tile-retrieval process with the *getTile* function, the tile's reference count is incremented by 1. The virtual memory manager prevents access to a shared resource from other threads using the *EnterCriticalSection* and *LeaveCriticalSection* Win32 functions. As an advantage with respect to most operating system level virtual memory systems, the tiles to be prefetched are determined by the algorithm from the $flist$ array containing tile ids. Prefetching occurs in parallel. The next tile on the list is prefetched by the virtual memory manager. After loading each tile, the virtual memory manager checks the memory limit and if this has been exceeded swaps out the least-recently accessed tile not in use. This strategy is a heuristic attempt to ensure that the next tiles needed by the compute threads will be in memory when they are requested.

- $unlockTile(tid)$: After the compute thread is finished with the tile, the tile is explicitly released by the owner thread. The virtual memory manager decrements the tile's reference count where the tile is requested with *getTile* function. When the tile's reference count reaches 0, the tile is returned to the memory pool. Then, virtual memory manager decides to keep the tile in memory or swaps out to the disk based on the page replacement algorithm to free memory space.

- $saveTile(tid)$: If the tile is finished completely by the compute thread, the result is saved to the disk via *saveTile* function and then the tile is marked as *finished* state which is the final state of a tile.

- $performLoadBalancing(tiles[], ctid)$: Given a set of tile ids $tiles[]$ and compute thread id *ctid*, *performLoadBalancing* returns the working set of tiles for the compute thread *ctid*. The virtual memory manager distributes the tiles evenly among the compute threads. Load balancing is a program option that can be disabled if desired.

- $finalizeMemoryManager()$: The memory resources used by the virtual memory manager are freed.

2.5.2 Load balancing

The variable nature of the topography that the algorithm operates on gives rise to variability in the number of cells on the unresolved stack for each tile at each iteration. Load balancing has been applied for concurrent MPI execution streams that exhibit irregular structure and dynamic load patterns [29] and to ensure that the load on each core is proportional to its computing power in multi-core architectures [30]. We developed a load balancing mechanism that is specifically tailored for the parallel modified PD algorithm. The modified PD algorithm applied to a tile iteratively processes unresolved cells whose indices are stored on a stack. The size of the stack on a tile indicates the work load per tile. The load balancing rule distributes the tiles based on tile stack size ($|S|$). Figure 2.5.2 illustrates the load balancing mechanism. Let T be the set of all tiles and CT be the set of compute threads. Then $\sum |S| = \sum_{i=1}^{|T|} |S|_i$ is the sum of stack sizes, and $\overline{|S|} = \sum |S| \, / \, |CT|$ is used as the target number of elements in the stacks per compute thread. The load balancing algorithm assigns consecutive tiles to each compute thread to take advantage of CPU caches through the principle of memory locality. The number of assigned cells might be greater than or equal to $\overline{|S|}$ as the tile granularity affects the distribution of the load.

2.6 Experimental Results

Numerical experiments were performed to evaluate the virtual memory manager using five DEM datasets obtained from the National Elevation Dataset web site (`http://ned.usgs.gov`) (Table 2.1). These ranged from a relatively small test dataset to a dataset that

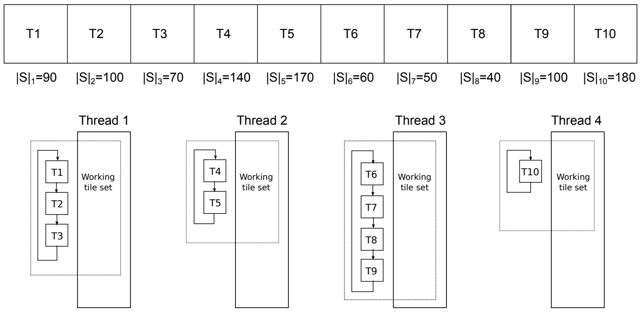
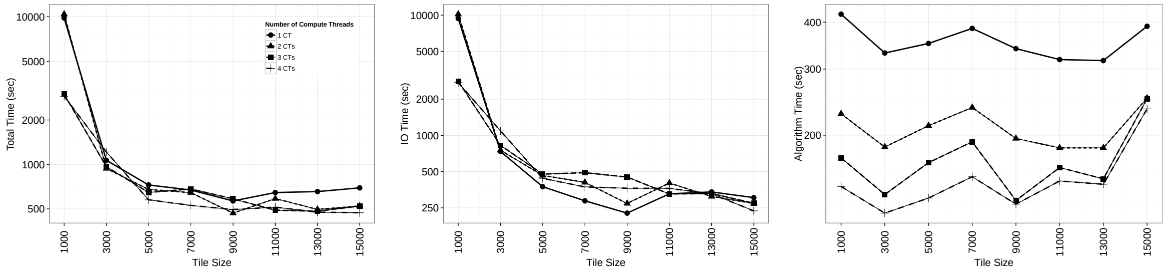


Figure 2.5: Illustration of load balancing among the compute threads; In this example, $\sum |S| = 1000$, $\overline{|S|} = 250$ for 4 compute threads and 10 tiles



(a) (b) (c)

Figure 2.6: Time results in log10 seconds for DEM3 as a function of tile size (a) Total execution time (b) Waiting time (c) Algorithm time. (for varying number of compute threads up to 4 and prefetch count $\alpha = 1$)

exceeded the physical memory capacity of the test computer. Tile sizes, number of compute threads and memory capacity were all varied in the runs. The experiments were conducted on a machine equipped with an Intel Core I7 processor with 3.40 GHz and configured with either 4 GB or 16 GB of RAM. We used the 64-bit version of Microsoft Windows 7. Although not popular in HPC systems, Windows is widely used for desktop Geographic Information System terrain analysis and enabling the processing of large DEMs on these systems was a goal of this work.

Tile size is one of the most important parameters impacting the performance of the parallel PD algorithm. To examine the impact of tile size, the parallel PD algorithm was executed with shared virtual memory management enabled and a varying number of compute threads up to 4 using DEM3 with a range of tile sizes from 1000 to 15000 on a machine with 16 GB RAM. The maximum memory capacity for this experiment was set to half memory size of DEM3 dataset for these runs. Maximum memory is the size of the allocated memory for the entire page structure. In this experiment it was purposely set less than the total memory required by the algorithm, around 4 times the file size, or 16 GB to induce swapping and exercise the user level virtual memory management.

The total execution time is a combination of *Algorithm time* and I/O *Waiting time* (i.e., the elapsed time between the time a tile is requested and the completion of that request by the I/O thread). In Figure 2.6(a), the total execution time is seen to decrease sharply when more than 2 compute threads are utilized for a tile size of 1000. Parallelization is seen here to be effective in overlapping I/O operations with computation by the I/O thread. We observe that total execution time continues to decrease as the tile size increases up to 9000, in which case the minimum time is achieved with 468 seconds when 2 compute threads are used. After a tile size of 9000, the total execution time begins to increase due to a *thrashing* effect because the memory manager performs an excessive number of disk swapping operations. The waiting time accounts for the majority of this total execution time, indicating an increased I/O overhead cost associated with small tile sizes and correspondingly more distinct disk accesses (Figure 2.6(b)). The algorithm time is also dependent on tile size, reflecting the additional computation involved with edge exchanges for smaller tile sizes, although this effect is significantly smaller than the I/O effect quantified by the wait time (Figure 2.6(c)). These experiments show that the tile size can have a dramatic effect on performance of the program but this effect diminishes as the tile size increases, and then becomes more prominent due to better data locality. This experiment suggests limiting tile sizes to approximately 9000 for this system for the best performance using 3 compute threads. Multithreading also helps decrease the algorithm time. However, we observe that as the memory limit decreases, the performance benefit of multithreading diminishes because the algorithm becomes more I/O intensive. Because I/O overhead increases as tile sizes decrease, experiments for tile sizes less than 1000 were not performed. Conversely, note that when the tile size is 15000, the algorithm execution times with 3 and 4 compute threads approach the algorithm time with 2 compute threads because of the unfair load distribution among the threads. We also performed the experiment for tile size 24000 to investigate the effect of the biggest tile size on the virtual memory manager but the experiments failed due to insufficient memory.

Table 2.1: Properties of DEM datasets used in the experiments

Label	Location	Domain Size	File Size
DEM1	GSL Basin (100 m cells)	4045x7402	117 MB
DEM2	GSL Basin (27.3 m cells)	14849x27174	1.61 GB
DEM3	Boise River Basin (10 m cells)	24856x41000	3.98 GB
DEM4	Wasatch Front (10 m cells)	34649x44611	6.05 GB
DEM5	Chesapeake Bay (10 m cells)	45056x49152	8.65 GB

Memory use during the run of the parallel PD algorithm to evaluate the effect of tile size with the DEM3 dataset was profiled to find the average and maximum memory used (Table 2.2) where the memory limit was set to 2000 MB. In practice, because memory is allocated in tile size increments for each thread, the maximum memory usage is larger than the target memory capacity parameter. The amount of this excess varies with tile size. When we set the tile size to 24000, the algorithm is failed to allocate memory space because the test machine doesn't have sufficient memory space and, therefore, we completely lost the benefit of the virtual memory system. The experiments show that the amount by which actual memory use exceeds target memory capacity increases with tile size. Maximum memory use exceeding the target parameter is a disadvantage of this virtual memory management approach as it limits the control on memory use by the program, resulting in potential for operating system virtual memory to kick in or for the program to crash due to insufficient memory.

In a second experiment we evaluated the parallel performance of the algorithm on DEM datasets that fit within RAM and where swapping was not needed. With the hardware reconfigured to 16 GB RAM we tested the speed on the first four DEM datasets. Although we achieved best time result with tile size 9000, tile size was set to 1000. Speed up ratios for this test (Figure 2.6) show a beneficial speed-up relative to the sequential modified PD

Table 2.2: Memory profile result of DEM3 in which expected maximum memory limit is 2000 MB

Tile Size (q)	Avg. Memory Usage	Max. Achieved Memory Usage
1000	1828 MB	2046 MB
3000	1851 MB	2396 MB
5000	1780 MB	3477 MB
7000	2089 MB	4955 MB
9000	2249 MB	6392 MB
11000	2438 MB	9253 MB
13000	3308 MB	12242 MB
15000	3456 MB	13683 MB
24000	×	×

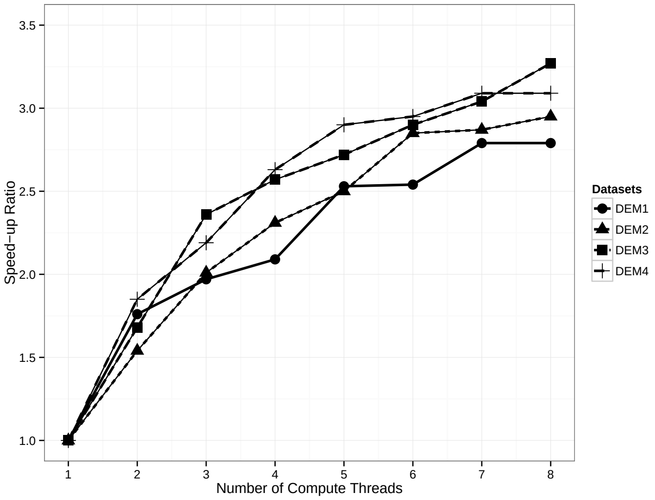


Figure 2.7: Speedup Ratio of the parallel PD algorithm for DEM1, DEM2, DEM3 and DEM4

algorithm with one I/O thread. A speed-up ratio of around 3 was obtained with 8 compute threads. Speed-up ratios are generally slightly more for the larger datasets, though this pattern is not consistent across the full range of the number of threads tested, the variability presumably being due to differences in the data. The machine used in the study has four cores, but the experiments were expanded to 8 compute threads to observe the effect of the hyper-threading feature of the processor, which improves processor performance by enabling the execution of pipeline-scaled interleaving of multiple threads on a single core, resulting in a modest performance benefit.

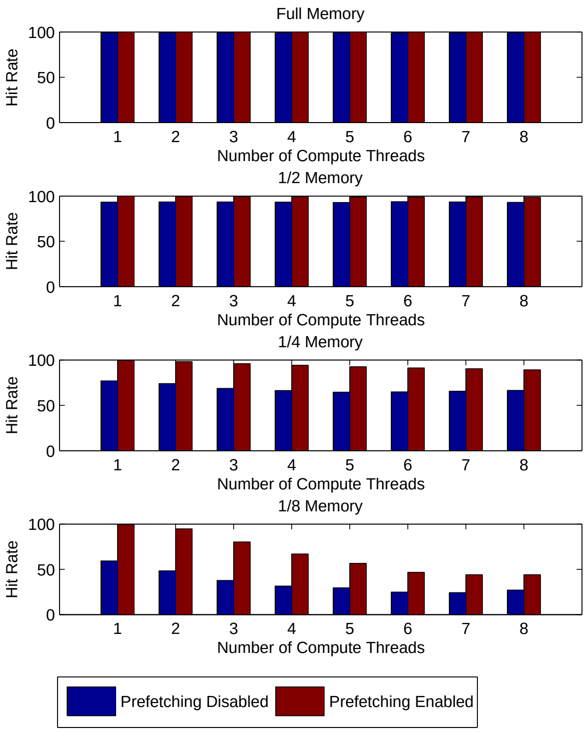
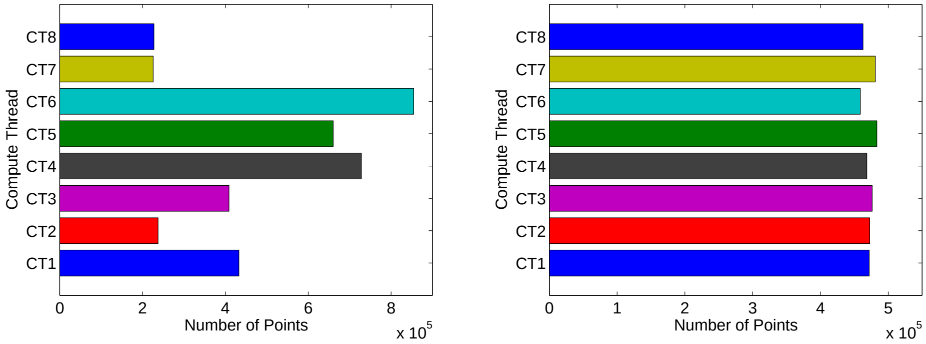


Figure 2.8: Hit rate results in percentage for virtual memory manager without prefetching (blue bars) and with prefetching (red bars) with respect to memory capacity and number of compute thread for DEM1

Next, the prefetching capability of the shared virtual memory manager module, described at the end of Section 3, was evaluated (Figure 2.6). The experiments were conducted with respect to the fraction of tiles that can be *present* in the available system memory. For example, 1/4 memory means there can be at most 1/4 of total number of tiles present in memory at the same time. The experiments show that as the size of the memory limit allowed to be used by the application decreases, the hit rate (i.e., the fraction of tiles already in memory when requested) also decreases due to an increasing number of swapping out operations. This may lead to higher loads on the I/O thread and disk. By contrast, because the implemented system uses one I/O thread, prefetching more than one tile ahead may also result in other threads experiencing higher I/O times. This I/O bottleneck is exacerbated



(a) (b)

Figure 2.9: Compute thread loads with respect to a number of processed points (a) Without load balancing (b) With load balancing

in the presence of multiple compute threads and low memory limits.

Table 2.3 shows average ($\overline{HRD}$) and standard deviation ($\sigma_{\overline{HRD}}$) values of the difference of hit rates (HRD) between the VM with prefetching (P) and without prefetching (WP) in percentage with respect to memory capacity (M) and a number of compute threads (CT). The values are computed as follows:

$$HRD(CT, M) = (HR_{CT,M,P} - HR_{CT,M,WP}) \tag{2.1}$$

$$\overline{HRD}(M) = \frac{\sum_{i=1}^{CT} HRD_{i,M}}{CT} \tag{2.2}$$

$$\sigma_{\overline{HRD}}(M) = \sqrt{\frac{\sum_{i=1}^{CT} (HRD(i,M) - \overline{HRD}(M))^2}{CT-1}} \tag{2.3}$$

As the virtual memory limit decreases, the average of difference of hit rate values increase that shows the benefit of using prefetching for low memory limit. However, the variation of hit ratios also increase. This is because increasing compute threads affects the hit ratio more under lower memory limits when compared to higher memory limits.

Table 2.3: Average ($\overline{HRD}$) and standard deviation ($\sigma_{\overline{HRD}}$) of the difference of hit ratio between the VM with prefetching and without prefetching in percentage with respect to a number of compute threads for varying memory limits

Statistics	Full Mem.	1/2 Mem.	1/4 Mem.	1/8 Mem.
Average	0.784	5.834	22.664	31.279
Std. Dev.	0.015	0.450	9.355	11.301

Table 2.4: DEM datasets for which Pitremove was successfully run using TauDEM 5.1, the parallel PD algorithm with Operating System's built-in virtual memory system (WVM) and tiled virtual memory (TVM) approach

	DEM1 (0.1 GB)	DEM2 (2.3 GB)	DEM3 (4 GB)	DEM4 (6 GB)	DEM5 (8.7 GB)
TauDEM 5.1	✓	✓	×	×	×
PD Algorithm with WVM	✓	✓	✓	×	×
PD Algorithm with TVM	✓	✓	✓	✓	✓

We evaluated the load balancing mechanism in which the tiles are distributed among the compute threads. Without a load balancing mechanism the number of points (grid cells) evaluated by each compute thread is quite unevenly distributed (Figure 2.9 a), resulting in the threads with fewer points to evaluate waiting for the threads with more points to evaluate before each synchronization that occurs with each call of the exchangeBorders function. When load balancing is not enabled, the compute thread 6 (CT6) processes nearly 6×10^5 more cells than the compute thread 7 (CT7). However, the load balancing approach resulted in a much more even distribution of the number of points processed by each thread (Figure 2.9 b) where the difference of processed cells between the compute threads dropped significantly. Fair data distribution can be enhanced by the use of smaller tile size that affects the granularity of the virtual memory.

The objectives of the study were to develop a capability to run the PD algorithm for large datasets without relying on the operating system's built-in virtual memory system. We performed an experiment using three approaches for the five DEMs listed in Table 2.1. We used a machine configured with 4 GB RAM. The last two DEM datasets require memory significantly more than this. DEM4 is 6.05 GB and requires nearly 21 GB of memory for the algorithm, while DEM5 is 8.65 GB and requires 34 GB of memory for the algorithm. We set the target memory capacity for our tiled virtual memory (TVM) approach to 2 GB and used a tile size of 1000 because of its better memory usage efficiency (see Table 2.2). We also performed a run with the tiled virtual memory disabled so that the Windows 7 operating system's built-in virtual memory manager (WVM) was invoked. The default Windows 7 paging file size was used. The third comparison ran the same problem using TauDEM 5.1 [4] which uses an MPI message-passing approach to implement the PD parallel algorithm. It has no virtual memory management so also relies on the Windows 7 virtual memory manager for swapping where the total memory demanded by all processes is greater than the physical memory capacity. As illustrated in Table 2.4, we were able to successfully run the PD algorithm with the developed system for all the datasets, the largest of which demanded memory more than five times the physical memory available. By contrast, the TauDEM 5.1 package failed to process DEM3, DEM4 and DEM5, and the PD algorithm with WVM failed to process DEM4 and DEM5.

2.7 Conclusion and Discussion

The increased availability of high resolution DEMs is driving the need for software to process and analyze these on all systems including desktop PC systems with limited memory. In this chapter we introduced a virtual tile memory manager that can be used with hydrologic terrain analysis programs to provide user level memory management and enable the processing of large DEMs on a PC, where operating system virtual memory management and multi-process domain partitioning fail. This was illustrated using a parallel pit removal algorithm for hydrologically conditioning DEM datasets. Elimination of pits is an essential step in hydrologic terrain analysis, required before applying other hydrological algorithms to calculate flow paths, slopes, and delineation of watersheds and sub-watersheds. We used a parallel implementation of a modified Planchon and Darboux algorithm [10] for pit removal in shared-memory multithreaded systems, designed to run on desktop computers having limited system memory. system. The system is optimized to retrieve tiles with a high hit ratio. The main conclusions of the system developed are:

1. We are able to run this hydrologic terrain analysis algorithm on limited memory systems for datasets so large that previously available algorithms fail. This overcomes the main obstacle of the memory capacity of the machine, by utilizing a user-level shared virtual memory system.
2. While evaluated with pitremove, there is nothing in the user-level virtual memory system that should preclude it from use with other algorithms which may require modification of the page replacement algorithm based on algorithm-specific data access patterns to maximize system throughput.
3. Efficiencies were achieved using algorithm refinements such as load balancing and the pre-fetching approach used in the page replacement algorithm of the user-level shared virtual memory system that increased hit rate and reduced wait time.

We conclude from the experimental results that the implemented parallel application is beneficial to geoscientists and researchers for computing raster-based computations in very large DEM datasets on a single machine with limited memory. We believe that the implemented system also can be used for other flow algebra algorithms used in TauDEM [7] with slight modification. For other algorithms the pre-fetching approach will need to be modified according to the algorithm-specific data access patterns. Furthermore, it may be beneficial to explore adaptive page replacement algorithms that change the replacement algorithm at run time based on the tile-access pattern of the system.

REFERENCES

[1] I. D. Moore, R. B. Grayson, and A. R. Ladson, “Digital terrain modelling: A review of hydrological, geomorphological, and biological applications,” *Hydrological Processes*, vol. 5, no. 1, pp. 3–30, 1991.

[2] T. Hengl and H. I. Reuter, *Geomorphometry: Concepts, Software, Applications.* Elsevier, 2009, vol. 33.

[3] J. P. Wilson and J. C. Gallant, Eds., *Terrain Analysis: Principles and Applications.* Wiley, 2000.

[4] D. G. Tarboton, *Terrain Analysis Using Digital Elevation Models (TauDEM)*, Utah Water Research Laboratory, Utah State University, 2014. [Online]. Available: http://hydrology.usu.edu/taudem/

[5] T. K. Tesfa, D. G. Tarboton, D. W. Watson, K. A. Schreuders, M. E. Baker, and R. M. Wallace, "Extraction of hydrological proximity measures from dems using parallel processing," *Environmental Modelling and Software*, vol. 26, no. 12, pp. 1696 – 1709, 2011.

[6] C. Wallis, R. Wallace, D. G. Tarboton, D. W. Watson, K. A. T. Schreuders, and T. K. Tesfa, "Hydrologic terrain processing using parallel computing." 18th World IMACS / MODSIM Congress, 2009, pp. 2540 – 2545. [Online]. Available: http://www.mssanz.org.au/modsim09/F13/wallis.pdf

[7] D. G. Tarboton, K. A. T. Schreuders, D. W. Watson, and M. E. Baker, "Generalized terrain-based flow analysis of digital elevation models." 18th World IMACS / MODSIM Congress, 2009, pp. 2000–2006. [Online]. Available: http://www.mssanz.org.au/modsim09/F4/tarboton_F4.pdf

[8] D. Engler, S. Gupta, and M. Kaashoek, "Avm: application-level virtual memory," in *Hot Topics in Operating Systems, 1995. (HotOS-V), Proc. the 5th Workshop on*, May 1995, pp. 72–77.

[9] A. Marowka, "Back to thin-core massively parallel processors," *Computer*, vol. 44, no. 12, pp. 49 –54, December 2011.

[10] O. Planchon and F. Darboux, "A fast, simple and versatile algorithm to fill the depressions of digital elevation models," *CATENA*, vol. 46, no. 2?3, pp. 159 – 176, 2002.

[11] GDAL Development Team, *GDAL - Geospatial Data Abstraction Library*, Open Source Geospatial Foundation, 2014. [Online]. Available: http://www.gdal.org

[12] S. K. Jenson and J. O. Dominque, "Extracting topographic structure from digital elevation data for geographic information system analysis," *Photogrammetric Engineering and Remote Sensing*, vol. 54, no. 11, pp. 1593–1600, 1988.

[13] D. G. Tarboton, R. L. Bras, and I. Rodriguez-Iturbe, "On the extraction of channel networks from digital elevation data," *Hydrological Processes*, vol. 5, no. 1, pp. 81–100, 1991.

[14] L. Wang and H. Liu, "An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling," *Int. J. Geographical Information Sci.*, vol. 20, no. 2, pp. 193–213, 2006.

[15] R. Barnes, C. Lehman, and D. Mulla, "Priority-flood: An optimal depression-filling and watershed-labeling algorithm for digital elevation models," *Computers and Geosciences*, vol. 62, pp. 117–127, Jan. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.cageo.2013.04.024

[16] X. J. W. Liu Yong He, Zhang Wan Chang, "Another fast and simple dem depression-filling algorithm based on priority queue structure," *Atmospheric and Oceanic Sci. Letters*, vol. 2, no. 4, p. 214, 2009.

[17] H.-T. Do, S. Limet, and E. Melin, "Parallel computing flow accumulation in large digital elevation models," *Procedia Computer Sci., Proc. Int. Conf. on Computational Sci., ICCS 2011*, vol. 4, no. 0, pp. 2277 – 2286, 2011.

[18] R. Xu, X. Huang, L. Luo, and S. Li, "A new grid-associated algorithm in the distributed hydrological model simulations," *SCIENCE CHINA Technological Sci.*, vol. 53, pp. 235–241, 2010, 10.1007/s11431-009-0426-4.

[19] H. Wang, Y. Zhou, X. Fu, J. Gao, and G. Wang, "Maximum speedup ratio curve (msc) in parallel computing of the binary-tree-based drainage network," *Computers and Geosciences*, vol. 38, no. 1, pp. 127 – 135, 2012.

[20] L. Ortega and A. Rueda, "Parallel drainage network computation on cuda," *Computers and Geosciences*, vol. 36, no. 2, pp. 171 – 178, 2010.

[21] J. F. O'Callaghan and D. M. Mark, "The extraction of drainage networks from digital elevation data," *Computer Vision, Graphics, and Image Processing*, vol. 28, no. 3, pp. 323 – 344, 1984. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0734189X84800110

[22] S. K. Reinhardt, J. R. Larus, and D. A. Wood, "Tempest and typhoon: User-level shared memory," *SIGARCH Comput. Archit. News*, vol. 22, no. 2, pp. 325–336, Apr. 1994. [Online]. Available: http://doi.acm.org/10.1145/192007.192062

[23] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient virtual memory for big memory servers," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 237–248, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508148.2485943

[24] H. Huang, P. Pillai, and K. G. Shin, "Design and implementation of power-aware virtual memory," in *Proc. Annual Conf. on USENIX Annual Technical Conf.*, ser. ATEC '03. USENIX Association, 2003, pp. 5–5. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247340.1247345

[25] J. Pardy and K. Hartling, *Efficient Data Management and Analysis with Geoprocessing*, ESRI, 2014. [Online]. Available: http://proceedings.esri.com/library/userconf/devsummit13/papers/devsummit-086.pdf

[26] A. S. Tanenbaum, A. S. Woodhull, A. S. Tanenbaum, and A. S. Tanenbaum, *Operating Systems: Design and Implementation.* Prentice-Hall, 1987, vol. 2.

[27] K. Li, "Ivy: A shared virtual memory system for parallel computing." *ICPP (2)*, vol. 88, p. 94, 1988.

[28] A. J. Smith, “Sequential program prefetching in memory hierarchies,” *Computer*, vol. 11, no. 12, pp. 7–21, 1978.

[29] M. Bhandarkar, L. Kal, E. de Sturler, and J. Hoeflinger, “Adaptive load balancing for mpi programs,” in *Computational Sci. - ICCS 2001*, ser. Lecture Notes in Computer Sci., V. Alexandrov, J. Dongarra, B. Juliano, R. Renner, and C. Tan, Eds. Springer, 2001, vol. 2074, pp. 108–117. [Online]. Available: http://dx.doi.org/10.1007/3-540-45718-6_13

[30] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn, “Efficient operating system scheduling for performance-asymmetric multi-core architectures,” in *Proc. 2007 ACM/IEEE Conf. on Supercomputing*, ser. SC ’07. ACM, 2007, pp. 53:1–53:11. [Online]. Available: http://doi.acm.org/10.1145/1362622.1362694

CHAPTER 3

Parallel Data Reduction Techniques for Big Datasets[*]

3.1 Abstract

Data reduction is perhaps the most critical component in retrieving information from big data (i.e., petascale-sized data) in many data-mining processes. The central issue of these data reduction techniques is to save time and bandwidth in enabling the user to deal with larger datasets even in minimal resource environments, such as in desktop or small cluster systems. In this chapter, we will examine the motivations behind why these reduction techniques are important in the analysis of big datasets. Then we will present several basic reduction techniques in detail, stressing the advantages and disadvantages of each. We also consider signal processing techniques for mining big data by the use of discrete wavelet transformations and server-side data reduction techniques. Lastly, we include a general discussion on parallel algorithms for data reduction, with special emphasis given to parallel wavelet-based multi-resolution data reduction techniques on distributed memory systems using MPI and shared memory architectures on GPUs along with a demonstration of the improvement of performance and scalability for one case study. The contribution of this chapter is to demonstrate parallel WaveCluster algorithm employing Discrete Wavelet Transform to reduce data for clustering big cloud dataset using MPI. We processed this dataset on a computer cluster system where the memory is limited in single processors within the available hardware.

3.2 Introduction

With the advent of information technologies, we live in the age of data. Data need to be processed and analyzed efficiently to extract useful information for innovation and

---

[*] Authors: Ahmet Artu Yıldırım, Cem Özdoğan, Daniel Watson

decision-making in corporation and scientific research. While the term of big data is relative and subjective and varies over time, a good working definition is the following:

Big data is data that takes an excessive amount of time/space to store, transmit, and process using available resources.

One remedy in dealing with big data might be to adopt a distributed computing model to utilize its aggregate memory and scalable computational power. Unfortunately, distributed computing approaches such as grid computing and cloud computing are not without their disadvantages (e.g., network latency, communication overhead, and high-energy consumption). An "in-box" solution would alleviate many of these problems, and GPUs (Graphical Processing Units) offer perhaps the most attractive alternative. However, as a cooperative processor, GPUs are often limited in terms of the diversity of operations that can be performed simultaneously and often suffer as a result of their limited global memory as well as memory bus congestion between the motherboard and the graphics card. Parallel applications as an emerging computing paradigm in dealing with big datasets have the potential to substantially increase performance with these hybrid models, because hybrid models exploit both advantages of distributed memory models and shared memory models.

A major benefit of data reduction techniques is to save time and bandwidth by enabling the user to deal with larger datasets within minimal resources available at hand. The key point of this process is to reduce the data without making it statistically indistinguishable from the original data, or at least to preserve the characteristics of the original dataset in the reduced representation at a desired level of accuracy. Because of the huge amounts of data involved, data reduction processes become the critical element of the data mining process on the quest to retrieve meaningful information from those datasets. Reducing big data also remains a challenging task that the straightforward approach working well for small data, but might end up with impractical computation times for big data. Hence, the phase of software and architecture design together is crucial in the process of developing data reduction algorithm for processing big data.

In this chapter, we will examine the motivations behind why these reduction techniques are important in the analysis of big datasets by focusing on a variety of parallel computing models ranging from shared-memory parallelism to message-passing parallelism. We will show the benefit of distributed memory system in terms of memory space to process big data because of the systems aggregate memory. However, although many of todays computing systems have many processing elements, we still lack data reduction applications that benefit from multi-core technology. Special emphasis in this chapter will be given to parallel clustering algorithms on distributed memory systems using the MPI library as well as shared memory systems on graphics processing units (GPUs) using CUDA (Compute Unified Device Architecture developed by NVIDIA).

3.3 General Reduction Techniques

Significant CPU time is often wasted because of the unnecessary processing of redundant and non-representative data in big datasets. Substantial speedup can often be achieved through the elimination of these types of data. Furthermore, once non-representative data is removed from large datasets, the storage and transmission of these datasets becomes less problematic.

There are a variety of data reduction techniques in current literature; each technique is applicable in different problem domains. In this section, we provide a brief overview of *sampling* by far the simplest method in implementation but not without intrinsic problems and *feature selection* as a data reduction technique, where the goal is to find the best representational data among all possible feature combinations. Then we examine *feature extraction* methods, where the aim is to reduce numerical data using common signal processing techniques, including discrete Fourier transforms (DFTs) and discrete wavelet transforms (DWTs).

3.3.1 Sampling and Feature Selection

The goal of sampling is to select a subset of elements that adequately represents the complete population. At first glance, sampling may appear to be overly simplistic, but these methods are nonetheless a very usable class of techniques in data reduction, especially with the advent of big data. With the help of sampling, algorithms performed on the sampled population retain a reasonable degree of accuracy while yielding a significantly faster execution.

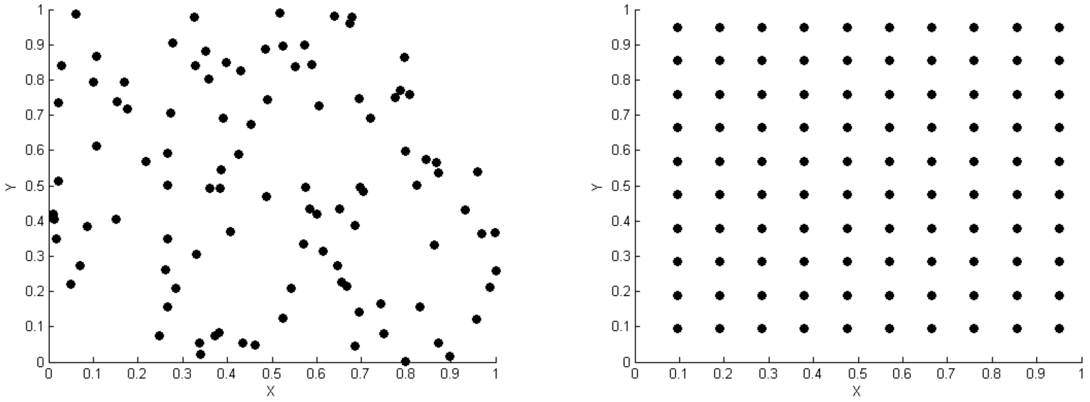
Perhaps the most straightforward sampling method is simple random sampling (Algorithm 4), in which sample tuples are chosen randomly using a pseudorandom number generator. In simple random sampling, the original big data population is randomly sampled, and those sampled units are added to the (initially empty) sampled population set with (or without) replacement. With replacement, as sampled units are placed into the sampled population set, if an identical unit already exists in that set, it is replaced by the sampled unit. Thus, one unit is randomly selected first with probability of $1/N$ where N is the size of population, and there are no duplicate elements in the sampled population. Without replacement, sampled units are added to the sampled population set without prejudice, and thus identical elements may exist in the resulting set. In simple random sample without replacement, because there are $\binom{N}{n}$ possible samples, where n is sample size, the probability of selecting a sampled unit is $1 \ / \ \binom{N}{n}$ [1].

It is worth noting that random number generation on parallel computers is not straightforward, and care must be taken to avoid exact duplication of the pseudorandom sequence when multiple processing elements are employed. The reader is directed to [2–4] for more detailed information on parallel pseudorandom number generation.

Algorithm 4 Simple random sampling algorithm

```
Require: Set of tuples $X$, size of samples $n$
Ensure: Set of sample tuples $S$
$S \leftarrow \emptyset$
for $i = 1 \rightarrow n$ do
    $x \leftarrow selectRandomTuple(X)$
    $S \leftarrow S \cup \{x\}$
end for
```

Another sampling method is systematic sampling, illustrated in Algorithm 5. The advantage of this method over the simple random method is that systematic sampling does not need a parallel random number generator. Let $k = N/n$ and R be a random number



(a)

(b)

Figure 3.1: Sampling illustrations on 2D domain (a) Simple random sampling (b) Systematic sampling

between 1 and k, then every ith tuple, $R + i \times k$, in database is selected according to the equation $R + i \times k$. Because systematic sampling performs selection of the sampled units in a well-patterned manner, the tuples must be stored in a random fashion in the database; otherwise, systematic sampling may introduce unwanted bias into the sampled set. For example, as a web server records the information of the visitors, the order of information stored in the database might produce bias if the employment status of the visitors affects this ordering because of heterogeneous characteristics of the population.

Algorithm 5 Systematic sampling algorithm

```
Require: Set of tuples $X$, size of population $N$, size of samples $n$
Ensure: Set of sample tuples $S$
$S \leftarrow \emptyset$
$k \leftarrow N/n$
$R \leftarrow randomNumber(1, k)$
for $i = 0 \rightarrow n - 1$ do
    $x \leftarrow X(R + i \times k)$
    $S \leftarrow S \cup \{x\}$
end for
```

In stratified sampling, shown in Algorithm 6, population units are divided into disjoint, homogeneous groups, called *strata* in which a sampling technique such as simple random sampling or systematic sampling is applied on those strata independently. One of the advantages of stratified sampling is to minimize estimation error. For example, suppose the manager of a web site wishes to obtain sampled users represented fairly in a sampled population, and that the information of gender proportions is known in advance. Therefore, the group (strata) consisting of all males are sampled together, and then the same sampling operation is applied on the remaining portion of database tuples (i.e., females), independently.

Algorithm 6 Stratified sampling algorithm

Require: Set of tuples X, size of population N, size of strata h, set of strata $Z = \{z_1, z_2, \ldots, z_h\}$ and filter operation σ

Ensure: Set of sample tuples S

```
$S \leftarrow \emptyset$
for all $z_i \in Z$ do
    $X^{'} \leftarrow \sigma_{z_i=true}(X)$
    $S^{'} \leftarrow applySampling(X^{'})$
    $S \leftarrow S \cup S^{'}$
end for
```

While sampling techniques are straightforward and efficient in reducing data, they may introduce a biased result, because the sampled set does not represent the population fairly. Thus, feature subset selection (FSS) techniques come up as a promising way to deal with this problem. A *feature* is defined as an attribute of the data specifying one of its distinctive characteristics. The objective of an FSS is to find the feature subset with size d representing the data best among all possible feature set combinations, $\binom{n}{d}$, over n features. To measure how well the selected feature subset represents the data, an objective function is used to evaluate the goodness of the candidate subset. Feature subset Y maximizes the objective function J(Y).

Exhaustive search, shown in Algorithm 7, is one straightforward way to find the optimum feature subset of size d. This algorithm is able to obtain the optimal feature set of size d over all possible feature subsets. The algorithm is considered a top-down approach where it starts from an empty feature subset and incrementally finds the solution. The most optimal feature set is the set having maximum goodness value. While exhaustive search guarantees the best solution, it is obvious that it exhibits poor performance due to its exponential algorithm complexity of $O(J(Y_i)) \times O(\binom{n}{d}) = O(J(Y_i)\binom{n}{d})$. Therefore, exhaustive search can be used only in practice if the data has a small number of features. The search space of exhaustive search is illustrated in Figure 3.3.1 where every leaf node is traversed to find the optimum solution.

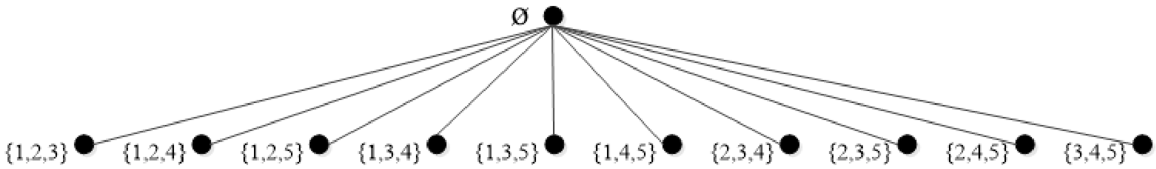


Figure 3.2: Search space of exhaustive search where $n = 5$ and $d = 3$

Algorithm 7 Exhaustive feature selection algorithm

```
Require: Set of features $X$, size of feature set $n$, size of target feature subset $d$, set of possible feature subsets, $F$, of $X$ where each subset is of size $d$
Ensure: Optimum feature subset $Y_{opt}$ of size $d$
$Y_{opt} \leftarrow \emptyset$
$G_{opt} \leftarrow -\infty$
for all $Y_i \in F = \{Y_0, Y_1, \ldots, Y_k\} \mid k = \binom{n}{d}$ do
    $G_i \leftarrow J(Y_i)$
    if $G_i > G_{opt}$ then
        $Y_{opt} \leftarrow Y_i$
        $G_{opt} \leftarrow G_i$
    end if
end for
```

In contrast to the exhaustive search approach, the branch and bound algorithm for future subset selection [5] finds the optimal feature set in a bottom-up manner using a tree structure. Hence, the algorithm discards m features to find the feature subset with d dimensions where $m = n - d$. The monotonicity property of feature subsets with respect to objective function, J, is satisfied at each iteration in which only one feature is discarded. Thus, the objective function is maximized at each iteration as shown in Equation 3.1.

$$J_1(Y_{i_1}) \leq J_2(Y_{i_1}, Y_{i_2}) \leq \cdots \leq J_m(Y_{i_1}, Y_{i_2}, \ldots, Y_{i_n}) \tag{3.1}$$

The search space of the branch and bound algorithm is illustrated as a tree structure form in Figure 3.3.1. The tree has $m + 1$ levels where $0 \leq i \leq m$. Each leaf node represents a candidate feature subset. The label near each node indicates the feature to be discarded. Traversing begins from the root node and switches to the next level. When the goodness-of-fit value of the feature set is less than the current maximum value or there are no nodes left to be visited at level i, the algorithm backtracks up to the previous level. The maximum goodness-of-fit value is calculated when the leaf node is reached for the corresponding candidate feature set. The algorithm finishes when it backtracks to level zero and the candidate feature set with highest goodness-of-fit value is considered the optimal feature set.

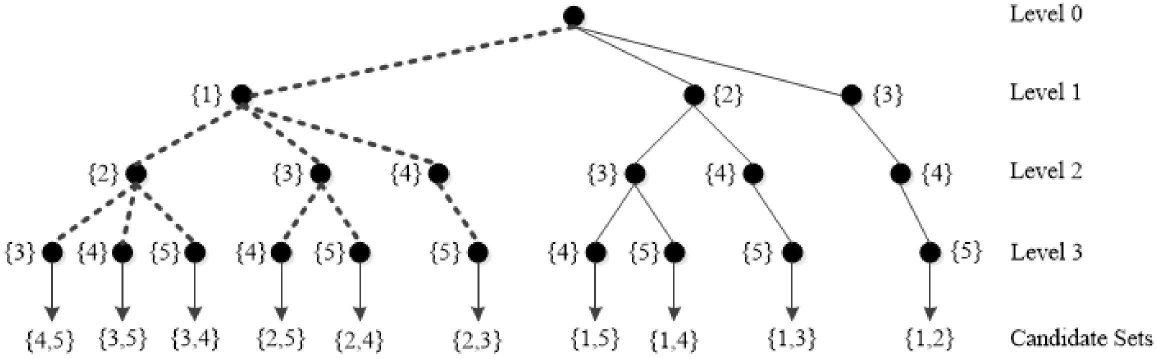


Figure 3.3: Search space of branch and bound algorithm where $n = 5$ and $d = 2$. Sample traversed path is shown by dashed lines

Although the branch and bound algorithm is superior to the exhaustive search algorithm in terms of algorithmic efficiency, the objective function is invoked not only for the leaf nodes, but also for other nodes in the tree. In addition, there is no guarantee that every node is visited. In order to alleviate this problem, Somol et al. [6] introduced a fast branch and bound algorithm (FBB) for optimal subset selection. The FBB algorithm uses a prediction mechanism to save the computation time of excessive invocation of objective function by monitoring goodness-of-fit changes.

Another data reduction approach is to use heuristic feature selection algorithms such as sequential forward selection (SFS, shown in Algorithm 8) and sequential backward selection (SBS, shown in Algorithm 9), which employ a greedy approach by respectively adding or removing the best features in an incremental fashion. The SFS algorithm starts with an empty set and then adds features one by one, whereas the SBS algorithm starts with all feature sets and then yield a suboptimum feature set by incrementally discarding features. Because heuristic algorithms substantially reduce the search space (for example by three orders of magnitude when $n = 20$, $d = 10$ [7], they are considered faster than exhaustive search at the expense of optimality. Furthermore, because forward selection starts with small subsets and then enlarges them, while backward selection starts with large subsets and then shrinks them, experiments show that forward selection is typically faster than backward selection because of the computational cost in repetitively processing large subsets [8]. As a forward selection approach, FOCUS by Almuallim and Dietterich [9] is a quasi-polynomial algorithm that starts searching from the bottom when the size of candidate feature subsets is one, then increments the size until the minimum goodness is satisfied.

Algorithm 8 Sequential forward feature selection algorithm

```
Require: Set of features $X = \{x_0, x_1, \ldots, x_n\}$, size of feature set $n$, size of target feature subset $d$
Ensure: Suboptimum feature subset $Y_{subopt}$ of size $d$
$Y_{subopt} \leftarrow \emptyset$
for $j = 1 \rightarrow d$ do
    $x \leftarrow \max(J(Y_{subopt} \cup \{x_i\})) \mid x_i \in X \, and \, x_i \notin Y_{subopt}$
    $Y_{subopt} \leftarrow Y_{subopt} \cup \{x\}$
end for
```

Algorithm 9 Sequential backward feature selection algorithm

```
Require: Set of features $X = \{x_0, x_1, \ldots, x_n\}$, size of feature set $n$, size of target feature subset $d$
Ensure: Suboptimum feature subset $Y_{subopt}$ of size $d$
$Y_{subopt} \leftarrow X$
for $j = 1 \rightarrow n - d$ do
    $x \leftarrow \min(J(Y_{subopt} - \{x_i\})) \mid x_i \in X \, and \, x_i \in Y_{subopt}$
    $Y_{subopt} \leftarrow Y_{subopt} - \{x\}$
end for
```

The drawback of these heuristic algorithms is a problem called nesting, in which the operation of adding (SFS) or removing (SBS) cannot be undone once the operation is finished. This situation is a primary cause of a suboptimal solution. To overcome the nesting problem, some algorithms have been proposed, such as the Plus-L-Minus-R introduced by Stearns [10] in which the algorithm performs l adding operations and r removing operations.

If $l > r$, it functions like the forward selection algorithm, otherwise it functions like the backward selection algorithm. One drawback of the Plus-L-Minus-R algorithm is that the parameters of l and r must be determined to obtain the best feature set. To compensate for this problem, a floating search method is proposed in which the algorithm changes l and r values at run time to maximize the feature set optimality [11].

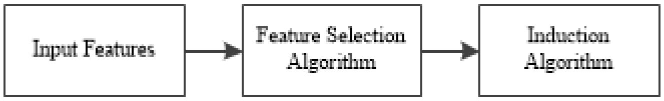


Figure 3.4: Flow of the filter model

As stated by John et al. [12], the approach of any data reduction algorithm falls into two models for selecting relevant feature subsets. The first is the filter model (see Figure 3.3.1) in which the phase of feature selection algorithm is applied independently of the induction algorithm as a preprocessing step. This is contrasted with the wrapper model (see Figure 3.3.1), which wraps the induction algorithm into the feature selection step. In the wrapper model, the evaluation phase of the goodness function for the feature subset is performed according to the induction algorithm. Each model has its own strengths and weaknesses; the filter model may introduce irrelevant features that may impose performance problems on the induction algorithm because of its separate existence but still performs relatively faster than the wrapper model. Even so, the wrapper model has the ability to find more relevant subsets at a reasonable cost to performance.

The wrapper model utilizes an n-fold cross validation technique where the input feature set is divided into n partitions of equal sizes. Using those partitions, the algorithm uses $n1$ partitions as training data and performs a validation operation on one partition. This operation is repeated n times for each of the other partitions and the feature subset with the maximum averaged goodness-of-fit value is regarded as the best solution. Cross validation can also avoid the phenomena of over-fitting; the problem of fitting noise into the model. While the wrapper model may not be ideal for big data due to its performance inefficiencies, some feature pruning techniques or sampling techniques applied to training data may yield faster execution times.

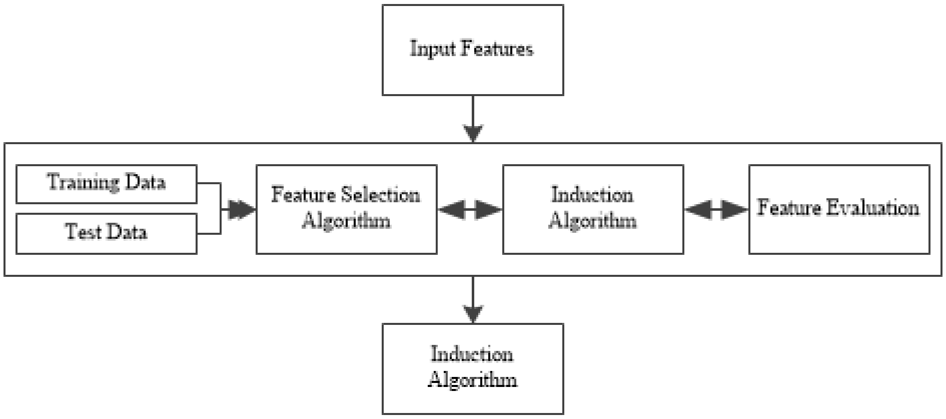


Figure 3.5: Flow of the wrapper model with cross validation

3.3.2 Data Reduction using Signal Processing Techniques

Another way to reduce big data to a manageable size is to use contemporary signal processing techniques. Although those methods fall into the class of feature extraction methods in the context of dimensionality reduction [13], they are considered here as an alternative approach to feature selection techniques. Feature extraction methods reduce the dimensionality by mapping a set of elements in one space to another; however, feature selection methods perform this reduction task by finding the best feature subset. Additionally, because feature selections methods require an objective function to evaluate the goodness of the subset, the output of the reduction operation therefore depends on the objective function. By contrast, feature extraction methods solve the reduction problem on another space on transformed data without using objective function. Because of the fact that signal processing techniques do not work on non-numerical data, they require a quantization step to process non-numerical data as a preprocessing step before applying the signal processing technique.

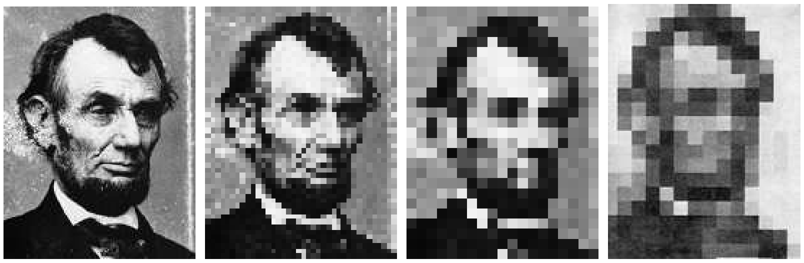
The Discrete Fourier Transform (DFT) method is used in many signal-processing studies that seek a reduction in data, for example in reducing time series data (that naturally tends toward large amounts of data) without sacrificing trend behavior [14]. The main aim of applying a DFT is to convert a series of data from the time domain into the frequency domain by decomposing the signal into a number of sinusoidal components. The DFT extracts features from sequences in which each feature in the transformed feature space refers to one DFT coefficient.

While the DFT is useful tool for analyzing signal data, they often fail to properly analyze non-stationary data that might be frequently encountered in time series data. To overcome this problem, another signal processing technique, the Discrete Wavelet Transform (DWT) can be employed to analyze the components of non-stationary signals [15]. Moreover, while DFTs transform the signal from the time/space domain into frequency domain, DWTs transform the signal into time/frequency domain. Thus, DWTs are beneficial by preserving the spatial property of the data on the transformed feature space (unlike DFTs). DWTs are location-sensitive unlike DFTs. Wu et al. [16] show that the DWT is a better tool than the DFT for the application of time series analysis because DWTs reduce the error of distance estimates on the transformed data.

DWTs decompose signals into high-frequency and low-frequency components using filtering techniques. The low-frequency component represents a lower resolution approximation of the original feature space on which the data-mining algorithm is applied to obtain (upon re-application of the DWT on the low-frequency component) solutions at different scales from fine (as in the case of the original dataset) to coarse as in the case of multiple applications of the DWT). In addition, a lookup table can be used to map the units from the transformed feature space back to a unit in the original feature space, providing a useful indexing capability of the resulting units in the transformed space.

As implied above, wavelet transforms can be applied to the input data multiple times on the low-frequency component in a recursive manner, creating a multi-resolution sampling of the original feature set. This recursive operation of decomposing the signal is called a filter bank. Each operation of wavelet transform constitutes half-sized objects in the transformed feature space for each dimension as compared to previous dimension of the feature space.

Figure 3.6 illustrates the low-frequency component extracted from the image of Abraham Lincoln. As shown in the Figures, as the scale level of the wavelet transform increases, we obtain coarser representation of the original data in which typically a data-mining algo-



(a) (b) (c) (d)

Figure 3.6: Illustration of Low-frequency Component in Art (a) Original image (b) Low-Frequency Component (scale level = 3) (c) Low-Frequency Component (scale level = 4) (d) Low-frequency Component in Art, mosaic painting of Abraham Lincoln by Leon Harmon, which is small in resolution and a coarser representation of the original image

rithm is applied to extract the information from this reduced data. For example, in Figure 3.6(b), DWT is applied three times on the low-frequency component (scale level is 3), and in Figure 3.6(c), DWT is applied one more time using Figure 3.6(b), so the scale level becomes four. Thus, the reduced data in Figure 3.6(c) has half-sized cells as compared to the Figure 3.6(b) and Figure 3.6(c) has coarser representation of the input data as compared to the previous low-frequency component (Figure 3.6(b)). For comparison, Figure 3.6(d) is actually a rather famous illustration by Leon Harmon (1973) created to illustrate his article on the recognition of human faces.

There are many popular wavelet algorithms including Haar wavelets, Daubechies wavelets, Morlet wavelets and Mexican Hat wavelets. As stated by [17], Haar wavelets [18] are used in many studies because of its simplicity, fast and memory-efficient characteristics. It should also be mentioned that it is necessary to scale source datasets linearly to power of two for each dimension, since the signal rate decreases by a factor of two at each level. Let $X = \{x_1, x_2, \ldots, x_n\}$ be an input data of size n. Then the low frequency component of the data can be extracted via averaging operation as shown in Equation 3.2.

$$\begin{pmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0.5 & 0.5 \end{pmatrix} \bullet \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \frac{x_1+x_2}{2} \\ \frac{x_3+x_4}{2} \\ \vdots \\ \frac{x_{n-1}+x_n}{2} \end{pmatrix} \tag{3.2}$$

Each object in the low-frequency component of the feature space W_φ^j, is also contained in the low-frequency component of the previous transformed feature space W_φ^{j-1}. Hence, the feature spaces are nested [18], that is:

$$W_\varphi^j \subset W_\varphi^{j-1} \subset W_\varphi^{j-2} \subset \cdots \subset W_\varphi^0 \tag{3.3}$$

The main advantages of using DWT over DFT are listed below [19].

- DWT is able to capture time-dependent local properties of data, whereas DFT can only capture global properties.
- The algorithmic complexity of DWT is linear with the length of data, which is superior to even fast Fourier transform with $O(nlogn)$ complexity.
- While DFT provides the set of frequency components in the signal, DWT allows us to analyze the data at different scales. The low-frequency component of the transformed space represents the approximation of the data in another scale.
- Unlike DFTs, DWTs have many wavelet functions. Thus, they provide access to information that can be obscured by other methods.

3.4 Parallel Data Reduction Algorithms

3.4.1 Parallel Processing

Despite to substantial improvements on processor technology in terms of speed, data reduction algorithms may still not complete the required task in a reasonable amount of time at big data. Besides, there may not have enough available memory resources to hold all the data on a single computer. One of the possible solutions to overcome those issues and efficiently mine big data is to make utilization of parallel algorithms [20]. Parallel processing techniques are extensively used to divide the task into smaller subtasks and then executing them simultaneously to cope with memory limits and to decrease the execution time of that task.

Parallel computer architectures are classified according to the memory sharing models. One class of parallel architectures is distributed memory systems (DMS) containing processors or nodes which are connected with each other by means of high-speed network infrastructure. In this model, each processor has its own memory. Traditionally, data in these architectures are shared among processors using message passing via the Message Passing Interface (MPI) [21] library, typically used to construct the communication among the processors by sending/receiving messages.

Another class of parallel architectures are shared memory systems (SMS). As the name implies, each processor communicates each other via dedicated shared memory over a data bus. Advantages of SMS over DMS the absence of the need to store duplicate memory and much faster communication for the processors. On the other hand, distributed memory architectures fully utilize the larger aggregate memory that allows us to solve larger problems which typically imply large memory needs.

As a distributed memory system, MapReduce [22] promises to provide a way of dealing with big data. In this model, the programmer is not required to specify the parallelism in code. At its core, there are only two functions which are map and reduce. The map function processes key/value pairs and divides the data into a group of sub-data where each sub-data is represented by a key value. The sub-data, also known as intermediate values, are stored in a distributed file system by a key-value pair. By contrast, the reduce function merges all intermediate values associated with the same key. In the starting phase of the job, the library splits the input files into pieces of typically 16 megabytes to 64 megabytes. It then starts up the many copies of the program on a cluster. The implementation has been developed based on master/worker model. Master node makes assignments of map and reduce tasks to available worker jobs. Each worker writes the intermediate results of map and reduce operations to local disks which can be accessed by all nodes in the cluster. One of the characteristics of the system is that processors are not allowed to communicate with each other directly. Thus, while this strategy has been criticized to be too low-level and not-novel, the model provides us scalability to support the ever-growing data. It also ensures fault-tolerance by storing intermediate files on distributed file system and its dynamic job scheduling system.

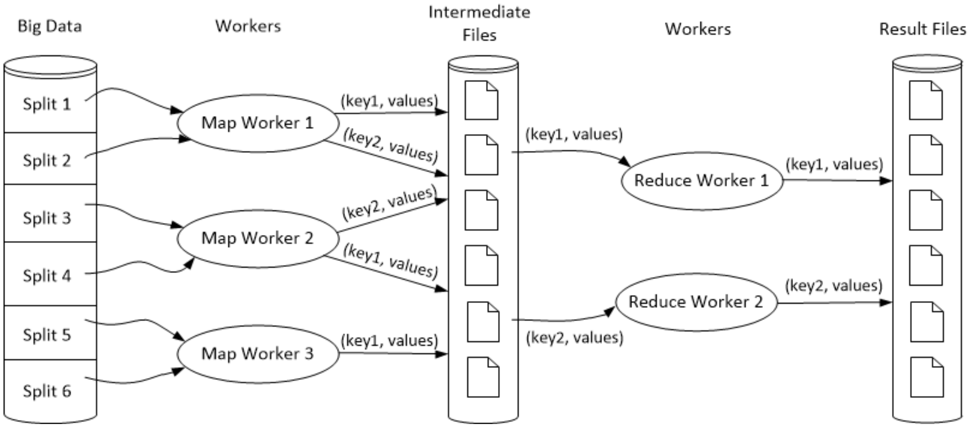


Figure 3.7: Illustration of map-reduce model where the number of pieces, $M = 6$

We have plenty of model choices for shared memory systems. Threading is one way to run concurrent processes on multicore processors where each thread might run on different cores in a single process. Threads share the process address space and can access other process resources as well. Another choice is OpenMP [23]. OpenMP seeks to speed up the processing of loops by using pragma directives. As an application programming interface (API), OpenMP provides the programmer flexibility, ease of use and transparency in the creation of CPU threads.

With the advent of many-cored graphical processing units (GPUs), auxiliary processing units (such as GPUs) can be utilized for general purpose computation. In addition to its traditional usage of accelerating the graphic rendering, GPUs can be harnessed as co-processors through the use of languages such as CUDA. NVIDIA introduced CUDA [24] (Compute Unified Device Architecture) to enable these general purpose computations on

NVIDIA GPUs in an efficient way. In CUDA programming, the programmer partitions the problem into subproblems that can be solved independently in parallel on GPU cores. In practice, the programmer writes a function named *kernel* and accesses the data using thread IDs in a multidimensional manner. The advantage of this model is that once the parallel program is developed, it can scale transparently to hundreds of processor cores independent on any CUDA-capable NVIDIA graphic processors.

There are three main stages in CUDA program flow. First: input data is copied from local memory (RAM) to the global memory of the graphic processor so that processing cores can access the input data quickly. Secondly, the CUDA machine executes the parallel program given. Lastly, data residing on the global memory of the GPU need to be copied from global memory to local memory. The primary disadvantage of this model is that this mechanism may incur performance bottlenecks as a result of extra memory copy operations. In some cases this may decreases the speedup ratio of the parallel program to tht epoint where it is comparable compared to sequential algorithms. Another disadvantage of CUDA is that the CUDA kernel is not allowed to access the disk, or call the functions of operating system or libraries developed for CPUs.

OpenCL [25] (Open Computing Language) is the first open standard for general-purpose parallel programming of heterogeneous systems. It enables the programmer to write portable parallel code for machines with multicore CPUs and GPUs as well. The generalization of the OpenCL interface allows efficient mapping to a wide range or hardware. Like the CUDA model, a kernel function needs to be specified in which each unit of concurrent execution accesses the data via thread ids. OpenCL applications can work with large arrays or multidimensional matrices.

Since GPU hardware implements a SIMD (Single instruction, multiple data) based parallel execution model, each control unit in the GPU broadcasts 1 instruction among the processor units inside the same processor group, – for which the instruction is executed in parallel. This architectural difference from the traditional CPU architecture requires non-trivial parallel programming models. It also comes with its intrinsic restrictions in developing a parallel program such as synchronization issues among the GPUs and the complications of having different level of memory models. Furthermore, many classical programming techniques such as recursion may not be easily applied in GPU computing. Thus, GPU programs have to be developed in accordance with the underlying architecture.

3.4.2 Server-side Parallel Data Reduction Method for Data Centers

Due to often massive bandwidth requirements, the operation of data movement occurring between hosting datacenters and workstations may require a substantial amount of time. One remedy is to run data reduction applications on the data centers themselves to take advantage of data locality. Wang et al. [26] introduced the Script Workflow Analysis for MultiProcessing (SWAMP) system. In SWAMP system, the user is required to write a shell script and then the script is submitted to the OpenDAP data server using Data Access Protocol (DAP). The SWAMP parser has the ability to recognize netCDF options and parameters. The execution engine of SWAMP manages the script execution and builds dependency trees. The SWAMP engine can potentially exploit parallel execution of the script when possible.

Singh et al. [27] introduced parallel feature selection algorithm to evaluate quickly billions of potential features for very large data sets using a *map-reduce* framework. In the *mapping phase*, each map worker iterates over the training records of the block and produces intermediate data records for each new feature. In the *reduce* phase, each reduce worker computes an estimated coefficient for each feature. Finally, post-processing is performed to aggregate the coefficients for all features in the same feature class.

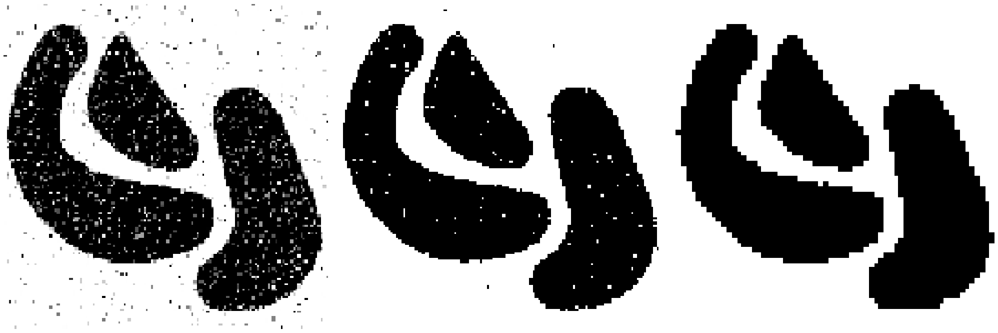
3.5 Parallel WaveCluster Algorithm

3.5.1 WaveCluster Algorithm

Clustering is a common data mining technique that used for information retrieval by grouping similar objects into disjoint classes or clusters [28]. There has been a rapid growth of very large or big datasets in scientific and commercial domains with the recent progress in data storage technology. Because cluster analysis has become a critical task for the mining of the data, a considerable amount of research has been carried out in developing sequential clustering analysis methods. Clustering algorithms have been highly utilized in various fields such as satellite image segmentation [29], unsupervised document clustering [30], and in the clustering of bioinformatics data [31]. The WaveCluster algorithm is a multi-resolution clustering algorithm introduced by [32]. This algorithm is designed to perform clustering on large spatial datasets by taking advantage of discrete wavelet transforms. Thus, the algorithm has the ability of detecting arbitrary shape clusters at different scales and can handle noise in an appropriate way.

WaveCluster algorithms contains three phases. In the first phase, algorithm quantizes the feature space and then assigns objects to the units. This phase affects the performance of clustering for different values of interval sizes. In the second phase, discrete the wavelet transform is applied on the feature space multiple times. Discrete wavelet transforms are a powerful tool for time-frequency analysis that decomposes the signal into average subband and detail subbands using filtering techniques. The WaveCluster algorithm gains the ability to remove outliers with the wavelet transform and detects the clusters at different levels of accuracy (multi-resolution property), from fine to coarse, by applying wavelet transforms multiple times. Following the transformation, dense regions (clusters) are detected by finding connected components and labels are assigned to the units in the transformed feature space. Next, a lookup table is constructed to map the units in the transformed feature space to original feature space. In the third and last phase, The WaveCluster algorithm assigns the cluster number of each object in the original feature space. In Figure 3.8, the effect of the wavelet transformation on source dataset (see Figure 3.8(a)) is demonstrated.

The results of the WaveCluster algorithm for different values of ρ is shown in Figures 3.8(b) and 3.8(c); where ρ (the scale level) represents how many times wavelet transform is applied on the feature space. There are 6 clusters detected with $\rho = 2$ (Figure 3.8(b)) and 3 clusters are detected with $\rho = 3$ (Figure 3.8(c)). In the performed experiments,



(a) (b) (c)

Figure 3.8: WaveCluster algorithm multi-resolution property. (a) Original source dataset. (b) $\rho = 2$ and 6 clusters are detected. (c) $\rho = 3$ and 3 clusters are detected. (where ρ is the scale level)

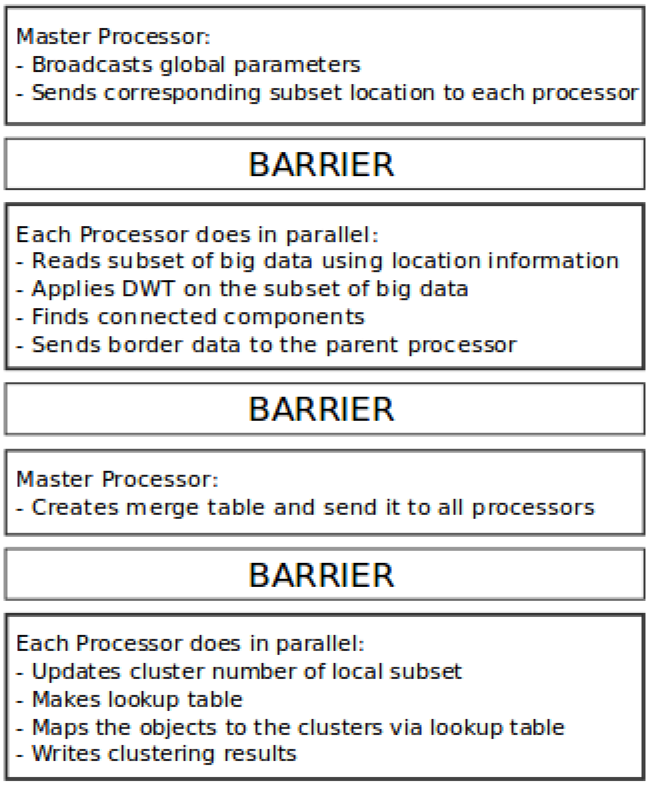
connected components are found on average subbands (feature space) using classical two-pass connected component labelling algorithm [33] at different scales.

3.5.2 Parallel WaveCluster Algorithm on the Distributed Memory System

We report now our developed parallel WaveCluster algorithm based on the message-passing model using the MPI library [34]. This algorithm scales linearly with the increasing number of objects in the dataset. One may conclude that it makes mining big datasets possible via the parallel algorithm on distributed memory systems, without having restrictions due to the dataset size and other relevant criteria.

In the message-passing approach, each copy of the single program runs on processors independently, and communication is provided by the manner of sending and receiving messages among nodes. In our implementation, we followed a master-slave model. Each processor works on a specific partition of the dataset and executes the discrete wavelet transform and connected component labeling algorithm. The obtained results from each processor are then locally correct, but might not be globally correct. Therefore, processors exchange their local results and then check for correctness. This operation is achieved simply by sending the border data of the local results to the master node, called the parent node. This border data consists of transformed values and their corresponding local cluster numbers. After processors send the border data to the parent node, the parent node creates a merge table for all processors with respect to the global adjacency property of all cells. This requirement for sustaining consistency in effect creates a barrier primitive. Accordingly, all processors wait for the parent node to receive merged table. Then, all processors update the cluster numbers of the units on local feature space. Lastly, processors map the objects of the original feature space to the clusters using lookup table. In the lookup table, each entry specifies the relationships of one unit in the transformed feature space to the corresponding units of the original feature space. The flow of the algorithm is depicted in Figure 10.

Algorithm 10 Parallel WaveCluster Algorithm for Distributed Memory Systems



Master Processor:
- Broadcasts global parameters
- Sends corresponding subset location to each processor

BARRIER

Each Processor does in parallel:
- Reads subset of big data using location information
- Applies DWT on the subset of big data
- Finds connected components
- Sends border data to the parent processor

BARRIER

Master Processor:
- Creates merge table and send it to all processors

BARRIER

Each Processor does in parallel:
- Updates cluster number of local subset
- Makes lookup table
- Maps the objects to the clusters via lookup table
- Writes clustering results

Experiments for this technique were conducted and obtained performance behaviors were presented for 1, 2, 4, 8, 16 and 32 core cases on a cluster system having 32 cores with a 2.8 GHz clock speed and fast Ethernet (100 Mbit/sec) as underlying communication. Our experiments have shown that the cluster shape complexity of the dataset has minimal impact on the execution time of the algorithm. Datasets below are named as DS32 (1073741824 objects) and DS65 (4294967296 objects) according to the dataset size. The DS 65 case is beyond the limit of dataset size that fits into the memory of a single processor within the available hardware, and is studied by running that dataset with 8, 16 and 32 processors.

Table 3.1: Execution times and speed-up ratios for varying dataset sizes and number of processors (np)

Dataset	np = 1	np = 2	np = 4	np = 8	np = 16	np = 32
Execution Times (in seconds)						
DS65	-	-	-	17.59	9.70	5.21
DS32	35.19	17.96	9.15	4.41	2.50	1.30
Speed-up Ratios						
DS65	-	-	-	1	1.81	3.37
DS32	1	1.95	3.84	7.97	14.07	27.06

Table 3.1 shows execution times and speed-up ratios with respect to the number of objects in the dataset for varied numbers of processors. Analysis of the dataset DS65 is only possible at or above the size of eight processors in our computer cluster system. The benefit of the distributed memory system is easily observed above in processing big data. Experimental results have shown that this parallel clustering algorithm provides a superior speed-up and linear scaling behavior (time complexity) and is useful in overcoming space complexity constraints via aggregate memory of the cluster system.

3.5.3 Parallel Data Reduction on GPUs

In this section, the parallel implementation of the WaveCluster algorithm on GPUs is presented [35]. The WaveCluster approach first extracts the low-frequency component from the signal using the wavelet transform. This phase is considered to be a data reduction stage in which the low-frequency component is representative of the dataset to be processed in another phase. The phase of applying wavelet transform may be invoked multiple times, consequently yielding a clustering result from fine to coarse with respect to the predefined scale level m which reflects the feature of multi-resolution of the algorithm. In the WaveCluster algorithm, a *cluster* is defined as a group of connected units in the transformed feature space. In this study, we used a Haar wavelet [36], because of its initial small window width and easy implementation. Then a lookup table is built that maps more than one unit in the original dataset to one unit in the transformed dataset. After this phase, a data mining algorithm is performed on the low-frequency component and the resulting units are transformed back to the original resolution using a look-up table. To find the connected units, the algorithm uses standard Connected Component Labeling algorithm.

Algorithm 11 CUDA Algorithm of Low-Frequency Component Extraction

```
Input: $W_{\varphi}^{j-1}, islastscale, TH$
Output: $W_{\varphi}^{j}$
declare $I[dim.y * 2][dim.x * 2]$ in shared memory
declare $H[dim.y * 2][dim.x]$ in shared memory
load thread-related disjoint 2x2 points of $V^{j}$ into buffer $I$
apply one-dimensional wavelet transform to each column of points of 2x2
  field and store values into buffer $H$
apply one-dimensional wavelet transform to each row of points over H and
  store approximation value in local memory $m$
if $islastscale = true$ then
    if $val > TH$ then
        $m \leftarrow MAXFLOAT$
    else
        $m \leftarrow threadindex$
    end if
end if
$W_{\varphi}^{j}[threadindex] \leftarrow m$
```

Algorithm 11 shows the pseudo-code of the extraction of low-frequency component on CUDA machine. The kernel is invoked j times and each invocation results in the extraction of coarser component. The kernel also removes outliers on the transformed feature space in which the threshold value TH is chosen as arbitrary. Each CUDA thread is responsible for finding one approximation value extracted from the feature space of 2x2 size. Before kernel invocation, the input feature space is transferred from host memory to the CUDA global memory and the output buffer is allocated in the device memory for storing the transformed feature space. Because the input data is 2-dimensional, the wavelet transform is applied twice. Each CUDA thread applies one-dimensional wavelet transform to each column of the local feature space and stores intermediate values in the shared memory buffer H. The final approximation value is eventually calculated by applying second one-dimensional wavelet transform on each row of the points on H. Hence, buffer H is used to store temporal results. To facilitate the data access operation, the algorithm benefits from shared memory located on the CUDA machine.

We implemented the CUDA version of the algorithm, connected component labeling, and the look-up phase, and then evaluated the performance of each CUDA kernels on very large synthetic datasets. CUDA experiments were conducted on a Linux workstation with 2 GB RAM, Intel Core2Duo (2 Cores, 2.4 GHz, 4MB L2 Cache) processor and NVIDIA GTX 465 (1 GB memory, 352 computing cores, each core runs at 1.215 GHz) with compute capability 2.0 and runtime version 3.10. In this section, we present the extraction phase of the low frequency component as a data reduction algorithm on GPUs. In these experiments, two-dimensional fog datasets (DataSet1; DS1 and DataSet2; DS2) are used. These datasets have been obtained from the web site of NASA Weather Satellite Imagery Viewers (2011). As a distinctive property, DS1 has more clusters than DS2, but the size of clusters is bigger in DS2. The larger datasets (DS(1_ or 2_) with the size of 4096, 2048, 1024 and 512) have been obtained by scaling the datasets linearly.

Execution times (in microseconds) and the corresponding kernel speed-up values for the low-frequency extraction algorithm are presented in Table 3.2.

Table 3.2: Performance results of CUDA and CPU versions of Low-Frequency Component Extraction for scale level 1 on datasets (DS1 & DS2); times in microseconds

Dataset	Number of Points	Execution Time (CPU)	Execution Time (GPU)	Kernel Speed-up
DS1_4096	16777216	121286	735	165.01
DS1_2048	4194304	30739	228	134.82
DS1_1024	1048576	7742	94	82.36
DS1_512	262144	1958	52	37.65
DS2_4096	16777216	121267	735	164.98
DS2_2048	4194304	30406	230	132.20
DS2_1024	1048576	7710	95	81.15
DS2_512	262144	1956	54	36.22

The obtained results indicate achievement of up to a 165.01x kernel speed-up ratio in dataset DS1 with the size 4096 in the kernel of low-frequency extraction. We obtained high speed-up ratios as the number of points increased in the dataset. This result indicates that the operation of the signal component extraction in the wavelet transform is suitable for execution on CUDA devices.

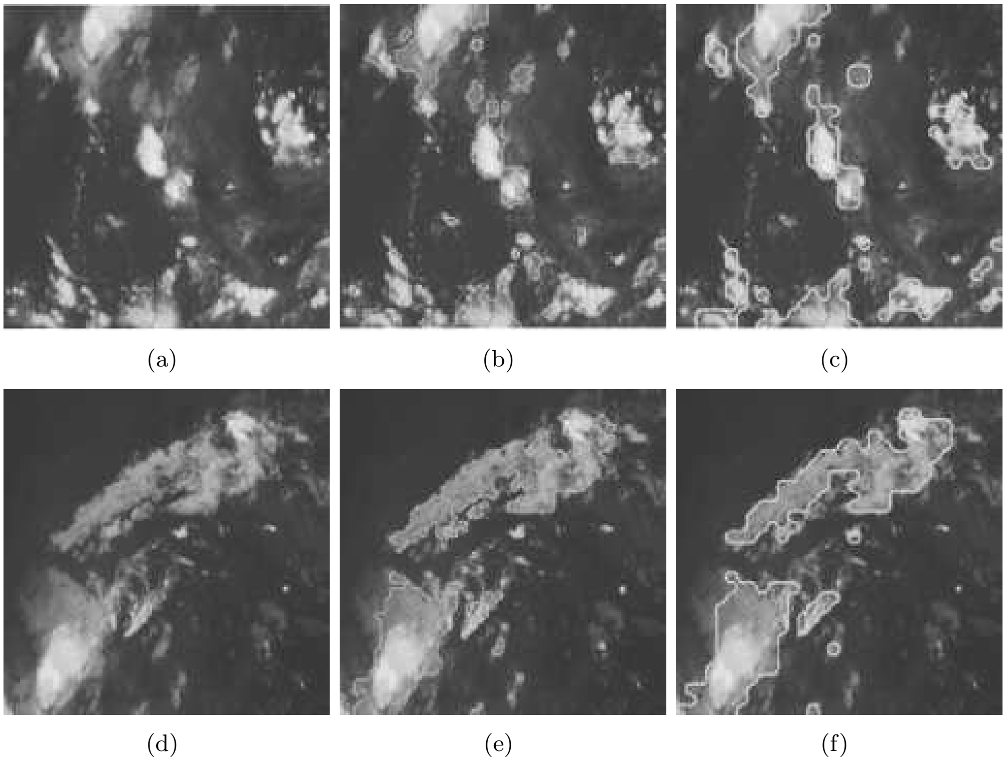


Figure 3.9: Application of WaveCluster approach on fog datasets. (a) DS1 dataset. (b) $\rho = 3$ and K = 35 for DS1. (c) $\rho = 4$ and K = 15 and for DS1. (d) DS2 dataset. (e) $\rho = 3$ and K = 21 for DS2. (f) $\rho = 4$ and K = 6 for DS2. (where ρ is the scale level and K is the number of clusters detected)

The detected numbers of clusters (K) for varying scale level ρ are depicted in Figure 3.9. As shown in the figures, there is a negative correlation between the scale level, ρ, and number of clusters K. Because resizing the dataset results in a coarser representation of the original dataset, the number of clusters decreases with the increasing scale level ρ. This phenomenon is a natural consequence of applying wavelet transforms on the low-frequency component of the original signal.

3.6 Conclusion

In this chapter, we have investigated the study of data reduction methodologies for big datasets both from the point of theory as well as application with special emphasis on parallel data reduction algorithms. Big datasets need to be processed and analyzed efficiently to extract useful information for innovation and decision-making in corporate and scientific research. However, because the process of data retrieval on big data is computationally expensive, a data reduction operation can result in substantial speed up in the execution of the data retrieval process. The ultimate goal of the data reduction techniques is to save time and bandwidth by enabling the user to deal with larger datasets within minimal resources without sacrificing the features of the original dataset in the reduced representation of the dataset at the desired level of accuracy.

Despite substantial improvements on processor technology in terms of speed, data reduction algorithms may still not complete the required task in a reasonable amount of time. Additionally, there may not be enough available memory resources to hold all the data on a single computer. One of the possible solutions to overcome these issues and efficiently mine big data is to implement and use parallel approaches [20] that work by dividing the task into smaller subtasks and then executing them simultaneously to cope with memory limits and to decrease the execution time of given task.

Graphical processing units (GPUs) are regarded as co-processors of the CPU and have tremendous computational power. NVIDIA introduced CUDA, providing a programming model for parallel computation on GPUs. In this chapter, we have presented a CUDA algorithm of a wavelet transform that is used in the WaveCluster algorithm for reducing magnitude of the original dataset. The WaveCluster approach first extracts the low-frequency components from the signal using a wavelet transform and then performs connected component labeling on the low-frequency component to find the clusters represented in the dataset. The algorithm has a multi-resolution feature. Thus, the algorithm has the ability of detecting arbitrarily-shaped clusters at different scales and can handle noise in an appropriate way.

Due to limited memory capacity of the video cards, CUDA devices may not be a remedy to all problems in processing big data. Much larger datasets could be mined on distributed memory architectures with the aggregate memory of the system using message passing APIs such as MPI. We have also implemented the WaveCluster algorithm for distributed memory models using MPI [34] in which a master-slave model and replicated approach are followed. A dataset that does not fit into the available memory is processed by taking advantage of the aggregate memory on the distributed memory system.

Each parallel memory architecture has intrinsic advantages over other models. With increasingly hybrid architectures, an MPI model (or a map-reduce model) can be used to achieve data distribution between GPU nodes where CUDA functions as the main computing engine [37]. Hence, they can be considered complementary models for maximizing the use of system resources.

REFERENCES

[1] S. Lohr, *Sampling: Design and Analysis*, ser. Matemáticas Thomson. Duxbury Press, 1999.

[2] S. Aluru, G. Prabhu, and J. Gustafson, “A random number generator for parallel computers,” *Parallel Computing*, vol. 18, no. 8, pp. 839 – 847, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016781919290030B

[3] T. Bradley, J. du Toit, R. Tong, M. Giles, and P. Woodhams, “Chapter 16 - parallelization techniques for random number generators,” in *GPU Computing Gems Emerald Edition*, W. mei W. Hwu, Ed. Morgan Kaufmann, 2011, pp. 231 – 246. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123849885000164

[4] A. D. Matteis and S. Pagnutti, “A class of parallel random number generators,” *Parallel Computing*, vol. 13, no. 2, pp. 193 – 198, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016781919090146Z

[5] P. Narendra and K. Fukunaga, “A branch and bound algorithm for feature subset selection,” *IEEE Trans. on Computers*, vol. C-26, no. 9, pp. 917 –922, Sept. 1977.

[6] P. Somol, P. Pudil, F. J. Ferri, and J. Kittler, “Fast branch & bound algorithm in feature selection,” in *Proc. SCI 2000. The 4th World Multiconference on Systemics, Cybernetics and Informatics*, B. Sanchez, M. J. Pineda, and J. Wolfmann, Eds. IIIS, July 2000, pp. 646–651.

[7] A. Whitney, “A direct method of nonparametric measurement selection,” *IEEE Trans. on Computers*, vol. C-20, no. 9, pp. 1100 – 1103, Sept. 1971.

[8] A. Jain and D. Zongker, “Feature selection: Evaluation, application, and small sample performance,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 2, pp. 153–158, Feb. 1997. [Online]. Available: http://dx.doi.org/10.1109/34.574797

[9] H. Almuallim and T. G. Dietterich, “Learning with many irrelevant features,” in *Proc. Ninth National Conf. on Artificial Intelligence (AAAI-91)*, vol. 2. AAAI Press, 1991, pp. 547–552. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.2488

[10] S. D. Stearns, “On selecting features for pattern classifiers,” in *Proc. 3rd Int. Conf. on Pattern Recognition (ICPR 1976)*, 1976, pp. 71–75.

[11] P. Pudil, J. Novovicova, and J. Kittler, “Floating search methods in feature selection,” *Pattern Recognition Letters*, vol. 15, no. 11, pp. 1119 – 1125, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167865594901279

[12] G. H. John, R. Kohavi, and K. Pfleger, “Irrelevant Features and the Subset Selection Problem,” in *Int. Conf. on Machine Learning*, 1994, pp. 121–129. [Online]. Available: http://citeseer.ist.psu.edu/john94irrelevant.html

[13] P. Pudil and J. Novovičová, “Novel methods for feature subset selection with respect to problem knowledge,” in *Feature Extraction, Construction and Selection.* Springer, 1998, pp. 101–116.

[14] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, “Fast subsequence matching in time-series databases,” *SIGMOD Rec.*, vol. 23, no. 2, pp. 419–429, May 1994. [Online]. Available: http://doi.acm.org/10.1145/191843.191925

[15] M. Sifuzzaman, M. Islam, and M. Ali, “Application of wavelet transform and its advantages compared to fourier transform,” *J. of Physical Sci.*, vol. 13, pp. 121–134, 2009.

[16] Y.-L. Wu, D. Agrawal, and A. El Abbadi, “A comparison of dft and dwt based similarity search in time-series databases,” in *Proc. 9th Int. Conf. on Information and Knowledge Management*, ser. CIKM ’00. ACM, 2000, pp. 488–495. [Online]. Available: http://doi.acm.org/10.1145/354756.354857

[17] T. Huffmire and T. Sherwood, “Wavelet-based phase classification,” in *PACT ’06: Proc. 15th Int. Conf. on Parallel Architectures and Compilation Techniques.* ACM, 2006, pp. 95–104.

[18] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin, “Wavelets for computer graphics: A primer, part 1,” *IEEE Comput. Graph. Appl.*, vol. 15, no. 3, pp. 76–84, 1995.

[19] I. Popivanov and R. Miller, “Similarity search over time-series data using wavelets,” in *Proc. 18th Int. Conf. on Data Engineering*, 2002, pp. 212 –221.

[20] S. Reese Hedberg, “Parallelism speeds data mining,” *Parallel Distributed Technology: Systems Applications, IEEE*, vol. 3, no. 4, pp. 3 –6, winter 1995.

[21] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, 1994.

[22] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[23] B. Chapman, G. Jost, and R. v. d. Pas, *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation).* MIT Press, 2007.

[24] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with cuda,” *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500

[25] B. Gaster, L. Howes, D. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL.* Elsevier Sci., 2011. [Online]. Available: http://books.google.com/books?id=qUJVU8RH3jEC

[26] D. Wang, C. Zender, and S. Jenks, “Server-side parallel data reduction and analysis,” *Advances in Grid and Pervasive Computing*, pp. 744–750, 2007.

[27] S. Singh, J. Kubica, S. Larsen, and D. Sorokina, “Parallel large scale feature selection for logistic regression,” in *SIAM Int. Conf. on Data Mining (SDM)*, 2009.

[28] U. Fayyad, G. Piatetsky-shapiro, and P. Smyth, “From data mining to knowledge discovery in databases,” *AI Magazine*, vol. 17, pp. 37–54, 1996.

[29] A. Mukhopadhyay and U. Maulik, “Unsupervised satellite image segmentation by combining sa based fuzzy clustering with support vector machine,” *Int. Conf. on Advances in Pattern Recognition*, vol. 0, pp. 381–384, 2009.

[30] M. Surdeanu, J. Turmo, and A. Ageno, “A hybrid unsupervised approach for document clustering,” in *KDD ’05: Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery in Data Mining.* ACM, 2005, pp. 685–690.

[31] S. C. Madeira and A. L. Oliveira, “Biclustering algorithms for biological data analysis: A survey,” *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, vol. 1, pp. 24–45, 2004.

[32] G. Sheikholeslami, S. Chatterjee, and A. Zhang, “Wavecluster: a wavelet-based clustering approach for spatial data in very large databases,” *The VLDB J.*, vol. 8, no. 3-4, pp. 289–304, 2000.

[33] L. Shapiro and G. Stockman, *Computer Vision.* Prentice Hall, 2001.

[34] A. A. Yıldırım and C. Özdoğan, “Parallel wavecluster: A linear scaling parallel clustering algorithm implementation with application to very large datasets,” *J. of Parallel and Distributed Computing*, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2011.03.007

[35] A. A. Yıldırım and C. Özdoğan, “Parallel wavelet-based clustering algorithm on gpus using cuda,” *Procedia Computer Sci.*, vol. 3, pp. 396–400, 2011.

[36] A. Haar, “Zur theorie der orthogonalen funktionensysteme,” *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, 1910. [Online]. Available: http://dx.doi.org/10.1007/BF01456326

[37] N. Karunadasa and D. Ranasinghe, “Accelerating high performance applications with cuda and mpi,” in *Int. Conf. on Industrial and Information Systems (ICIIS)*, 2009, pp. 331 –336.

CHAPTER 4

A comparative study of the parallel wavelet-based clustering algorithm on 3 dimensional dataset[*]

4.1 Abstract

Cluster analysis – as a technique for grouping a set of objects into similar clusters – is an integral part of data analysis and has received wide interest among data mining specialists. The parallel wavelet-based clustering algorithm using Discrete Wavelet Transforms (DWT) has been shown to extract the approximation component of the input data on which objects of the clusters are detected based on the object connectivity property. However, this algorithm suffers from inefficient I/O operations and performance degradation due to redundant data processing. We address these issues to improve the parallel algorithm's efficiency and extend the algorithm further by investigating two merging techniques (both merge-table and priority-queue based approaches), and apply them on 3-dimensional data. In this study, we compare two parallel WaveCluster algorithms and a parallel K-means algorithm to evaluate the implemented algorithms' effectiveness.

4.2 Introduction

Cluster analysis is a widely-used technique that is employed to map a set of objects into groups (i.e., clusters) based on their similar properties, such as spatial and temporal similarity. Sheikholeslami et al. [1] introduced a novel unsupervised clustering approach, called WaveCluster, utilizing a discrete wavelet transform (DWT) that enables data analysts to perform clustering in a multi-level fashion. This method has the ability to discover clusters with arbitrary shapes and can deal with outliers effectively.

---

[*] Ahmet Artu Yıldırım, Daniel Watson

We previously reported a parallel wavelet-based clustering algorithm to process large datasets using a message-passing library (MPI) for distributed memory architectures [2]. While this algorithm achieves linear behaviour in terms of algorithm complexity, it suffers from inefficient I/O operations over 2-dimensional datasets and performance degradation due to redundant data processing. In this study, we introduce two parallel wavelet-based clustering algorithms that address these issues by benefiting collective MPI I/O capabilities and efficient usage of data structures. Additionally, we cluster 3-dimensional datasets with the new algorithms.

To illustrate the effectiveness of this approach, a comparison is performed between our two new parallel WaveCluster algorithms and parallel k-means clustering algorithm. Because one of the fundamental reasons of employing parallelism is to overcome space restrictions by exploiting aggregate memory on the distributed memory systems, and to obtain scalable performance, we take these two important metrics into consideration to measure the performance of the parallel algorithms.

4.3 Wavelet Transforms

Wavelet transforms (WT) are a mathematical technique to analyze non-stationary data in order to extract frequency information at different resolution scales from the original signal. WTs are commonly used in variety of areas such as in image compression [3], speech recognition [4], and in the analysis of DNA sequences [5]. WTs can be considered a complementary approach to Fourier transforms (FTs). One disadvantage of FTs is that they cannot determine which frequency band exists at any specific time interval in the signal. Short-time Fourier transforms (STFTs) might provide a remedy for time-frequency analysis by dividing the signal into successive segments and then performing an FT on each one, but STFTs suffer from the requirement of choosing the 'right' window width, where a narrow window leads to good time resolution but poor frequency resolution, and vice versa. A more detailed discussion of this effect can be found in [6–8].

The main idea of the Wavelet transform is based on the dilation and translation of the wavelet Ψ ('small wave' function) (i.e., the mother wavelet) continuously over the signal

$f(t)$ [9];

$$W_f(s,\tau) = \int_{-\infty}^{\infty} f(t)\Psi_{s,\tau}^{*}(t)\mathrm{d}t \tag{4.1}$$

The mother wavelet Ψ is defined as;

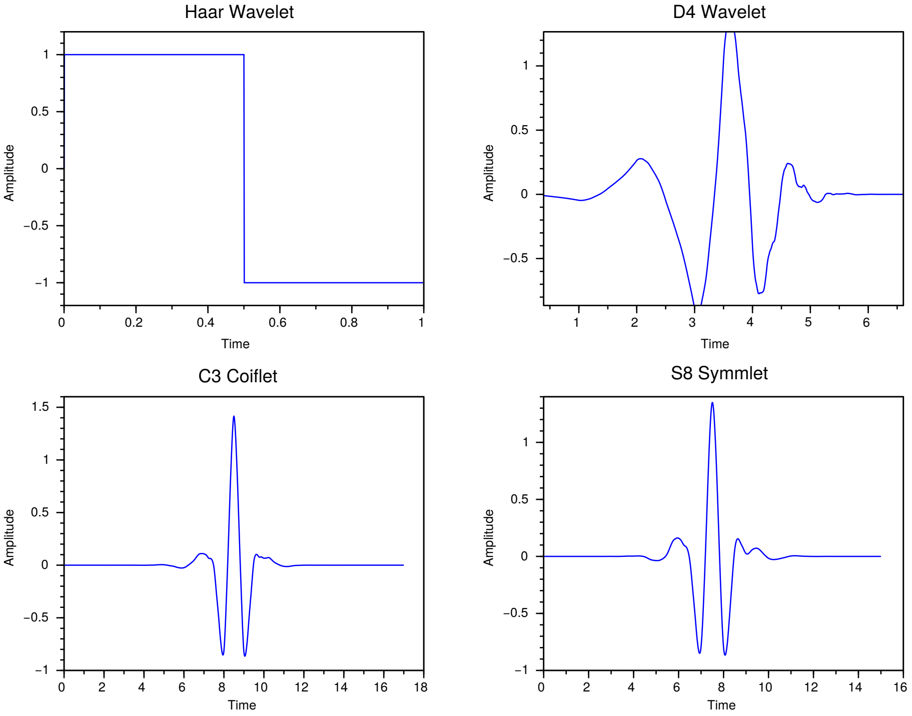
$$\Psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\Psi\left(\frac{t-\tau}{s}\right) \tag{4.2}$$

where s is the dilation or scaling factor determining the window width, and the factor τ manages the translation of the wavelet function Ψ. $\frac{1}{\sqrt{s}}$ is for energy normalization.

There are many mother wavelet functions, such as Haar, Daubechies, Morlet and Mexican hat. Some mother wavelets are depicted in Figure 4.1. In order to be regarded as a mother wavelet, the function must satisfy the two conditions in Equation 4.3a, which indicates the net area of the corresponding wavelet graph must be zero, but the absolute area cannot be zero (4.3b). Thus, it must be an oscillating function and have unit energy [10].

$$\int_{-\infty}^{\infty} \Psi(t)\,\mathrm{d}t = 0 \tag{4.3a}$$

$$\int_{-\infty}^{\infty} |\Psi(t)|^2\,\mathrm{d}t = 1 \tag{4.3b}$$


Haar Wavelet

D4 Wavelet

C3 Coiflet

S8 Symmlet

Figure 4.1: Some mother wavelet functions: Haar, Daubechies 4, C3 Coiflet, S8 Symlet

In this chapter, the approximation coefficients will be referred to as the ‘low-frequency component’, and the term of ‘high-frequency component’ will be used interchangeably as the detail coefficients of the signal. By means of the mother wavelet function, detail coefficients can be detected at different time intervals. However, we need another function used to retrieve average coefficients of the signal, which is referred to as the scaling function (i.e., the father wavelet) Φ. The father wavelet is orthogonal to the mother wavelet in that both wavelet functions form the basis for the multiresolution analysis. In fact, the father wavelet is not considered as a ‘wavelet function’ because it does not satisfy the condition (Equation 4.3a). However, by the combination of both wavelet functions, we gain the ability to decompose the signal into low-frequency and high-frequency components.

Algorithm 12 shows the procedure to compute discrete wavelet transforms (DWTs) that return the list of low-frequency components and high-frequency components. In DWTs, the size of the input signal must be power of two in each dimension to perfectly decompose

Algorithm 12 Discrete Wavelet Transform (DWT) algorithm

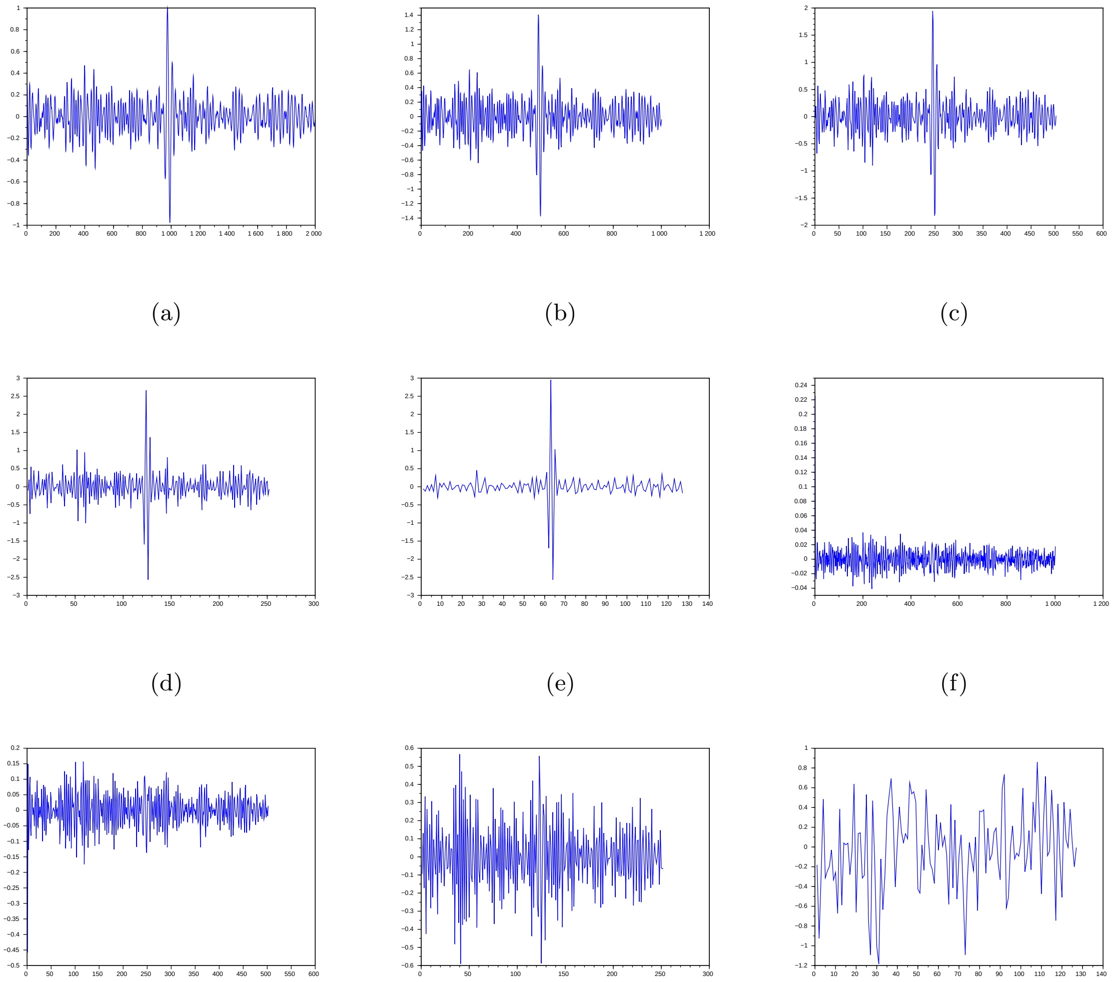
```
Require: Signal $X$ with size $N = 2^{level}$ where $level$ is integer
Ensure: List of low-frequency components $A$ and high-frequency components $D$
$A_0 \leftarrow X$
$D \leftarrow \emptyset$
for $j = 1 \rightarrow level$ do
    $[A_j, D_j] \leftarrow DWT_{\Phi,\Psi}(A_{j-1})$
end for
```

the signal, while this is not a requirement in continuous wavelet transforms (CWTs) due to continuous nature of the signal and the factors of s and τ. A_j denotes the approximation coefficients and D_j denotes detail coefficients at level j. After each level, the components of A_j and D_j are extracted using A_{j-1} in a recursive manner where A_0 refers to the original input signal.

Figure 4.2 shows the decomposition of a sample non-stationary signal by means of Daubechies wavelet function. Because the width of the wavelet window is doubled at each level, the time resolution is halved because of downsampling by two; thus we obtain coarser representations of the original signal; however, the frequency resolution is doubled in that the approximation components contain the lowest half of the frequency and the detail components take the other half.

4.4 Wavelet-based Clustering Algorithm

Cluster analysis is a widely-used technique that is employed to map a set of objects into groups (i.e., clusters) based on their similar properties, such as spatial and temporal similarity. Sheikholeslami et al. [1] introduced a novel unsupervised clustering approach, called WaveCluster, using DWTs that enable data analysts to perform clustering in a multi-level fashion. The method can discover clusters with arbitrary shapes and can deal with outliers (data points that don't belong to any cluster) effectively.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 4.2: Wavelet decomposition of a sample non-stationary signal. (a) Original signal (A_0). (b) Approximated coefficients at level 1. (A_1). (c) Approximated coefficients at level 2 (A_2). (d) Approximated coefficients at level 3 (A_3). (e) Approximated coefficients at level 4 (A_4). (f) Detailed coefficients at level 1 (D_1). (g) Detailed coefficients at level 2 (D_2). (h) Detailed coefficients at level 3 (D_3). (i) Detailed coefficients at level 4 (D_4).

WaveCluster defines the notion of a cluster as a dense region consisting of neighboring objects (in the 8-connected neighborhood) in the low-frequency component of the data at level j (A_j). The low-frequency component represents a lower resolution approximation of the original feature space on which connected component labeling algorithm is performed to detect clusters at different scales from fine to coarse. Hence, the clustering algorithm gains multi-resolution properties by means of the DWT. The corresponding algorithm with its multi-resolution property is illustrated in Figure 4.3. The algorithm discards detail coefficients and uses approximation values (low-frequency component) that are extracted by the low-pass filter operation $L[n]^j$ at level j. The algorithm applies thresholding to the approximation coefficients in the last level to remove the outlier data (i.e, *nodata* vertices whose values are less than threshold).

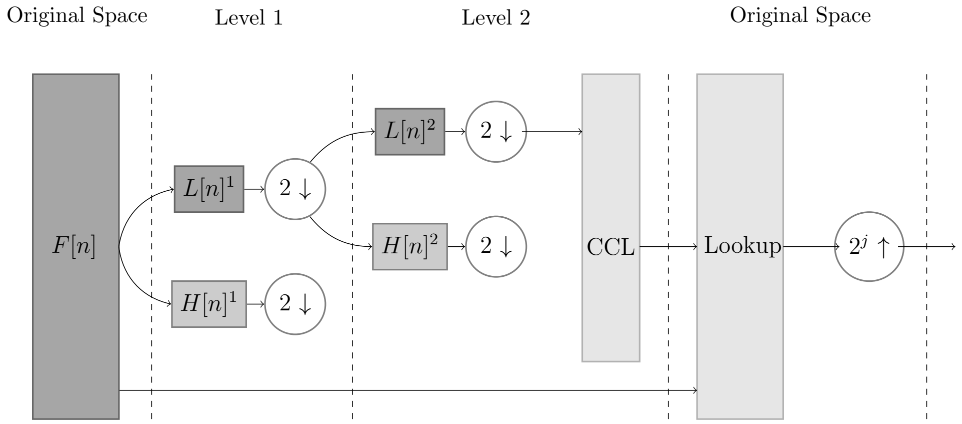


Figure 4.3: Multiresolution property of the WaveCluster algorithm at level 2 on 1 dimensional data

In our implementation, we use the Haar wavelet [11], whose scaling function is described as:

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

The Connected Component Labeling algorithm (CCL) is illustrated in Algorithm 13. The algorithm traverses through a list of vertices V in the WaveCluster algorithm such that the list is substituted by the low-frequency component of the input data. In the initialization phase between Line 1 and 4 of the algorithm, all vertices are marked as unvisited and their cluster numbers are set to a special value *nocluster* indicating no vertex is associated with any cluster. The algorithm uses stack S to avoid recursive calls and to keep track of the connected vertices. When there is no element in the stack (stack is empty) in Line 6, the algorithm finds another vertex that is not visited yet and has no *nodata* value. In Line 12, the algorithm pushes the adjacent vertices v_k to the stack where each vertex has no *nodata* value. Thus, all connected vertices are traversed and assigned a unique cluster number to represent distinct clusters over V. The algorithmic complexity of the CCL algorithm is $O(N)$ where N is the size of the input list.

Algorithm 13 Connected Component Labeling (CCL) algorithm

```
Require: $V$ is a list of vertices with size $N$ where the vertices $v_i, v_j, v_k \in V$ and vertex $v_k$
    is adjacent to $v_j$. $i$, $j$ and $k$ are indices that identify the vertices over $V$.
Ensure: $C$ stores the cluster numbers of the vertices where the value $nocluster$ indicates
    the vertex $v_i$ is not associated with any cluster.
 1: for $i = 1 \to N$ do
 2:     $visited[i] \leftarrow false$
 3:     $C[i] \leftarrow nocluster$
 4: end for
 5: $clusternumber \leftarrow 1$
 6: for all $v_i \in V$ where $visited[i] \neq true$ and $v_i \neq nodata$ do
 7:     $S \leftarrow push(S, v_i)$
 8:     while $S$ is not empty do
 9:         $v_j \leftarrow pop(S)$
10:         $visited[j] \leftarrow true$
11:         $C[j] \leftarrow clusternumber$
12:         $S \leftarrow push(S, v_k)$ where $v_k$ is neighbour of $v_j$ and $v_k \neq nodata$
13:     end while
14:     $clusternumber \leftarrow clusternumber + 1$
15: end for
```

In the final phase, mapping the units in the transformed feature space to the original feature space, a lookup procedure is carried out that one object in the low-frequency component at level j corresponds to $2^{(j \times d)}$ objects in the original signal where d is the dimension of the data. If the object is not represented in the transformed feature space (outlier), the object in the original space is assigned a *nocluster* value.

4.5 Parallel Wavelet-based Clustering Algorithms

We previously reported parallel wavelet-based clustering algorithm using a message-passing library (MPI) for the distributed memory architecture. While the algorithm possesses linear behaviour in terms of algorithm complexity, it suffers from the inefficient I/O operations and performance degradation due to redundant data processing. In this study, we address those issues by improving two parallel WaveCluster algorithms that are based on merge tables and priory-queues, respectively. The two parallel algorithms differ from the way they handle the merging phase, which is fundamental in the context of parallelism.

In the previous algorithm [2], the master processor reads the whole 2-dimensional input dataset and then distributes each stripe of data to the slave processors. This approach leads to inefficient I/O operation. In order to minimize I/O times, both algorithms benefit from collective MPI I/O capabilities in which each processor reads its local data in parallel collectively via *MPI_File_Read* function. We also extend the study by processing larger and 3-dimensional datasets.

4.5.1 Priority-queue Based Parallel WaveCluster Algorithm

Algorithm 14 Parallel WaveCluster Algorithm using priority-queue structure; where ρ is target wavelet level and i denotes MPI process number i, that is, $A_{j,i}$ is the partition of low-frequency component in level j, C_i is the output of the CCL algorithm at the first invocation, and then the output of $Merge$ algorithm in loop, CR_i is the final clustering result of partition i, all processed by MPI processes in parallel

```
$A_{0,i} \leftarrow loadLocalDomain\,(A_0,\ i)$
$A_{\rho,i} \leftarrow DWT(A_{0,i},\ \rho,\ threshold)$
$C_i \leftarrow CCL(A_{\rho,i})$
repeat
    $G_i \leftarrow swapGhostRegion()$
    $C_i \leftarrow Merge(C_i, G_i)$
until $no\ cluster\ value\ is\ changed\ globally$
$CR_i \leftarrow lookup(A_{0,i},\ C_i,\ \rho)$
$writeClusteringResult(CR_i, i)$
```

We implemented a new parallel WaveCluster algorithm whose merging phase is performed using priority-queue data structure. As algorithmic improvements, in the merging phase of the previous parallel WaveCluster algorithm [2], the master processor creates the merging table with respect to the adjacency relations of bordering data for each slave processor. Then, the algorithm distributes the result that each processor updates its local clustering numbers using the merging table. This approach leads to inefficiency due to the idle time of slave processors during the execution of merging phase at master processor. We employ cooperative merging operation between adjacent processors to alleviate this execution inefficiency. In this approach, each processor maintains its local clustering result. Ultimately, this leads to a globally correct result as ghost regions are swapped among adjacent processors.

The main algorithm is given in Algorithm 14. All operations in the algorithm figure are performed in parallel. The partitions of $A_0 = A_{0,1} \cup A_{0,2} \cup \ldots \cup A_{0,n}$ are distributed evenly among the MPI processes in a striped fashion where n is the number of MPI processes. Each process performs discrete wavelet transform using its own input partition data independently and returns the local low-frequency component with level ρ in Line 2. The value of input vertex over A_0 has either 1 or 0 that indicates whether there is a vertex or not at a particular index at A_0. The algorithm uses *threshold* value to remove the outlier vertices. The algorithm considers all vertices in the domain to be 'outliers' when threshold value is 1. When the value is 0, the algorithm does not discard any vertex from the domain. In Line 3 CCL function finds the connected vertices using local transformed feature space.

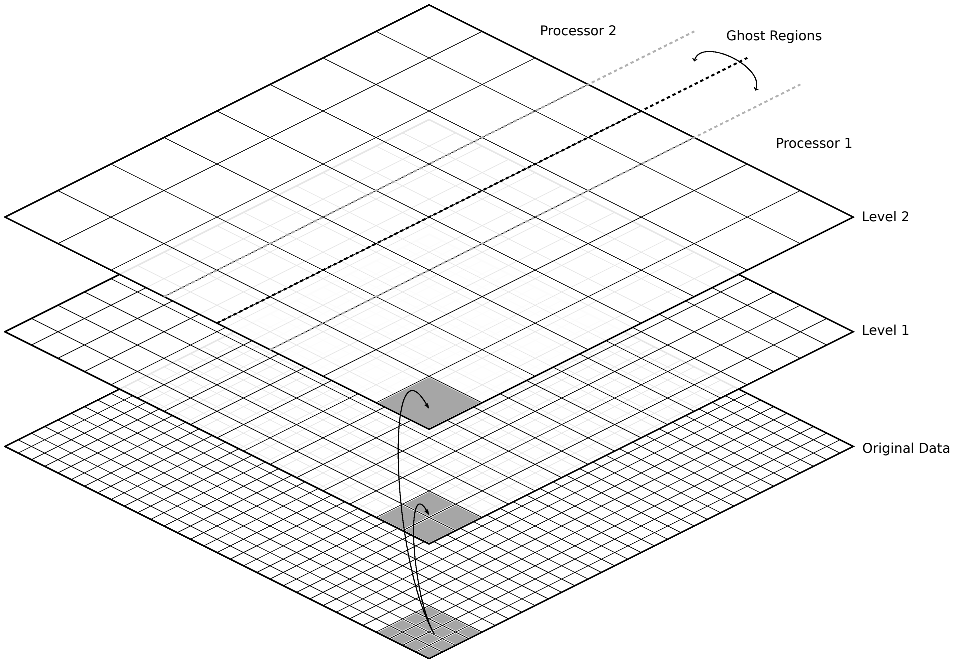


Figure 4.4: Parallel WaveCluster Algorithm with ghost data illustration using 2 processors

Subsequent calls, between Lines 4 and 7, deal with merging of the connected vertices that might possibly run through the process borders. The parallel merge phase is performed to merge the clusters globally, and propagates the minimum cluster number through the connected vertices of the local domain. In this phase, the parallel algorithm first calls *swapGhostRegions* in Line 5, whose goal is to exchange ghost data among the neighbouring processes, as illustrated in Figure 4.4. The ghost region is a low-frequency component with 1-vertex thickness that is adjacent to the local low-frequency component of the neighbouring MPI process. By swapping the ghost regions, the algorithm propagates the minimum cluster number of the connected vertices through the processes, which possibly can span the whole domain. Then, the process calls *merge* function in Line 6 to merge the clusters in the local transformed domain using the latest ghost data.

In the *merge* function (shown in detail in Algorithm 15), we use priority queue structure PQ to retrieve the minimum cluster number on the ghost data (Line 7), and stack S to propagate the cluster number among the connected vertices. If the cluster number of the popped ghost vertex is smaller than the neighbouring vertex of the local domain (Line 12),

Algorithm 15 Priority-queue based cluster merging algorithm

Require: C is a list of local cluster numbers of the vertices with size N where cluster numbers of c_i, c_j, c_k, c_g, $\in C$ for the vertices v_i, v_j, v_k and v_g respectively. G is the ghost region retrieved from the neighbouring MPI processes where the vertex $v_g \in G$, PQ is the priority queue structure whose function $pop(PQ)$ returns and removes the vertex with minimum cluster number at the priority queue, S is stack structure.

Ensure: The algorithm returns the updated list of clustering result C

```
$ischanged \leftarrow false$
for $i = 1 \rightarrow N$ do
    $visited[i] \leftarrow false$
end for
$PQ \leftarrow push(PQ, G)$ where $v_g \in G$ and $c_g \neq nocluster$
while $PQ$ is not empty do
    $v_g \leftarrow pop(PQ)$
    $S \leftarrow push(S, c_i)$ where $c_i$ is neighbour of $v_g$ and $c_i \neq nocluster$
    while $S$ is not empty do
        $c_j \leftarrow pop(S)$
        $visited[j] \leftarrow true$
        if $c_k < c_j$ then
            $C[j] \leftarrow c_k$
            $ischanged \leftarrow true$
            $S \leftarrow push(S, c_k)$ where $v_k$ is neighbour of $v_j$ and $c_k \neq nocluster$
        end if
    end while
end while
```

the merge function updates all the connected vertices' cluster numbers, and then marks those vertices as visited –which insures that those vertices are never visited for one iteration of the merging phase only. This merging phase runs until all clusters are detected globally, keeping track with the variable *ischanged* locally and by reducing all *ischanged* variables to the single *globalischanged* variable via *MPI_Reduce* function globally.

Finally, the lookup procedure is called in Algorithm 14 in Line 8 to map the vertices in the transformed feature space to the original feature space. Those that are considered 'outliers' based on *threshold* value, are marked with special *nocluster* value, and the results are written to the disk in parallel.

4.5.2 Merge-table Based Parallel WaveCluster Algorithm

The second parallel WaveCluster algorithm is based on merge-table which is similar to the previous algorithm introduced in [2]. We observe that memory consumption on the merging phase is too high due to data sparsity because of the nature of lookup table usage with $O(N)$ space complexity where N is the number of all vertices on input dataset. We addressed this memory consumption issue using a hash table storing only the cluster numbers of vertices which do not have *nodata* values.

The main phases of the merge-table based WaveCluster algorithm are shown on Algorithm 16. The merge table is the data structure that stores one record for each vertex and each record associates the initial cluster number with the final cluster number. At each iteration to merge the local clustering results, the algorithm repeatedly propagates the minimum cluster number through the connected vertices in the local transformed domain. The algorithm maintains the merge table using a hash table rather than the lookup table used in the previous implementation to keep track of the proposed cluster number that is smaller than the current cluster number. The hash table key is the initial cluster number and the associated values are the proposed/final cluster number and the coordinate information of the vertex residing on the ghost data. This hash table is initialized before the MPI process starts performing the merging phase. In subsequent iterations, this merge table is updated based on the ghost data in $updateMergeTable$ function. The merging phase continues until no changes occur in the merge tables globally. Finally, using this merge table, each process changes the cluster numbers of the connected vertices to reflect the correct cluster numbers in $updateClusterNumbers$, an operation performed using a stack data structure.

4.6 Performance Evaluation

In this section, we present the performance comparisons of the two parallel WaveCluster algorithms by means of the elapsed algorithm time and their speed-up ratios for synthetically generated 3-dimensional datasets. We implemented a synthetic data generation program that allowed us to minimize application-specific artifacts and concentrate more on properties and benefits of proposed algorithms. Generated input datasets have equal length for each

Algorithm 16 Merge-table based Parallel WaveCluster Algorithm; where MT is the merge-table

```
$A_{0,i} \leftarrow loadLocalDomain\ (A_0,\ i)$
$A_{\rho,i} \leftarrow DWT(A_{0,i},\ \rho,\ threshold)$
$C_i \leftarrow CCL(A_{\rho,i})$
$MT = initializeMergeTable(C_i)$
repeat
    $G_i \leftarrow swapGhostRegion()$
    $MT \leftarrow updateMergeTable(G_i, MT)$
until $any\ merge\ table\ is\ not\ updated\ globally$
$C_i \leftarrow updateClusterNumbers(C_i, MT)$
$CR_i \leftarrow lookup(A_{0,i},\ C_i,\ \rho)$
$writeClusteringResult(CR_i, i)$
```

three dimension and power of two as a condition of discrete wavelet transform to fully exploit multi-resolution analysis. The vertex value of dataset is either 1 with probability 0.3 or 0. Thus, the objects over the dataset are evenly distributed. The discussion about the effects of data distribution on the parallel WaveCluster algorithm can be found in [2]. In these experiments, we generated three datasets named Dataset1, Dataset2 and Dataset3 where number of objects –vertices with value 1– are nearly 40K, 80K and 161K, respectively. The details of the datasets are shown in Table 4.1.

Table 4.1: Datasets used in the experiments

Name	Size	Number of Objects
Dataset1	512 MB	40,268,934
Dataset2	1 GB	80,536,099
Dataset3	2 GB	161,067,172

We compare the parallel WaveCluster algorithm that has better benchmark results with the common parallel clustering algorithm – parallel K-Means algorithm. The experiments are conducted on a cluster where each node is equipped with two Quad-Core AMD Opteron(tm) Processor 2376 (8 total cores/node) and 16 GB memory. The interconnection among the nodes is achieved over double data rate (DDR) infiniband. The main performance measures that we consider are then running time of the parallel algorithms and their speed-up ratio with respect to serial execution of the algorithm.

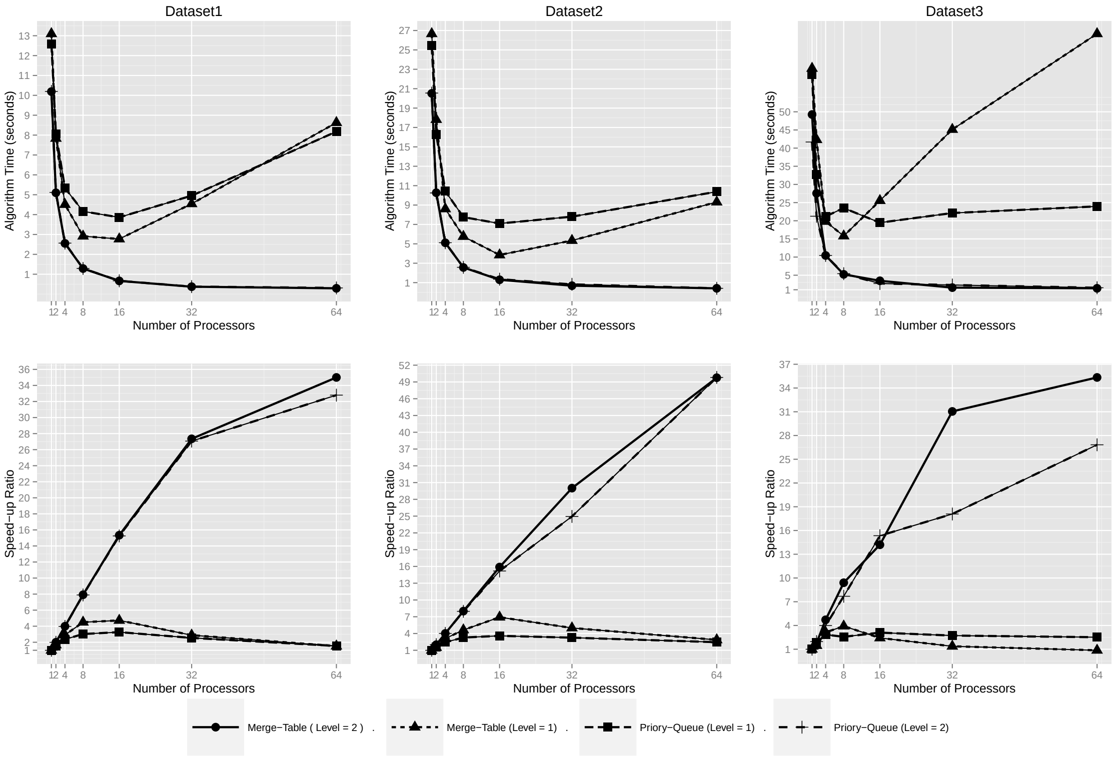


Figure 4.5: Parallel WaveCluster Algorithms with varying input dataset files and wavelet levels

In both parallel WaveCluster algorithms, the input on the original space is initially evenly distributed among p processors in a striped fashion. Figure 4.5 shows the benchmark results of the two parallel WaveCluster algorithms with different merging approaches called the priority-queue approach and the merge-table approach. The experiments are conducted for wavelet levels 1 and 2 for all datasets. Both parallel WaveCluster algorithms do not scale well for wavelet level 1 because of the high communication time in exchanging the border data. Furthermore, the run time of the algorithm starts to increase when more than 16 processors are used for Dataset1 and Dataset2, and 8 processors for Dataset3. The reason is that the communication time dominates the computation time that is required to obtain a globally correct result. Although the algorithms do not pose effectiveness in the context of parallelism for wavelet level 1, we did not observe this behaviour when the wavelet level is 2.

Note that the number of objects to be exchanged between the neighboring processes are decreased by factor of 2^d at each level where d is the dimension of the input dataset. In our experiments, the communication time is decreased by a factor of 8 as wavelet level increases. The communication overhead is reduced with the cost of coarser clustering analysis of original data. This shows that the wavelet level highly significantly affects the scaling behaviour of the parallel algorithm due to adverse effect of communication time. While we obtain identical speed-up results on Dataset1 and Dataset2, in the largest one, Dataset3, the merging approach performs better than the priority-queue approach when the wavelet level is 2.

We performed comparison experiments between the two parallel WaveCluster algorithms and the parallel K-means algorithm [12]. We have chosen the parallel K-means algorithm as a representative of a classical parallel clustering algorithm. We define the speedup in comparisons as a fraction of the execution times of the parallel K-means algorithm relative to the parallel WaveCluster algorithm for a varying number of processors and wavelet levels. The speed-up equation is shown below:

$$S_p = \frac{T_{pkmeans,\, p}}{T_{pwavecluster,\, p}} \tag{4.5}$$

Table 4.2: Execution times (in seconds) of the parallel WaveCluster (PWC) algorithms using priority queue (PQ) and merge table (MT) approaches for wavelet levels (ρ) 1 and 2, and the parallel K-means algorithm with a varying number of processors (np) on Dataset1

Exec. Time/np	1	2	4	8	16	32	64
P. K-means	2387.46	395.15	144.31	98.53	45.67	24.98	20.01
PWC-PQ ($\rho = 1$)	12.59	8.06	5.33	4.16	3.86	4.95	8.17
PWC-PQ ($\rho = 2$)	10.19	5.11	2.56	1.29	0.67	0.37	0.31
PWC-MT ($\rho = 1$)	13.10	7.83	4.50	2.91	2.77	4.54	8.62
PWC-MT ($\rho = 2$)	10.18	5.09	2.55	1.28	0.66	0.37	0.29

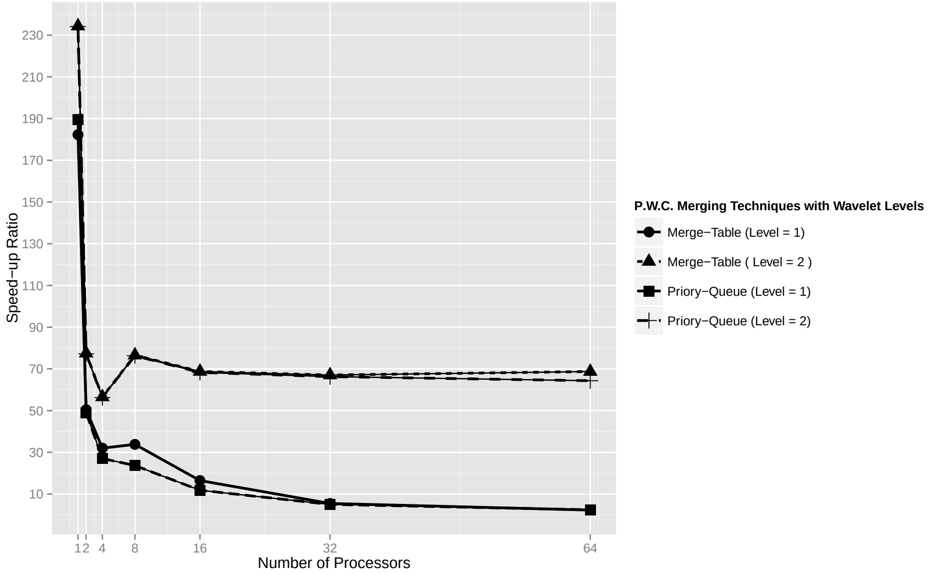


Figure 4.6: Speed-up Comparison between Parallel WaveCluster Algorithms and Parallel K-Means algorithm; where wavelet levels 1, 2 on Dataset1, $K = 20$

The parallel K-means algorithm must be configured to detect K distinct clusters where K is fixed. However, the number of clusters detected by the WaveCluster algorithm is determined by the wavelet level and the threshold value which is used to remove outlier objects that do not belong to any clusters. The performance of both parallel WaveCluster algorithms can be better seen in Table 4.2 for varying processors and wavelet levels. The execution time of parallel K-means algorithm are also included. Note that both parallel WaveCluster algorithms perform significantly better than parallel K-means algorithm on finding clusters. Figure 4.6 shows that both parallel WaveCluster algorithms are nearly 70 times faster than the parallel K-means algorithm on wavelet level 2 and 8 times faster on wavelet level 1 where maximum number of processors are utilized. We choose the parameter $K = 20$ in the experiments for K-means algorithm. Because the speed-up values are measured in terms of parallel performance of the algorithms $(\frac{T_{pkmeans,\, p}}{T_{pwavecluster,\, p}})$, a slight speed-up drop occurs because parallel K-means algorithm (to which our algorithm is compared) experiences performance comparatively greater increase when 4 processors are used – even though the wall clock time decreases in both cases. This result shows the effectiveness of the parallel WaveCluster algorithm when compared to the parallel K-means algorithm.

4.7 Conclusion

Parallel WaveCluster algorithms with their detailed description and comparison study have been presented here. We have improved upon previous work by investigating two different merging techniques, namely priority-queue and merge-table approaches that play important role in the parallel algorithm. These parallel algorithms find distinct clusters using discrete wavelet transformation on distributed memory architectures using the MPI API. Due to high compute times, we did not obtain scalability when wavelet level is 1. However, the scaling behavior is acquired for wavelet level 2.

We have chosen parallel K-means algorithm as a classical parallel clustering algorithm to compare the performance of two parallel WaveCluster algorithms. As an inherent property of the WaveCluster algorithm opposed to the K-means algorithm, it is capable of removing outlier objects that do not belong to any clusters. The effectiveness of the parallel WaveCluster algorithms as compared to the parallel K-means algorithm are shown in the study to have achieved up to 70x speed-up ratio where wavelet level is 2 on Dataset1.

REFERENCES

[1] G. Sheikholeslami, S. Chatterjee, and A. Zhang, “Wavecluster: a wavelet-based clustering approach for spatial data in very large databases,” *The VLDB J.*, vol. 8, no. 3-4, pp. 289–304, 2000.

[2] A. A. Yıldırım and C. Özdoğan, “Parallel wavecluster: A linear scaling parallel clustering algorithm implementation with application to very large datasets,” *J. of Parallel and Distributed Computing*, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2011.03.007

[3] A. S. Lewis and G. Knowles, “Image compression using the 2-d wavelet transform,” *IEEE Trans. on Image Processing*, vol. 1, no. 2, pp. 244–250, 1992.

[4] Z. Tufekci and J. Gowdy, "Feature extraction using discrete wavelet transform for speech recognition," in *Southeastcon 2000. Proc. IEEE*, 2000, pp. 116–123.

[5] A. Arneodo, E. Bacry, P. V. Graves, and J. F. Muzy, "Characterizing Long-Range Correlations in DNA Sequences from Wavelet Analysis," *Physical Review Letters*, vol. 74, pp. 3293–3296, Apr. 1995.

[6] I. Shim, J. J. Soraghan, and W. Siew, "Detection of pd utilizing digital signal processing methods. part 3: Open-loop noise reduction," *Electrical Insulation Magazine, IEEE*, vol. 17, no. 1, pp. 6–13, 2001.

[7] L. Cohen, "The uncertainty principles of windowed wave functions," *Optics Communications*, vol. 179, no. 16, pp. 221 – 229, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0030401800004545

[8] P. Loughlin and L. Cohen, "The uncertainty principle: global, local, or both?" *IEEE Trans. on Signal Processing*, vol. 52, no. 5, pp. 1218–1227, 2004.

[9] C. Valens, "A really friendly guide to wavelets," *C. Valens@ mindless. com*, vol. 2004, 1999.

[10] C. Torrence and G. P. Compo, "A practical guide to wavelet analysis," *Bull. Am. Meteorological Soc.*, vol. 79, pp. 61–78, 1998.

[11] A. Haar, "Zur theorie der orthogonalen funktionensysteme," *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, 1910. [Online]. Available: http://dx.doi.org/10.1007/BF01456326

[12] Y. Liu, J. Pisharath, W.-k. Liao, G. Memik, A. Choudhary, and P. Dubey, "Performance evaluation and characterization of scalable data mining algorithms," in *Proc. IASTED*, 2004, note: Source code can be found in http://users.eecs.northwestern.edu/∼wkliao/Kmeans/.

CHAPTER 5

HydroGate: A Service-oriented Gateway to HPC Resources to Facilitate Environmental Research

5.1 Abstract

Environmental researchers, modelers, water managers, and users often require access to high performance computing (HPC) resources for running data and computationally intensive models without being an HPC expert. To remedy this, science web portals have been created that integrate scientific models, data analysis, and tools to visualize results via web browsers. in this study, we present a general-purpose science gateway service named HydroGate created to access heterogeneous HPC storage and computational resources. HydroGate abstracts away many details and complexities involved in the use of HPC systems including authentication, authorization, data and job management, and encourages the sharing of scientific applications and promotes collaboration among scientists via unified access to HPC centers. HydroGate has been developed as part of the CI-WATER project which aims to broaden the application of cyberinfrastructure (CI) and HPC techniques into the domain of integrated water resources modeling. HydroGate is capable of dealing with various batch job schedulers and CI storage systems through a JSON-based manifest.

5.2 Introduction

Cyberinfrastructure provides scientists a set of middleware services to access software tools and data for collaborative scientific research over the high-speed network [1]. This work is part of the CI-Water project that has developed cyberinfrastructure to enhance access to data and high performance computing for water resources modeling. We present a science gateway web service named HydroGate that facilitates the ability of environmental researchers, modelers, water managers, and users to benefit from high-performance computing (HPC) resources for running computationally intensive models without an HPC expert. Integration of data, hardware, simulation models and software tools to visualize and disseminate results impact science into policy, management, and decisions. In cyberinfrastructure, state-of-the-art high-performance computing (HPC) facilities play important role for running computationally or memory intensive simulations.

At the core of cyberinfrastructure system as well as in grid computing [2], Service-Oriented Architectures (SOAs) play an important role whose goal is to achieve loose coupling among interacting software agents [3]. With a uniform interface using standard HTTP methods (i.e. *PUT*, *GET*, *POST*, *DELETE*) and a lightweight infrastructure, Restful web services [4] encourage widespread adoption among gateway designers [5,6].

In this study, we present a science gateway for heterogeneous HPC storage and computational resources. This gateway software, named HydroGate, is a Restful web service that takes input via HTTP methods, and then performs tasks on the selected HPC center on behalf of the web service user. HydroGate abstracts away many details and complexities involved in the use of HPC systems including authentication, authorization, data and job management via unified access to HPC centers. It virtualizes a variety of storage and job scheduler systems in which the web service user accesses all these resources through a unified web interface and uses a system-independent JSON manifest to describe the HPC jobs.

5.2.1 Problem Description

1. HPC centers have a mix of heterogeneous job resource managers and scripts used to define HPC job parameters (i.e., job limits, program parameters, and number of compute nodes and processors to utilize) and security requirements in accessing the resources. It is an important feature for a HPC gateway to support widely-used software systems on HPC centers and to allow gateway users to access HPC

centers' computing and storage resources through a unified application protocol and job description language in a secure fashion.

2. Storage systems are utilized in cyberinfrastructure systems that provide durable, secure and scalable scientific data storage to compute services over the network.

3. Another challenge [7] is that a HPC gateway service should enable scientific applications to be rapidly deployed on computational resources, exposing these capabilities as a web service to science web portals and applications.

4. Optimization is a desirable feature in handling high traffic for gateway services. We optimize data retrieval operations by transferring large package files without staging files via data streaming directly to HPC centers. This approach avoids the need to cache the file before transferring, thus eliminating redundant I/O.

5.2.2 Our Solutions to the Problems

1. HydroGate provides a single point of access to the HPC centers with PBS [8] and SLURM [9] job resource managers using a JSON-based manifest to define HPC jobs in which the data communication between HydroGate and HPC centers is achieved using a secure shell (SSH) [10] for secure remote login and remote command execution.

2. HydroGate transfers the input files required for HPC execution from these cyberinfrastructure storage servers to the HPC center's storage servers. The input files are compressed in a zip format which is termed a *package* in the context of *CI-Water*. An input package can reside on a local disk, an SFTP server, a web server, a distributed file system manager, or online cloud storage system. HydroGate recognizes the underlying storage protocol using the URL's scheme and begins transferring files over the secure shell (SSH) to HPC centers.

3. Scientific applications are installed in the central location of HPC center that HydroGate authorize users to access these applications in a unified fashion. Each scientific

application is installed once on the HPC center and then is used by all accounts. Besides, no HydroGate software component need be installed – particularly on HPC systems – other than scientific programs (zero-installation philosophy). This is a key feature of HydroGate that distunguishes it from other toolkits [5, 6, 11].

4. HydroGate is a database-driven web service, where it holds the records of HPC centers, service users, program parameters, HPC jobs, package information, etc., and establish many connections to HPC centers using secure shell (SSH) where the establishment of connections to HPC centers as well as to the database server are expensive. We implement resource pooling mechanisms for the intent of minimizing connection resources on memory.

5.2.3 Functionalities of the HydroGate

HydroGate utilizes the secure shell (SSH) protocol which helps standardize access to HPC centers. It provides flexibility, interoperability and easy integration with new HPC resources. The core functionalities of HydroGate are listed below:

- **Security** using token-based authentication to the HydroGate service and SSH-based authentication to HPC centers
- **File transfer** back and forth between HPC storage and CI storage, which is transparent to the service user
- **Submission of jobs** that the user has permission to perform on the specified HPC center
- **Monitoring of job status** by means of a URL callback mechanism, carried out by HydroGate to avoid requiring end users to poll job status continuously, that notifies service users when the status of job is changed and the output package is transferred to CI storage
- **Automatic batch script generation** based on the HPC center preferences and program requirements using a JSON-based manifest

- **Discovery** functions to determine the capabilities of HPC centers, HPC programs, program parameters and so on that the gateway user can query.

5.3 Related Works

Gateway software services have been proved as a way to facilitate interaction and integration with the complex systems for the service users. Although the feature of loose coupling is necessary in building a successful web service due to the interoperability nature of the Web, gateway services also impose a well-defined software architecture and protocol to communicate with the underlying heterogeneous systems.

Unicore [11] aims to provide a seamless batch interface for German HPC centers in a Grid Computing Environment. The goals of building such a software include allowing the service users to create, manipulate and control complex batch jobs for the execution at heterogeneous systems, use of existing and emerging technologies with minimal intrusion into existing computing practices and policies. However, because Unicore relies on a tightly-coupled architecture, Unicore clients requires its server components to be installed on HPC centers.

NEWT [5] is a web service that is used to access HPC resources in a web browser environment using AJAX and JSON at the National Energy Research Scientific Computing (NERSC) center. It exposes the similar resources as URIs with HydroGate including authentication, files/directories, batch jobs and UNIX shell commands. NEWT provides a fine level of access to the resources, for example, it allows the service clients to download and upload files at a given URI which are mapped to actual filesystem resources on the backend systems and allows to execute jobs given parameters. Although this approach might lead to more flexible access to the resources at HPC centers, without a higher level of abstraction, it might increase the complexity in the integration of future HPC centers due to the heterogeneous nature of these systems. Similar to UNICORE, NEWS also relies on software components to be installed on server side, which is Globus grid toolkit [12] for accessing compute and data resources.

GISolve [13] provides a set of REST Web APIs for CyberGIS [14] authentication, application integration, and cyberinfrastructure-based computation. As an architectural difference with HydroGate, GISolve is a gateway as a widely accessible platform mainly benefiting end-users to access CyberGIS resources, on contrary, HydroGate advocates a diverse architecture that might contain a various types of software systems and protocols in order to be used by scientific portal applications through an unified interface.

5.4 Science Grid Gateway to HPC Centers

Accessing and using High Performance Computer (HPC) centers pose inherent challenges for non HPC specialists. HPC users typically perform authentication, data transfers, program installation and job management using a terminal user interface and difficult to use commands whose communication is established over secure shell (SSH). To remedy this problem, science web portals [13, 15, 16] have been introduced that integrate scientific models, data analysis and tools to visualize results via web browsers.

Grid computing provides a software abstraction layer that isolates all of the details through a unified interface to access heterogeneous computer systems (i.e. multiple HPC centers). The Globus Toolkit has emerged as a de Facto Standard for grid computing [2] by providing a set of tools for application programming (API) and software development kits (SDKs). However, given the steep learning curve for service commands and program interfaces to use Globus directly [17], challenges to integrating Globus with grid platforms and, difficulties in solving the sociological and institutional problems, researchers seek easy-to-use and concise APIs in the form of URLs to expose grid computing capabilities over the web [5, 6, 11, 18]. Moreover, HPC users are mostly obliged to use existing software tools installed on HPC centers that highly affects the design of HPC gateway services.

HydroGate is designed to transcend difficulties in accessing HPC centers via an unified RestFul interface and allows rapid integration of HPC centers with heterogeneous software systems for science web portals. In this model, administrators are in charge of installing programs onto HPC centers and entering necessary records to the HydroGate database to introduce the scientifi applications and the parameters to the system. In this section, we report the inner-workings of the web service and show a typical Cyberinfrastructure (CI) architecture that HydroGate is designed to run on.

5.5 Architecture

HydroGate has been developed as part of the CI-WATER project which aims to broaden the application of cyberinfrastructure and HPC techniques into the domain of integrated water resources modeling. Initial implementation of HydroGate is as part of the CI-WATER web portal which is the service user of HydroGate. However we believe that HydroGate can be utilized by not only water resource modeling, but can also serve as general-purpose grid middleware for scientific web portals/applications.

A number of web standards are used in science grid gateways such as Java servlet engine [19], OGSA [20], WS-Resource framework [21], SOAP [22] and WPS [23]. While each has its own advantages and disadvantages for grid gateway programmers, we adopt the Representational State Transfer (REST) architectural style that attempts to minimize latency and network communication, while maximizing the independence and scalability of component implementations [24]. RESTful web services also encourage the integration of web service and client applications through standard HTTP methods (i.e. GET, POST, PUT, DELETE).

Figure 5.1 shows a Cyberinfrastructure (CI) architecture that HydroGate is designed to run on. A CI user interracts with the CI system over a modern web browser. The web server passes a user's HPC-related requests to the HydroGate web service that performs on behalf of the user on the specified HPC center. The CI web server stores all the package files on a CI storage. There are several approaches to storing files on the CI storage including a local file system, distributed file system, SFTP server, SCP server or online cloud storage. The distributed file system that HydroGate can interoperate is iRods. iRods [25] is used for managing, sharing, publishing and preserving distributed data collections as a cyberinfrastructure. Data are distributed among a varying type of storage systems across geographically-dispersed data sources.

In order to submit a HPC job, a user defines a set of input files, a model/scientific program to run, parameters of the HPC job and optionally a HPC center that the user

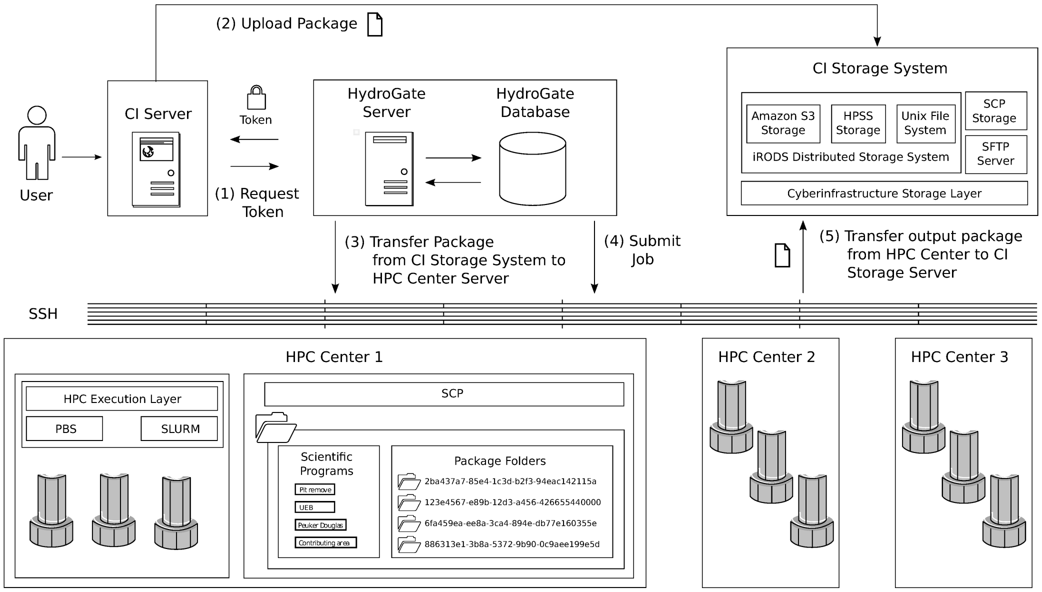

(2) Upload Package
CI Storage System
CI Server
Token
HydroGate Server
HydroGate Database
Amazon S3 Storage
HPSS Storage
Unix File System
SCP Storage
SFTP Server
iRODS Distributed Storage System
Cyberinfrastructure Storage Layer
User
(1) Request Token
(3) Transfer Package from CI Storage System to HPC Center Server
(4) Submit Job
(5) Transfer output package from HPC Center to CI Storage Server
SSH
HPC Center 1
HPC Center 2
HPC Center 3
HPC Execution Layer
PBS
SLURM
SCP
Scientific Programs
Pit remove
UEB
Peuker Douglas
Contributing area
Package Folders
2ba437a7-85e4-1c3d-b2f3-94eac142115a
123e4567-e89b-12d3-a456-426655440000
6fa459ea-ee8a-3ca4-894e-db77e160355e
886313e1-3b8a-5372-9b90-0c9aee199e5d

Figure 5.1: A Cyberinfrastructure architecture in submitting job to HPC centers using CI server, CI storage system, HydroGate and multiple HPC centers; (1) CI user requests token from HydroGate and submit a job (2) CI server uploads the input package to CI storage server or creates the input package using existing files on the storage (3) HydroGate transfers the input package to a specified HPC center (4) HydroGate creates job script on the HPC center (5) After the job is completed, HydroGate creates the output package on the HPC center and transfers it to the CI storage

is allowed to utilize, and ultimatelly submits a HPC job to the CI server using the web interface. Then, the CI server creates a zip file (a package) containing all required input files on the CI storage and authenticates to HydroGate through a token-based authentication mechanism. Job submission is carried on in two steps: Transferring the package from the CI storage to the HPC storage via *upload_package* function and describing the HPC job to Hydrogate (i.e. input package, program name, program parameters, number of compute nodes and processors to utilize etc.) via *submit_job* function. This approach provides us a package-reusing capability in which redundant package transfers are avoided for the HPC job requests using the same input files, but differ by job/program parameters. For example, the user can instruct the gateway to transfer the input package to a specified HPC center and then run the models on the uploaded package that differs by the parameters such as number of processors used during the run or program parameters. After the invocation of *submit_job* function, HydroGate creates a job script file based on the specifications of the underlying job scheduler, submits the job the HPC center and starts monitoring the job status. As the status of the job changes, HydoGate updates the related record on the HydroGate database. Whenever the job execution is completed on the HPC center, HydroGate creates the output package containing the output file of the job and automatically transfers the output package from the HPC center to the CI storage.

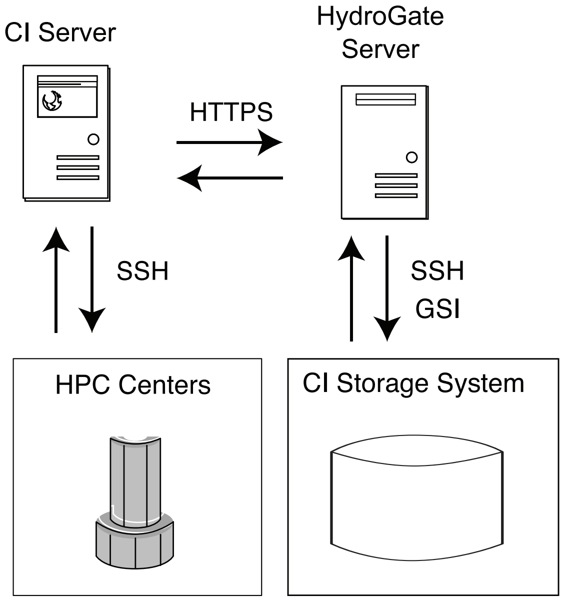


Figure 5.2: Security protocols among the software components in the architecture of HydroGate

Figure 5.2 shows the security protocols used in data transportation among the software components in the architecture. The Restful interface exposed by HydroGate is accessed over HTTPS protocol for secure transformation of the messages. Although we take advantage of SSH in accessing the HPC centers, the network protocol between HydroGate and the CI storage system depends on the underlying storage software system. If the CI storage is consists of SCP or SFTP server, SSH protocol is used in exchanging data, however, if iRODS is utilized, HydroGate benefits Grid Security Infrastructure (GSI) for authentication.

5.5.1 Authentication

Authentication is an important aspect of the security in the design of web services. In a grid environment, it is required for a service to handle multiple security mechanisms [26]. HydroGate applies two authentication methods; the one required when a user accesses to the service using token-based authentication and the other one in accessing HPC resources using the secure shell (SSH) [27].

Given the user name and password identifying a user using *request_token* function over HTTPS channel, HydroGate signs and issues a security software token which is one-time password that is valid for a constant period of time to achieve higher security. This approach allows the CI server not to hold user credential information in memory during the use of service and simplifies the access to HydroGate.

HydroGate establishes the connection to the HPC centers using SSH-based authentication that is ubiquitous for encrypting and transferring data and commands over insecure network. Because establishing SSH connections is memory-intensive and relatively slow operation, we implement SSH pooling mechanism holding the existing connections that aims to reuse existing SSH connections whenever necessary with minimum SSH connections.

We have considered two ways for the gateway to authenticate science client applications; through an individual account that every science web portal user must have, or a general account in which there is one HPC account used in accessing a particular HPC center for all service users that have right to access. Because HydroGate modifies the structure of the home directory (i.e. creating package and job folders, installing scientific applications locally), the implications of files being changed in individual user accounts or a general account need to be evaluated. We implement general account approach where user records including credential information, access rights etc. are stored on HydroGate database. Considerations are:

1. In the case of an individual account, possible changes to files being used by HydroGate by the user working separately from HydroGate.

2. The burden associated with establishing an HPC user account.

3. Some CI users may be reluctant to share their HPC account with the science web portal.
4. Maintaining separate SSH connections for each user in the connection pool may be less efficient.

On the other hand, there are also the concerns of many HPC centers that they may want to know who is using their system that the individual account approach might be necessary in this case.

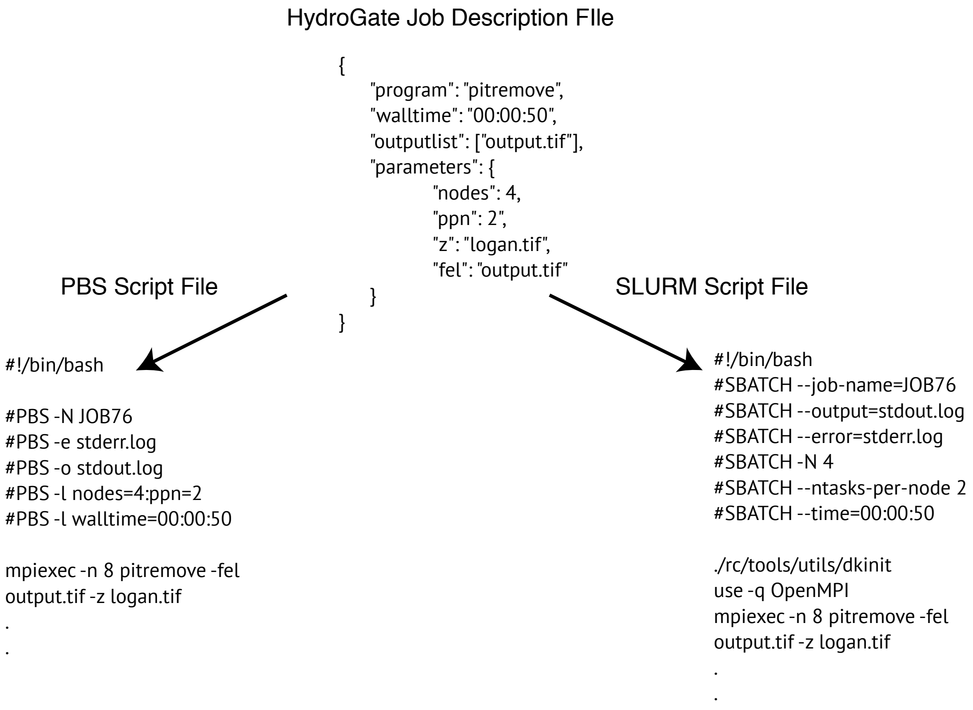
5.5.2 Job Description Language

Job scheduler systems are used to allocate computer cluster resources and schedule the execution of batch jobs on HPC centers. A typical HPC user is required to create a script file that contains job limits, executable path, program parameters etc. and then run the designated tool of the job scheduler to submit a job to the system by passing a script file as parameter.

We implement a general-purpose JSON-based HydroGate job description language (HJDL) that defines a batch job including programs to run, program parameters and scheduler-specific information. The HJDL language not only supports to describe a single batch job, but also capable to define multiple HJDL jobs called a *workflow* whose programs run sequentially defined in single script file.

HJDL script text is passed to *submit_job* functionas a parameter. Using HJDL script, HydroGate creates a corresponding job script file on a HPC center based on the specifications of the job scheduler system that can be either PBS or SLURM in the current version (Figure 5.3). We store HPC center specific parameters and template scripts on HydroGate database. This approach provides us flexibility in which each job scheduler system on a HPC center might define its own settings, for example, a queue name that the project is allowed to use and required modules needs to be loaded before the batch job runs.

There are a set of defined variables in the template files whose values are set by HydroGate when creating a job script on the file system of the HPC center. Definition of the



```
{
    "program": "pitremove",
    "walltime": "00:00:50",
    "outputlist": ["output.tif"],
    "parameters": {
            "nodes": 4,
            "ppn": 2",
            "z": "logan.tif",
            "fel": "output.tif"
    }
}
```

```
#!/bin/bash

#PBS -N JOB76
#PBS -e stderr.log
#PBS -o stdout.log
#PBS -l nodes=4:ppn=2
#PBS -l walltime=00:00:50

mpiexec -n 8 pitremove -fel
output.tif -z logan.tif
.
.
```

```
#!/bin/bash
#SBATCH --job-name=JOB76
#SBATCH --output=stdout.log
#SBATCH --error=stderr.log
#SBATCH -N 4
#SBATCH --ntasks-per-node 2
#SBATCH --time=00:00:50

./rc/tools/utils/dkinit
use -q OpenMPI
mpiexec -n 8 pitremove -fel
output.tif -z logan.tif
.
.
```

Figure 5.3: HydroGate Job Description Language (HJDL) script and the corresponding batch scripts of PBS and SLURM

Table 5.1: Template variables with their definition

Template Variable	**Definition**
$HGW_JOB_NAME	Uniquely assigned job name
$HGW_ERROR_LOG_FILE	File name of error log
$HGW_OUTPUT_LOG_FILE	File name of standard error
$HGW_NODES	Number of nodes
$HGW_PPN	Number of processor per node
$HGW_WALLTIME	Maximum allowed walltime
$HGW_RUNNER	Program runner or interpreter
$HGW_RUNNER_PARAM	Runner/Interpreter parameter
$HGW_PROGRAM	Path of the scientific program to run
$HGW_PROGRAM_PARAM	Program parameter
$HGW_RESULT_ZIP_FILE_NAME	Name of output package file
$HGW_RESULT_FILE_LIST	Output files contained in the output package

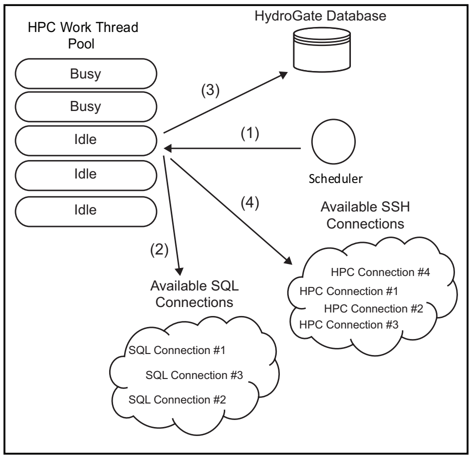
template variables is shown in Table 5.1. HydroGate is capable to run a variety of programs including Python programs and MPI programs. For example, to run a Python program on a HPC center, the variable of $HGW_RUNNER set to *python* which is interpreter of Python programs and $HGW_RUNNER_PARAM contains mostly empty string. However, in case of MPI, $HGW_RUNNER is *mpiexec* and $HGW_RUNNER_PARAM includes MPI options such as number of processors that the MPI program requires to run on. We show a sample PBS template file to describe single batch job in Listing 5.1.

Listing 5.1: PBS Template

```
#!/bin/sh
#PBS -N $HGW_JOB_NAME
#PBS -A CI-WATER
#PBS -e $HGW_ERROR_LOG_FILE
#PBS -o $HGW_OUTPUT_LOG_FILE
#PBS -l nodes=$HGW_NODES:ppn=$HGW_PPN
#PBS -l walltime=$HGW_WALLTIME

cd $PBS_O_WORKDIR

$HGW_RUNNER $HGW_RUNNER_PARAM $HGW_PROGRAM $HGW_PROGRAM_PARAM

rc=$?

[ $rc -ne 0 ] && echo $rc > $HGW_ERRORCODE_FILE && zip -q stderr.
    log stdout.log && exit $rc

echo 0 > $HGW_ERRORCODE_FILE
zip -q $HGW_RESULT_ZIP_FILE_NAME $HGW_RESULT_FILE_LIST
```

5.5.3 Resource Pooling Mechanism

HydroGate is a multi-threaded web service maintaining a thread pool with a number of work thread items that are managed by the scheduler component. The goal of this thread pool is to minimize the turnaround time. The scheduler passes compute and connection intensive operations to the work item. If the number of allowed work thread items is exceeded, the request is queued to the FIFO queue to be assigned later when an available work item exists. Currently, we take advantage of thread pool in transferring the package between HPC centers and CI storage systems and submitting batch job.



1. Scheduler directs the request to an available thread
2. Thread retrieves a ready-to-use SQL connection if available, otherwise pool component creates a connection
3. Thread obtains required information from HydroGate database
4. Thread retrieves an available SSH connection from SSH pool to connect to a HPC center. Otherwise it creates a SSH pool using the credential information obtained from HydroGate database

Figure 5.4: HydroGate Pooling Mechanism

Because HydroGate accesses HPC centers using the secure shell, these precious resources needs to be used effectively. To mitigate that, we maintain SSH resource pool containing HPC specific ready-to-use SSH connections. When an HPC connection is requested from the work thread pool for a HPC center, the SSH connection pool checks and then returns if the available connection exists. Otherwise, the SSH connection pool constructs the connection and adds it to the pool. This mechanism is shown in Figure 5.4. HydroGate also maintains a pool for SQL connections to be a fully capable web service with minimum operating system resources.

5.6 Case Study

In this section, we present a case study to demonstrate the usage and show its effectiveness in accessing HPC centers. We choose TauDEM (Terrain Analysis Using Digital Elevation Models) software suite [28] that is a set of Digital Elevation Model (DEM) tools for the extraction and analysis of hydrologic information from topography as represented by a DEM. TauDEM consists of a set of command line MPI [29] programs that are capable to run on HPC centers as well as desktop computers. We installed TauDEM programs on USU HPC center for demonstration. HydroGate is a database-driven gateway service. Thus, to introduce TauDEM programs to Hydrogate, we inserted required records of the TauDEM executables (etc. installation locations, parameters of the programs) on the HydoGate database.

In science, a workflow that defines a set of consecutive operations to achieve a particular objective, is common. HydroGate supports a workflow mechanism through its JSON-based job definition language (HJDL). We compute the contributing area of the selected region using *aread8* program. However, *aread8* requires a D8 flow directions grid as input that is produced by *d8flowdir* program, in turn, *d8flowdir* depends on *pitremove* program to obtain the hydrologically correct elevation grid. In this case, the workflow to generate the contributing area consists of a set of atomic operations to be executed on the HPC centers.

The following demonstrates the steps of the HydroGate web service functions calls to compute *aread8* workflow using USU HPC center which is using SLURM job scheduler system. We take advantage of iRODS storage system installed in Utah State University to store the packages in a distributed fashion. The web service functions are invoked using *curl* [30] command line tool. Variable naming conventions used in the listings is shown in Table 5.2.

1. **Token Request:** Service users need to obtain a security token to access the services of HydroGate. The token is only valid for a limited time and then expires. Given credential information as username and password, *request_token* function returns token string and expire duration in minutes. username parameter is the account name of a

Table 5.2: Variable naming conventions in the listings

Variable	**Definition**
$HOSTNAME	HydroGate host address
$TOKEN	Token string
$USERNAME	Account name of the user
$PASSWORD	Account password of the user
$JOBDEF	Batch job definition

service user that is managed by HydroGate. Error convention is the same throughout the returned JSON data in which if the value of *status* key is *failed*, *description* key shows the error information. However, the detailed error information is logged on the server for security concerns.

Listing 5.2: Token request script

```
Input:
curl –k –X POST ––data ”username=$USERNAME&password=$PASSWORD
    ” https://$HOSTNAME/hydrogate/request_token/

Output:
{”lifetime”:500,”status”:”success”,”token”:”$TOKEN”,”username
    ”:”$USERNAME”}
```

2. **Package Upload:** As a prerequisite of the batch submission, the package file containing the input files needs to be transferred from the CI storage –which is using iRODS in this case– to a HPC center storage. HydroGate follows a asynchronous processing model that *upload_package* returns immediately once the task is assigned to one of the thread in the thread pool. The *upload_package* function returns a package ID that identifies the package uniquely on the all HPC centers that HydroGate recognizes. The HPC center that the package is to be transferred to is given in *hpc* parameter. The *package* key contains the URL of the package that in this case, it's the location of the package in iRODS file system. HydroGate is also capable to transfer

the packages in various protocols (e.g *HTTP*, *HTTPS* and *FTPS*) that is recognized using the schema name the URL string.

Listing 5.3: Package transfer script

```
Input:
curl -k -X POST --data "token=$TOKEN&package=/usu/home/rods
    /75ec4ba2b75c316c64f4c6ba/dem31937741937071.tif.zip&hpc=
    USU" https://$HOSTNAME/hydrogate/upload_package/


Output:
{"packageid":"112","status":"success"}
```

3. **Check Package Status (Optional):** HydroGate has capabilities to inform the service user about the status of a package during the transfer in two ways. First, the user can poll the status by invoking *retrieve_package_status* function given package ID. This function returns the status as a string in *state* key of the response JSON data (e.g. *PackageInQueue*, *PackageTransferDone*, *PackageTransferError*) However, this approach might cause the congestion on the network and might not be an effective solution. Thus, we also implemented HTTP callback mechanism that given the client address, HydroGate passes the status and package id information in the form of query string by calling the URL.

Listing 5.4: Check package status script

```
Input:
curl -k -X GET "https://$HOSTNAME/hydrogate/
    retrieve_package_status?token=$TOKEN&packageid=112"


Output:
{"packageid":"112","status":"success"}
```

4. **Job Submission:** Once the input package is transferred to the HPC center, the user can submit job by describing the batch job in *HJDL* language. Because we separate the package and the job concept, It allows the service users to submit multiple jobs using the same package, but the jobs might differ by program and job parameters. This approach avoids redundant package transfer to the HPC center and provides effective usage of the HPC storage capacity.

   submit_job function requires the job definition which is passed in *jobdefinition* key. The contents of the *JOBDEF* variable in Listing 5.5 is shown in Listing 5.6. The tasks of the workflow are defined separately with a unique key value and its parameters and the sequence of the tasks are defined in *workflow* key. The file names that are given as parameters of the tasks exist in the package file. The *walltime* value is given as a expected total wall time of the workflow. Upon the completion of the workflow successfully, HydroGate creates a output package containing the files given in *outputlist* key and starts transferring the package to the CI storage automatically. The client user is able to check the status of the job via *retrieve_job_status* function by passing the unique job id returned from $submit_job$ function, or through URL callback mechanism.

Listing 5.5: Job submission script

```
Input:
curl −k −X POST −−data ”token=$TOKEN=112&jobdefinition=
    $JOBDEF” https://HOSTNAME/hydrogate/submit_job/


Output:
{”jobid”:”79”,”outputpath”:”/usu/home/rods/79/result.zip”,”
    status”:”success”}
```

Listing 5.6: Job description ($JOBDEF) used in the workflow to compute the contributing area

```
{
  "program": "myworkflow", "walltime": "00:05:00", "
      outputlist": ["d8flow.tif", "d8contarea.tif"],

  "task-pitremove" : {
         "program": "pitremove",
         "parameters": {"z": "logan.tif", "fel": "pitfilled.
             tif"}
  },

  "task-d8flowdir" : {
         "program": "d8flowdir",
         "parameters": {"fel": "pitfilled.tif", "sd8": "
             sd8slope.tif", "p": "d8flow.tif" }
  },

  "task-aread8" : {
         "program": "aread8", "parameters": {"p": "d8flow.tif"
             , "ad8": "d8contarea.tif"}
  },

  "workflow": ["task-pitremove", "task-d8flowdir", "task-
      aread8"],
}
```

5.7 Future Research

Access control for gateway systems is an important part in building HPC gateway services. HydroGate advocates to follow global account approach that one HPC account is necessary per HPC center. While this approach encourages the sharing of scientific applications and data, and provides an eco-system fully controlled by HydroGate, there might some necessities for some users in accessing to HPC centers with their account. Besides, using separate accounts might be required by the system administrators of HPC centers. We are planning to implement account-based access control in the future releases.

Currently, database records are added to the database without the aid of a client program. Thus, a tool is necessary to manage the records of HPC, program parameters and so on for better management of system.

5.8 Conclusion

We introduced HydroGate gateway service to facilitate environmental research using high-performance computing centers. HydroGate is designed to transcend difficulties in accessing heterogeneous storage systems and HPC centers via an unified RestFul interface and allows rapid integration of HPC centers with heterogeneous software systems for science web portals. We believe HPC gateway systems play an important role to make the complex HPC systems accessible for service clients by allowing them to execute compute and data intensive programs through an unified and elegant web interface.

In this chapter it has shown the capabilities of HydroGate and the case study to compute the contributing area have been demonstrated. We benefit USU HPC center that is using SLURM job scheduler system in the case study, but the concept is also proved to be effective on other HPC systems with PBS job scheduler.

REFERENCES

[1] T. Hey and A. E. Trefethen, "Cyberinfrastructure for e-science," *Science*, vol. 308, no. 5723, pp. 817–821, 2005. [Online]. Available: http://www.sciencemag.org/content/

308/5723/817.abstract

[2] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez *et al.*, *Introduction to Grid Computing with Globus.* IBM Corporation, Int. Technical Support Organization, 2003.

[3] H. He, "What is service-oriented architecture," *Publicação eletrônica em*, vol. 30, p. 50, 2003.

[4] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. big web services: Making the right architectural decision," in *Proc. 17th Int. Conf. on World Wide Web*, ser. WWW '08. ACM, 2008, pp. 805–814. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367606

[5] S. Cholia, D. Skinner, and J. Boverhof, "Newt: A restful service for building high performance computing web applications," in *Gateway Computing Environments Workshop (GCE), 2010*, Nov. 2010, pp. 1–11.

[6] S. Wang, M. Armstrong, J. Ni, and Y. Liu, "Gisolve: a grid-based problem solving environment for computationally intensive geographic information analysis," in *Proc. Challenges of Large Applications in Distributed Environments*, July 2005, pp. 3–12.

[7] W. Wu, T. Uram, M. Wilde, M. Hereld, and M. E. Papka, "Accelerating science gateway development with web 2.0 and swift," in *Proc. 2010 TeraGrid Conf.*, ser. TG '10. ACM, 2010, pp. 23:1–23:7. [Online]. Available: http://doi.acm.org/10.1145/1838574.1838597

[8] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten, "Portable batch system: External reference specification," Technical Rep., MRJ Technology Solutions, Tech. Rep., 1999.

[9] A. Yoo, M. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in

Computer Sci., D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Springer Berlin Heidelberg, 2003, vol. 2862, pp. 44–60.

[10] T. Ylonen and C. Lonvick, "The secure shell (ssh) protocol architecture," The Internet Society, Tech. Rep., 2006. [Online]. Available: https://tools.ietf.org/html/rfc4251

[11] D. W. Erwin and D. F. Snelling, "Unicore: A grid computing environment," in *Euro-Par 2001 Parallel Processing*. Springer, 2001, pp. 825–834.

[12] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *Network and Parallel Computing*, ser. Lecture Notes in Computer Sci., H. Jin, D. Reed, and W. Jiang, Eds. Springer Berlin Heidelberg, 2005, vol. 3779, pp. 2–13.

[13] S. Wang, L. Anselin, B. Bhaduri, C. Crosby, M. F. Goodchild, Y. Liu, and T. L. Nyerges, "Cybergis software: a synthetic review and integration roadmap," *Int. J. of Geographical Information Sci.*, vol. 27, no. 11, pp. 2122–2145, 2013.

[14] S. Wang, "A cybergis framework for the synthesis of cyberinfrastructure, gis, and spatial analysis," *Annals Assoc. Am. Geographers*, vol. 100, no. 3, pp. 535–557, 2010.

[15] S. Krishnan, C. J. Crosby, V. Nandigam, M. Phan, C. Cowart, C. K. Baru, and J. R. Arrowsmith, "Opentopography: a services oriented architecture for community access to lidar topography." in *COM.Geo*, ser. ACM Int. Conf. Proc. Series, L. Liao, Ed. ACM, 2011, p. 7.

[16] D. Blodgett, N. Booth, T. Kunicki, J. Walker, and J. Lucido, "Description of the u.s. geological survey geo data portal data integration framework," *IEEE J. of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 6, pp. 1687–1691, Dec. 2012.

[17] Y. Zhu and P. Shen, "Bring ajax to web application based on grid service," in *Int. Conf. on Intelligent Human-Machine Systems and Cybernetics*, vol. 2, Aug. 2009, pp. 162–165.

[18] S. Pallickara and M. Pierce, "Swarm: Scheduling large-scale jobs over the loosely-coupled hpc clusters," in *IEEE 4th Int. Conf. on eScience*, Dec. 2008, pp. 285–292.

[19] M. E. Pierce, C. Youn, and G. C. Fox, "The gateway computational web portal," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1411–1426, 2002. [Online]. Available: http://dx.doi.org/10.1002/cpe.681

[20] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, no. 6, pp. 37–46, June 2002.

[21] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke, "From open grid services infrastructure to ws-resource framework: Refactoring & evolution," 2004. [Online]. Available: http://toolkit.globus.org/wsrf/

[22] S. Majithia, M. Shields, I. Taylor, and I. Wang, "Triana: a graphical web service composition and execution toolkit," in *Proc. IEEE Int. Conf. on Web Services*, July 2004, pp. 514–521.

[23] B. Baranski, "Grid computing enabled web processing service," *Proc. 6th Geographic Information Days, IfGI prints*, vol. 32, pp. 243–256, 2008.

[24] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May. 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[25] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert *et al.*, "irods primer: integrated rule-oriented data system," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 2, no. 1, pp. 1–143, 2010.

[26] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for grid services," in *Proc. 12th IEEE Int. Symposium on High Performance Distributed Computing*, June 2003, pp. 48–57.

[27] D. J. Barrett and R. E. Silverman, *SSH, the Secure Shell: The Definitive Guide.* O'Reilly Media, Inc., 2001.

[28] T. K. Tesfa, D. G. Tarboton, D. W. Watson, K. A. Schreuders, M. E. Baker, and R. M. Wallace, "Extraction of hydrological proximity measures from dems using parallel processing," *Environmental Modelling and Software*, vol. 26, no. 12, pp. 1696 – 1709, 2011.

[29] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel Computing*, vol. 22, no. 6, pp. 789–828, 1996.

[30] Curl and libcurl, http://curl.haxx.se, 2015, [Online; accessed 15-Feb-2015].

CHAPTER 6

Conclusions

This dissertation presented novel approaches for dealing with big data using algorithmic and architectural techniques. From Chapter 1, we show from experimental results that the implemented virtual memory manager is beneficial for geoscientists and researchers in processing raster data – in this case very large DEM datasets – to fit in the memory of a single machine. We are able to run this hydrologic terrain analysis algorithm on limited memory systems for datasets so large that previously available algorithms fail. This overcomes the main obstacle of the memory capacity of the machine, by utilizing a user-level shared virtual memory system. Efficiencies were achieved using algorithm refinements such as the load balancing and the pre-fetching approach used in the page replacement algorithm of the user-level shared virtual memory system that increased hit rate and reduced wait time.

In Chapter 2, we showed that a big dataset which is beyond the limit of the memory of a single node in the HPC cluster system is processed using parallel WaveCluster algorithm by benefiting 8, 16 and 32 processors for which we achieved linear scalability behavior. In Chapter 3, we extended the parallel WaveCluster algorithm and presented detailed description and comparison study of this algorithm. We improved upon previous work by investigating two different merging techniques, namely priority-queue and merge-table approaches that play an important role in the parallel algorithm. These parallel algorithms find distinct clusters using discrete wavelet transformations on distributed memory architectures using the MPI API. The effectiveness of the parallel WaveCluster algorithms as compared to the parallel K-means algorithm are shown in our study to have achieved dramatic speed-up ratios where wavelet level is 2 on Dataset1.

In Chapter 4, we introduce the HydroGate gateway service to facilitate environmental research using high-performance computing centers. HydroGate is designed to transcend difficulties in accessing heterogeneous storage systems and HPC centers via a unified RestFul interface and allows rapid integration of HPC centers with heterogeneous software systems for science web portals. We believe HPC gateway systems play an important role to make complex HPC systems more accessible for service clients by allowing them to execute compute and data intensive programs through an unified and elegant web interface. It is demonstrated that the capabilities of HydroGate and the case study data can be used to compute the contributing area. We made use of the USU HPC center that is using SLURM job scheduler system in the case study, but the concept is also proven to be effective on other HPC systems with PBS job scheduler.

APPENDIX

Appendix A

Reprint Permission

SPRINGER LICENSE
TERMS AND CONDITIONS

Feb 26, 2015

This is a License Agreement between Ahmet A Yildirim ("You") and Springer ("Springer") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Springer, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	3576330226952
License date	Feb 26, 2015
Licensed content publisher	Springer
Licensed content publication	Journal of Supercomputing, The
Licensed content title	A comparative study of the parallel wavelet-based clustering algorithm on three-dimensional dataset
Licensed content author	Ahmet Artu Yıldırım
Licensed content date	Jan 1, 2015
Type of Use	Thesis/Dissertation
Portion	Full text
Number of copies	1
Author of this Springer article	Yes and you are the sole author of the new work
Order reference number	None
Title of your thesis / dissertation	Advancement of Computing on Big Data via Parallel Computing and Distributed Systems
Expected completion date	May 2015
Estimated size(pages)	120
Total	0.00 USD
Terms and Conditions	

Introduction

The publisher for this copyrighted material is Springer Science + Business Media. By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the Billing and Payment terms and conditions established by Copyright Clearance Center, Inc. ("CCC"), at the time that you opened your Rightslink account and that are available at any time at http://myaccount.copyright.com).

Limited License
With reference to your request to reprint in your thesis material on which Springer Science and Business Media control the copyright, permission is granted, free of charge, for the use indicated in your enquiry.

Licenses are for one-time use only with a maximum distribution equal to the number that you identified in the licensing process.

This License includes use in an electronic form, provided its password protected or on the university's intranet or repository, including UMI (according to the definition at the Sherpa website: http://www.sherpa.ac.uk/romeo/). For any other electronic use, please contact Springer at (permissions.dordrecht@springer.com or permissions.heidelberg@springer.com).

The material can only be used for the purpose of defending your thesis limited to university-use only. If the thesis is going to be published, permission needs to be re-obtained (selecting "book/textbook" as the type of use).

Although Springer holds copyright to the material and is entitled to negotiate on rights, this license is only valid, subject to a courtesy information to the author (address is given with the article/chapter) and provided it concerns original material which does not carry references to other sources (if material in question appears with credit to another source, authorization from that source is required as well).

Permission free of charge on this occasion does not prejudice any rights we might have to charge for reproduction of our copyrighted material in the future.

Altering/Modifying Material: Not Permitted
You may not alter or modify the material in any manner. Abbreviations, additions, deletions and/or any other alterations shall be made only with prior written authorization of the author(s) and/or Springer Science + Business Media. (Please contact Springer at (permissions.dordrecht@springer.com or permissions.heidelberg@springer.com)

Reservation of Rights
Springer Science + Business Media reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.

Copyright Notice:Disclaimer
You must include the following copyright and permission notice in connection with any reproduction of the licensed material: "Springer and the original publisher /journal title, volume, year of publication, page, chapter/article title, name(s) of author(s), figure number(s), original copyright notice) is given to the publication in which the material was originally published, by adding; with kind permission from Springer Science and Business Media"

Warranties: None

Example 1: Springer Science + Business Media makes no representations or warranties with respect to the licensed material.

Example 2: Springer Science + Business Media makes no representations or warranties with respect to the licensed material and adopts on its own behalf the limitations and disclaimers established by CCC on its behalf in its Billing and Payment terms and conditions for this licensing transaction.

Indemnity
You hereby indemnify and agree to hold harmless Springer Science + Business Media and CCC, and their respective officers, directors, employees and agents, from and against any and all claims arising out of your use of the licensed material other than as specifically authorized pursuant to this license.

No Transfer of License
This license is personal to you and may not be sublicensed, assigned, or transferred by you to any other person without Springer Science + Business Media's written permission.

No Amendment Except in Writing
This license may not be amended except in a writing signed by both parties (or, in the case of Springer Science + Business Media, by CCC on Springer Science + Business Media's behalf).

Objection to Contrary Terms
Springer Science + Business Media hereby objects to any terms contained in any purchase order, acknowledgment, check endorsement or other writing prepared by you, which terms are inconsistent with these terms and conditions or CCC's Billing and Payment terms and conditions. These terms and conditions, together with CCC's Billing and Payment terms and conditions (which are incorporated herein), comprise the entire agreement between you and Springer Science + Business Media (and CCC) concerning this licensing transaction. In the event of any conflict between your obligations established by these terms and conditions and those established by CCC's Billing and Payment terms and conditions, these terms and conditions shall control.

Jurisdiction
All disputes that may arise in connection with this present License, or the breach thereof, shall be settled exclusively by arbitration, to be held in The Netherlands, in accordance with Dutch law, and to be conducted under the Rules of the 'Netherlands Arbitrage Instituut' (Netherlands Institute of Arbitration).***OR:***

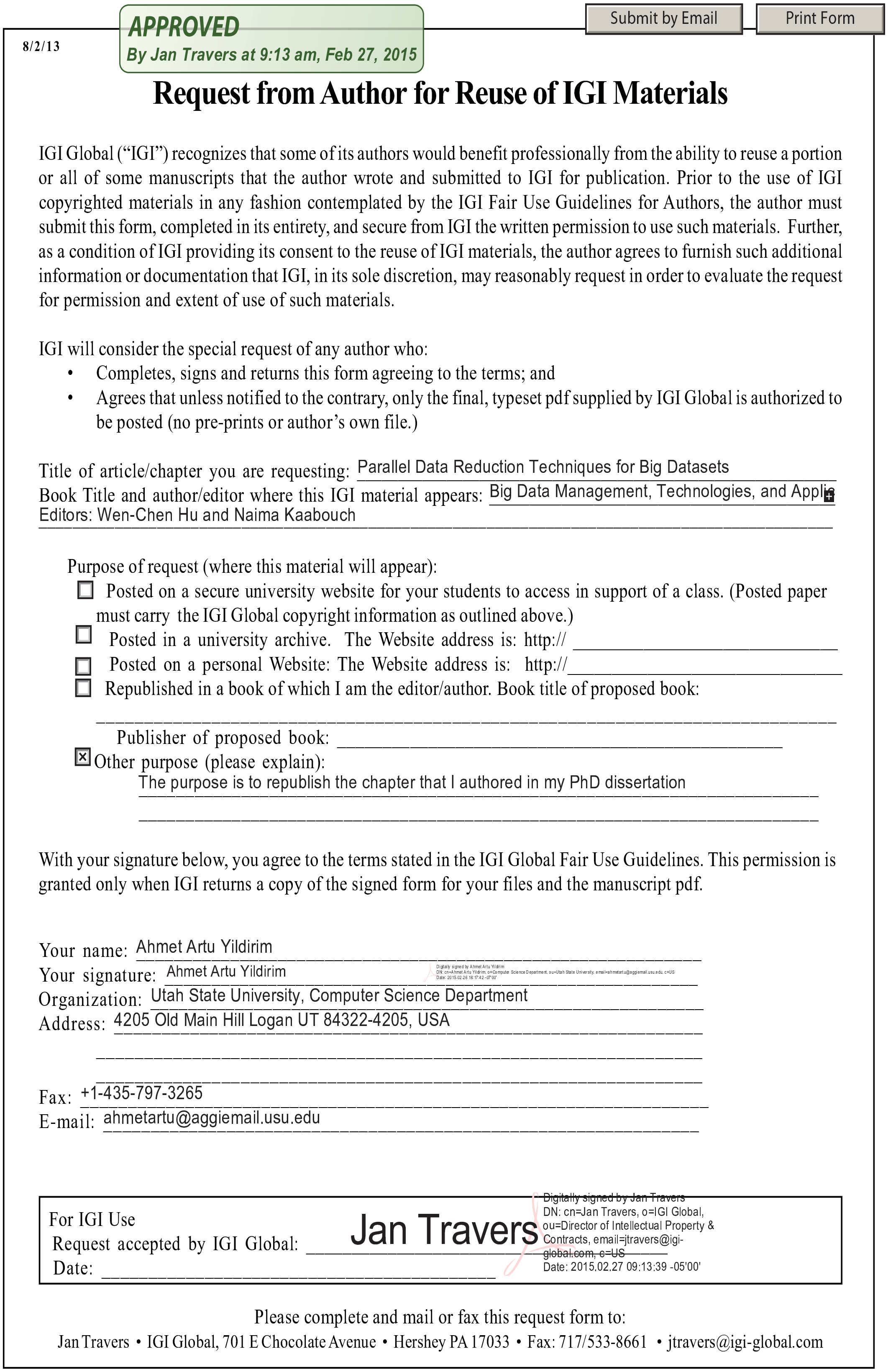
All disputes that may arise in connection with this present License, or the breach thereof, shall be settled exclusively by arbitration, to be held in the Federal Republic of Germany, in accordance with German law.

Other terms and conditions:

v1.3

Questions? customercare@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777.

Gratis licenses (referencing $0 in the Total field) are free. Please retain this printable license for your reference. No payment is required.



Submit by Email | Print Form

8/2/13

APPROVED
By Jan Travers at 9:13 am, Feb 27, 2015

Request from Author for Reuse of IGI Materials

IGI Global ("IGI") recognizes that some of its authors would benefit professionally from the ability to reuse a portion or all of some manuscripts that the author wrote and submitted to IGI for publication. Prior to the use of IGI copyrighted materials in any fashion contemplated by the IGI Fair Use Guidelines for Authors, the author must submit this form, completed in its entirety, and secure from IGI the written permission to use such materials. Further, as a condition of IGI providing its consent to the reuse of IGI materials, the author agrees to furnish such additional information or documentation that IGI, in its sole discretion, may reasonably request in order to evaluate the request for permission and extent of use of such materials.

IGI will consider the special request of any author who:

- Completes, signs and returns this form agreeing to the terms; and
- Agrees that unless notified to the contrary, only the final, typeset pdf supplied by IGI Global is authorized to be posted (no pre-prints or author's own file.)

Title of article/chapter you are requesting: Parallel Data Reduction Techniques for Big Datasets
Book Title and author/editor where this IGI material appears: Big Data Management, Technologies, and Applic
Editors: Wen-Chen Hu and Naima Kaabouch

Purpose of request (where this material will appear):
☐ Posted on a secure university website for your students to access in support of a class. (Posted paper must carry the IGI Global copyright information as outlined above.)
☐ Posted in a university archive. The Website address is: http:// ______
☐ Posted on a personal Website: The Website address is: http://______
☐ Republished in a book of which I am the editor/author. Book title of proposed book: ______
Publisher of proposed book: ______
☒ Other purpose (please explain):
The purpose is to republish the chapter that I authored in my PhD dissertation

With your signature below, you agree to the terms stated in the IGI Global Fair Use Guidelines. This permission is granted only when IGI returns a copy of the signed form for your files and the manuscript pdf.

Your name: Ahmet Artu Yildirim
Your signature: Ahmet Artu Yildirim
Digitally signed by Ahmet Artu Yildirim
DN: cn=Ahmet Artu Yildirim, o=Computer Science Department, ou=Utah State University, email=ahmetartu@aggiemail.usu.edu, c=US
Date: 2015.02.26 16:17:42 -07'00'
Organization: Utah State University, Computer Science Department
Address: 4205 Old Main Hill Logan UT 84322-4205, USA
Fax: +1-435-797-3265
E-mail: ahmetartu@aggiemail.usu.edu

For IGI Use
Request accepted by IGI Global: Jan Travers
Digitally signed by Jan Travers
DN: cn=Jan Travers, o=IGI Global, ou=Director of Intellectual Property & Contracts, email=jtravers@igi-global.com, c=US
Date: 2015.02.27 09:13:39 -05'00'
Date: ______

Please complete and mail or fax this request form to:
Jan Travers • IGI Global, 701 E Chocolate Avenue • Hershey PA 17033 • Fax: 717/533-8661 • jtravers@igi-global.com

Ahmet Artu Yildirim
1157 E Stadium Dr
Logan, UT 84341
Phone: (435) 881-9964
Email: ahmetartu@aggiemail.usu.edu

Assoc. Prof. Dr. Cem Özdoğan
Cankaya University
Material Science and Engineering
Office:N-B08 / N-B18
EskişehirYolu 29. Km, YukarıyurtçuMahallesi
MimarSinanCaddesi No:4 N Blok Zemin Kat (B Katı),
06790, Etimesgut/Ankara Turkey

Assoc. Prof. Dr. Cem Özdoğan,

I am preparing my Dissertation in the Department of Computer Science at Utah State University. I am anticipating completion of my degree in April of 2015.

A book chapter, *Parallel Data Reduction Techniques for Big Datasets*, of which I am the first author, and which appears in the book of Big Data Management, Technologies, and Applications, IGI Global, reports a part of my Dissertation research. I would like to reprint it as a chapter in my Dissertation (reprinting the paper necessitates some revision.)

Please indicate your approval of this request by signing in the space provided, and attach any other form necessary to confirm permission. If you charge a reprint fee for use of the chapter by the author, please indicate that as well.

Best Regards,

Ahmet Artu Yildirim

I hereby give permission to Ahmet Artu Yildirim to reprint the requested chapter in his Dissertation

Signed:
Date: April 24, 2015
Fee: No Fee

CURRICULUM VITAE

Ahmet Artu Yıldırım

EDUCATION

Ph.D., Computer Science. Utah State University, Logan, UT. Present.

M.S., Computer Engineering. Cankaya University, Ankara, Turkey. 2011.

B.S., Industrial Engineering. Cankaya University, Ankara, Turkey. 2003.

RESEARCH INTERESTS

Distributed and parallel systems/algorithms, internet computing, cloud computing, data mining.

BOOK CHAPTERS

Yıldırım, A.A., and Watson, D. (2015) A Cloud-Aware Distributed Object Storage System to Retrieve Big Data via HTML5-Enabled Web Browsers. Managing Big Data in Cloud Computing Environments, IGI Global (Under Review)

Yıldırım, A.A., Ozdogan, C., and Watson, D. (2014) Parallel Data Reduction Techniques for Big Datasets, IGI Global

JOURNAL PUBLICATIONS

Yıldırım, A.A., Tarboton, D., and Watson, D. (2015) Virtualized Access to HPC Resources via Service Oriented Architecture to Facilitate Environmental Research, Earth Science Informatics, Springer (In Preperation)

Yıldırım, A.A., and Watson, D. (2015) A Comparative Study of the Parallel Wavelet-based Clustering Algorithm on 3-dimensional Dataset, Journal of Supercomputing, Springer

Yıldırım, A.A., Watson, D., Tarboton, D., and Wallace, R.M., (2015) A Virtual Tile Approach to Raster-based Calculations of Large Digital Elevation Models in a Shared-Memory System, Computers & Geosciences, Elsevier (Under Review)

Yıldırım, A.A., and Ozdogan, C., (2011) Parallel Wavelet-based Clustering Algorithm using CUDA, Procedia-Computer Science Journal, Elsevier

Yıldırım, A.A., and Ozdogan, C., (2011) Parallel WaveCluster: A Linear Scaling Parallel Clustering Algorithm Implementation with Application to Very Large Datasets, Journal of Parallel and Distributed Computing, Elsevier

CONFERENCE PUBLICATIONS

Yıldırım, A.A., Tarboton, D., Dash, P., and Watson, D. (2014) Design and Implementation of a Web Service-Oriented Gateway to Facilitate Environmental Modeling using HPC Resources, 7th International Congress on Environmental Modelling and Software (iEMSs), San Diego, California

Fan, Y., Liu, Y., Wang, S., Tarboton, D, Yıldırım, A.A., and Wilkins-Diehr, N. (2014) Accelerating TauDEM as a Scalable Hydrological Terrain Analysis Service on XSEDE, XSEDE 14, Atlanta, GA

JOB EXPERIENCE

Graduate Research Assistant. Utah State University. 2013 to Present.

Instructor. Utah State University. 2013.

Graduate Research Assistant. Utah State University. 2012 to 2013.

Graduate Teaching Assistant. Utah State University. 2011 to 2012.

Lead Software Engineer. Polar Information Technologies. Ankara. Turkey. 2006 to 2009.

Software Engineer. Polar Information Technologies. Ankara. Turkey. 2004 to 2006.