AUTONOMIC PROGRAMMING PARADIGM FOR HIGH PERFORMANCE COMPUTING

By

Yaser Jararweh

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2010

THE UNIVERSITY OF ARIZONA

GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Yaser Jararweh entitled “AUTONOMIC PROGRAMMING PARADIGM FOR HIGH PERFORMANCE COMPUTING” and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy

_____Date: 08/09/2010

Salim Hariri

_____Date: 08/09/2010

Ali Akoglu

_____Date: 08/09/2010

Janet Wang

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

_____Date: 08/09/2010

Dissertation Director: Salim Hariri

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under the rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the author.

SIGNED: Yaser Jararweh

ACKNOWLEDGEMENTS

First of all, I'm grateful to the GOD, most mighty, most merciful, who created me and gave me the ability to think and search for the truth.

I would like to thank my Ph.D. advisor, Professor Salim Hariri, for his continued support and guidance during the course of my Ph.D. studies. Professor Hariri has allowed me the freedom to pursue my research interests, and provided me with encouragement and patience. I have benefited from him not only research wisdom, but also research ethics that I treasure most.

In addition, I would like to thank Professor Ali Akoglu, Professor Janet Wang, Professor Ivan Djordjevic, and Professor Ferenc Szidarovszky for serving on my written and oral committees and offering valuable suggestions to improve this dissertation.

My fellow lab mates in Professor Hariri's group, I am indebted to my friend, instructor, critique, and mentor Professor Youssif Al-Nashif, for all what he did for me during the course of my Ph.D. studies. Last but not least, I am indebted to my parents, brothers and sisters, my wife, and my daughter "Ayah" for always being there for me. My achievements would not have been possible without their emotional support and encouragement.

To my Parents and Family

TABLE OF CONTENTS

TABLE OF CONTENTS – Continued

LIST OF FIGURES

LIST OF FIGURES - Continued

LIST OF TABLES

ABSTRACT

The advances in computing and communication technologies and software tools have resulted in an explosive growth in networked applications and information services that cover all aspects of our life. These services and applications are inherently complex, dynamic and heterogeneous. In a similar way, the underlying information infrastructure, e.g. the Internet, is large, complex, heterogeneous and dynamic, globally aggregating large numbers of independent computing and communication resources, data stores and sensor networks. The combination of the two results in application development, configuration and management complexities that break current computing paradigms, which are based on static behaviors, interactions and compositions of components and/or services. As a result, applications, programming environments and information infrastructures are rapidly becoming fragile, unmanageable and insecure. This has led researchers to consider alternative programming paradigms and management techniques that are based on strategies used by biological systems to deal with complexity, dynamism, heterogeneity and uncertainty. Autonomic programming paradigm is inspired by the human autonomic nervous system that handles complexity, uncertainties and abnormality, and aims at realizing computing systems and applications capable of managing themselves with minimum human intervention. The overarching goal of the autonomic programming paradigm is to help building systems and applications capable of self-management. In this thesis we focus on the Self-optimizing, and Self-Configuring properties of the autonomic system. Firstly, we investigated the large-scale scientific

computing applications which generally experience different execution phases at runtime and each phase has different computational, communication and storage requirements as well as different physical characteristics. Consequently, an optimal solution or numerical scheme for one execution phase might not be appropriate for the next phase of the application execution. In this dissertation, we present Physics Aware Optimization (PAO) paradigm that enables programmers to identify the appropriate solution methods to exploit the heterogeneity and the dynamism of the application execution states. We implement a Physics Aware Optimization Manager to exploit the PAO paradigm. On the other hand we present a self configuration paradigm based on the principles of autonomic computing that can handle efficiently complexity, dynamism and uncertainty in configuring server and networked systems and their applications. Our approach is based on making any resource/application to operate as an Autonomic Component (that means it can be self-managed component) by using our autonomic programming paradigm. Our POA technique for medical application yielded about 3X improvement of performance with 98.3% simulation accuracy compared to traditional techniques for performance optimization. Also, our Self-configuration management for power and performance management in GPU cluster demonstrated 53.7% power savings for CUDA workload while maintaining the cluster performance within given acceptable thresholds.

1 INTRODUCTION

1.1 Problem Statement

The increase in complexity, interconnectedness, dependency and the asynchronous interactions between systems components that include hardware resources (computers, servers, network devices), and software (application services, scientific simulation, middleware, web services, etc.) makes the system management and configuration a challenging research problem. As the scale and complexity of these systems and applications grow, their development, configuration and management challenges are beginning to break current static programming and management paradigms, overwhelm the capabilities of existing tools and methodologies, and rapidly render the systems and applications fragile, unmanageable and insecure.

In large-scale distributed scientific application, the computational complexity associated with each computational region or domain varies continuously and dramatically both in space and time throughout the whole life cycle of the application execution. In general, large-scale scientific computing applications experience different execution phases at runtime and each phase has different computational, communication and storage requirements as well as different physical characteristics. Consequently, an optimal solution or numerical scheme for one execution phase might not be appropriate for the next phase of the application execution. Choosing the ideal numerical algorithms and solutions for all application runtime phases remains an active research challenge.

On the other hand, current information infrastructure, e.g. the Internet, globally aggregating large numbers of independent computing and communication resources, data stores and sensor networks.

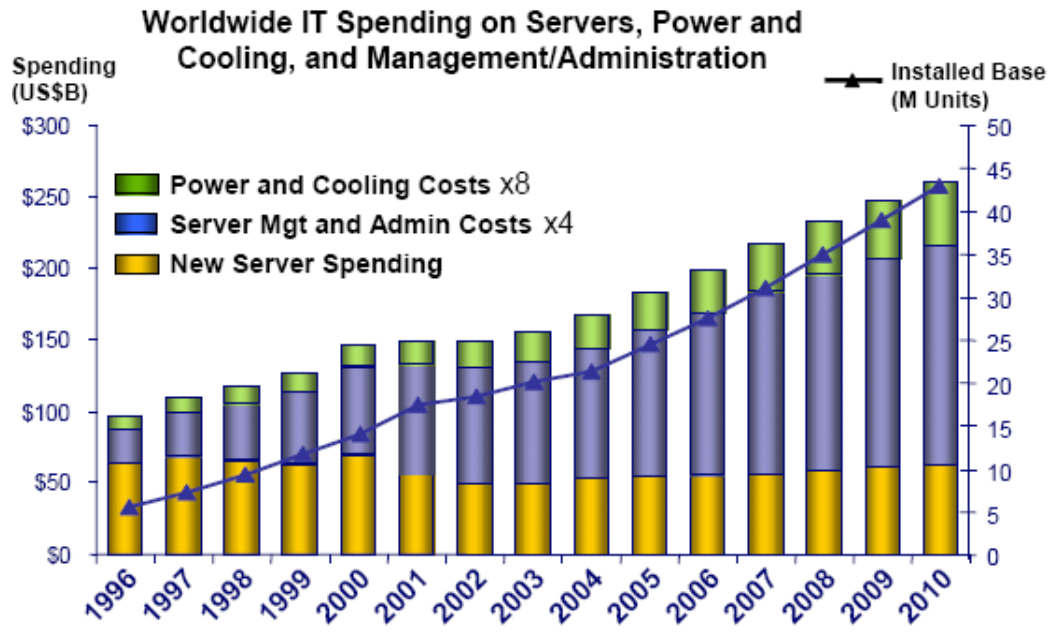


Figure 1-1. Worldwide IT spending. Source: http:idc 2006

The number of computing devices in use is keep growing and there average complexity is increasing as well. Also, large scale computer networks for communication and computation are employed in large companies, universities, and institutions. In addition, these networks are pervaded in mobile computing and communication using laptops, PDAs, or mobile phones with diverse forms of wireless. It results in complex, large, heterogeneous, and dynamic networks of large numbers of independent computing and communication resources and data stores [52]. While the infrastructure is growing to be larger and more complex, the applications running on computer systems and networks are

also turning out to be more and more complex. These applications vary with many different functions, ranging from internal control processes to presenting web content and to customer support. For typical information system consisting of an application server, a web server, messaging facilities, and layers of middleware and operating systems, the number of tuning parameters and their combinations can be hundreds of thousands and reaching a level that exceeds the ability of human to comprehend and act on in a timely manner. Currently this volume and complexity is managed by highly skilled system administrators but the demand for skilled IT personnel is already outstripping supply, with labor costs exceeding equipment costs. Figure 1-1 shows the cost of system management comparing to the hardware cost. Computing advancement has brought great benefits of speed and automation but there is now an overwhelming need to automate and control most of management activities. This has led researchers to consider alternative programming paradigms and management techniques that are based on strategies used by biological systems to deal with complexity, dynamism, heterogeneity and uncertainty.

1.2 Research Objectives

The goal of this research is to develop a theoretical and experimental framework and general methodology for autonomic management framework for run time performance optimization, and an integrated power and performance management of high-performance GPU server platforms to achieve (a) online modeling, monitoring, and analysis of application phases, application power consumption and performance; (b) automatic detection of application execution phases and properties for a wide range of

workloads and applications;(c) employing knowledge of application execution phases to dynamically identify strategies that improve the application performance, and minimize power consumption while maintaining the required quality of service (d) dynamically reconfigure high-performance GPU cluster system according to the selected configuration\optimization strategies; and (e) seek new and effective optimization techniques that can scale well and can perform in a dynamic and continuous fashion, making the system truly autonomic. Our methodology exploits emerging hardware and software standards for GPU servers that offer a rich set of hardware and software features for effective power and performance management. We consider power consumption as a first-class resource that is managed autonomously along with performance for high-performance GPU based computing systems. We incorporate power and performance management within an autonomic management framework that is modeled after the human autonomic nervous system. This management framework can be extended for the joint management of multiple autonomic objectives such as power, performance, fault and security.

1.3 Research Challenges

This work develops innovative techniques to address the following research challenges for both Self-Optimization and Self-Configuration in today computing systems and applications:

Self-Optimization

The research described in the first part of this dissertation developed the expertise, technology, and infrastructure to optimize very large multi-dimensional multi-phase numerical and medical simulations. Realizing these realistic simulations by best utilizing the computing power in a most efficient way without jeopardizing the accuracy of the simulation presents grand research challenges. These challenges are due to:

1. Exceptional scales in domain size and resolution of the applications.
2. Unprecedented heterogeneity and dynamics in time, space and state of applications and systems.
3. Unprecedented complexity in physical models, numerical formulations and software implementations.
4. Unique programmability, manageability and usability requirements in allocating, programming and managing very large numbers of resources.

In fact, we have reached a level of complexity, heterogeneity and dynamism that our programming environments and infrastructure are becoming fragile, unmanageable and insecure. There is a need for a fundamental change in how these applications are formulated, composed, managed and optimized.

Self-Configuration for Power and Performance Management.

Automatic modeling and online analysis of multiple objectives and constraints such as power consumption and performance of high-performance computing platforms is a challenging research problem due to the continuous change in workloads and applications requirements, continuous changes in surrounding environment, the variety of services and

software modules being offered and deployed, and the extreme complexity and dynamism of their computational workloads. In order to develop an effective autonomic control and management of power and performance, it becomes highly essential for the system to have the functionality of online monitoring; adaptive modeling and analysis tailored towards real-time processing and proactive management mechanisms. The second part of this work develops innovative techniques to address the following research challenges in power and performance management in GPU computing system:

1. How do we efficiently and accurately model power consumption from a GPU platform-level perspective that involves the complex interactions of different classes of devices such as CPUs, main memory, GPU cards, and I/O? Current research focuses on modeling conventional computing system (CPU-only systems) and data center. The research challenge is to accurately incorporate and model the emerging GPU cards power and performance management and the complex interactions between the GPU cards and various device classes in the host system. Modeling this accurately is a significant research and development challenge.
2. How do we predict in real-time, the behavior of GPU server resources and their power consumptions as workloads change dynamically. We address this challenge by utilizing a novel concept called application execution flow (Appflow) that characterizes both resource requirements and the spatial and temporal behaviors of applications during their life-time.

3. How to design “instantaneous” and adaptive optimization mechanisms that can continuously and endlessly learn, execute, monitor, and improve in meeting the collective objectives of power management and performance improvement? We have exploited mathematically rigorous optimization techniques to address this research challenge.

1.4 Dissertation Organization

The rest of the dissertation is organized as follows.

In Chapter 2, we introduce related work on autonomic programming systems. The Self-Configuration management techniques, we categorize the configuration management techniques as initial configuration and dynamic configuration management. We introduce related work on Self-optimization system. We categorize the self-optimization techniques into three major categories based on the knowledge exploited by the optimization techniques: architecture aware, error aware and application aware. We reviewed the work related to power and performance management for computing systems and GPU based system. Also, we introduce the human cardiac modeling and simulation. Finally we introduce the case base reasoning learning technique.

In Chapter 3, we lay the foundations of the autonomic programming paradigm. We present the human autonomic nervous system, and then we present the properties of an autonomic programming system. Finally, we present the conceptual architecture of the autonomic programming system:

In Chapter 4, we present our Physics Aware Optimization (PAO) technique and introduce the theoretical background of this methodology and some examples on how this methodology will benefit the developers of scientific applications without sacrificing the accuracy and convergence of the application. We also discuss the implementation of the PAO for a human heart simulation. Finally, we present some experimental results and evaluation for the PAO paradigm.

In Chapter 5, we investigate the GPU-based Personal Supercomputing as an emerging concept in high performance computing that provides an opportunity to overcome most of today HPC data centers problems. We explore and evaluate the GPU- based cluster system. Finally, we investigate the problems of the GPU based clusters and there current available solutions.

In Chapter 6, we focus on the power and performance management of the GPU cluster system. We present a novel self-configuration technique that employs dynamic resource scaling for the GPU cluster component. We introduce a detailed model for the GPU cluster from the power point of view. Also we introduce **MPAlloc** and **PERFORMWATT** algorithms, and an implementation details for our power and performance Manager. Finally, we present some experimental results and evaluation.

In Chapter 7, we conclude this dissertation and give future research directions.

2 BACKGROUND AND RELATED WORK

2.1 Autonomic Programming Paradigm

2.1.1 Autonomic Programming Paradigm Research

As autonomic programming paradigm is a multi-disciplinary research area, a significant number of projects in computer science, software engineering, and artificial intelligence are developed based on this paradigm. The following lists some autonomic programming paradigm projects in academia and in the industrial field.

AUTONOMIA [27] at the University of Arizona, is an autonomic control and management framework that provides users with all the capabilities required to perform the suitable control and management schemes and the tools to configure the required software and network resources.

AUTOMATE [78] at Rutgers University, aims at developing conceptual models and implementation architectures that can enable the development and execution of self-managing Grid applications. Specifically, it investigates programming models, frameworks and middleware services that support the definition of autonomic elements, the development of autonomic applications as the dynamic and opportunistic composition of these autonomic elements, and the policy, content and context driven definition, execution and management of these applications.

Astrolabe [81] is a distributed information management system which collects large-scale system state, permitting rapid updates and providing on-the-fly attribute aggregation.

This latter capability permits an application to locate a resource, and also offers a scalable way to track system state as it evolves over time.

Recovery-Oriented Computing at UC Berkeley/Stanford [76] investigates novel techniques for building highly dependable Internet services, recovery from failures rather than failure-avoidance. This philosophy is motivated by the observation that even the most robust systems still occasionally encounter failures due to human operator error, transient or permanent hardware failure, and software anomalies resulting from software aging.

The goal of Autonomizing Legacy Systems project at Columbia University [77] is to retrofit autonomic computing onto such systems, externally, without any need to understand or modify the code, and in many cases even when it is impossible to recompile. The project presents a meta-architecture implemented as active middleware infrastructure to explicitly add autonomic services via an attached feedback loop that provides continual monitoring and, as needed, reconfiguration and/or repair. The lightweight design and separation of concerns enables easy adoption of individual components, as well as the full infrastructure, for use with a large variety of legacy, new systems, and systems of systems.

IFLOW [79] is an autonomic middleware for implementing distributed information flow services. It provides the functionality of dynamically associating its abstract information graph to physical network nodes based on utility functions.

VIOLIN [80] is a system that supports self-adaptation of allocation of computing resources across multi-domains by considering the dynamic availability of infrastructure resources and the current applications' resource requirement.

QoS Management at Virtualized Data Centers proposed a two-level autonomic resource management system that enables automatic and adaptive resource provisioning in accordance with Service Level Agreements (SLA) specifying dynamic tradeoffs of service quality and cost based on fuzzy model.

On the industrial side, large enterprise-wide vendors such as IBM, Sun Microsystems, HP, Microsoft, Intel, Cisco, and several other vendors have developed variety of management systems and solutions. We focus on four major vendors whose products are leading the market, namely: IBM, Sun, HP and Microsoft with their major projects as follows.

Storage Management Analytics and Reasoning Technology (SMART) [82] developed by IBM reduces complexity and improves quality of service through the advancement of self-managing capabilities within a database environment.

Another IBM project, Oceano [83] designs and develops a pilot prototype of a scalable, manageable infrastructure for a large scale computing utility power plant.

Optimal Grid [84] aims to simplify the creation and management of large-scale, connected, parallel grid applications by optimizing performance and includes autonomic grid functionality as a prototype middleware.

AutoAdmin developed by Microsoft [85] makes database systems self-tuning and self-administering by enabling them to track the usage of their systems and to gracefully to adapt to application requirements.

N1 [86] developed by Sun manages data centers by including resource virtualization, service provisioning, and policy automation techniques.

The Adaptive Enterprise [87] developed by HP helps customers to build a system in three levels namely business (e.g., customary relation management), service (e.g., security), and resource (e.g., storage). Most industry projects focused on self-configuring and self-optimizing and exploited product specific autonomic framework, instead of being a general management framework that addresses issues such as self-healing and self-protecting.

In our work, we mainly focused on self-configuring and self-optimizing properties of the autonomic system. In the following section we present a detailed review for self-configuring and self-optimizing techniques and their related work.

2.1.2 Autonomic Programming Paradigm Concepts

Autonomic programming paradigm and management technique is based on strategies used by biological systems to deal with complexity, dynamism, heterogeneity and uncertainty. It is inspired by the human autonomic nervous system that handles complexity and uncertainties, and it aims at realizing computing systems and applications capable of managing themselves with minimum human intervention.

The autonomic programming paradigm must have a mechanism whereby changes in its essential variables (e.g., performance, power, fault, security, etc.) can cause changes to the behavior of the computing system such that the system is brought back into equilibrium with respect to the environment. This state of stable equilibrium is essential in the organism nervous system for the survivability of the organism. In the case of an autonomic computing system, we can consider the survivability as the system's ability to maintain its optimal performance, with minimum recourses over provisioning, protect itself from all types of attacks, recover from faults, reconfigure as required by changes in the environment. Both the internal environment (e.g. excessive CPU utilization, high power consumption) and the external environment (e.g. protection from an external attack, spike in incoming workload) impact its equilibrium. The autonomic computing system requires sensors to sense the changes in managed component and effectors to react to the changes in the environment by changing itself so as to counter the effects of changes in the environment and maintain equilibrium. The changes sensed by the sensor have to be analyzed to determine if any of the essential variables has gone out of their viability limits. If so, it has to trigger some kind of planning to determine what changes to inject into the current behavior of the system such that it returns to equilibrium state within the new environment. This planning would require knowledge to select just the right behavior from a large set of possible behaviors to counter the change. Finally, the system executes the selected change. MAPE which stands for 'Monitor', 'Analyzing', 'Planning', 'Execute' and 'Knowledge' are in fact the main functions used to identify an autonomic system [4].

Properties of an Autonomic Computing System

An autonomic system/application can be a collection of autonomic components, which can manage their internal behaviors and relationships with others in accordance to predefined policies. The autonomic computing system has the following properties [4] [5] [27]:

1. Self-optimizing

   Self-optimizing components can tune themselves to meet end-user or business needs. The tuning actions could mean reallocating resources such as in response to dynamically changing workloads to improve overall utilization.

2. Self-protecting

   Self-protecting components can detect abnormal behaviors as they occur and take corrective actions to stop or mitigate the impacts of attacks that triggered the anomalous events. The abnormal behaviors can be triggered by an unauthorized access and use of a file, virus infection and proliferation, and denial-of-service attacks.

3. Self-configuring

   Self-configuring components adapt dynamically to changes in the environment and policies. Such changes could include the deployment of new components or the removal of existing ones, or dramatic changes in the system execution environment.

4. Self-healing

   Self-healing components can detect system malfunctions and initiate policy-based corrective action to tolerate hardware and/or software failures.

It is an important to note that an autonomic computing system addresses these issues in an integrated manner rather than being treated in isolation as is currently done. Consequently, in autonomic programming paradigm, the system design paradigm takes a holistic approach that can integrate all these attributes seamlessly and efficiently.

Autonomic Computing System Architecture

Figure 2-1 shows the architecture of an autonomic computing system from the conceptual point of view. It consists of the following modules.

The Environment

The environment represents all the factors that can impact the managed resource. The environment and the managed resource can be two subsystems forming a stable system. Any change in the environment causes the change on the system stable operating point. This change required a counter changes in the managed resource causing the system to move back to its stable state.

Managed Resources

The managed resources represent any existing hardware or software resource that is modified to manage itself autonomously. Any managed resource will be affected by its surrounding environment.

MAPE Components

At runtime, the high performance computing environment can be affected in different ways, for example, it can encounter a failure during execution, it can be externally attacked or it may slow down and affect the performance of the entire application. It is the job of the Control to manage such situations as they occur at runtime. The MAPE has the following components to execute its functionalities:

1. Monitoring: Monitors the current state of the managed resources through its sensors.

2. Analysis: Analyzes managed resource current state behavior to detect if there are any anomalies or state divergence from the normal state (i.e. component failure, degradation in performance, power over provisioning).

3. Planning: Plans alternate execution strategies (by selecting appropriate actions) to optimize the behavior (e.g. to self-heal, self-optimize, self-protect etc.) and operation of the managed system.

4. Knowledge: Provides the decision support with the appropriate knowledge to select the appropriate rule to improve performance..

5. Execute: Execute the planned actions on the managed resource.

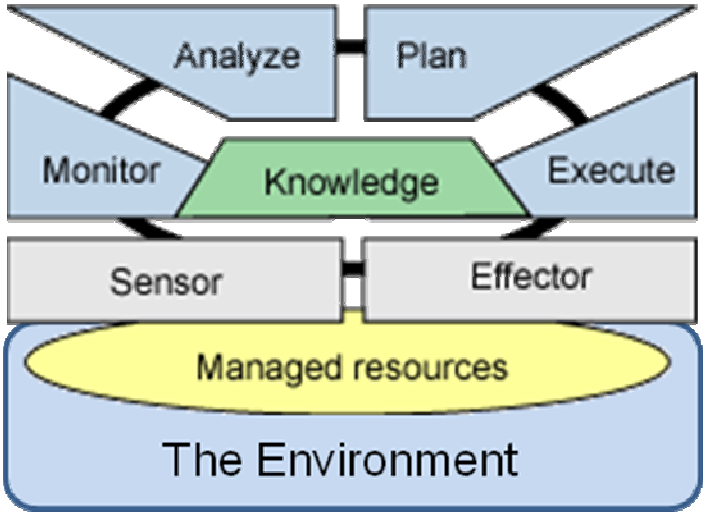


Figure 2-1: Autonomic computing system [99].

2.2 Self-Configuration and Self-Optimization Management Techniques

2.2.1 Self-Configuration Management Techniques

Self-Configuration management of an applications and systems can be defined as a set of activities to deploy, configure, activate and maintain the life cycle of its goals, services and resources. Recently, Self-Configuration approach introduced in different domains in computing systems like networked systems and Applications [55][98]. Self configuring principles, as observed in natural systems (e.g. nerves system), and many engineered

systems like unmanned vehicles, robotics, and data centers, are increasingly applied to the computing systems. Self configuring components should provide strength to the system at all, that must cope with unstable and dynamic environments, while keeping simplicity in individual system components. Currently, many distributed managed systems\application are based on centralized framework without any global system overview, they are ad hoc and hard to be used to the other systems [53][55].

Self-configuration perform configuration actions automatically according to a predefined policies to handle environment changes and\or topology change as a new components join the system, and\or an old components deleting from the system [98]. Self-Configuration can be performed in two stages: initial configuration and dynamic configuration at runtime [27]. Initial configuration is a process consisting of deployment, configuration and activation of a set of interrelated services or resources during the setting up phase of the system execution environment. Initial configuration approaches can be classified into manual configuration and autonomic configuration. Manual configuration needs human involvement during the process of installation and configuration, which is inefficient and error prone because of the absence of global system view. Autonomic configuration keeps track for the dependencies of services and resources, setups the running environment, and activates them automatically based on predefined policies 58] [59]. Autonomic initial configuration management approaches can be classified into three types: i) Script based approach in which the automation introduced by specifying configuration attributes and a set of configuration actions. ii)

Language-based approach that extends the script-based approach by introducing lifecycle management through the use of dependencies and system state monitoring. iii) Model-based approach that uses models for the creation of deployment and configuration instantiations.

Many configuration management systems [58] [59], only support initial configuration without any sort of support for dynamic configuration. In order to support dynamic configuration, a system needs advanced feedback control mechanisms to monitor its state and its system parameters and take suitable actions. Such configuration actions could mean adjusting configuration parameters, reallocating resources to improve overall utilization, or a complete reorganization of the system configuration. The dynamic configuration involves changing the system services or resource configurations dynamically to adapt in real time to the changes in the environment, current system loads and policies. In general, any dynamic configuration management approach can be classified based on its approach to implement i) Configuration change policy (when to make configuration changes), ii) Configuration adaptation policy (what to change in current configuration) and iii) Configuration implementation policy (how to execute the recommended changes) [27].

Self-configuration management shows a great potential in the domain of Mobile Ad-hoc Networks (MANETs) due to and the continues changes in network topologies [98].

2.2.2 Self-Optimizing Management Techniques

Runtime optimization of parallel and distributed applications has been an active research area and has been approached using different techniques. Also, self-optimization architecture introduced in IP-based network to manage themselves [101]. The knowledge exploited by the optimization algorithms can be classified into three major categories: Application Aware, Error Aware, and Architecture Aware (see Figure 2-2).

Application Aware algorithms: The Physics Aware Optimization (PAO) methodology belongs to this approach of optimization. Also, this approach includes Self-Adapting Numerical Software (SANS) [92], SANS is built based on Common Component Architecture (CCA) [93] with intelligent agents that analyze the application "metadata". The intelligent agents of SANS will also propose self-tuning rules to improve application performance after consulting with a knowledge base associated with the application. Our method, Physics Aware Optimization (PAO) exploits not only the application current state, but also the temporal/spatial characteristics that can be accurately predicted if one takes into consideration the application physics.

Error Aware algorithms: Adaptive Mesh Refinement (AMR) [88], Finite Element Method (FEM) [89], are the main representatives of this approach. Error aware algorithm required a grid refinement at runtime based on the error rate in certain computational regions as in AMR. But in the case FEM its statically minimize the grid size for all the computational domains if the error exceeds predetermined threshold value. AMR, and FEM cannot determine the grid size that produces the desired accuracy; only after the application executes and the error in the computations is calculated, the grid size can be

refined to improve the accuracy. All these techniques refine the computational mesh to meet the desired accuracy level.

Architecture Aware algorithms: The representatives of this approach include PHiPAC [90], and ATLAS [91]. PHiPAC provides a static optimization technique to achieve machine-specific near-peak performance. The user provides search scripts to find the best parameters for a given system executing the code. Also, the Automatically Tuned Linear Algebra Software (ATLAS) [91] provides a code generator coupled with a timer routine which takes parameters such as cache size, length of floating point and fetch pipelines etc., and then evaluates different strategies for loop unrolling and latency based on the environment states.

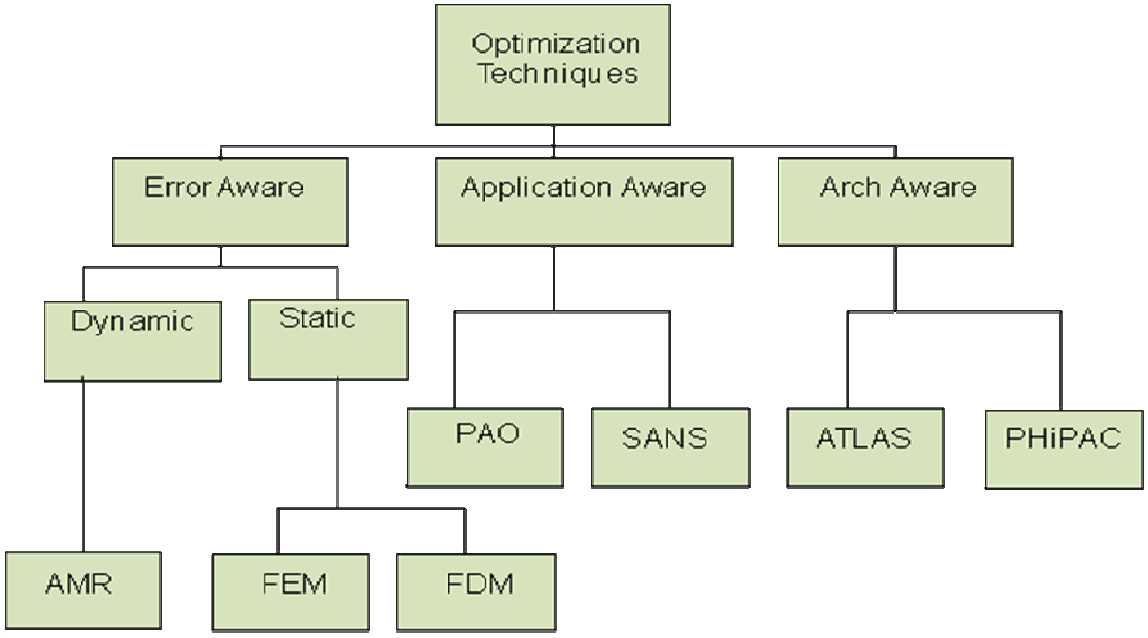


Figure 2-1. Different optimization classified by knowledge exploited [3]

2.3 Power Management Techniques for Computing Systems

Power consumption and thermal management become a critical issue in modern HPC data centers. The main reasons for that is the rising price of electricity, with the cooling infrastructure and cost. The cost of running a data centre through the typical three year life cycle of hardware is becoming comparable to the cost of the data centre itself. Manufactures and researchers are already thinking about an Exaflops system and electrical power is one of their concerns. The most obvious and straightforward way of reducing power is to improve Power Usage Effectiveness of the data centre which is the ratio of total facility power to the power of computer equipment. This is mostly about very efficient cooling methods and power supplies [32].

Power management techniques can be broadly classified into software techniques and hardware techniques. Hardware techniques deal with hardware design of power-efficient systems such as low-leakage power supplies, power-efficient components such as transistors, DRAMs and so on. Software techniques on the other hand, deal with power management of hardware components (processor, memory, hard disk, network card, display, graphics cards etc) within a system – such as a server, laptop, PDA or system-of-systems such as e-commerce data centers, by transitioning the hardware components into one of the several different low-power states when they are idle for a long period of time. This technique is also known as Dynamic Power Management (DPM). DPM is by far the most popular software power management technique and can be grouped into three most discernable sub-classes – predictive techniques, heuristic techniques and QoS and energy trade-offs. The main distinction between these techniques is in the manner in which they

determine when to trigger power state transitions for a hardware component from a high-power to a low-power state or vice-versa. Whereas heuristic techniques are more ad hoc and static in their approach, predictive techniques employ simple to sophisticated predictive mechanisms to determine how long into the future the hardware component is expected to stay idle and use that knowledge to determine when to reactivate the component into a high power state so that it can service jobs again. Recently researchers have started looking into QoS and energy trade-offs as a DPM technique to determine how long a hardware component can be put to 'sleep' such that it does not hurt the performance seen by the applications [51]. This is a more aggressive power management technique that takes advantage of the fact that "acceptable performance degradations" can be employed to let the hardware components sleep longer and hence save more power. This is specifically true for power-hungry domains such as huge data centers that have Service Level Agreement (SLA) contracts with their customers that clearly define the "acceptable performance degradations" such as 97% response time, .05% MTBF (mean time between failures) and so on. On the other hand, a GPU cluster system become very popular in HPC systems today, these systems suffer from high power consumption that can be up to 10 times the power consumption of CPU-only cluster. QoS and energy trade-off DPM techniques can be either Control-theoretic or Optimization based. Our approach to power and performance management is Optimization-based with Self-configuration enabled GPU cluster. We formulate the power and performance management problem as an optimization problem, where we constantly maintain the

GPU components in a power state such that they consume the minimal possible power while maintaining the performance within the acceptable threshold bounds.

Software power management techniques can also be subdivided based on where they reside within the system. They could be power-aware applications, power-aware compilers that perform optimizations on the code for power efficiency, power-aware operating systems that employ the DPM techniques mentioned earlier. And then we have the hardware design optimizations for power-efficiency sitting at the very bottom. Note that, it is possible for a high-level technique to build on top of low-level techniques or share borders across techniques. For example, power-aware compilers can perform code transformations for power-efficiency such that the targeted hardware component can sleep longer. It then employs the power-aware operating system to actually trigger the transition of the hardware component into the low-power sleep state and reactivate it back when required[51].

2.3.1 **Power Management for CPU-Only Systems**

Most software power management techniques exploit the over-provisioning of components, devices or platforms for power savings. This technique also known as Dynamic Power Management (DPM) is extensively used for reducing power dissipation in systems by slowing or shutting-down components when they are idle or underutilized. DPM techniques can be used in isolation for power management of a single system component such as the processor, system memory or the NIC card. They can also be used for joint power management of multiple system components or power management of the whole system [51]. Most DPM techniques utilize power management features supported

by the hardware. For example, frequency scaling, clock throttling, and dynamic voltage scaling (DVS) are three processor power management techniques [94] extensively utilized by DPM. Many works for example, extends the operating system's power manager by an adaptive power manager that uses the processor's DVS capabilities to reduce or increase the CPU frequency thereby minimizing the overall energy consumption [95]. Other approaches is to combine the DVS technique at the processor-level together with a server turn on/off technique at the cluster-level to achieve high power savings while maintaining the response time for server clusters [96]. Also, a very useful scheme is to concentrate the workload on a limited number of servers in a cluster such that the rest of the servers can remain switched-off for a longer time. Similarly, for dynamic memory power management, the usage of multiple power modes of RDRAM memory and dynamically turns off memory chips with power-aware page allocation in operating system shows a promising opportunity for power saving in server system [94][96].

Researchers have also explored joint power management techniques that involve techniques to jointly maintain power consumption of multiple system components such as the memory and the hard disk [51].

2.3.2 Power Management for GPU Based Systems

GPU cluster promise to boost the performance per watt of different application kernels. This means that the rate or the amount of computation that can be delivered by using the GPU cluster will be bigger than any conventional computing system for every watt of power consumed. Currently, the cost of GPUs is of the same order as CPUs but the

theoretical peak performance of GPUs per Watt is roughly an order of magnitude higher [32]. However, the GPU cluster still consuming more power than other accelerators Cluster (i.e. FPGA clusters). FPGA devices offer a significant advantage (4X-12X) in power consumption over GPUs [34]. The computational density per Watt in FPGAs is much higher than in GPUs [33]. This is even true for 32-bit integer and floating-point arithmetic (6X and 2X respectively), for which the raw computational density of GPUs is higher [34].

The power consumed by the GPU card like the T10 series, is significant even when they are idle. The researchers in [38] reported some data about GPU cluster power consumption with idle, active, and loaded server, their results show that the idle power measured for single host server was 230 Watt, without the GPU cards. The idle power of a host server with four-Tesla cards attached to it was measured at about 380 W. However once the cards were activated the power went up to 570 Watt and stayed there. In other words, even if the Tesla server is not used it may be dissipating up to 350 Watt in addition to whatever is being dissipated by the host. When the GPU server is loaded, the power goes up to around 900 W and when the job is done it drops down to around 580 Watt. After the completion of the task execution on the server and all the GPU cards are idle, the Tesla server still consumes about 350 Watt extra.

To mitigate the power issue in GPU cluster, researchers introduce a hybrid cluster that combines GPUs and FPGAs accelerators. Such type of hybrid clusters is motivated by ability of FPGA devices to reduce power consumption significantly over GPUs [34]. An

example of a hybrid cluster is the 16-node cluster combines four NVIDIA GPUs and one Xilinx FPGA accelerators for each of the nodes at NCSA-UIUC [36].

The Idle power consumption become a crucial threaten for the GPU cluster computing, in some cases the power loss share of the idle GPU card could reach up to 60% of the total server power needs. Therefore either the GPU card need to be busy all the time by loading it with a fair share of work, or idle power needs to be much lower by optimizing the SDK drivers. Currently the only alternative is to shut down the host server, switch off the power from the Tesla server through remote management in the PDU (power distribution unit) and power the host back again now without the GPU card [38].

In [65] the energy efficiency of GPUs systems for scientific application was presented, it concluded that using all the cores provides the best efficiency without considering any bandwidth limitation effects that we consider in our work.

The study in [64] presents a thermal management technique for GPUs. A work based on empirical CPU power modeling presented in [97], they introduce a GPU model that does not require performance measurements as other models, by integrating an analytical model for GPU [47] and an empirical power model. There work was mainly focused on the streaming multi-processor components only and they did not consider other GPU cluster components. A feedback driven threading mechanism presented [66]. The feedback system decides how many (cores) can run without degrading performance in order to reduce power consumption by monitoring the bandwidth utilization. There work was done for CPU-only cluster and they did not consider the GPU cluster case.

2.4 Human Cardiac Modeling and Simulation

Cardiovascular disease become a major challenges in the modern health care services, sudden cardiac death is one of the leading causes of mortality in the industrialized world. Each year, heart disease kills more Americans than cancer [73]. In a most of the cases sudden cardiac death is caused by the occurrence of a cardiac arrhythmia called ventricular fibrillation. It is now widely accepted that most dangerous cardiac arrhythmias are associated with abnormal wave propagation caused by reentrant sources of excitation. The mechanisms underlying the initiation and subsequent dynamics of these reentrant sources in the human heart are largely unidentified, primarily due to the limited possibilities of invasively studying cardiac arrhythmias in humans. Consequently, a lot of research has been performed on animal hearts, like the mouse, rabbit, dog to the pig heart. However, the arrhythmias studied in animal hearts are not necessarily the same as those which occur in the human heart [67].

Another major drawback of any experimental study of human heart arrhythmias is that patterns of excitation can be recorded with reasonable resolution only from the surface of the heart, whereas the underlying excitation patterns are 3D. Although some work has been done to overcome these limitations by either using transmural plunge needle electrodes in combination with optical mapping the spatial resolution of current 3D measurement techniques is insufficient to identify the reentrant sources of arrhythmias and to study their dynamics. Computer modeling, especially detailed quantitative modeling of the human heart, can play an important role in overcoming these types of limitations [67].

Mathematical models and computer simulations play an increasingly important role in cardiac arrhythmia research. Major advantages of computer simulations are the ability to study wave propagation in the 3D cardiac wall, which is currently still impossible in experiments, and the ability to bridge the gap between changes in ionic currents and ion channel mutations at a sub-cellular and cellular level and arrhythmias that occur at the whole organ level. Further important application of modeling is studying arrhythmias in the human heart, given the limited possibilities for experimental and clinical research on human hearts.

Cardiac electrical activity arises from the movement of ions across the cell membrane (through ion channels, exchangers and pumps), gap junctions and the extracellular space [67]. Ion channels are complex proteins that regulate the flow of ions (mainly sodium, potassium and calcium) across the cell membrane. Ions flow thought the pores formed by ion channels. Those pores are gated by voltage so that a change in the membrane potential will change the gate status and hence the flow of ions, this in turn will change the membrane potential and so forth. An ion channel can be in an open state and allow the flow of ions, a resting state where is nonconductive but can be opened when its activation threshold voltage is reached or in an inactive state where it will not open at all. Multiple other pumps and exchangers (mainly Na-K-ATPase pump and Na-Ca exchanger) also contribute to ion movement across the membrane.

The resting potential is (-80to -95) mVolt in a normal myocardial cell with the cell interior being negative compared to the extracellular space. It results from the equilibrium potentials of multiple ions mainly the potassium. The equilibrium potential

for particular channel is calculated using the Nernst equation. During the rest state potassium channels are open and the membrane potential is close to the equilibrium potential for potassium.

Action potential (AP) results from opening the sodium and calcium channels which brings the membrane potential closer to the equilibrium potential of sodium and calcium (approximately +40 mV and +80 mV, respectively). This rise in the membrane potential is called depolarization. The depolarization phase of action potential of the fast response tissues (includes the epicardial cells that we are studying) depends on the voltage sensitive and kinetically rapid sodium channels. Depolarization is followed by repolarization during which the membrane potential becomes more negative mainly due to the outflow of the potassium.

The core of any cardiac arrhythmia modeling study is a model describing electrodynamical properties of the cardiac cell. Due to the limited availability of human cardiomyocytes for experimental research, most detailed electrophysiological models have been formulated for animal cardiomyocytes [67]. The Noble model [68] and the Luo–Rudy model [69] were formulated for guinea pig ventricular cells. However, animal cardiomyocytes differ from human ones in important aspects such as action potential shape and duration, range of normal heart rates, action potential restitution and relative importance of ionic currents in the action potential generation. As these factors may influence the mechanism of arrhythmia initiation and dynamics, models of human ventricular myocytes are much needed [67].

In recent years more and more anatomical data on human ionic currents have been gathered from human cardiomyocytes. In addition, a new technique has been developed, involving the cloning of human ion channels and heterologously expressing them in another cell type from which then voltage clamp measurements can be made[67]. As a consequence, in recent years, several models for human ventricular cells have been formulated. In 1998 Priebe and Beuckelmann published the first model for human ventricular myocytes (PB model) [70]. Their model was largely based on the Luo–Rudy model for guinea pig ventricular cells [69] in which formulations for the major ionic currents were adjusted to the scarce data available for human ventricular cells at that time. In addition, for the computer power available at that time, the model was too complex for large-scale spatial simulations of reentrant arrhythmias [67].

This limitation was overcome in a reduced version of the PB model proposed by Bernus et al. [71] (redPB model), where the number of variables was reduced from 15 to 6 by reformulating some currents and fixating intracellular ionic concentrations. A new model for human ventricular myocytes by Ten Tusscher, Noble, Noble and Panfilov (TNNP model) appeared in 2004[72]. This model uses new formulations for all major ionic currents based on a now much wider basis of experimental data, largely from human ventricular cell experiments but also from ion channel expression experiments. The TNNP model was formulated for the purpose of performing large-scale spatial simulations. Therefore, it was constructed to form a compromise between a considerable level of physiological detail and computational efficiency [67]. Later on in 2004 another model for human ventricular myocytes (IMWmodel) was published in [71].

Experimentally recorded data was used by the human ventricular cell models [70] to validate the accuracy of their models; many parameters were used to validate mathematical model accuracy such as the Voltage during the diastolic interval (V_{dia}), Action Potential Amplitudes (APA), and the Action Potential Duration (APD). A second version of the TNNP ventricular cell model presented in [74], introduced by the same group with more extensive and realistic description of intracellular calcium dynamics, including subspace calcium dynamics fast and slow voltage inactivation gate. Many researcher focuses on introducing a minimal, reduced, and computing efficient models, a reduced version of the TNNP ventricular cell model was introduced in [72].

2.5 Case Based Reasoning (CBR)

Rule-based decision support systems draw conclusions by applying generalized rules, step-by-step, starting from scratch. Although there were success in many application areas, such systems have met several problems in knowledge acquisition and system implementation [7-10]. Inspired by the role of reminding in human reasoning, Case based reasoning CBR systems have been proposed as an alternative to rule-based systems, where knowledge or rule elicitation are a difficult or intractable process. In principle, the CBR approach takes a different view, in that reasoning and problem-solving are performed by applying the experience that a system has acquired. Case-based reasoning means using old experiences to understand and solve new problems. In case-based reasoning, a CBR engine remembers a previous situation similar to the current one and uses that to solve the new problem. CBR can mean adapting old solutions to meet new

demands; using old cases to explain new situations; or reasoning from precedents to interpret a new situation (much like lawyers do) or create an equitable solution to a new problem [8][10] . In the CBR system, each case is a piece of information and the case base is a format for cases representation and retrieval. This approach focuses on how to exploit human experience, instead of rules, in problem- solving, and thus improving the performance of decision support systems. Four main processes involved in CBR system as follow:

1. Retrieve: Retrieve the most similar case(s).
2. Reuse: Reuse the case(s) to attempt to solve the problem.
3. Revise: Revise the proposed solution if necessary.
4. Retain: Retain the new solution as a part of a new case.

In summery; A new problem is matched against cases in the case base and one or more similar cases are retrieved. A solution suggested by the matching cases is then reused and tested for success. Unless the retrieved case is a close match the solution will probably have to be revised producing a new case that can be retained. Usually a case retrieval process selects the most similar cases to the current problem, mostly relying on nearest neighbor techniques (a weighted sum of features in the input case is compared with the ones that identify the historical cases).

Case Representation

A case is a contextualized piece of knowledge representing an experience. It contains the past lesson that is the content of the case and the context in which the lesson can be. Typically a case comprises:

- The problem that describes the state of the world when the case occurred.
- The solution which states the derived solution to that problem.

Cases which comprise problems and their solutions can be used to derive solutions to new problems. Whereas cases comprising problems and outcomes can be used to evaluate new situations. If, in addition, such cases contain solutions they can be used to evaluate the outcome of proposed solutions and prevent potential problems. There is exact format for information represented in the case. However, two pragmatic measures can be taken into account in deciding what should be represented in cases: the functionality and the ease of acquisition of the information represented in the case [10].

Case Retrieval

Given a description of a problem, a retrieval algorithm should retrieve the most similar cases to the current problem or situation. The retrieval algorithm relies on direct search about potentially useful cases. CBR system will be useful for large scale problems only when retrieval algorithms are efficient at handling thousands of cases. Unlike database or serial searches that target a specific value in a record, retrieval of cases from the case-base must be equipped with heuristics that perform partial matches, since in general there is no existing case that exactly matches the new case [12]. The cases retrieval in CBR

system required the ability to retrieve relevant cases from its case base quickly and accurately:

- Response time is really important in real-time systems like monitoring and control systems.
- Accuracy is necessary because reporting utilities are useful only if they are reliable.

  The main part of the case retrieval is defining the similarity function, in what follow we describe the similarity function.

Similarity Function

The similarity function or similarity measure used to quantify the degree of resemblance between a pair of cases plays a very important role in case retrieval. Therefore CBR systems are sometimes called similarity searching systems [11]. The dominant approach in CBR similarity function is the distance-based or computational approach, which calculates the distance between cases from a number of objects constituting the cases. The most similar case is determined by the evaluation of a similarity measure.

Among well known methods for case retrieval are: nearest neighbor, induction, knowledge guided induction and template retrieval. These methods can be used alone or combined into hybrid retrieval strategies [10]. In our work we focused on the nearest neighbor algorithm.

Nearest neighbor Algorithm (NN)

This approach involves the assessment of similarity between stored cases and the new input case, based on matching a weighted sum of features. The biggest problem here is to determine the weights of the features [54]. The limitation of this approach includes problems in converging on the correct solution and retrieval times. In general the use of this method leads to the retrieval time increasing linearly with the number of cases. Therefore this approach is more effective when the case base is relatively small [53]. Several CBR implementations have used this method to retrieve matching cases. A typical algorithm for calculating nearest neighbor matching is the one used in [11], where w is normalized and its represent the importance weighting of a feature, Sim(x, y) is the similarity function, and X and Y are the values for feature i in the input and retrieved cases respectively, where i = 1, 2, . . . , N,, as N being the number of features in the case, as shown in function.

$$\mathbf{Sim}(x,y) = \sum_{i=1}^{N} \mathrm{W_i}\ \mathbf{dist}\ (\mathrm{x_i}\ ,\mathrm{y_i})$$

The normalized dist(x, y) is often represented as

$$\mathrm{dist}(\mathrm{x_i}, \mathrm{y_i}) = \frac{|\mathrm{x_i} - \mathrm{y_i}|}{|\mathrm{x_{max}} - \mathrm{y_{max}}|}$$

For numerical features $\mathrm{x_{max}}$, $\mathrm{y_{max}}$ are the maximum values of the $\mathrm{i^{th}}$ feature for x and y respectively.

3 PHYSICS AWARE OPTIMIZATION FOR LARGE-SCALE SCIENTIFIC AND MEDICAL APPLICATIONS

3.1 Introduction

Large scale scientific and medical applications generally experience different phases at runtime and each phase has different computational requirements. An optimal solution or numerical scheme at one execution phase might not be appropriate for the next phase of application execution. Choosing the ideal numerical algorithms and solutions for an application remains an active research area because of the heterogeneous execution behavior of applications at runtime especially for time-varying transient applications. In this chapter, we will introduce our proposed innovative programming paradigm, Physics Aware Optimization (PAO), as a Self-Optimization technique to model and simulate cardiac electrophysiology. This problem is very important because cardiovascular disease is the No. 1 killer in the United States with heart rhythm disorders being a significant direct cause of death [73]. Understanding the fatal heart rhythm disorders is essential for developing effective treatment and prevention plans.

An innovative Physics Aware Optimization (PAO) approach that dynamically changes the application solution parameters (temporal or spatial parameters) or solution methods in order to exploit the physics properties being simulated to achieve efficient simulation of large scale scientific and medical applications. In this approach, the application execution is periodically monitored and analyzed to identify its current execution phase (e.g., current phase of the cardiac action potential phases) and predict its next execution phase. For each change in the application execution phase, the simulation parameters

(e.g., Δt, Δx, Δy, Δz) and/or numerical algorithms/solvers that are appropriate to the current phase and application physics characteristics are chosen. The development of real-time and interactive simulation of human cardiac electrical activities based on PAO methodology will lead to a significant reduction in the computational requirements without affecting the accuracy.

3.2 Physics Aware Optimization (PAO) Research

Introduction and Motivation

In the domain of scientific computations, transient problems are usually encountered in a large class of problems [39][40]. However, transient applications are generally time dependent and their volatile nature make them hard to be solved. As time evolves, transient problems will evolve to different phases with different physical characteristics. Most of the current research typically uses one algorithm to implement all the phases of the application execution, but a static solution or numerical scheme for all the execution phases might not be ideal to handle the dynamism of the problem.

Some techniques use application states to refine the solution as in Adaptive Mesh Refinement (AMR) [41] where the mesh size is refined to achieve a smaller error rate. In this scheme, the error is used to drive the dynamic changes in the grid point size. In our approach, we take a more general runtime optimization methodology that is based on the current application phase and the application physics temporal and spatial characteristics. For transient problems, a different solution that can meet the desired accuracy and performance will be determined and then applied for each application execution phase. Our method can be applied to heart simulations, Finite Element Method, Domain

Decomposition Method, Finite Volume Approximations [42], etc. Our method can also be applied to steady state problems, but the assigned solution will not be changed at runtime due to the static nature of the steady state problems.

We determine the application execution phase by monitoring the application execution, identifying the application phase changes and predicting its next phase by exploiting the knowledge about the application physics (e.g., the cardiac action potential application goes through 5 phases) and how it behaves at runtime, and then use the appropriate parameters (e.g., Δt, Δx, Δy, Δz) and/or numerical solution/algorithm for each predicted phase during the application execution. For each application execution phase, different numerical schemes and solvers that exploit its physics characteristics and its state were chosen. POA results show that the PAO implementation has very little overhead and its solution accuracy is comparable to the solution obtained by using the finest grid size.

3.2.1 **Physics Aware Programming (PAO)**

In order to explain our PAO methodology, we consider the example presented in [3] for a diffusion application that utilizes routines that are widely used in many different real-world applications (e.g. sideway heat equation [44], boundary-layer problems [45]).

Let us consider a heat diffusion problem shown in Figure 3-1.

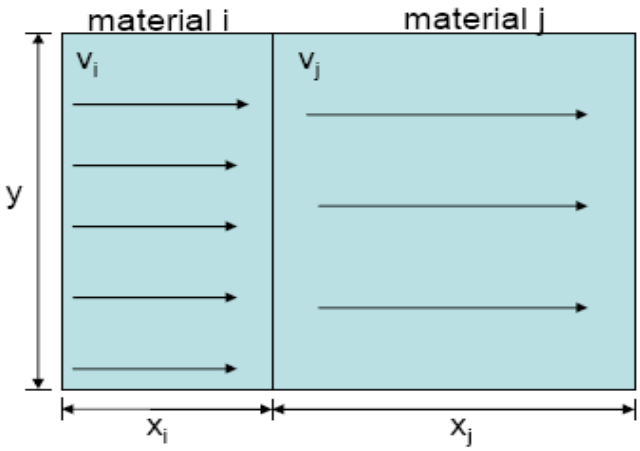


Figure. 3-1: An example of diffusion problems

The heat propagation problem can be modeled as a diffusion-type problem and it is governed by Equation (1).

$$\alpha_x \frac{\partial^2 u}{\partial x^2} + \alpha_y \frac{\partial^2 u}{\partial y^2} + \alpha_z \frac{\partial^2 u}{\partial z^2} = \frac{\partial u}{\partial t} \tag{1}$$

Where u is the temperature, α_x, α_y and α_z are the thermal conductivity in the x, y and z coordinate directions. Let us also assume the heat is propagating through two different materials i and j. In reality, more mixed materials can be encountered, but we will only consider two materials in this illustrative example for simplicity. The thermal conductivity α will be used to determine the temperature change in these two materials. The Partial Differential Equations (PDE) shown in Equation (1) are usually solved by applying Taylor's theorem [46],

$$u_t(x,t) = \frac{u(x,t+k) - u(x,t)}{k} + TE$$
$$TE = -\frac{k}{2} u_{tt}(x,\tau), t < \tau < t+k \tag{2}$$

Where TE is a truncation error, i.e., an analytical PDE solution equals numerical solution plus the truncation error (TE).

In PAO methodology, we exploit the heterogeneous properties of the material to reduce the computational complexity of the algorithms modeling the heat diffusion in these materials. In what follows, we show how to apply the PAO approach to choose the spatial and temporal characteristics of the solutions based on the thermal conductivity α associated with media type in order to improve the performance of Equation 2 solver.

Determining the Temporal Characteristics (Δt)

In our PAO implementation for large scale medical application, we consider the temporal characteristics of our application. The temporal characteristics of the solution depend primarily on the required accuracy. For temporal characteristics with respect to one dimension as the case in our medical application, we discretize the problem domain into small grids. On each grid point, u_n^j represents the analytical solution. Suppose that v_n^j represents a measured or computed value of u_n^j that differs from the exact u_n^j value by an error amount e_n^j that is,

$$u_n^j = v_n^j + e_n^j \qquad (3)$$

Usually, the user will specify a small constant E as the upper bound on the acceptable error for the computation, thus for the errors e_n^j, we have

$$| e_n^j | \leq E \quad \forall \text{ n, j} \qquad (4)$$

Finally, assume that M is an upper bound for $u_{tt}(x,t)$. Notice that u_{tt} is the acceleration of the heat transmission in the problem, this value can be obtained by solving the problem for one time step and then use it to estimate the maximum acceleration of the heat.

$$|u_{tt}(x,t)| \le M \quad \forall \text{ x, t} \qquad (5)$$

Now, if we apply the forward difference formula to estimate $u_t(x_n,t_j)$, we get

$$u_t(x_n,t_j) = \frac{u_n^{j+1}-u_n^j}{\Delta t} - \frac{\Delta t}{2}u_{tt}(x_n,\tau_j), t_j < \tau_j < t_{j+1} \qquad (6)$$

This is the same as Equation (2). According to Equation (3), Equation (6) can be rewritten into

$$u_t(x_n,t_j) = \frac{v_n^{j+1}-v_n^j}{\Delta t} + \frac{e_n^{j+1}-e_n^j}{\Delta t} - \frac{k}{2}u_{tt}(x_n,\tau_j) \qquad (7)$$

From Equation (7), the u_t consists of three parts: a computed difference quotient, a rounding-measurement error and a truncation error. Hence the total error incurred in using the difference quotient $(v_n^{j+1}-v_n^j)/\Delta t$ to approximate $u_t(x_n,t_j)$ can be bounded as follows:

$$\left|\frac{e_n^{j+1}-e_n^j}{\Delta t} - \frac{\Delta t}{2}u_{tt}(x_n,\tau_j)\right| \le \left|\frac{e_n^{j+1}-e_n^j}{\Delta t}\right| + \left|\frac{\Delta t}{2}u_{tt}(x_n,\tau_j)\right| \le \frac{2E}{\Delta t} + \frac{M\Delta t}{2} \qquad (8)$$

In order to satisfy the desired accuracy, we choose Δt that minimizes the maximum total error. In this example, $\Delta t_{opt} = 2\sqrt{E/M}$ where M is an upper bound on the heat transmission acceleration. The M value depends on the physics properties of each material. For a fixed error E, Δt can be optimally chosen to improve solution performance by increasing Δt (solution uses less number of time steps) without compromising the accuracy of the

solution. For example, Δt for the rubber material can be made much larger than the Δt for the aluminum material. If physics properties are not exploited, which is normally the case, the domain scientists will choose the smallest Δt for all the materials shown in Figure 4.1. In PAO approach, the thermal conductivity of each material is utilized to determine the optimal time step of the solution method associated with each sub-domain that minimizes the maximum total error. For example, assuming we have a computation domain composed by two different materials with a constant heat boundary, if we found the heat transmission acceleration in each material is 0.2 and 7.2 respectively through our calculation, then the time step in the slow transmission material could be set 6 times faster than the other one. This makes sense since in a faster heat transmission; we need a smaller time step to capture the heat variation between each time step.

Determining the Solution spatial characteristics (Δx, Δy, Δz)

The appropriate spatial characteristics Δx, Δy, Δz of the solution depends on the physical properties of the material and the numerical method being used. If implicit method is used, for our example, the stability criterion for the x direction can be determined by:

$$\frac{2\alpha\Delta t}{(\Delta x)^2} \leq 1 \qquad (9)$$

Where α is the heat conductivity along the x direction. The same criteria can be applied for y and z directions. The intuition behind equation (9) is that as long as within one time step, the heat transmission distance is within Δx, the computation should be stable, i.e., Δx captures the heat transmission distance during one time step. Traditionally, domain scientists will apply the smallest spatial characteristics h = min(Δx, Δy, Δz) for the entire

domain to achieve stability. As discussed in the previous section, different thermal conductivity materials will have different time steps and thus their spatial characteristics could be chosen such that the computational complexity is reduced while maintaining stability. In the example shown in Figure 3-2, PAO-based solution will use larger spatial characteristics for material i since it has a higher thermal conductivity (i.e., it has slow heat transmission rate). For example, if we keep Δt as a constant, the material with larger heat conductivity could have a larger spatial characteristic.

3.3 POA for Human Heart Simulation

3.3.1 Human Ventricular Epicardia Myocyte Model

Cardiac electrical activity arises from the movement of ions across the cell membrane (through ion channels, exchangers and pumps), gap junctions and the extracellular space [39]. Ion channels are complex proteins that regulate the flow of ions (mainly sodium, potassium and calcium) across the cell membrane. Ions flow thought the pores formed by ion channels. Those pores are gated by voltage so that a change in the membrane potential will change the gate status and hence the flow of ions, this in turn will change the membrane potential and so forth. An ion channel can be in an open state and allow the flow of ions, a resting state where is nonconductive but can be opened when its activation threshold voltage is reached or in an inactive state where it will not open at all. Multiple other pumps and exchangers (mainly Na-K-ATPase pump and Na-Ca exchanger) also contribute to ion movement across the membrane.

The resting potential is 80-95 mV in a normal myocardial cell with the cell interior being negative compared to the extracellular space. It results from the equilibrium potentials of

multiple ions mainly the potassium. The equilibrium potential for particular channel is calculated using the Nernst equation. During the rest state potassium channels are open and the membrane potential is close to the equilibrium potential for potassium.

Action potential (AP) results from opening the sodium and calcium channels which brings the membrane potential closer to the equilibrium potential of sodium and calcium (approximately +40 mV and +80 mV, respectively). This rise in the membrane potential is called depolarization. The depolarization phase of action potential of the fast response tissues (includes the Epicardial cells that we are studying) depends on the voltage sensitive and kinetically rapid sodium channels. Depolarization is followed by repolarization during which the membrane potential becomes more negative mainly due to the outflow of the potassium.

Action potential consists of 5 phases (see Figure 3-2):

Phase 0 (Rapid Depolarization) is a quick rise in membrane potential due to the opening of the sodium channel and the inflow of sodium.

Phase 1 is a partial repolarization due to the activation of the transient outward potassium currents and the rapid decay of the sodium current.

Phase 2 (Plateau) results from the balanced calcium inflow and potassium outflow.

Phase 3 is a complete repolarization down to the resting potential resulting from the outflow of potassium.

Phase 4 is the resting phase.

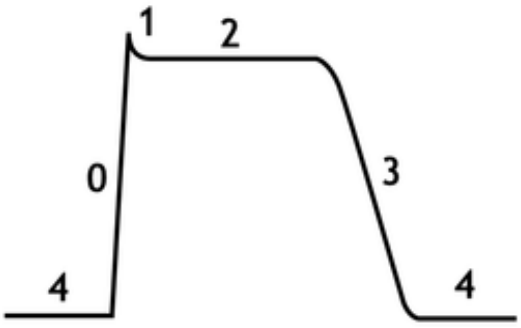


Figure. 3-2. The cardiac action potential phases

The behavior of ion channels, pumps and exchangers has been studied and mathematical models were developed to predict the reciprocal effects of voltage and ion flow. However, because of the complexity of these models it is practically impossible to solve them without computation.

Computerized models have been used to study rare channel disorders, drug effects [48] and the basic mechanisms of arrhythmias. The anatomically correct models have large number of cells in three dimensional distributions. Solving such models requires long periods of time even using supercomputers [74]. To overcome this problem multiple "simplified" models have been develop. However, Simplified models increase the speed of simulation at the cost of a decrease in the accuracy.

We investigated the use of the PAO programming paradigm to improve the performance of the simulation without compromising its accuracy.

We used the human ventricular model by ten Tusscher. Please refer to ten Tusscher et al. for a detailed description of ion channels and their gates [75]. We used the status of selected gates **m, h, j, d,** and **f** as the parameter that would differentiate one AP phase from another. The phase detection algorithm uses the distinct values for these gates

during each phase to detect the current phase. The basic model for human ventricular tissue presented in [74] assumes a fixed Δt during the simulation. In the following subsections, we will discuss how to use data mining techniques to detect the current phase of the Ten-Tusscher simulation as it is required by the PAO, and then the simulation parameters are changed for each phase of the simulation to improve performance.

3.3.2 **AP Phase Detection Technique**

Data mining involves analyzing a given data set to discover previously unknown, valid patterns and relationships in this data set [49][60]. The analysis uses statistical models, mathematical algorithms, and machine learning methods (algorithms that improve their performance automatically through experience, such as neural networks or decision trees) [49]. Data mining applications can use a variety of parameters to examine the data. They include association (patterns where one event is connected to another event), sequence or path analysis (patterns where one event leads to another event), classification (identification of new patterns), clustering (finding and visually documenting groups of previously unknown facts), and forecasting (discovering patterns from which one can make reasonable predictions regarding future activities) [50]. Some of the most common data mining algorithms in use today include statistical modeling, constructing decision trees, constructing rules, mining association rules; instance based learning, linear models as linear and logistic regression, and clustering. In this application, we use the Nearest Neighbors (NNG) [53] algorithm to mine the data sets generated by the medical application.

The Nearest Neighbors algorithm (NNGs) is used to generate the AP phase rules for a given data set [50]. In this scheme, to predict the value in an unclassified record, the algorithm looks for records with similar predictor values in the database and uses the prediction value that presented in the record that is nearest to the unclassified record. On other words, the nearest neighbor prediction algorithm simply states that the objects that are "near" to each other will have similar prediction values as well. Thus if you know the prediction value of one of the objects you can use it to predict its nearest neighbors. The distance function that is used to quantify the neighbors is based on Euclidean distance. We applied this algorithm to detect the action potential phase during each time step of the Ten-Tusscher's model of a human ventricular epicardial myocyte simulation. The simulation was paced at a varied cycle length (1000 to 50 ms) using the dynamic restitution pacing protocol [74]. The data sets are generated from an accurate simulation with a fixed Δt of 0.001 ms. Data about transmembrane voltage (Vm), and intracellular calcium (C_{ai}) and GS were recorded. Action Potential (AP) was visually inspected and AP Phases 0,1,2,3 and 4 were marked using Matlab software to produce (V_APP) data set that will be used to compare with the results produced when we apply the PAO approach to implement the same simulation. The V_APP data sets contain information about the voltage, gates values, and the corresponding AP phase (V_APP).

Our phase detection approach is based on mining the **H, J, M, D** and **F** gate values that have unique values for each phase. We identify a set of rules that map accurately the gate values to the appropriate Action Potential phase. For example, phase 4 has the following gates values (phase 4 rule) ranges **(58671.86616<=h<=77292.2916,**

11976.6527<=j<=77282.979, 136.753414<=m<=161.9938871, 1.96122901<=d<=2.190402, 47087.29658 <=f<=99981.83352).

Consequently, at any instant of time in the simulation, by knowing the gate values, we will be able to determine accurately the current AP phase. The next step is then to determine for each phase, what is the appropriate simulations time step that speedups the simulation while maintaining the required accuracy.

3.3.3 PAO implementation of the Ten-Tusscher's simulation

The PAO implementation of the simulation is shown in Figure 3-3. The application's execution state is continuously monitored as well as the current gate values (**H, J, M, D** and **F**) (steps 2 and 3). Once the current gate values are determined, the data mining rules will be invoked to predict the AP phase and consequently the appropriate Δt to run the simulation in the next iteration of the simulation (Steps 4-6). The algorithm continues its execution with the new Δt (Steps 6 and 7).

PAO_Phase Algorithm

```
1: While (true)
2:      Current_Phase       ← monitoring (Appl)
3:      Current_Gate_State  ← monitoring (Appl)
4:      APP_Phase_Prediction = analysis  (Current_Gate_State)
5:      if (APP_Phase_Prediction ≠ Current_Phase)
6:          Δt = planning (APP_Phase_Prediction)
7:          Execute (Appl, Δt)
8:      end if
9:  end while
```

Fig. 3-3: PAO-based Algorithm

3.4 Experimental Results and Evaluation

To evaluate the performance of the PAO implementation, we experiment with three cases to map the Δt to each AP phase:

Case 0: Reference Implementation – In this case, we apply the smallest time step, Δt=0.001, to all phases.

Case 1: Two Δt values are used (Δt=0.001 for phases 0, 1, 2 and 3, Δt=0.1 for phase 4)

Case 2: Three Δt values are used (Δt=0.001 for both phases 0 and 1, Δt=0.1 is used for both phases 2 and 4, Δt=0.01 for phase 3)

Case 3: Four Δt values are used (Δt=0.001 for both phases 0 and 1, Δt=0.1 for phase 2, Δt=0.01 for phase 3, and Δt=1 for phase 4).

We compare the performance of the PAO implementation of the algorithm with the reference implementation (**Case 0**), that is considered the most accurate implementation, and as well as the misclassification rate that defines the number of cases in which the rules misclassified the current AP phase of the application with respect to the overall number of phases. The simulation time, misclassification rate and speedup with respect to **Case 0** are shown in Table 3-1. It is clear from Table 4 that **Case 1** gives the best misclassification rate (around 1.7%) while achieving a speedup of 2.89X when its simulation time is compared to **Case 0** simulation time. **Case 3** has the best speedup but the worst misclassification rate.

Table 3-1. Evaluation of PAO implementations

Case No.	Simulation time (sec)	Misclassification Rate %	Speedup
1	**16.988**	**1.7**	**2.89 X**
2	**12.37**	**6.96**	**3.97 X**
3	**8.108**	**16.38**	**6.06 X**

The next important issue in our analysis is to quantify the accuracy of the PAO implementation when compared with **Case 0** implementation. We use statistical correlations to evaluate the accuracy of PAO implementation. Table 3-2 shows the statistical correlation between the voltage (v), intracellular calcium concentration (Cai), Intracellular sodium concentration (Nai), and the Intracellular potassium concentration (Ki). A perfect correlation (correlation=1, which indicate that both signals are identical) for V, Cai, Nai, Ki, in phases 0, 1. These results come from the fact that in phases 0 and 1, the PAO and the reference case implementations use the same value of Δt. The correlations values for phase 4 and 3 in PAO Cases 1 and 2 give a very close to the perfect correlation while they improve the performance by almost 3 and 4 times, respectively.

Table 3-2. Quantifying the accuracy of the PAO implementation.

phase	V			Cai			Nai			Ki		
	Case 1	Case 2	Case 3	Case 1	Case 2	Case 3	Case 1	Case 2	Case 3	Case 1	Case 2	Case 3
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
2	**1**	**0.90**	**0.89**	**1**	**0.89**	**0.92**	**1**	**0.93**	**0.91**	**1**	**0.94**	**0.87**
3	**1**	**0.97**	**0.95**	**1**	**0.98**	**0.95**	**1**	**0.99**	**0.95**	**1**	**0.98**	**0.95**
4	**0.98**	**0.98**	**0.92**	**0.98**	**0.97**	**0.94**	**0.98**	**0.97**	**0.94**	**0.98**	**0.98**	**0.95**

4 PERSONAL SUPERCOMPUTING AND GPU CLUSTER SYSTEM

Complexity of applications has increased exponentially with a growing demand for unlimited computational resources. Cluster based supercomputing systems and high performance data centers have traditionally been the ideal resources to fulfill the ever increasing computing demand. Such computing resources require multimillion dollar investment for building the system. The cost increases with the amount of power used for operating and cooling the system along with the maintenance activities. In this Chapter we investigate the Personal Supercomputing as an emerging concept in high performance computing that provide an opportunity to overcome most of these problems. We explore and evaluate the GPU- based Personal Supercomputing system, and we compare it with conventional high performance cluster based computing system. Our evaluations show promising opportunities in using GPU based cluster for high performance computing

4.1 Introduction

In many fields of computing, universities, research institutions, and industries face the grand challenges of responding to the computation demand of today's applications.. A recent trend in high-performance computing is the development and use of architectures and accelerators that combine different degrees of parallelism granularity using thousands of heterogeneous and distinct processing elements. Primarily the demands for these accelerators is driven by consumer applications, including large scale scientific applications like geosciences simulation, molecular biology, medical imaging, computer gaming and multimedia, just to name a few. Many high performance accelerators are

available today such as graphics processing units (GPUs), field programmable gate arrays (FPGAs), and the Cell Broadband Engines (IBM Cell BEs). These processing elements are available as accelerators or many-core processors, which are designed with the objective of achieving higher performance for data parallel applications. Compared to conventional CPUs, the accelerators can offer an order-of-magnitude improvement in performance per dollar and per watt [13].

Massively parallel Graphics Processing Units (GPUs) have become a common device in most of today's computing systems ranging from personal workstation to a high performance computing clusters. For example, NVIDIA Tesla C1060 GPU contains 240 Streaming Processor (SP) cores with memory bandwidth of 102 GB/sec delivers approximately one TFLOPS peak performance. The emerging general-purpose programming models for the GPUs like CUDA and OpenCl provide a great opportunity for the programmers to utilize these massively parallel GPUs, as a computing resource for many compute intensive applications, and reduce the development cycle cost compared to the early methods (i.e. the usage of graphics API). Many researchers and developers have reported that a significant speedup could be achieved by using the GPUs [14-16], and there are hundreds of applications with the reported 100X and over speedups over the CPU-only implementations.

In this study we explore emerging concept of the Personal Supercomputing (PSC) as a contrast to conventional CPU-based high performance clusters and data centers. For many years, researchers and developers relied on a shared and an intensive computing resource to fulfill the computing needs for their complex application. Today with the

introduction of unprecedented powerful GPUs (e.g Tesla and Fermi), a GPU-based personal supercomputing (PSC-GPU) have become a reality, and it's being adopted by more and more people in both academic and industrial communities.

4.2 GPU and CUDA Programming

Modern GPUs such as Tesla 10 series and Fermi are very efficient in manipulating scientific applications based on their highly parallel structure. The efficient parallel structure makes GPUs more effective than traditional CPUs for a many data parallel applications. As shown in Figure 4-1, GPU is composed of a set of multiprocessors MPs; where each multiprocessor has its own streaming processors (SPs) and shared memory. A global memory is shared between all the processors.

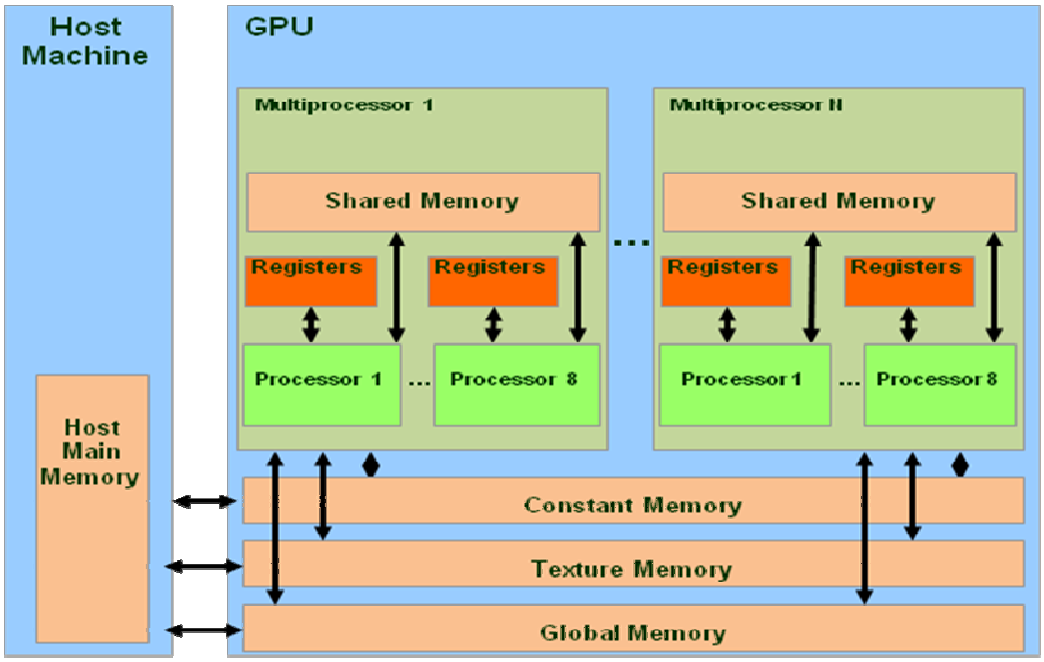


Figure 4-1: GPU Architecture

The concurrent execution of thousands of threads hides the memory latency, e.g. global memory latency is up to 600 cycles. In addition, a set of registers and shared memory are partitioned among the executing threads. CUDA is the programming environment for exploiting the parallel compute engine in NVIDIA GPUs. A CUDA program consists of one or more phases in which typically the serial components are executed on the host CPU and the data parallel sections are executed on the GPU card. Though a given SP will operate on its own unique piece of data, each instruction unit must issue the same single instruction to its group of streaming processors at once. Here, we focus on NVIDIA's architecture and programming environment (CUDA). The selection of the NVIDIA technology is based on two major reasons. 1) It has the largest market share, 2) CUDA programming model is the most mature GPU programming environment with the availability of training and development support [17]. An application utilizing the GPU goes thorough different phases, and the number of phases depends on the underlying architecture (i.e. single GPU or multi GPU). In the case of a single GPU, there are five main phases: preprocessing, Data movement to the GPU, GPU processing, data movement from the GPU, and post-processing [17]. But in the case of multi GPU we have an extra phase after the GPU processing which is the GPU-GPU data movement. One important observation is that in the case of multi GPU, the GPU-processing and the GPU-GPU data movement are repeatable phases during the execution of the application, but in the case of single GPU, only the GPU processing phase can be a repeatable phase. Table 4-1 shows the main phases for applications utilizing the GPU.

Table 4-1. GPU application processing phases

Phase	Sub-phase	Description
1. Preprocessing	1.1 GPU initialization	GPU initialization
	1.2 Host memory allocation	Memory allocation at the host and the GPU
	1.3 Pre-processing	Application-specific data preprocessing on the CPU
2. Data Movement to GPU	2.1 Data movement to GPU	Data movement from host's memory to GPU global memory
3.GPU Processing	3.1 Data movement to shared Memory	Data movement from global GPU memory to shared memories.
	3.2 GPU Processing	Application 'kernel' processing
	3.3 Data transfer to device global memory	Result movement from shared memory to global memory
4. GPU-GPU Data Movement	4.1 GPU-GPU data movement	GPU-GPU data movement
	4.2 Go to sub phase 3.1	Repeat GPU processing phase
5.Data Movement from GPU	5.1 Output data transfer	Movement the results to the host system memory
6. Post processing	6.1 Post-processing	Application-specific post processing and CPU resource deallocation

4.3 Personal Supercomputing and GPUs

. Personal Supercomputing (PSC) concept emerged as a new trend in today's high performance computing. One definition for a PSC is that "a system is a PSC if it's in form of a standard desktop or workstation, and it's able to deliver the equivalent

computing power of a cluster, with approximately 1/10th of the cost, space, and power consumption"[100]. Another simpler definition is that "a system is a PSC if it's cost as workstation and performs as a cluster". With the NVIDIA, Tesla 10 series GPUs (i.e. Tesla C1060), the age of Personal supercomputing using GPU has started, and it has kept evolving to this date and it's expected to keep evolving in the near future. In order to make the GPU-based HPC system available and cost effective for the users and developers, NVIDIA developed a programming language called CUDA (Compute Unified Device Architecture), which allows the GPU to be programmed like an x86 CPU. Officially and during Super Computing 2008 in Austin (SC 08), NVIDIA unveiled their Personal Supercomputer. A system with four C1060 cards delivers up to about 4 TFLOPS. A preconfigured cluster was the next generation of the PSC. Preconfigured cluster follows the model of the Personal Supercomputer, and it's known as "Accessible Supercomputing". Preconfigured cluster built from NVIDIA's S1070 cards coupled with CPU servers [18] is now widely deployed all over the globe.

Personal Supercomputing computing framework provides an opportunity to overcome most of the issues associated with data center and supercomputing clusters. In the next sub section we present the benefits of GPU systems over conventional computing system with examples.

4.3.1 **Personal Supercomputing Benefits**

CPU/GPU clusters with their supercomputing power are utilized to solve today's critical problems like Biomolecular Protein Folding for medical companies, explosion

simulations and nuclear armament for the US Military, and financial simulations. The customers are motivated by high availability, less footprint area, and impressive power and cost savings. Three different examples were reported in [18][19]. We will evaluate the systems employed by the BNP Paribas for Corporate and Investment Banking (CIB) , Hess Corp. for oil and gas, and the TACC Ranger [20].

Even with huge efforts to optimize the power consumption, energy costs are the fastest-rising cost element in the data centers. Today, data centers generate a large amounts of heat that requires huge cooling systems with high cooling costs. PSC has overcome this issue by providing a low power (**Performance Per Watt**) system that has limited cooling requirements compared to data centers and supercomputing clusters. For the same amount of workload PSC delivers more than 90% savings in power compared to the CPU-only systems as illustrated in Table 4-2.

Table 4-2. CPU-only and PSC-GPU power comparison

CASE	CPU Only (Watt)	PSC-GPU (Watt)	Power Saving %
BNP Paribas Corp.	37.5 K	2.8 K	92.5
Hess Corp.	1.2 M	45 K	96.2

The cost to build and operate a modern Data Center continues to increase. This Total Cost of Ownership (TCO) includes capital and operational expenses of data centers. Table 4-3 shows more than 90% saving in TCO for PSC-GPU.

Table 4-3. CPU-only and PSC-GPU TCO comparison

CASE	CPU Only ($)	PSC-GPU ($)	Cost Saving %
BNP Paribas Corp.	0.6 M	24 K	96
Hess Corp.	8 M	400 K	95

Table 4-4 shows more than 90% saving in Area for PSC-GPU.

Table 4-4. CPU-only and PSC-GPU Area comparison

CASE	(CPU-Core)	PSC-GPU (Tesla S1070)	Area Saving %
BNP Paribas Corp.	500	2	90
Hess Corp.	8000	32	90
TACC Ranger	62,976	145	95

In summary, as shown in the tables 4-2, 4-3, and 5-4 above, the PSC-GPU system can shrink the foot print area down to 10% or less of the floor space, along with the cost, power and cooling requirements.

4.4 GPU Cluster

The adoption of GPUs in high performance computing forced the development of a new software framework to make using the GPUs available for every programmer in the HPC

community. CUDA, AMD/ATI's Close to Metal (CTM) and OpenCL have become more known within the HPC community as programming frameworks for the CPU- GPU clusters [21]. NVIDIA CUDA SDK is the dominant framework in this field and it has approximately 50% of the market share. CUDA is a set of tools, libraries, and C language extensions that allow developers have easier access to the GPU hardware than typical graphics libraries such as OpenGL.

The Tesla S1070 is considered as the first teraflop many-core processor, with 960 processor cores and a standard C compiler that simplifies application development. It's available for HPC as either a standard add-on board, or in high-density self-contained 1U rack mount cases containing four GPU (Tesla S1070) devices with independent power and cooling infrastructure [21]. Zhe Fan et.al. presented their first attempt to introduce GPU clusters to HPC community for solving the computation demand of visualization systems in [22]. Over time, HPC research community started to move applications from the conventional CPU-only clusters to a GPU cluster [23-26][28]. Although with the booming use of GPUs as accelerators, GPU based clusters show a number of new problems in performance and power consumption, programming, resource allocation and sharing, security, workload scheduling, and resources utilization.

4.4.1 GPU Cluster Architecture

Several GPU clusters have been deployed since 2004. However, the majority of them were deployed as visualization systems [22]. Only recently attempts have been made to

deploy GPU based clusters for a wide range of applications. Early examples of such installations include a 160-node (based on NVIDIA QuadroPlex technology) “DQ” GPU cluster at LANL [23]. The majority of such installations are highly experimental in nature, and GPU clusters specifically deployed for production used in HPC environments are still rare [21]. In 2009 NCSA deployed two GPU clusters based on the NVIDIA Tesla S1070 Computing System: a 192-node production cluster “Lincoln”, and an experimental 32-node cluster “AC” [30]. Also, Science and Technology Facilities Council (STFC) in UK introduced 8 nodes with NVIDIA Tesla S1070. The CSIRO GPU supercomputer provided a128 Dual Xeon E5462 Compute Nodes with a total of 1024 2.8GHz compute cores, with 64 Tesla S1070 (256 GPUs with a total of 61440 streaming processor cores) connected with a DDR InfiniBand interconnect [29].

In General, a GPU cluster is composed of four major components: host nodes, GPUs, software tools, and interconnect. As a result of a high expectation for the GPUs to carry out a significant portion of the computations, the host memory, PCIe bus, and network interconnect performance need to be coordinated with the GPU performance in order to harness the GPUs powerful capabilities, and maintain a well cooperative system. In particular, high-end GPUs, such as the NVIDIA Tesla, require full-bandwidth PCIe 2 x16. Also, InfiniBand QDR interconnect is highly desirable to match the GPU-to-host bandwidth. Host memory also needs to at least match the amount of memory on the GPUs in order to enable their full utilization, and a one-to-one ratio of CPU cores to GPUs may be required from the software development perspective as it greatly simplifies the development of parallel applications (MPI, and OpenMP). However, in the actual

deployment, some of these requirements are hard to meet and issues other than performance considerations, such as system availability, resources utilization, cost effective computation, and power and thermal emergencies may become dominant. Figure 4-2 shows NVIDIA Tesla S1070 GPUs cluster architecture.

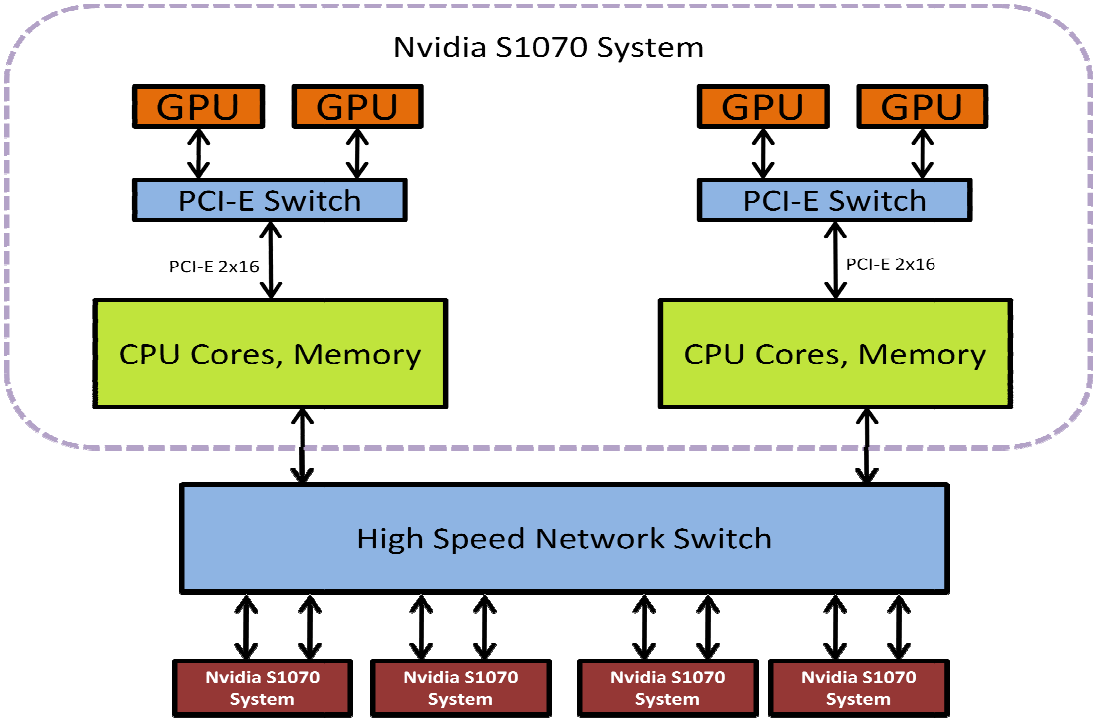


Figure 4-2: Tesla S1070 GPUs cluster architecture.

As shown in Figure 4-2, the NVIDIA Tesla S1070 Computing System is a 1U rack-mount system with four Tesla T10 computing processors, with a total of with 960 processor cores, and 16.0 GB of high speed memory, configured as 4.0 GB for each GPU. It can be connected to a single host system via two PCI Express (PCIe 2 x16) connections to that host, or connected to two separate host systems via one PCIe

connection to each host as shown in Figure 4-2. The host interface cards are compatible with both PCIe 1x and PCIe 2x systems. Each NVIDIA switch and corresponding PCIe cable connects to two of the four GPUs in the Tesla S1070. If only one PCI Express cable is connected to the Tesla S1070, only two of the GPUs will be used. To connect all four GPUs in a Tesla S1070 to a single host system, the host must have two available PCI Express slots and be configured with two cables [35].

4.5 GPU Clusters: Challenges and Problems

The usage of GPUs as accelerators in HPC clusters can present many advantages as outlined above, but GPU based clusters show a number of new problems in terms of the resource utilization, the application development process, workload scheduling, resource management, power consumption, and security. In this section we evaluate some of these issues.

1. Low GPU resources utilization

Low GPU resource utilization becomes a critical issue in today's GPU based cluster, in which millions of cycles are lost with idle Streaming Processors (SP) due to weak application development and system setup. To fully utilize the GPU device, the application developer needs to exploit large amount of data parallelism and create enough threads to fully utilize the hardware. This requires the developer to be aware of the CUDA architecture and the memory hierarchy [21][37]. GPU developers are facing a challenge in low level programming in CUDA, and memory hierarchy managements

[21]. From the cluster based system perspective host memory needs to match the amount of memory on the GPUs in order to accommodate their full utilization, and at least one-to-one ratio of CPU cores to GPUs may be needed from the software development perspective as it greatly simplifies the development of parallel applications (MPI, and OpenMP) [21].

2. GPUs and the host device bandwidth degradation

The achievable bandwidth between GPUs and the host device depends mainly on the PCIe interface mapping to the CPU cores. When the data is sent from the memory attached to the same CPU with the PCIe interface, a higher bandwidth is achieved. 20% bandwidth degradation was reported in [21], which represented about 1GB/s, when there is a separation between the CPU and PCIe interface.

3. Resource Allocation and Sharing

Most of applications or cluster resource management systems like Torque Batch system, have a full awareness about the CPU cores, storage, and memory as managed resources, but these systems have no such ability for GPUs. Cluster resource management systems can allow users to acquire nodes with the desired resources, but this by itself does not prevent users from interfering with each other while running applications in the GPUs. This fact prevents a dependable shared multi-user environment to be used when sharing the same node by multi users. Any cluster resource management system should have a

sufficient amount of information about underlying hardware such as CPUs to GPUs ratio, and GPU utilization.

4. Workload Scheduling

Current GPU programming frameworks like CUDA does not support time-slice or space-share of the GPUs between clients\loads, but runs each workload to completion before moving on to the next. The scheme for selecting which client to service next is not specified, but appears to follow a round robin or fair-share strategy. Also, current workload scheduling techniques in GPUs use simple first-in first-out (FIFO) scheduling, by which the workload is submitted to the GPU in order regardless of its priority or application specific constrains. FIFO with uncoordinated GPU resource scheduling could cause an application with a hard deadline constrains to miss these deadlines, and violate the application Service Level Agreement (SLA). One fact about workload scheduling in GPU clusters is the lack of global information about the total workload of the GPU cluster.

5. Power consumption and thermal emergencies

Power consumption and thermal management have become a critical issue in modern HPC data centers. The main reasons for that is the rising cost of electricity and the cooling infrastructure. The cost of running a data center through the typical three year life cycle of hardware is becoming comparable to the cost of the data center itself [21].

Manufactures and researchers have already started considering an Exaflops system. For such a system the power consumption is one of the major concerns. The most obvious and straightforward way of reducing power is to improve Power Usage Effectiveness of the data center which is the ratio of total facility power to the power of computer equipment. This is mostly about very efficient cooling methods and power supplies [32]. GPU cluster promise to boost the performance per watt. This means that the rate or the amount of computation that can be delivered by using the GPU cluster will be bigger than any conventional computing system for every watt of power consumed. Currently, the cost of GPUs is of the same order as CPUs but the theoretical peak performance of GPUs per Watt is roughly an order of magnitude higher [32]. However, the GPU based clusters are still consuming more power than other accelerator based clusters (i.e. FPGA clusters). FPGA devices offer a significant advantage (4X-12X) in power consumption over GPUs [34]. The computational density per Watt in FPGAs is much higher than in GPUs [33]. This is even true for 32-bit integer and floating-point arithmetic (6X and 2X respectively), for which the raw computational density of GPUs is higher [34]. In summary, while the GPUs provides an affordable hardware that is ready to be used in HPC clusters, it is important to be aware that the power consumed by the GPU servers is significant even when they are idle.

6. Idle GPU power consumption.

The power consumed by the GPU card such as the Tesla series, is significant even when it is in idle state. The researchers in [38] reported the power consumption in idle, active, and loaded server states. Their results show that the idle power measured for single host

server is 230 Watts, without the GPU cards. The idle power of a host server with four-Tesla cards attached to it is measured at about 380 W. However once the cards are activated the power goes up to 570 Watts and stays at that level. In other words, even if the Tesla server is not used, it may be dissipating up to 350 Watt in addition to the power dissipated by the host. When the GPU server is loaded, the power goes up to around 900 Watts and when the job is completed it drops down to around 570 Watt. After the completion of the task execution on the server and all the GPU cards are idle, the Tesla server still consumes about 350 Watts extra.

4.6 Current Solutions to Overcome GPU Cluster Challenges.

The challenges mentioned above threatened the development and the growth of GPU technology. To address these challenges GPU cluster community have started to introduce solutions to overcome these problems. In this section we present some of the current available solutions introduced by the research community and industry.

1. Low GPU resources utilization

GPU utilization limitation depends mainly on both, the application and the programmer; The GPU applications required a high level of parallelism to be inherent to the application to create enough threads to fully utilize the hardware and to enable exploitation of the many cores. Also, for programming of such type of accelerators, the application developer needs to be aware of the CUDA architecture and the memory hierarchy, which is a complex issue to some extent. Unfortunately, many of developer fail to optimize their applications to have fully utilization of the GPU, the developer are

facing a challenge in low level programming in CUDA. To overcome such issue, researchers in [37] introduce the aspect programming paradigm to the GPUs to reduce programming complexity and overhead induced by current SDK like CUDA.

2. GPUs and the host device bandwidth degradation

To overcome the bandwidth degradation, the basic fact is that the GPUs have better memory bandwidth performance to the host CPU cores depending on what CPU socket the process is running on. Researchers in [21] introduced and implemented proper affinity mapping, by supplying a file on each node containing the optimal mapping of GPU to CPU cores, in order to set process affinity for the CPU cores “closer” to the GPU being allocated.

3. Resource Allocation and Sharing

CUDA wrapper was introduced by [21], in order to provide a truly shared multi-user environment. CUDA wrapper is a library that works in synch with the batch system which is already available like the Torque Batch system. The wrapper overrides some CUDA device management API calls to ensure that the users access only to the GPUs allocated to them by the batch system. The CUDA wrapper library simplifies the GPU cluster usage by creating a GPU device virtualization framework. The user’s application sees only the virtual GPU devices, and the wrapper is responsible for the mapping between the virtual GPU and the physical GPU each time an application\process is launched.

4. Power consumption and thermal emergencies

To mitigate the power issue in GPU cluster, researchers introduce a hybrid cluster that combines GPU and FPGA accelerators. Such type of hybrid cluster is motivated by ability of FPGA devices to reduce power consumption significantly over GPUs [gc14]. An example of a hybrid cluster is the 16-node cluster combines four NVIDIA GPUs and one Xilinx FPGA accelerators for each of the nodes at NCSA-UIUC [36].

5. Idle GPU power consumption.

The idle power consumption has become a critical issue for the GPU cluster computing. In some cases the power loss share of the idle GPU card could reach up to 60% of the total that a server requires. Therefore either the GPU card needs to be fully utilized all the time by loading it with a fair share of workload, or idle power needs to be much lower by optimizing the SDK drivers. Currently the only alternative is to shut down the host server, switch off the power from the Tesla server through remote management in the PDU (power distribution unit) and power the host back again now without the GPU card [38].

4.6.1 **Problems with the current solutions.**

1. The current GPU cluster solutions have no global view for the cluster to optimize the cluster performance at all levels. Also, there is no comprehensive framework that addresses the GPU cluster issues in a consistent manner.
2. Using legacy software like the Torque batch system could prevent the proper exploitation of the computation power of the GPU as the batch system is not aware of the GPU resources. Therefore, there is a need to modify the batch system,
3. Some solutions require significant changes in the CUDA SDK and some of its basic APIs. On the other hand, some solutions require significant changes in the structure of the application.
4. Using hybrid clusters to reduce power consumption and cooling requirement over GPU only cluster suffer from performance degradation in the cluster, due to the lower FPGA throughput compared to the GPUs. Such approaches also suffer from the complexity of developing a hybrid application and managing the cluster during the run time.
5. All of the available solutions suffer from the lack of the coordination between GPU cluster components, workload characteristics, and the underlying hardware. Global administrator coordination for the purpose of load balancing and performance assurance. Has been ignored by most of these solutions.
6. None of the current solutions show any plans for exploiting GPU Heterogeneity (i.e. GT 80 series, with T10 series clusters) for better resource utilization, and power efficient computing.

7. Some solutions introduce runtime overhead for the CPU side of the code, which will impact the performance especially in the case of 1:1 ratio of CPU to GPU.
8. Switching off the power from the Tesla server through remote management in the PDU is a manual technique that can be automated.
9. Some of the problems might need an optimization from the SDK level, as the available room for optimization from the application point of view is still very narrow.

5 AUTONOMIC POWER & PERFORMANCE MANAGEMENT FOR GPU CLUSTERS

5.1 Introduction

In this chapter we present our approach to implement Self-Configuration of applications. Self-Configuration was adopted to achieve performance optimization and management of computing systems and their applications [58][59]. In this thesis we focus on power and performance management for GPU based clusters. We discuss how we apply the autonomic programming paradigm to design an autonomic GPU system within a high-performance GPU based cluster that jointly maintains its power and performance under varying workload conditions.

The energy consumed by GPU chips constitutes a major part of the total GPU based cluster power consumption [21][38].In our research group, an autonomic memory manager was developed to exploit advanced memory technologies such as Rambus DRAM (RDRAM) and Fully-Buffered DIMM (FBDIMM) to save energy and maintain performance by allocating just the required number of memory ranks to the applications at runtime and transitioning any unused memory ranks to low-power states [51].

However, given that GPU cards are often configured at peak performance, and consequently, they will be active all of the time. This introduces a challenge for the GPU based cluster power management problem. Keeping all the GPU cards active at run time provides less opportunity for energy saving. For example, an experimental study to measure the power consumption of a GPU cluster show a constant power consumption (maximum) regardless of the workload [21][36].

By intelligently managing the idle card modules we will save energy with a minimal or negligible impact on the application performance. This demonstrates an opportunity to maximize the GPU cluster performance-per-watt by dynamically scaling down the GPU

cluster resources to adapt to the application's\workload requirements at runtime. With this objective in mind, we design a Self-Configuration and an autonomic GPU cluster manager that addresses the following research challenges related to GPU power and performance management.

1. How do we exploit an application's or the workload behavior to guide our choice of an appropriate configuration for the GPU cluster? This depends on how the application impacts the power consumed by the GPU card and the application-level performance.
2. How do we design a Self-Configuration GPU cluster that effectively exploits the GPU cluster architecture to maximize its performance-per-watt?
3. What are the cost, run-time complexity and reconfiguration overhead associated with our technique and how they can be reduced to attain a greater return-on-investment?
4. How we can allocate the appropriate number of GPU cards that will reduce over provisioning and power consumption and maintain performance.

5.2 **Motivational Example**

In this Section we explain the problem of power and performance management of GPU cluster in detail with the aid of the following example. Let us consider a GPU server with 4 GPU cards (GPU_1, GPU_2, GPU_3, and GPU_4,) as shown in Figure 4-1 where each card is individually power-managed. The application can use the cards in parallel by creating 4 active threads with 1:1 ratio. Let us consider that the application is running in all the cards at run time, which is the general configuration for a GPU cluster, and then

regardless of the application intensity and workload, all the cards are active and will consume the approximately the maximum power even under low workload scenarios. On important observation that even with a zero workload, the GPU card consumes up to 91 watt of power. Consequently, with a zero work load for all the 4 GPU cards, the power consumption is approximately 367 watts as reported in [21]. Now, let us consider two time instants t_i and t_{i+1} during the application execution such that the application requires n_i cards at t_i and n_{i+1} cards at t_{i+1} to maintain performance and achieve the application SLA. Let us consider the case where all the cards are active. Hence, if $n_{i+1} < n_i$, we can save power by transitioning the extra GPU cards ($n_i - n_{i+1}$) into a low-power state.

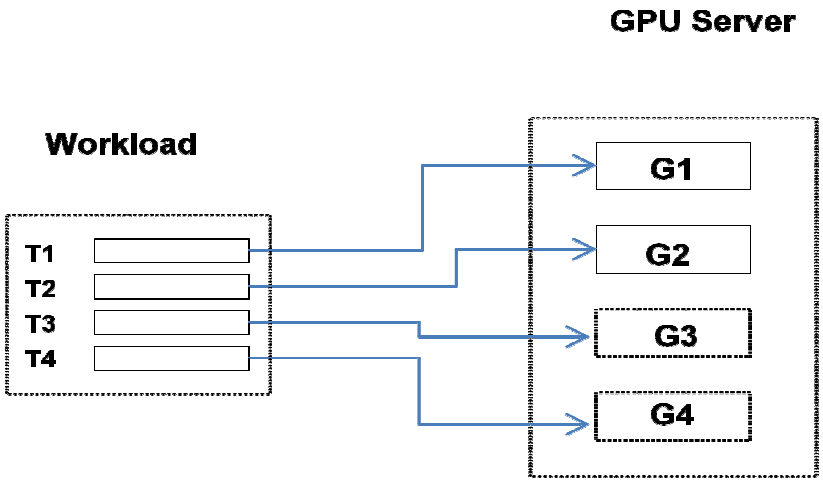


Figure 5-1: GPU server architecture with 4 GPU cards

We propose to create the opportunity for power saving in GPU based cluster by dynamically activating the minimum number of GPU cards that satisfy the application performance. For the example shown in Figure 5-1, we reduce the number of active

cards from 4 cards to 3 or even 2 by migrating the workload assigned to cards No.4 and 3 to cards No. 2 and 1. In this manner we can transition the unutilized cards to a low-power state and thus save energy without impacting the application performance.

5.2.1 Exploiting GPU Cluster Architecture

Reducing the number of active GPU cards decreases the application parallelization and that might impact the application performance. One way of reducing the impact on performance is to distribute the data in a manner that exploits any unique characteristics of the underlying cluster architecture, and the application communication model. For example in Figure 4-1, and in the case of data migration between different GPU cards, the data is moved first to the CPU main memory and then moved to the other GPU card. Hence, if there is a data communication between GPU_1 and GPU_2, it's better to migrate the workload from GPU_2 To GPU_1 or vice verses depending on the card with the high utilization; i.e. move the data from the low utilized card to the higher utilized card. We consider data migration between GPUs even if data migration is not possible in the current GPU architecture, but this will be possible by the new GPU architecture. We want to devise migration strategies that effectively exploit this characteristic. We can have many migration and distribution strategies based on our communication model, for example, Figure 5-2 shows three different redistribution\migration strategies. In strategy I, data is migrated from GPU_2 and GPU_4 to GPU_1 and GPU_3, respectively. This will move GPU_2 and GPU_4 to an idle state and thus it can be moved to a low power, but in

strategy II GPU_3 and GPU_4 will be moved to an idle state. In strategy III on the other hand, data is migrated from GPU_4 to both GPU_1 and GPU_2, and only GPU_4 will be idle.

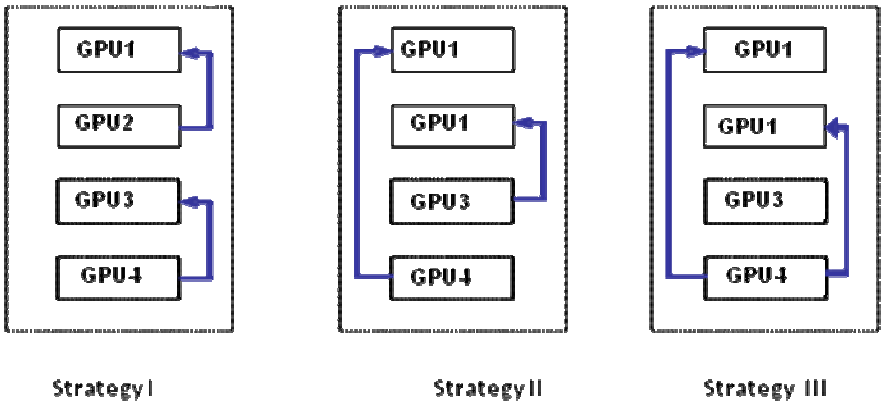


Figure 5-2: Three different redistribution\migration strategies

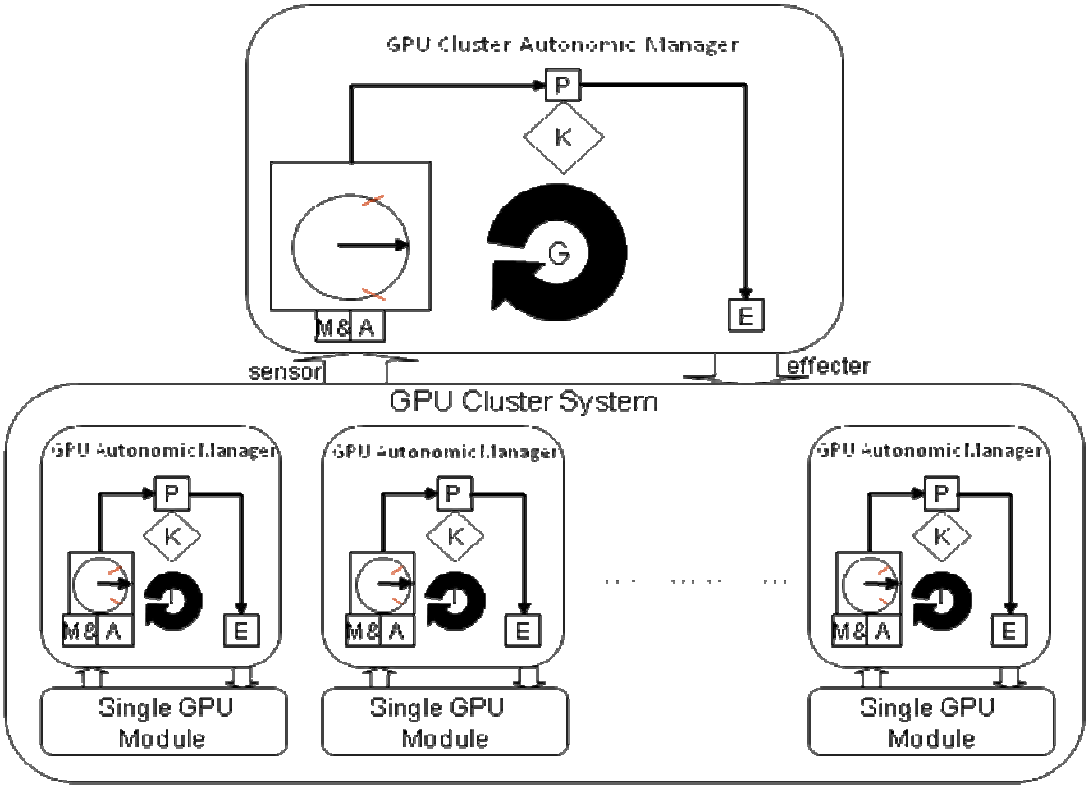
Given that the GPU_1, GPU_2, GPU_3, and GPU_4 have different PCIe connections; i.e. server with 4 C1060 cards, all the strategies will have the same impact on the performance and on data migration time. In the case of the Tesla S1070, there are two PCIe connections, one for GPU_1 and GPU_2, and the other for GPU_3 and GPU_4, strategy I will show a lower impact on both the performance and the migration time compared to strategy II because the data migration occurred within the same PCIe connection.

5.3 **Modeling the GPU Cluster Managed Components**

Figure 5-3 shows an autonomic GPU server system based on the approach discussed previously. It implements hierarchical management architecture. The top-level is the GPU Cluster autonomic manager that manages the entire GPU cluster. The bottom-level is the local autonomic manager, which we call the GPU Autonomic Manager that

manages an individual GPU card. In what follows, we describe each of the components in detail and explain how they work together to maintain the power and performance for the entire GPU cluster.

Figure 5-3: Autonomic GPU Cluster System



In this work we consider the NVIDIA GT200 GPU cards as our managed components, the NVIDIA GT200 GPU become very popular with the introducing of the Tesla 10 series [21] [27], because of its reliability, speed and density features. As shown in Figure 5-4, a Tesla 10 series based on the GT200 chips with 240 stream processors, packages up to 4GB GDDR3, and provides a 102GB\s of bandwidth with memory Interface of 512- bit. We consider the Tesla C1060 card as one managed component in

the cluster manager part, and we consider the streaming multi-processor (MP) as one managed component in the device level Manager. In our work, we assume no any management activity to the GDDR3 memory.

Figure 5-5 shows the model of our GPU cluster system. It is based on the GPU cluster architecture found in Supermicro 4U, 7046GT-TRF GPU server. The GPU cluster consists of four cores - Intel 2.4Ghz e5530 Nehalem Processor , Intel 5530 Chip set up to 6.4GT/s, 12 DIMM slots, up to 4 double-width GPU cards, 8x 3.5in H/S SATA HD trays (6x SATA on MB), 1400W PS. A one TB SATA 7200RPM Enterprise class driver, and 4x Tesla C1060 cards connected with four PCI-e 2 x16. The application can use the 4 GPU cards in parallel and access them in any combination. In our GPU cluster, we consider a GPU card as the smallest unit for power management. This is depicted as a single GPU card in Figure (6-4) and defines the Managed Component at the lower-level of the hierarchy.

A GPU chip supports four power states – offline (loaded drivers off, GDDR3 off, workload off), standby (loaded drivers on, GDDR3 off, workload off), suspend (loaded drivers on, GDDR3 on, workload off), active (loaded drivers on, GDDR3 on, workload on). Figure 5-6 shows the four power states, their power consumption, and potential state transitions. The GPU chip design brings a critical challenge to the HPC community as it consumes power even when there is no workload active, to maintain the data transfer link between the host system and the GPU devices.

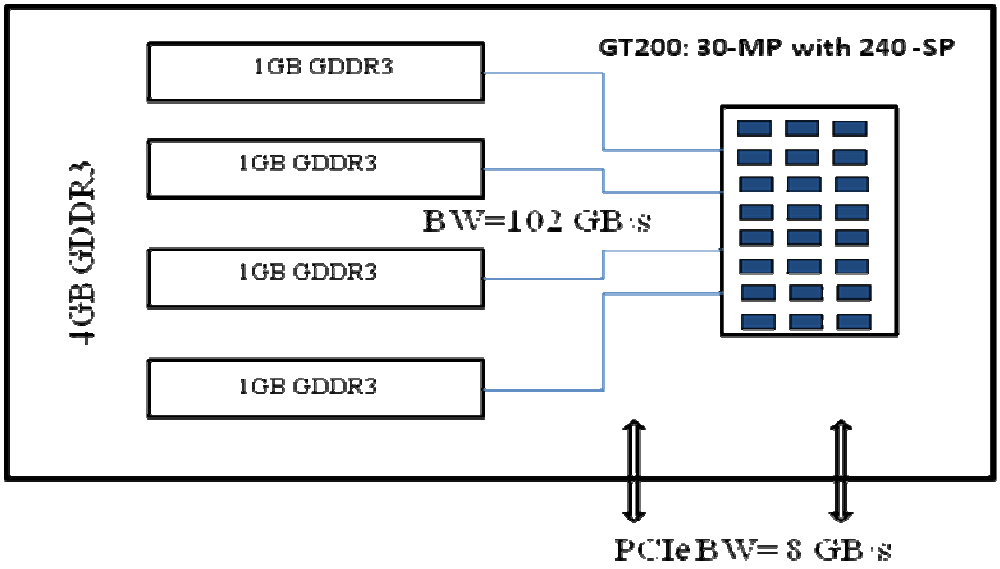


Figure 5-4: Tesla 10 series card based on the GT200 chips

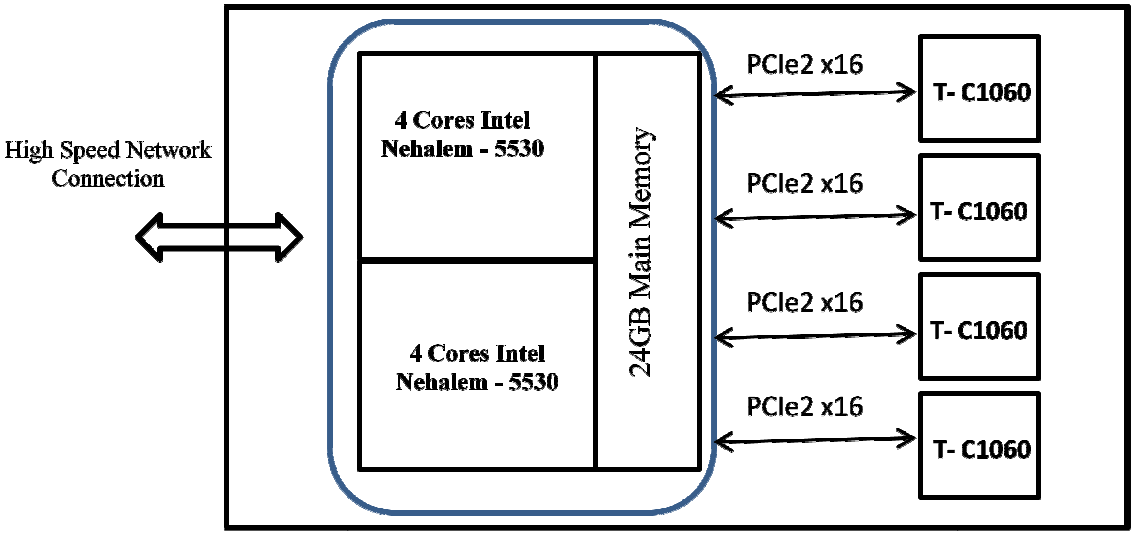


Figure 5-5: GPU Server based on T10 - series

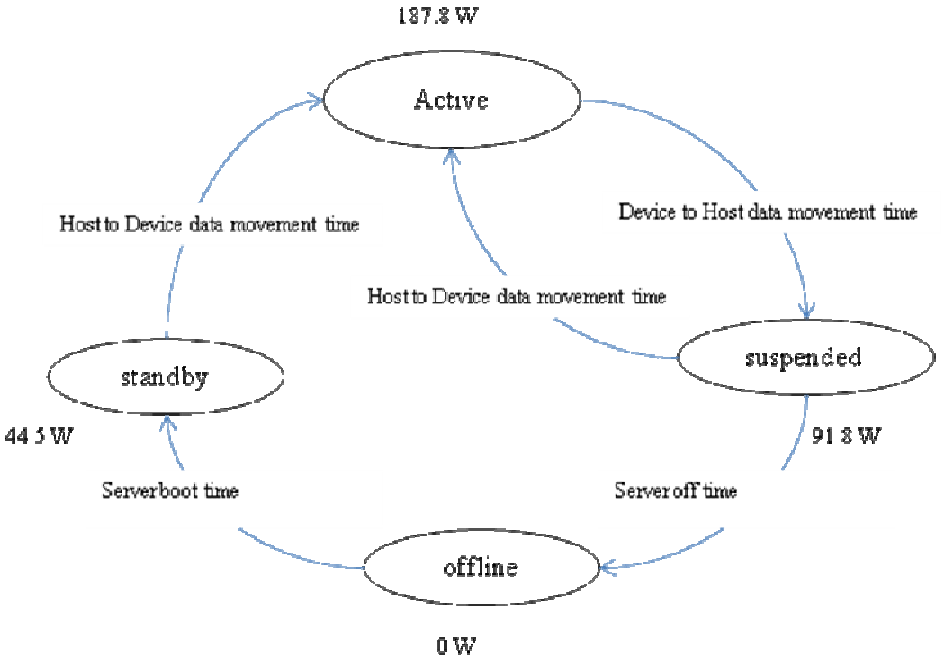


Figure 5-6: GPU chip power states and transitions

We model the performance of the GPU cluster in terms of end-to-end delay d . It is defined as the time from the instant a workload, i.e. CUDA kernel, sent to the GPUs to the time when the kernel processing is done in the GPUs and it consists of two delays – the data movement delay τ_m and the processing delay τ_p. τ_m is the time needed to move the data from the main host memory to the GPU card for processing or for data migration. τ_p is the time required to complete the kernel processing on the GPU card. This delay can be due to contention in the PCIe connection. Each state is defined by the number and the physical location of the GPU card in the 'active' power state. The remaining GPU cards can be in any of the low-power states – suspend standby and offline. A transition takes the system from one state to another. Each state s_k is defined

by a fixed base power consumption p_k and a variable end-to-end delay d_k. The state is defined as follows.

$$s_k = \{r_i \mid i:1 \rightarrow n_k, r_i \in N\}$$

$$p_k = n_k * p_a$$

$$d_k = \tau_{m_k} + \tau_{p_k}$$

where, N : Total GPU cards in the system

n_k : Total 'active' GPU cards in state s_k

p_a : Base power consumption in the active power state (187.8 W from Figure 5-6)

p_k : Total power consumed by the system in state s_k

d_k : End-to-end GPU access delay in state s_k

5.3.1 Modeling the GPU Autonomic Manager

The GPU autonomic manager uses the Multi-Processor Allocation (MPAlloc) algorithm to determine the appropriate number of MPs (N_{MP}) at runtime that can reduce power consumption, over-provisioning, and maintain performance simultaneously. It uses the memory BandWidth Utilization (BWU) metric [66] to predict the Multi-Processor requirements of the application. As shown in the MPAlloc algorithm below, the output of the algorithm is the optimal number of MPs to maintain the application performance and to reduce the number of allocated MPs and move the other MPs to a low power state to reduce the power consumption. The MPAlloc algorithm uses the BandWidth Utilization (BWU) metric to determine the optimal MPs number, and BWU concept emerges from

the fact that data-parallel applications where MPs operate on separate data require negligible data-synchronization. For such applications, contention for shared resources that do not scale with the number of MPs is more likely to limit performance. One such resource is the memory bandwidth. For applications with negligible data sharing, the demand for memory bandwidth increases linearly with the number of MPs. However, the memory bandwidth is not expected to increase as the number of cores because it is limited by the number of I/O pins [66]. Therefore, these applications become memory bandwidth limited. Once the memory bus saturates, no further performance improvement can be achieved by increasing the number of MPs. Thus, the performance of memory bus limited system is governed solely by the bus bandwidth and not by the number of MPs. However, having more MPs than required to saturate the bus only consume more power without contributing to performance. Where BWU is the bandwidth utilization, MP_{max} is the max number of available MP per card, BW_{max} is the maximum memory bandwidth per card, and the BW_per_MPi is the bandwidth utilization per card. In the case of the Tesla C1060, MP_{max}=30, BW_{max}=102 GB/sec, and the BW_per_MPi is application dependent. In the MPAlloc, as we increase the number of MPs, the corresponding BWU increases (lines 2-3). We keep checking when the BWU reaches 100 % or above in order to stop the algorithm, and the value of m-1 will be the optimum number of MPs. However, if the BWU still below 100%, then we can use MP_{max} as the optimum number of MPs (lines 10-11). The output of the algorithm is the optimal number of MPs (line 13) to maintain the application performance and to reduce the number of allocated MPs and move the other MPs to a low power state to reduce the power consumption

MPAlloc: Getting the optimal number of *Multi-Processor* (N_{MP}) with memory bandwidth utilization consideration

1. $N_{MP} = 0,\ BWU=0,\ n=MP_{max}$
2. For $(m=0, m <= n, m++)$
3. $BWU = \sum_{I=0}^{m} BW\_per\_MPi / BW\_max$
4. If$(BWU(m) > 100)$
5. *Break*
6. End If
7. End For
8. $N_{MP}=m-1$
10. if $(BWU<=100)$
11. $N_{MP}= MP_{max}$
12. End If
13. Return N_{MP}

5.3.2 Modeling the GPU Cluster Autonomic Manager

In this Section we describe how the cluster manager works with the individual GPU managers to collectively manage the power and performance of the GPU cluster. With every incoming workload, i.e. kernel, the cluster manager detects and predicts the number and the type (in the case of heterogonous server) GPU cards requirements of the application. It then identifies an optimal configuration/state (number, type and physical location of GPU cards) that meets the application processing requirements and at the

same time maintains the end-to-end access delay for the given workload. It uses this physical configuration to determine the degree of available GPU cards that would maximize the performance-per-watt for the given workload.

The server Manager uses the Application\Workload characteristics (Appflow) to determine when such a state transition/reconfiguration is required for the GPU cluster in order to maintain its power and performance. It determines the target state to transition to by using artificial intelligent techniques; in our case we used the Case Base Reasoning (CBR). The CBR database (CBR-DB) will store the optimal configurations for each workload scenario by solving a performance-per-watt optimization problem during the training phase. It readjusts the number of GPU cards available for the application by dynamically distributing the load to a selected GPU card(s) based on the distribution strategies, and how to transition the remaining GPU cards to appropriate low-power states.

In what follows we first discuss how the cluster manager predicts the application\Workload characteristics, and then discuss our formulation of the performance-per-watt optimization problem and our formulation of the CBR-DB. The dynamic cluster manager implements the workload distribution task for adaptive and dynamic server configuration that will result in moving the GPU cards to the appropriate low-power state based on the workload requirements. The individual GPU autonomic managers are responsible for allocating the optimal number of MPs based on our MPAlloc algorithm, moving unused MPs to a low power state. The Cluster manager can

be coupled with any suitable fine-grain power management techniques at the lower-level implemented in the GPU autonomic manager like our MPAlloc algorithm.

5.3.2.1 GPU Workload Modeling

For modeling the GPU workload before execution, we consider using the Ocelot framework for building an Appflow for data-parallel applications running in GPUs. The Ocelot framework [61] is an emulation and compilation infrastructure that implements the CUDA Runtime API. Ocelot is uniquely leveraged to gather instruction and memory traces from emulated kernels in unmodified CUDA applications, analyze control and data dependencies, and execute the kernel efficiently on CUDA-capable GPUs. The Ocelot framework enabling a detailed and comparative workload characterization using parallel thread execution ISA that will help us building GPU server Appflow.

NVIDIA Parallel Thread eXecution (PTX) [62] is a virtual instruction set architecture with explicit data-parallel execution semantics that are well-suited to NVIDIA's GPUs. PTX is composed of a set of RISC-like instructions for explicitly-typed arithmetic computations, loads and stores to a set of address spaces, instructions for parallelism and synchronization, and built-in variables. Functions implemented in PTX, known as kernels are intended to be executed by a large grid of threads arranged hierarchically into a grid of cooperative thread arrays (CTAs). CTAs in this grid may be executed concurrently or serially with an unspecified mapping to physical processors. Threads within a CTA are assumed to be executing on a single processor and may synchronize at programmer-inserted barriers within the kernel. Data may be exchanged between the threads of a CTA

by reading and writing to a shared scratchpad memory or by loading and storing to locations in global off chip memory; in either case, a barrier synchronization must be used to force all threads to complete their store operations before loads to the same location may be issued or the outcome of the resulting race condition is undefined.

GPU Appflow Metrics

Ocelot's PTX emulator instrumented the kernel (.ptx) with a set of user-supplied event handlers to generate detailed Kernel information of instructions and memory references, registers counts, etc. In our work we used the following metrics.

Memory Extent.

This metric uses pointer analysis to compute the working set of kernels as the number and layout of all reachable pages in all memory spaces (i.e. global). It represents the total amount of memory that is accessible to a kernel immediately before it is executed.

Registers per Thread.

This metric expresses the average number of registers allocated per thread. The large register files of GPUs will be partitioned into threads at runtime according to the number of threads per CTA. Larger numbers of threads increases the ability to hide latencies but reduces the number of registers available per thread.

Shared-Memory per Thread.

This metric represents the size of shared memory allocated per thread. The total shared memory in MP will be partitioned between threads at runtime according to the number of threads per CTA.

Instruction Count.

This metric includes the integer and the float instructions for each kernel before execution. For the integer instructions we distinguish between the number of integer arithmetic instructions, and the number of integer logical and compare instructions. Also we consider the single-precision arithmetic instructions for each kernel.

5.3.2.2 GPU Workload Data collection.

Figure 5-7 shows our experimental setup for training, data collection, and testing. In our work, we used two different CUDA applications for the purpose of training and data collection, i.e. workload characteristic, to build our Appflow. The first CUDA application is the A 3D heart simulation developed by the ACL group at the University of Arizona [42], the second CUDA application is the Nbody simulation from CUDA SDK [43]. For the data collection we followed the following steps.

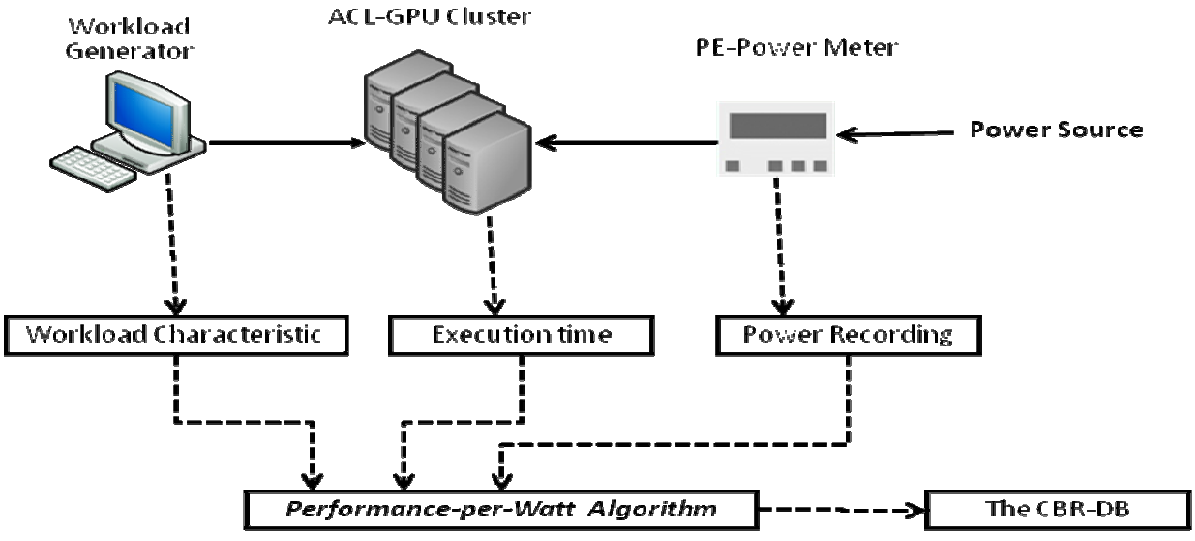


Figure 5-7: GPU Cluster testbed configuration

1. Using the workload generator and for each application, we generate more than a

hundred workload scenarios by changing the size of the heart tissue, i.e. changing the number of heart cells for our heart simulation, or by changing the number of simulated particles for the Nbody simulation. Each workload scenario will be represented by the Appflow metrics mentioned above.

2. For each workload scenario generated in step 1, we ran the workload scenario using four different GPUs configuration, and each GPU configuration differs from the other configurations by the number of GPU cards to be used to run the workload scenario. The GPU configuration can be extended to take into consideration the physical location of the card, and the card type, but in our case we have only one type of cards and all are symmetric regarding the physical location.
3. For each GPU configuration, we recorded the execution time, i.e. the delay, and the power consumption using the Brand Electronics ONE Meter [63].
4. The workload characteristic, the execution time and the power consumption data collected in the previous steps, will be the input to our Performance-Per-Watt algorithm discussed latter. The output of the Performance-Per-Watt algorithm will be stored in the Case Base Reasoning Database (CBR-DB).

5.3.2.3 Formulating the Optimization Problem for Performance-per-Watt Management

We formulate the GPU cluster configuration algorithm as a performance-per-watt optimization (maximization) problem with respect to the GPU card state space discussed earlier.

$$\text{Maximize} \quad ppw_t = \frac{1}{d_k * P_k}$$

s.t.

$$1.\ C_k \geq C_t$$

$$2.\ d_{\min} \leq d_k \leq d_{\max}$$

$$3.\ \sum_{k=1}^{N} x_{jk} = 1$$

$$4.\ x_{jk} = 0 \mid 1$$

Where, PPw_t is the performance-per-watt during interval, i.e. kernel t, d_k is the delay and p_k is the power consumed in state s_k.

. N: total number of system states, d_k is the summation of power consumed for data transfer and processing, $[d_{\min}, d_{\max}]$:the threshold delay range, x_{jk} : decision variable to chose from state s_j to s_k, C_k represent the state K computing capacity, and Ct_i represent the required computing capacity for the current state (kernel).

Constraint 1 of the optimization equation (4) states that the target state should have enough computing capacity to handle the Current kernel computing requirement. Constraint 2 states that in the target state, the delay should stay within the threshold range. Constraint 3 states that the optimization problem leads to only one decision. The decision variable corresponding to that is 1 and the rest are 0. Constraint 4 states that the decision variable is a 0-1 integer.

5.3.2.4 GPU Kernel Appflow

The cluster manager monitors the appropriate kernel .ptx file before kernel execution and predicted the CUDA application computing requirements using the Appflow metrics mentioned above. In addition, the cluster manager also monitors the end-to-end delay, d and the power consumed by the GPU cards in the current state, P. As shown in Figure 5-8, they constitute the GPU server operating point in a 3-dimensional space. The operating point changes in response to the incoming workload. The cluster manager triggers a state transition or reconfiguration whenever the operating point goes outside the safe operating zone. This is depicted by the decision, d_Z, in Figure 5-8. The decision is computed using the optimization approach discussed earlier.

Note that the safe operating zone is defined by threshold bounds along the delay axis given by [d_{min}, d_{max}], threshold bounds along the power axis given by [P_{min}, P_{max}] and the bounds along the application computing capacity axis given by [C_{min}, C_{max}]. For our work, the threshold delay bounds are predetermined and kept set at that value, the threshold Power bounds is defined by [$P_{min} = 0$, $P_{max} = N*p_a$], where P_{max} represents the

maximum power that is consumed by the GPU cluster when it is configured at its maximum capacity. The third threshold bound defined by the CUDA application computing capacity requirements however varies. Let us understand this with the aid of an example.

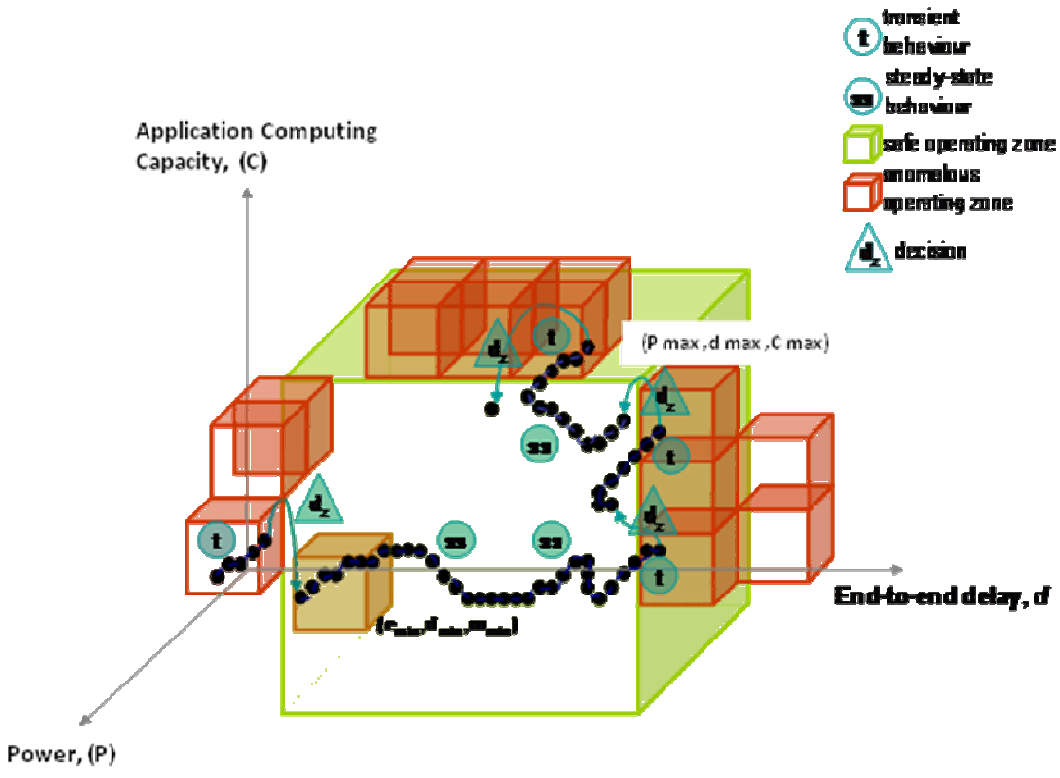


Figure 5-8: Trajectory of the GPU cluster operating point

5.3.2.5 PERFORMWATT: GPU Cluster Performance-per-Watt Management Algorithm

The **PERFORMWATT** algorithm represents the performance-per-watt algorithm in Figure 5-7, the output of the algorithm will be stored in the CBR database. The algorithm

below represents a general form for Performance-per-Watt Management that can be used for training (steps 6 -16), or for online management (steps 1 -16). In our work we used this algorithm for training purposes and for building the CBR database. The main steps of the algorithm, which is executed before the kernel execution, is shown below. Given the GPU server is in a current state s_c, the cluster manager initializes the maximum performance-per-watt $ppw_{\max}$. The cluster manager monitors the system operating point to determine if a reconfiguration may be required. If the application's computing requirement does not change and the measured delay in the current state d_{s_c} lies within the threshold delay range, the cluster manager maintains the same state (lines 2-4). Otherwise it looks for a state that gives the maximum performance-per-watt and also satisfies all the constraints. The Cluster manager visits each state in and computes the PERFORMWATT in that target state, if it is greater than the maximum performance-per-watt $ppw_{\max}$, the value of $ppw_{\max}$ is updated with the new value and the target state is set to the feasible state (line 10-11). This is repeated until all the states in the state space have been evaluated. Finally the target state that has the maximum performance-per-watt is returned (line 16).

The complexity of our algorithm can increase exponentially with the size of the state space. However, the state space is small for GPU cluster sizes generally available today.

PERFORMWATT: Getting the optimal GPU server configuration for maximum *performance-per-watt* with computing capacity consideration.

1. Current state = S_c, PPw_{max}=0,
2. If (d_{sc} => d_{min})&& (d_{sc} <= d_{max})&& (C_{Sc} == C_{kernel})
3. *target state*=S_C
4. End If
5. ELSE
6. For (i=0,i <= N, i++)
7. If (d_{sc} => d_{min})&& (d_{sc} <= d_{max})&& (C_{Si} == C_{kernel})
8. $PPw_{si} = 1/d_{si}*P_{si}$
9. If($PPw_{max} < PPw_{si}$)
10. $PPw_{max} = PPw_{si}$
11. *target state*= *Si*
12. End If
13. End If
14. End For
15. End Else
16. Return *target state*

5.3.2.6 Power and Performance Configuration Manager (PPCM).

In this section, we present the Power and Performance Management (PPM). Figure 5-9 shows the online power and performance Manager at run time, the PPM.

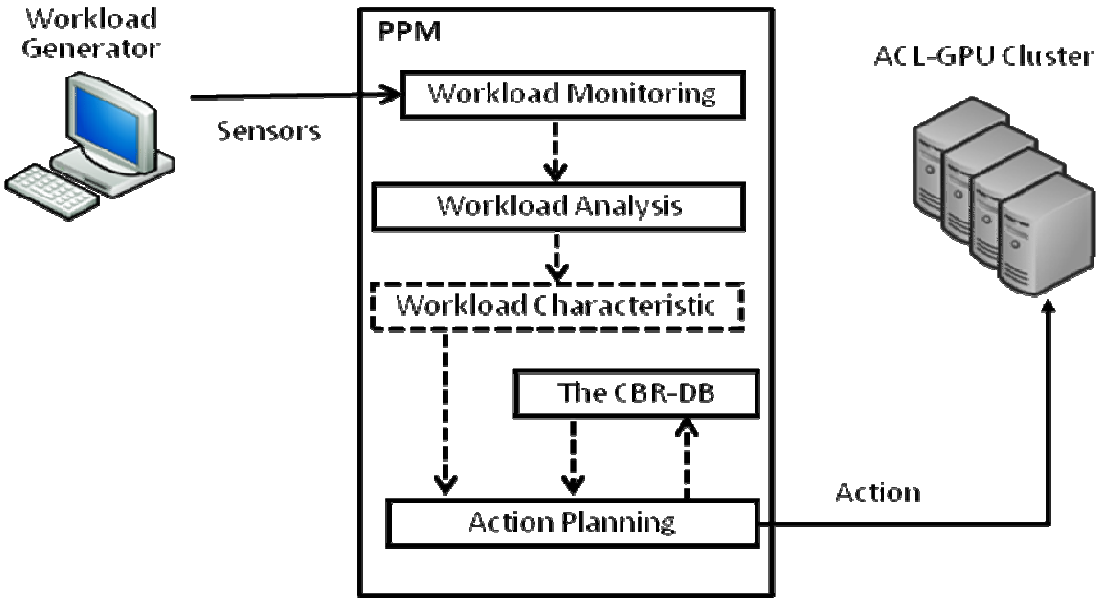


Figure 5-9: Power and Performance Manager (PPM).

For the purpose of evaluating our PPM system, we used three different CUDA applications. The first application is the A 3D heart simulation developed by the ACL group at the University of Arizona, the same application was used for data collection [42], the second application is the N-Body simulation. The third application is a fluid simulation available at CUDA SDK [43]. For the PPM evaluation we followed the following steps.

1. Using the workload generator and for each of the benchmark applications, we generated randomly different workload scenarios by changing the size of the problem.
2. PPM will monitor the application before kernel execution; each workload scenario (kernel) will generate a .ptx file that will be analyzed using the Ocelot framework to generate the Appflow metrics or the Workload characteristic.

3. The Workload characteristic generated in step 2 and the CBR-DB will be used by the Action Planning component in PPM, to find the optimal GPU cluster configuration, i.e. number of GPU cards, to run the current workload. The Action Planning component will update the CBR-DB with the new case based on the degree of similarity used.
4. The optimal configuration found in step 3 will be deployed in the GPU cluster to run the current workload.

5.4 Experimental Evaluation

5.4.1 GPU autonomic manager and MPAlloc aalgorithm rresults

To measure the power consumption of each MP, we controlled the number of used MPs by allowing only one CTA with 512 threads to be executed in each MP, in this way we were able to control the number of used MPs, by controlling the number of CTAs executing in the card with (1:1) mapping. Idle MPs do not consume as much power as the active MPs, as the idle MP do not change values in circuits as often as used MP [97]. Hence, there will be significant differences in the total power consumption depending on the number of allocated MPs in a GPU. Figure 5-10 shows an increase in power consumption as we increase the number of allocated MPs. the power consumption started from 91 watt (static power for C1060) and increased up to 170 watt, which is below the maximum power consumption for the C1060 which is about 187 watts.

Figure 5-11 shows the memory bandwidth utilization of two different applications when the number of MPs increases from 1 to 30. For the Matrix Multiplication, until 24 MPs, the bandwidth utilization increases linearly and then stays at 100% for more MPs. Therefore, having more than 24 MPs does not improve the performance. However, more MPs increase GPU power consumption which increases linearly with the number of MPs as shown in Figure 5-10. On the other hand, and for the 3D heart simulation, the memory bandwidth utilization increases linearly but it's still below 100%, and the application can use all the 30 MPs.

In Figure 5-12, with the MPs increase from 1 to 30, the execution time reduces until 24 MPs and then becomes constant for the Matrix Multiplication. This is due to the saturated memory bandwidth after 24 MPs. For the 3D heart simulation, the memory bandwidth utilization increases linearly but it's still below 100%, and the application can use all the 30 MPs, so that the application can gain more and more performance with the increasing number of MPs.

5.4.2 GPU Workload Modeling Results

The Ocelot's PTX kernel analyzer gave detailed Kernel characteristics for instructions, memory references, registers counts, etc. we analyzed more than 832 different kernels and we collected data for all the metrics discussed before. Figure 5-13 shows a memory extent values for different 25 random kernels, and Figure 5-14 shows the values of registers per thread for different 25 random kernels.

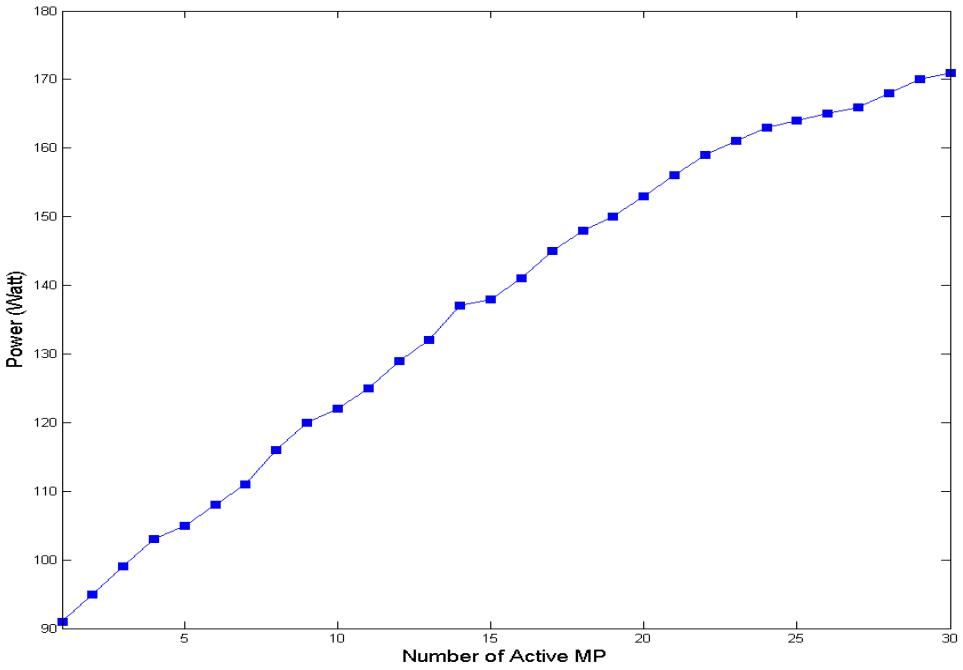


Figure 5-10: Power Consumption for Different Number of MPs

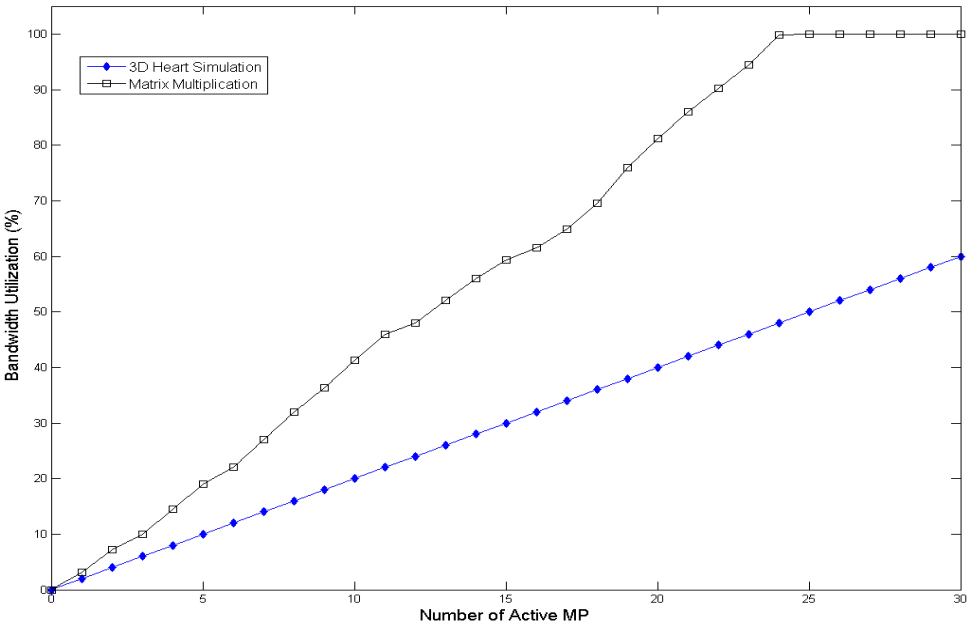


Figure 5-11: Bandwidth Utilization percentage for Different Number of MPs

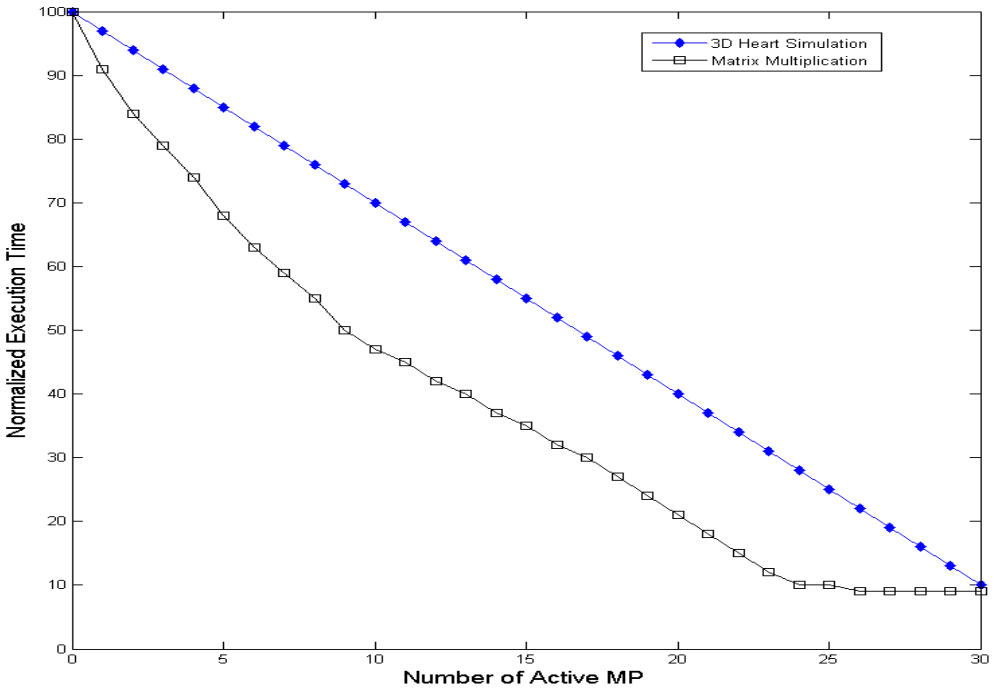


Figure 5-12: Normalized Execution Time for Different Number of MPs

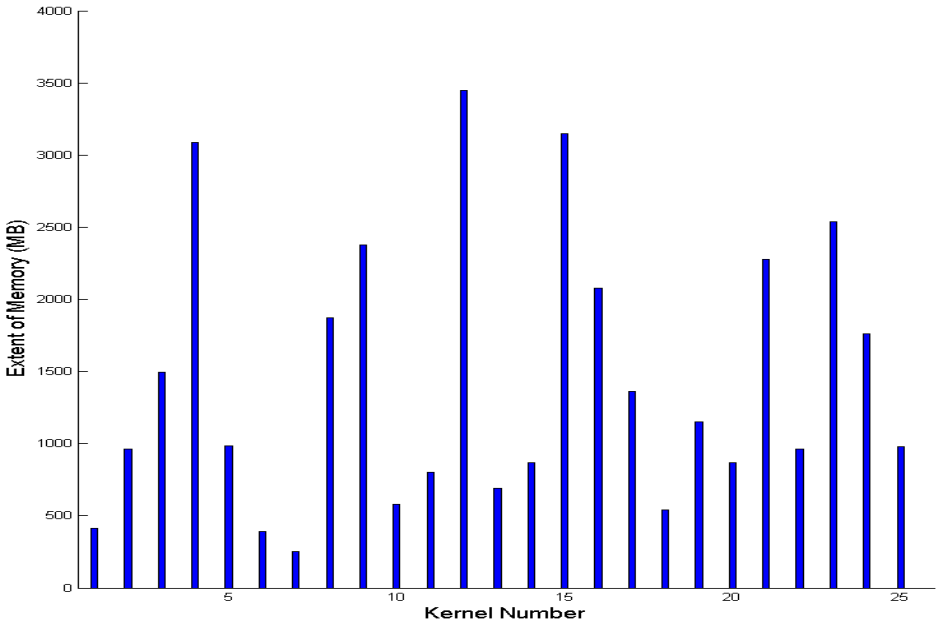


Figure 5-13: Extent of Memory values for 25 different kernels

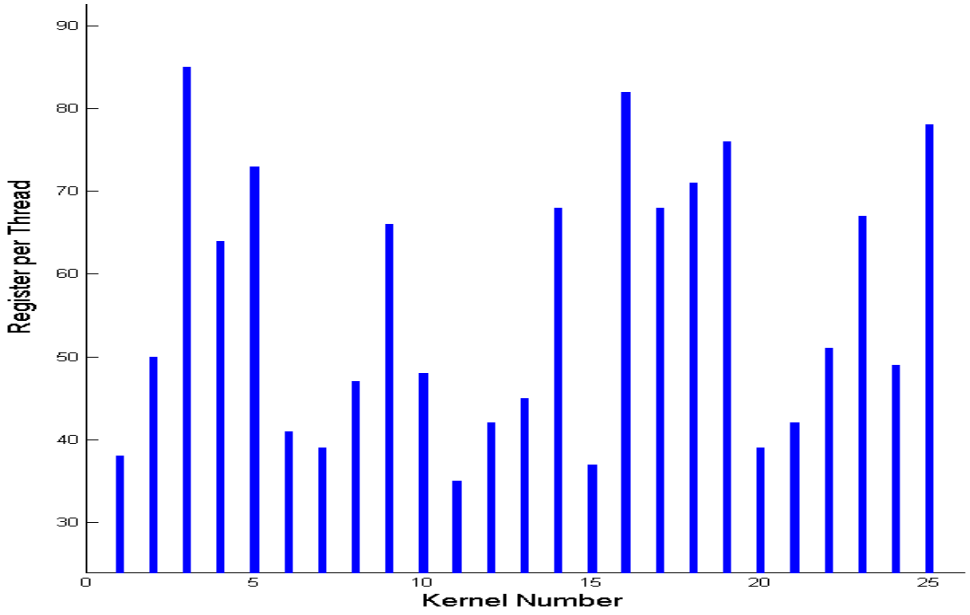


Figure 5-14: Register per Thread values for 25 different kernels

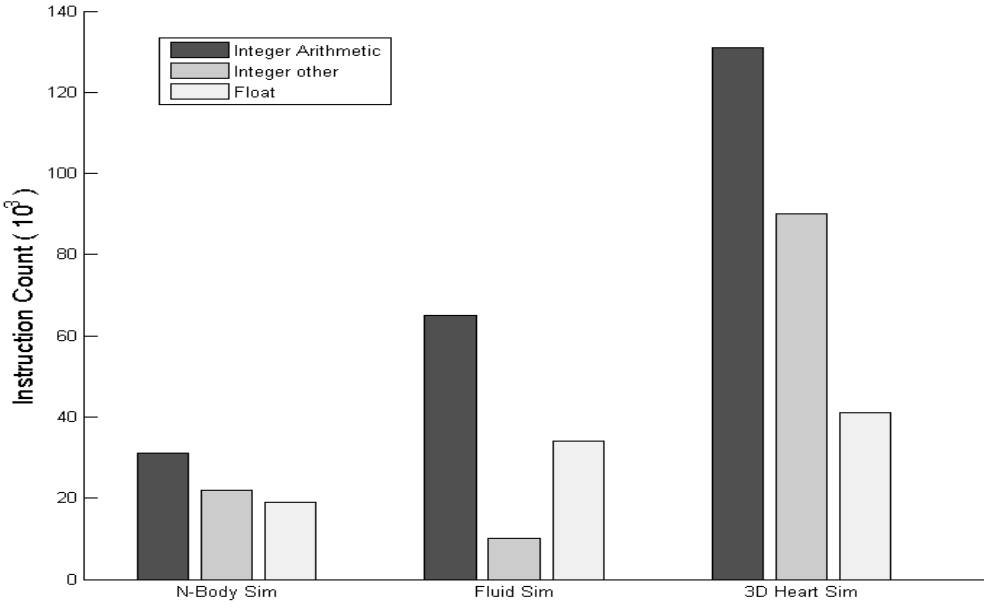


Figure 5-15: Instruction Counts for different Applications

It’s clear from Figures 6-13, 6-14 and 6-15 that each kernel is distinct from other kernels and we can easily use the workload metrics to identify any CUDA workload. An important point in Figure 5-14, that the minimum register per thread value is 32 and this value is related to the C1060 architecture, with different architecture like the C870, the minimum value is 10 register per thread.

5.4.3 Cluster manager and PERFORMWATT aalgorithm rresults

Optimal Cluster Configuration

Figure 5-16 plots the optimal number of cards that the GPU cluster will be configured during the run time for random 100 workload scenarios**.** Our algorithm adapts well to the nature GPU traffic. Figure 5-16 shows that about 65% of the workload scenarios will require 2 GPUs card at most, this fact will enable our PERFORMWATT algorithm to move the other 2 cards to a low power state. Also, only 15% of the scenarios required full cluster capacity with maximum power consumptions.

As recommended in CUDA programmer guide, the optimal number of threads per CTA is 192 threads, and this will results in about 85 registers per thread. From our experiments, we found that 85 registers per thread can be more than what actually needed by the application. As shown in Figure 5-17. We observed that our algorithm always determined the cluster configuration that gave the maximum performance-per-watt among all possible cluster configurations, with higher number of threads per CTA, this will reduce the execution time of the application as we are executing more threads at each iteration. Our results show that the 192 threads per CTA are very conservative and can impact the performance and waste the power.

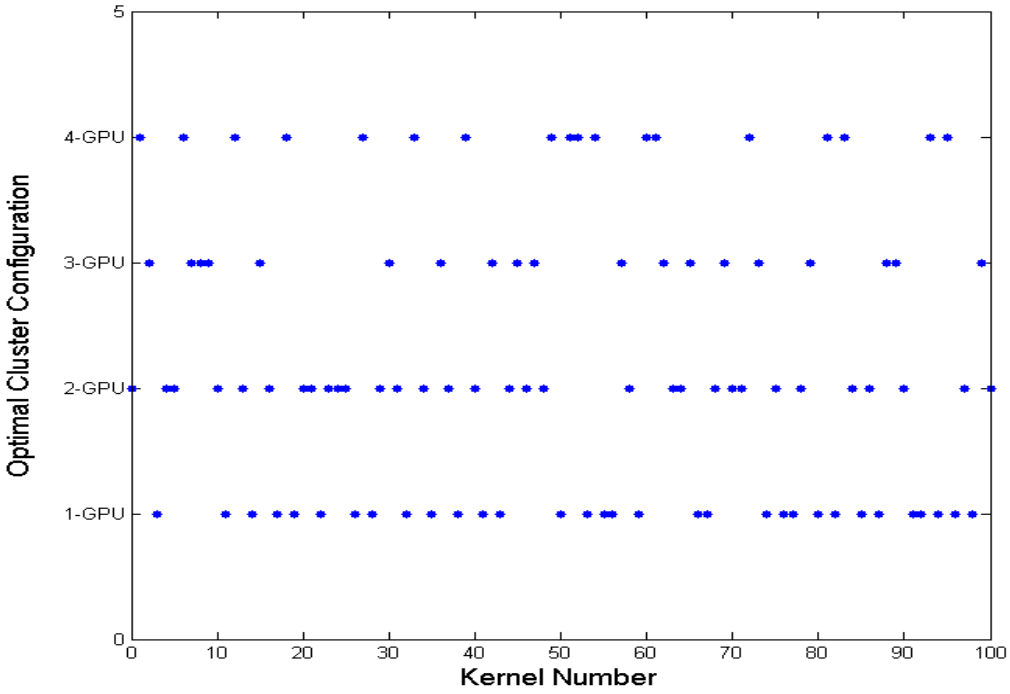


Figure 5-16: Optimal State Selection with 100 Workload Scenarios

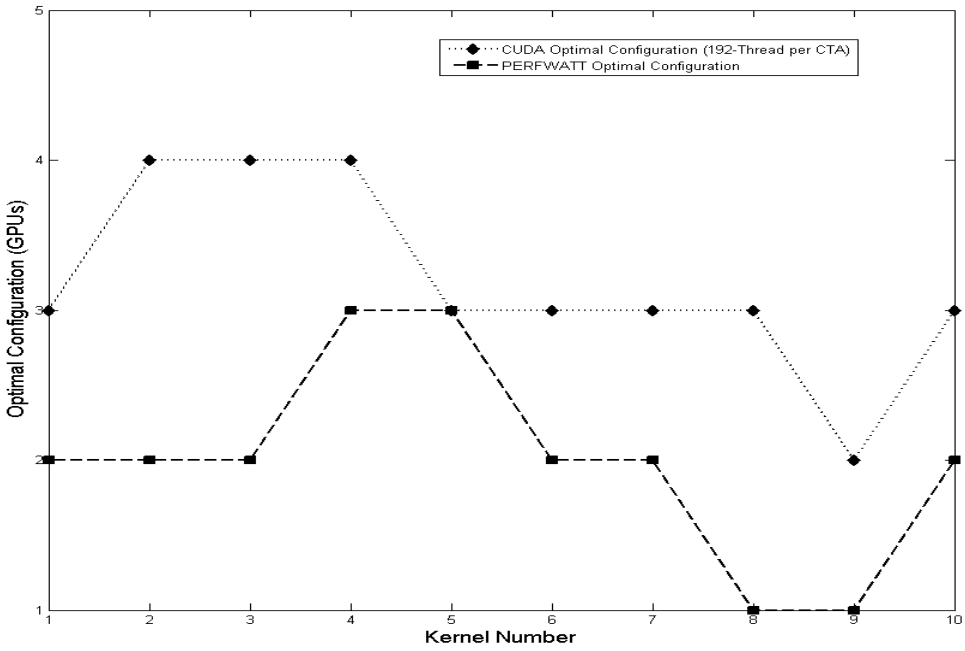


Figure 5-17: Optimal Cluster Configuration with Different CTA size

PERFORMWATT Algorithm Results

The output of our PERFORMWATT algorithm is the optimal number of GPU cards that maintain the application performance and reduce power consumption by moving the other GPUs to a low power state. Figure 5-18 shows for different workload scenarios PERFORMWATT values for different configurations. Scenario number one will perform better using only one GPU, as using two GPUs will increase the power consumption without any performance gain. For scenario number one, using full cluster capacity, i.e. 4 GPUs, will give only about 35% performance per watt and consequently that will lead to a huge power consumption that will reach up to 400 watt when compared to only 100 watt using one GPU. The same explanation is valid for the other workload scenarios.

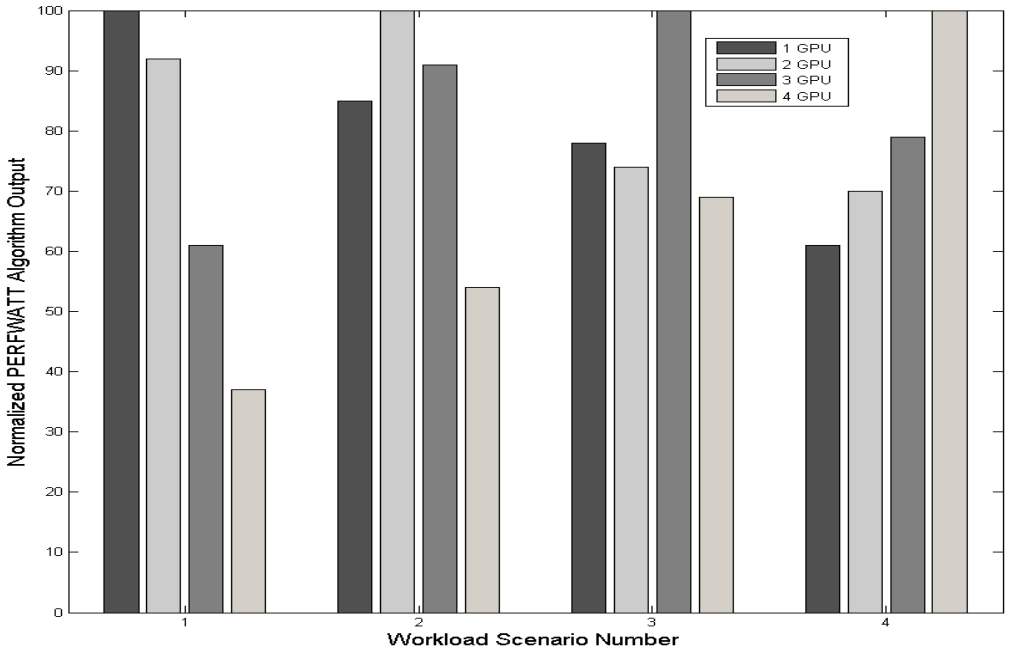


Figure 5-18: Normalized **PERFORMWATT** with Different Cluster Configuration

Features weighting and the CBR-Data Base sensitivity to the Degree of Similarity

Feature weighting or selection is a crucial method to recognize a significant subset of features from a data set. Removing irrelevant or redundant features can improve the retrieval process performance [57]. In our work, we used the Kononenko algorithm to calculate the features weighting values, in order to calculate the degree of similarity based on the nearest neighbor algorithm [53] presented in section 2.5, more details about Kononenko algorithm can be found in [54]. Table 5-1 shows the features weighting values for the workload metrics used in our work. The high degree of importance of register per thread, extent memory, and shared memory per thread features made them the most important metrics when we calculate the degree of similarity. The integer and the float instructions counts are less than 10% of the total weight, so they have no significant role in determining the degree of similarity and thus they can be ignored to decrease the computation time when we calculate the similarity degree.

Table 5-1: Feature Weighting Values

Feature Name	Weight
Register per Thread	**0.38**
Extent Memory	**0.28**
shared Memory Per Thread	**0.25**
Integer Instruction Count	**0.06**
Float Instruction Count	**0.03**

The Action Planning component in our PPM system uses the Workload characteristic and the CBR-Date Base to find the optimal GPU cluster configuration, i.e. number of GPU cards, to run the current workload. For each new case, the Action Planning component will update the CBR-DB with the new case based on the degree of similarity used. A lower degree of similarity will result in a huge number of retrieved cases, and these retrieved cases will contain many false cases, i.e. cases with the wrong configuration. If the retrieval process contains false cases the CBR-DB needs to be updated with the new case, and this will increase the size of the CBR-DB. A larger CBR-DB size will incur more overhead to search for similarity. As shown in Figure 5-18, a similarity degree of 50% will produce about 40% of false cases. A 95% of similarity degree will be the ideal degree in order to reduce significantly the false cases.

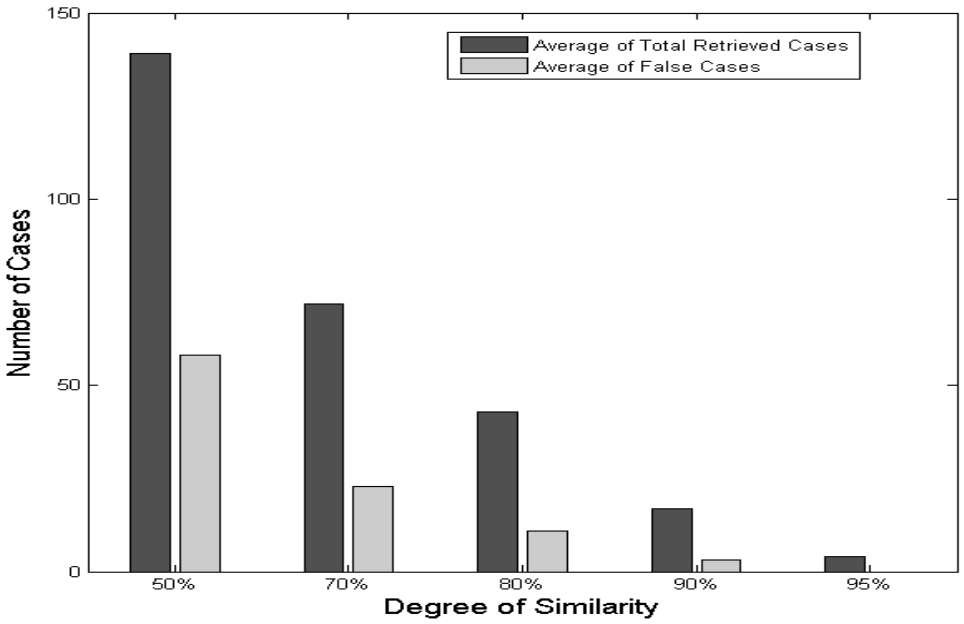


Figure 5-19: Degree of Similarity Effect on CBR-Data Base Growth

PERFORMWATT **Algorithm Comparison with Existing Techniques**

We compared the performance-per-watt obtained by running 4 different algorithms on the GPU cluster with 25 different kernels. Figure 5-20 plots the average performance-per-watt (PPW) given by each algorithm for the 25 kernels. The first algorithm ran the kernels using the low power configuration, i.e. 1 GPU, the second algorithm ran the kernels using full cluster capacity, i.e. using 4 GPUs. Algorithm 3 ran the kernels using the best performance configuration, i.e. the lowest execution time. Algorithm 4 in Figure 5-20 is our **PERFORMWATT** algorithm. As it can be seen from Figure 5-20, **PERFORMWATT** gives the maximum performance-per-watt when compared to all other algorithms. Figure 5-20 also plots the improvement in performance-per-watt given by **PERFORMWATT** as compared to each other algorithm. It gives a performance-per-watt improvement of 46% over the approach that optimizes only the performance. Also, using only one GPU as in the algorithm that optimizes only power or all the GPUs as in full capacity algorithm will result in 63% and 49% of PERFORMWATT waste, respectively when compared to **PERFORMWATT** algorithm.

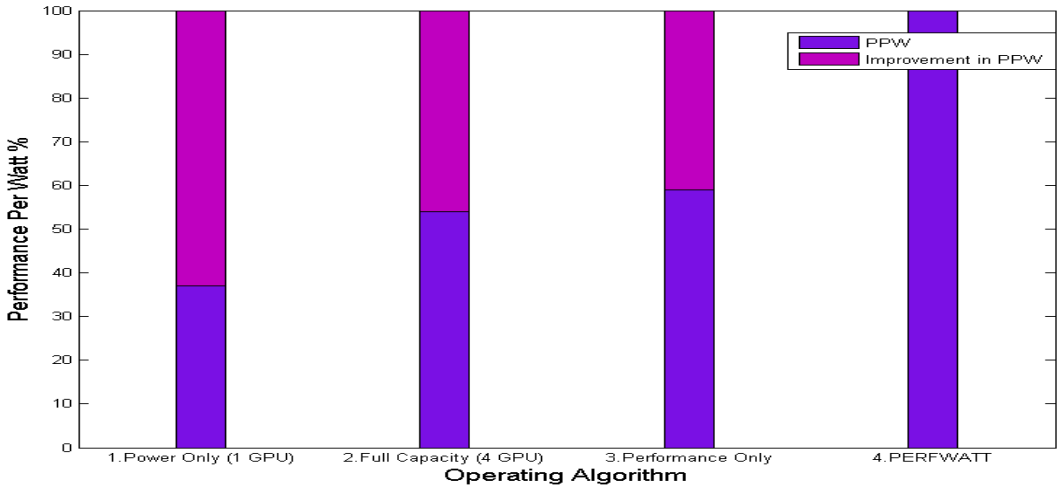


Figure 5-20: Comparison of performance-per-watt Algorithms

GPU autonomic manager and Cluster Manager Combined Results

Our hierarchal management framework for GPU based cluster yields power saving in the two levels of the hierarchy, the Cluster manager and the GPU autonomic manager. On the lower level of our framework, the individual GPU autonomic managers are responsible for allocating the optimal number of MPs based on our MPAlloc algorithm, moving unused MPs to a low power state. As shown in Figure 5-21, our MPAlloc algorithm yields in average 14.1% power saving and this value increased to 26.6% in average if we ignore the static power factor. On the upper level of our framework, the Cluster manager implements an adaptive and dynamic cluster configuration that will result in moving the GPU cards to the appropriate low-power state based on the workload requirements. The **PERFORMWATT** algorithm yields a 46.3% in power saving. The high percentage of power saving for **PERFORMWATT**

algorithm compared to the MPAlloc is related to the fact that we move a whole card to a low power state, which will results in about 100 watts in power saving, or 80 watts in the worse case, but in the case of MPAlloc, we only move number of the MPs to a low power state that will result in about 3 watts in the best case as shown in Figure 5-10. An important point here is that the amount of power saving in the device level is an application dependent, so that some of the applications may show a 0% power saving at the device level as shown in Figure 5-11. At the server level, the power saving is a workload intensity dependent. Our framework yields a 53.7% in power saving using the both level management as shown in Figure 5-21.

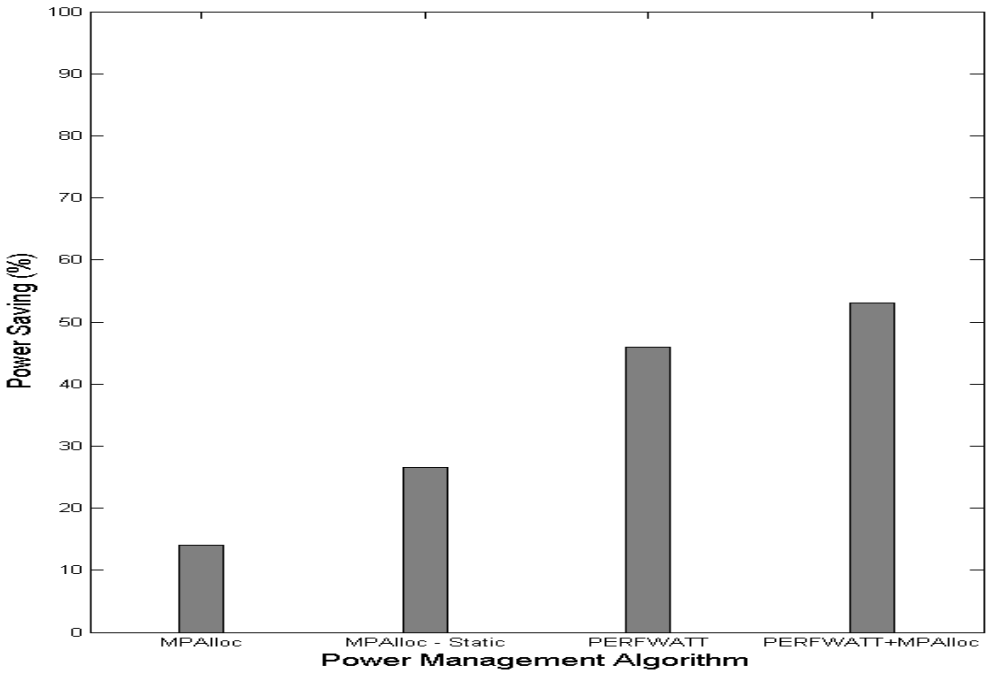


Figure 5-21: Comparison of MPAlloc and PERFORMWATT Algorithms

PERFORMWATT Algorithm Complexity Analysis

The complexity of our algorithm can be expressed as O(n), where the total number of states (n) is a liner function of the number of GPU cards N, GPU card type (T), and the GPU card location (L). It is expressed as follows

$$n \sim = N*T*L$$

The complexity increases with the increase of the total number of states.

6 CONCLUSION AND FUTURE RESEARCH DIRECTIONS

6.1 Summary

In this dissertation, we developed a theoretical and experimental framework and general methodology for physic aware optimization technique and an autonomic power and performance management of high-performance GPU cluster platforms to achieve (a) online modeling, monitoring, and analysis of power consumption and performance; (b) automatic detection of application execution phases and properties for a wide range of workloads and applications;(c) employing knowledge of application execution phases to dynamically identify strategies that minimize execution time and power consumption while maintaining the required quality of service and accuracy (d) dynamically reconfigure critical and compute intensive application according to the selected optimization strategies; and (e) effective optimization techniques that can scale well and can perform in a dynamic and continuous fashion, making the system truly autonomic. The main research contributions of this dissertation can be highlighted as follows:

1. Applying Physics Aware optimization to speedup the computations of large scale medical applications such as cardiac electrical activities simulation.

2. Hierarchical autonomic power and performance management. Our autonomic GPU cluster was built using smaller self-managing entities modeled based on autonomic computing principles. The autonomic GPU cluster consisted of multiple autonomic GPU cards that self-managed their power and performance according to the cluster

workload. This design is meant to demonstrate that while each individual entity within a GPU cluster is self-managing its own power and performance, they also co-operate and co-ordinate amongst themselves and with the upper and lower-level hierarchies to work towards the overall objective of reducing power consumption for the whole GPU cluster while maintaining performance. This hierarchical management scheme is a manifestation of the local and global control loops of our archetypal autonomic computing system.

3. Characterize dynamic (spatial and temporal) behaviors and resource requirements of applications. We devised an Application Flow (Appflow) methodology that characterizes the spatial and temporal behaviors and resource requirements of applications at runtime. An autonomic entity at any hierarchy uses this Appflow to develop the clairvoyance required to proactively adapt to changes in the incoming workload. We introduced the.

4. Power and performance management algorithms. As part of this dissertation, we devised and validated algorithms for autonomic power and performance management of GPU systems in a high-performance GPU cluster platform. We then built on that work to optimize power and performance management of a GPU cluster platform.

5. PERFORMWATT: Our power and performance management technique addressed GPU devices where existing power management techniques cannot be applied. We

dynamically scale the GPU cluster and the Streaming Multi Processor (SMP) to adapt to the incoming workload's requirements. The goal is to increase the idleness of the remaining SMP and\or GPU devices allowing them to transition to low-power states and remain in those states for as long as the performance remains within given acceptable thresholds. This delivers roughly the same performance, but with considerably less energy consumption consequently maximizing the platform's power and performance.

6. Our POA technique for medical application yielded about 3X improvement of performance with 98.3% simulation accuracy compared to the original application. Also, our Self-configuration management for power and performance management in GPU cluster demonstrated 53.7% power savings for CUDA workload while maintaining the cluster performance within given acceptable thresholds. Furthermore, the overhead of our approach is insignificant on the normal application\system operations and services.

6.2 Contributions

The main contributions of this work can be summarized as follows:

1. Hierarchical modeling of autonomic power and performance management. This approach ensures scalability of solutions and feedback driven decision-making.
2. Introduction of a novel technique called AppFlow that characterizes the spatial and temporal behavior and dynamic resource requirements of applications.

3. AppFlow also provides a coherent and consistent mechanism to capture management objectives while traversing across hierarchies of management without loss of information.
4. Using optimization techniques for runtime identification of appropriate power/performance management strategies that reconfigure the GPU cluster platforms to adapt to the incoming workload.
5. Holistic power management of a GPU cluster platform by collaborative power management of individual cluster component.
6. Physics Aware Optimization (PAO) paradigm that enables programmers to identify the appropriate solution methods to exploit the heterogeneity and the dynamism of the application execution states. We implement a Physics Aware Optimization Manager to exploit the PAO paradigm. It periodically monitors and analyzes the runtime characteristics of the application to identify its current execution phase (state). For each change in the application execution phase, PAO system will adaptively exploit the spatial and temporal attributes of the application in the current state to identify the ideal numerical algorithms/solvers that optimize its performance.

6.3 Future Research Directions

This work is guided by two research visions. The first is to address the problem of power and performance management in computing systems like the medical application and the GPU based cluster and the second is to develop a unified framework for

management of multiple objectives such as power, performance, fault-security etc. Towards that effect, we have demonstrated the simultaneous management of power & performance for GPU clusters. Going forward, we would like to incorporate other management objectives into the framework. This is a highly challenging task given the heterogeneity and dynamism of today's GPU platforms but we can leverage on and tie it together in a unified autonomic management framework.

We would like to expand in breadth and depth into the power management work itself by expanding our work to include other GPU platform components like the host main memory and CPU cores, NIC, hard-disks, power supply, fans. We would also extend the work to upper hierarchies within a GPU based data center. We would also like to incorporate the thermal element into our power management techniques without which power management schemes are quite incomplete.

Our algorithms proposed as part of this thesis are based on optimization approach. We would however like to experiment with other emerging approaches such as Game Theory and light-weight machine learning techniques that would make our algorithms more suitable for real-time use by keeping the complexity in check even when the size scales up.

REFERENCES

[1] Ashby W. R., “Design for a brain (Second Edition Revised 1960)”, published by Chapman & Hall Ltd, London, 1960.

[2] Kimball's Biology Pages, “Organization of the Nervous System”, http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/P/PNS.html#autonomic, October 2007.

[3] Yeliang Zhang. “PHYSICS AWARE PROGRAMMING PARADIGM AND RUNTIME SYSTEM”, PhD. Dissertation, the University of Arizona, 2007

[4] Kephart J.O. and Chess D.M., the Vision of Autonomic Computing, IEEE Computer, vol. 36(1) pp. 41–503, 2003.

[5] Horn P., “Autonomic Computing: IBM’s Perspective on the State of Information Technology”. http://www.research.ibm.com/autonomic/, October 2001.

[6] IBM An architectural blueprint for autonomic computing, April 2003.

[7] Daqing Chen, Phillip Burrell. “Case-Based Reasoning System and Artificial Neural Networks: A Review”. Neural Computing and Applications 10(3): 264-276,2001.

[8] Janet L. Kolodner: “An introduction to case-based reasoning”. Artif. Intell. Rev. 6(1): 3-34, 1992.

[9] Aamodt, A., Plaza, E., “Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches”. AI Communications, 7(i): pp 39-59.1994.

[10] Watson I., Marir F., “Case-Based Reasoning: A Review”. The Knowledge Engineering Review, 9(4), pp.355-381. 1994.

[11] Bonissone, P. P. and Ayub, S., “Similarity Measures for Case-Based Reasoning Systems”. In Proceedings of the 4th international Conference on Processing and Management of Uncertainty in Knowledge-Based Systems: Advanced Methods in Artificial intelligence, July 06 - 10, 1992.

[12] C. Balducelli, S. Bologna, L. Lavalle, G. Vicoli, “Safeguarding information intensive critical infrastructures against novel types of emerging failures, Reliability Engineering & System Safety”, Volume 92, Issue 9, Critical Infrastructures, Pages 1218-1229, ISSN 0951-8320, September 2007.

[13] Wu-Feng, Dinesh Manocha. “High performance computing using accelerators. Parallel Computing”, 33 645–647, 2007.

[14] John E. Stone, James C. Phillips, Peter L. Freddolino, David J. Hardy, Leonardo G. Trabuco, and Klaus Schulten. “Accelerating molecular modeling applications with graphics processors”. Journal of Computational Chemistry, 28:2618-2640, 2007.

[15] Christopher I. Rodrigues, David J. Hardy, John E. Stone, Klaus Schulten, and Wen-mei W. Hwu. “GPU acceleration of cutoff pair potentials for molecular modeling applications”. In CF'08:P roceedings of the 2008 conference on Computing Frontiers, pp. 273-282, New York, NY, USA, 2008.

[16] John Stone, Jan Saam, David Hardy, Kirby Vandivort, Wen-mei Hwu and Klaus Schulten. “High Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs”, Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units Pages 9-18 ,Washington, D.C. 2009.

[17] Samer Al-Kiswany, Abdullah Gharaibeh, Elizeu Santos-Neto, Matei Ripeanu, “On GPU's Viability as a Middleware Accelerator”, Journal of Cluster Computing, Springer, 2009.

[18]http://insidehpc.com/2009/05/05/introducing-the-personal-supercomputers-big-brother-nvidias-tesla-preconfigured-clusters.

[19]http://www.vizworld.com/2009/05/whats-the-big-deal-with-cuda-and-gpgpu-anyway/

[20]Texas Advanced Computing Center Ranger, 2010.

[21] V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, W. Hwu, “GPU Clusters for High-Performance Computing”, In Proc. Workshop on Parallel Programming on Accelerator Clusters, IEEE Cluster, 2009.

[22] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stove, “GPU Cluster for High Performance Computing,”, in Proc. ACM/IEEE conference on Supercomputing, 2004.

[23]Dominik Göddeke, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, Sven H.M. Buijssen, Matthias Grajewski, and Stefan Turek. “Exploring weak scalability for FEM calculations on a GPU-enhanced cluster”. Parallel Computing, Special issue: High-performance computing using accelerators, 33(10–11):685–699, November 2007.

[24]Dominik Göddeke, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, Hilmar Wobker, Christian Becker, and Stefan Turek. “Using GPUs to improve multigrid solver performance on a cluster”. International Journal of Computational Science and Engineering (IJCSE), 4(1):36–55, 2008.

[25] Dominik Göddeke, Hilmar Wobker, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, and Stefan Turek. “Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU”. International Journal of Computational Science and Engineering (IJCSE), 4(4):254–269, 2009.

[26] Xiong, H., Peng, H., Qin, A., and Shi, J. “Parallel occlusion culling on GPUs cluster”. In Proceedings of the 2006 ACM international Conference on Virtual Reality Continuum and Its Applications, (China). VRCIA '06. ACM, New York, NY, 19-26. 2006

[27] Huoping Chen. “Self-Configuration Framework For Networked Systems And Applications”. PhD Dissertation, The University of Arizona, February 2008

[28] Hagan, A. and Zhao, Y. “Parallel 3D Image Segmentation of Large Data Sets on a GPU Cluster”. In Proceedings of the 5th international Symposium on Advances in Visual Computing: Part II (Las Vegas, Nevada, November 30 - December 02, 2009.

[29] M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington, W. Hwu, “QP: A Heterogeneous Multi-Accelerator Cluster,” in Proc. 10th LCI International Conference on High Performance Clustered Computing, 2009.

[30] http://www.ncsa.illinois.edu/

[31]Toshio Endo and Satoshi Matsuoka. "Massive Supercomputing Coping with Heterogeneity of Modern Accelerators", IEEE International Parallel & Distributed Processing Symposium (IPDPS2008), the IEEE Press, April, 2008.

[32]I. N. Kozin, “Comparison of traditional and GPU-based HPC solutions from Power/Performance point of view”. Distributed Computing Group, STFC Daresbury Laboratory, Daresbury, Warrington, Cheshire, WA4 4AD, UK, 2009.

[33] J. Williams, A. George, J. Richardson, K. Gosrani, and S. Suresh, "Computational Density of Fixed and Reconfigurable Multi-Core Devices for Application Acceleration," Proc. of Reconfigurable Systems Summer Institute 2008 (RSSI), Urbana, IL, July 7-10, 2008.

[34] Alexandros Papakonstantinou, Karthik Gururaj, John A. Stratton, Deming Chen, Jason Cong, Wen-Mei W. Hwu, "FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs," Application Specific Processors, Symposium on, pp. 35-42, 2009 IEEE 7th Symposium on Application Specific Processors, 2009.

[35] http://www.NVIDIA.com/docs/IO/43395/SP-04154-001_v02.pdf

[36] M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington, W. Hwu, “QP: A Heterogeneous Multi-Accelerator Cluster”, Proceedings 10th LCI International Conference on High-Performance Clustered Computing, 2009.

[37] M. Wang, M. Parashar, “Programming GPU Accelerators with Aspects and Code Transformation” , Proceedings of the First Workshop on Programming Models for Emerging Architectures (PMEA), Co-located with the 18 the International Conference on Parallel Architectures and Compilation Techniques (PACT), Computer Society Press, Raleigh, North Carolina, USA, September, 2009.

[38] Distributed Computing Group of UK (STFC), “Comparison of traditional and GPU-based HPC solutions from Power/Performance point of view”. Report on power usage of Nehalem/Tesla servers November 2009

[39] George Hornberger and Patricia Wiberg, “Numerical Methods in the Hydrological Sciences”, American Geophysical Union, 2005.

[40]P. Bochev, M. Gunzburger and R. Lehoucq, “On stabilized finited element methods for transient problems with varying time scales”, European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS, 2004.

[41] Gerhard Zumbusch, “Parallel Multilevel Methods: Adaptive Mesh Refinement and Load balancing,” Teubner, 2003.

[42] Yaser Jararweh, Salim Hariri, Talal Moukabary. “Simulating of Cardiac Electrical Activity With Autonomic Run Time Adjustments.”AHSC Frontiers in Biomedical Research, 2009.

[43] Srivastava, R., and T.-C.J. Yeh, “A three-dimensional numerical model for water flow and transport of chemically reactive solute through porous media under variably saturated conditions,” Advances in Water Resources, Vol. 15, p 275-287, 1992.

[44]L. Eldén. Numerical Solution of the Sideways Heat Equation. “In Inverse Problems in Diffusion Processes”. Proceedings in Applied Mathematics, ed. H. Engl and W. Rundell, SIAM, Philadelphia 1995.

[45]J. C. Tannehill, D. A. Anderson, R. H. Pletcher. “Computational fluid mechanics and heat transfer”, 2nd. Edition, Taylor & Francis.

[46]Paul Duchateau, David Zachmann “Applied Partial Differential Equations”, Dover Publications, Inc. 1989.

[47] Sunpyo Hong, Hyesoon Kim, "An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness," Proceedings of the 36th

International Symposium on Computer Architecture (ISCA) , Austin, TX, June 2009.

[48]C.E. Clancy, Z.I. Zhu, and Y. Rudy, “Pharmacogenetics and anti-arrhythmic drug therapy: a theoretical investigation,” American journal of physiology. Heart and circulatory physiology, vol. 292, pp. H66-75. 2007

[49]Pieter Adriaans and Dolf Zantinge, “Introduction to Data Mining and Knowledge Discovery”, Third Edition, New York: Addison Wesley, 1996.

[50] Lan H. Witten and Eibe Frank. “Data Mining practical machine learning tools and technique”. Second edition, Morgan Kaufmann, 2005.

[51] Bithika Khargharia, Salim Hariri, Mazin S. Yousif: “An Adaptive Interleaving Technique for Memory Performance-per-Watt Management”. IEEE Trans. Parallel Distrib. Syst. 20(7): 1011-1022, 2009.

[52] S. Hariri, B. Khargharia, H. Chen , Y. Zhang, B. Kim, H. Liu and M. Parashar, "The Autonomic Computing Paradigm", Cluster Computing: The Journal of Networks, Software Tools and Applications, Special Issue on Autonomic Computing, Vol. 9, No. 2 , Springer-Verlag. 2006

[53] Sebastien Bubeck, Ulrike von Luxburg. “Nearest Neighbor Clustering: A Baseline Method for Consistent Clustering with Arbitrary Objective Functions”. Journal of Machine Learning Research 10, 657-698. 2009

[54] Robnik-Šikonja, M. and Kononenko, I. “Theoretical and Empirical Analysis of ReliefF and RReliefF”. Mach. Learn. 53, 1-2 , 23-69. 2003

[55]A. Huang and P. Steenkiste, “Building Self-adapting Services Using Service-specific Knowledge,” The 14th IEEE Int’ Symposium on High-Performance Distributed Computing (HPDC-14), Research Triangle Park, NC, July 24-27, 2005

[56]A. Huang and P. Steenkiste, “Building Self-configuring Services Using Service-specific Knowledge,” the 13th IEEE Int’ Symposium on High-Performance Distributed Computing (HPDC-13), Honolulu, HI, June 4-6, 2004.

[57] Yu, H., Oh, J., and Han, W. “Efficient feature weighting methods for ranking. In Proceeding of the 18th ACM Conference on information and Knowledge Management” (Hong Kong, China, November 02 - 06). CIKM '09. ACM, New York, NY, 1157-1166. 2009

[58] H. Chen, S. Hariri, and F. Rasal; "An Innovative Self-Configuration Approach for Networked Systems and Applications"; 4th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), 2006

[59]Goldsack, P., et al., “Configuration and Automatic Ignition of Distributed Applications”, HP Openview University Association conference. 2003

[60] http://msdn.microsoft.com/en-us/library/ms174949%28SQL.90%29.aspx.

[61] Andrew Kerr, Gregory F. Diamos, Sudhakar Yalamanchili. “Modeling GPU-CPU workloads and systems”. GPGPU workshop: 31-42. 2010

[62] NVIDIA. “NVIDIA Compute PTX: Parallel Thread Execution”. NVIDIA Corporation, Santa Clara, California, 1.3 edition, October 2008.

[63] http://www.brandelectronics.com/onemeter.html

[64]J. W. Sheaffer, D. Luebke, and K. Skadron. “A flexible simulation framework for graphics architectures”. In HWWS, 2004.

[65] S. Huang, S. Xiao, and W. Feng. “On the energy efficiency of graphics processing units for scientific computing”. In IPDPS, 2009.

[66] M. A. Suleman, M. K. Qureshi, and Y. N. Patt. “Feedback driven threading: Power-efficient and high-performance execution of multithreaded workloads on cmps”. In ASPLOS-XIII, 2008.

[67] Ten Tusscher K.H.& Bernus O.& Hren R. & Panfilov A.V. “Comparison of electrophysiological models for human ventricular cells and tissues”. Prog Biophys Mol Biol., 90: 326-345. 2006

[68] Noble, D., Varghese, A., Kohl, P., Noble, P., “Improved guinea-pig ventricular cell model incorporating a dyadic space, IKr and IKs, and length- and tension-dependent processes”. Can. J. Cardiol. 14, 123–134.1998.

[69] Luo, C., Rudy, Y., “A model of the ventricular cardiac action potential. Depolarization, repolarization, and their interaction”. Circ. Res. 68, 1501–1526.1991.

[70] Priebe, L., Beuckelmann, D.J. “Simulation study of cellular electric properties in heart failure”. Circ. Res. 82, 1206–1223. 1998

[71] Iyer, V., Mazhari, R., Winslow, R. “A computational model of the human left-ventricular epicardial myocyte”. Biophys. J. 87, 1507–1525. 2004

[72] Ten Tusscher KH, Panfilov AV. “Cell model for efficient simulation of wave propagation in human ventricular tissue under normal and pathological conditions”. Phys. Med. Biol., 51: 6141-6156, 2006.

[73]The American Heart Association, “Heart Disease and Stroke Statistics”, 2006.

[74]Ten Tusscher, K.H.W.J., Noble, D., Noble, P.J., Panfilov, A.V. “A model for human ventricular tissue”. Am. J. Physiol. Heart Circ. Physiol. 286, H1573–H1589 , 2004

[75] K.H.W.J. ten Tusscher et al., “A model for human ventricular tissue,” American journal of physiology. Heart and circulatory physiology, vol. 286, Apr. 2004, pp. H1573-89.

[76] D. A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, N. Treuhaft. “Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies,” UC Berkeley Computer Science Technical Report UCB//CSD-02-1175, March 15, 2002

[77] G. Valetto, G. Kaiser, D. Phung. “A Uniform Programming Abstraction for Effecting Autonomic Adaptations onto Software Systems,” Second International Conference on Autonomic Computing, 13-16 June 2005 Page(s): 286 – 297

[78] M. Parashar, Z. Li, H. Liu V. Matossian, and C. Schmidt. “Self-Star Properties in Complex Information Systems,” Volume 3460 of Lecture Notes in Computer 175 Science, chapter Enabling Autonomic Grid Applications: Requirements, Models and Infrastructures. Springer Verlag, 2005.

[79] Vibhore Kumar, Zhongtang Cai, Brian F. Cooper, Greg Eisenhauer, Karsten Schwan, Mohamed Mansour, Balasubramanian Seshasayee, Patrick Widener, “Implementing Diverse Messaging Models with Self-Managing Properties using IFLOW,” 3rd IEEE International Conference on Autonomic Computing (ICAC 2006), Dublin, Ireland, June 2006.

[80] Paul Ruth, Junghwan Rhee, Dongyan Xu, Rick Kennell, Sebastien Goasguen, "Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure", Proceedings of IEEE International Conference on Autonomic Computing (ICAC 2006), Dublin, Ireland, June 2006.

[81] Robbert van Renesse, Kenneth Birman, and Werner Vogels. “Astrolable: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining”, ACM Transactions on Computer Systems, 21(2), May 2003.

[82] G. M. Lohman and S. S. Lightstone, “SMART: Making DB2 (More) Autonomic,” In VLDB 2002, 28th International Conference on Very Large Data Bases, Hong Kong, China, August 20-23 2002.

[83] IBM Research. The Oceano Project. http://www.research.ibm.com/oceanoproject/

[84] IBM OptimalGrid. http://www.alphaworks.ibm.com/tech/optimalgrid

[85] The AutoAdmin project. http://research.microsoft.com/dmx/AutoAdmin

[86] N1, SUN Microsystems. http://www.sun.com/software/n1gridsystem/176

[87] Robert Paul Brettf, Subu Iyer, Dejan Milojicic, Sandro Rafaeli, Vanish Talwar. "Scalable Management," Technologies for Management of Large-Scale, Distributed Systems". Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)

[88]Gerhard Zumbusch, "Parallel Multilevel Methods: Adaptive Mesh Refinement and Loadbalancing," Teubner, 2003.

[89]J. H. Ferziger, "Numerical Methods for Engineering Application," John Wiley & Sons, Inc. 1998.

[90]Jeff Bilmes, Krste Asanovic, Chee-Whye Chin, Jim Demmel, "Optimizing Matrix Multiply Using PHiPAC: A Portable, High-Performance ACSI C Coding Methodology," In Proceedings of the International Conference on Supercomputing, Vienna, Austria, July 1997.

[91]R. C. Whaley, A. Petitet, and J. Dongarra, "Automated Empirical Optimizations of Software and the ATLAS Project," Parallel Computing, 27(1-2): 3-35, January 2001.

[92]Jack Dongarra, Victor Eijkhout, "Self-adapting numerical software for next generation applications," Lapack Working Note 157, ICL-UT-02-07

[93]Benjamin A. Allan, "The CCA Core Specification in a Distributed Memory SPMD Framework," Concurrency Computat. 2002;14:1-23.

[94] Miyoshi A., Lefurgy C., Van Hensbergen E., Rajamony R., Rajkumar R., "Critical power slope: understanding the runtime effects of frequency scaling", Proceedings of the 16th International Conference on Supercomputing, June 22-26, 2002, New York, USA.

[95] Shin H. and Lee J., "Application Specific and Automatic Power Management based on Whole Program Analysis", Final Report, http://cslab.snu.ac.kr/~egger/apm/final-report.pdf, August 20, 2004.

[96] Elnozahy E. N., Kistler M., Rajamony R., "Energy-Efficient Server Clusters". In Proceedings of the 2nd Workshop on Power-Aware Computing Systems, February 2002.

[97] Sunpyo Hong, Hyesoon Kim, "An Integrated GPU Power and Performance Model," ISCA-37, June 2010.

[98] Yang Yang, Jian Chen, Leiling Duan, Luoming Meng, Zhipeng Gao, Xuesong Qiu, “A Self-Configuration Management Model for Clustering-based MANETs”. International Conference on Ultra Modern Telecommunications & Workshops, 2009.

[99] M. Parashar and S. Hariri, Autonomic Computing: Concepts, Infrastructure, and Applications, CRC Press, Taylor & Francis Group, ISBN 0-8493-9367-1, 2007.

[100] NVIDIA Corporation. NVIDIA CUDA Programming Guide, 1.1 edition, 2007.

[101] Derbel, H., Agoulmine, N., and Salaün, M. ANEMA: Autonomic network management architecture to support self-configuration and self-optimization in IP networks. Comput. Netw. 53, 3 (Feb. 2009