

2013

Automatic Energy Saving Schemes for Parallel Applications

Vaibhav Sundriyal
Iowa State University

Follow this and additional works at: <http://lib.dr.iastate.edu/etd>

Part of the [Computer Engineering Commons](#)

Recommended Citation

Sundriyal, Vaibhav, "Automatic Energy Saving Schemes for Parallel Applications" (2013). *Graduate Theses and Dissertations*. Paper 13481.

This Dissertation is brought to you for free and open access by the Graduate College at Digital Repository @ Iowa State University. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Digital Repository @ Iowa State University. For more information, please contact hinefuku@iastate.edu.

Automatic energy saving schemes for parallel applications

by

Vaibhav Sundriyal

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:

Masha Sosonkina, Co-major Professor

Zhao Zhang, Co-major Professor

Philip Jones

Ahmed Kamal

Mark Gordon

Iowa State University

Ames, Iowa

2013

Copyright © Vaibhav Sundriyal, 2013. All rights reserved.

TABLE OF CONTENTS

LIST OF TABLES	v
LIST OF FIGURES	vii
ACKNOWLEDGEMENTS	xii
ABSTRACT	1
CHAPTER 1. INTRODUCTION	2
1.1 Effects of Dynamic Voltage and Frequency Scaling and CPU Throttling on Communication	5
1.1.1 Power Consumption Estimate for Multicore Nodes.	9
1.1.2 Types of Distributed Communication Operations	10
1.1.3 Runtime System Encompassing Point-to-point and Collective Communications	11
1.2 Thesis Organization	12
CHAPTER 2. Energy-Aware Collective Communication Algorithms	13
2.1 All-to-all	13
2.2 Allgather	16
CHAPTER 3. Runtime Procedure	19
3.1 Analysis	19
3.1.1 Sequence	20
3.1.2 Watching	22
3.1.3 Recording	22
3.1.4 Scaling	23

CHAPTER 4. GAMESS	30
4.1 Overview of Quantum Chemistry Package GAMESS	30
4.2 GAMESS Energy Characteristics	31
4.2.1 The 4x4 Execution Configuration	32
4.2.2 The 4x1 Execution Configuration	34
4.2.3 Power profile of Self Consistent Field Phases	35
4.2.4 Energy Consumption Model	37
4.2.5 Model Verification	42
4.3 Mapping	45
CHAPTER 5. Modified Runtime System	49
5.1 Communication Phase Detection	49
5.2 System Design	51
5.2.1 Phase Detection	51
5.2.2 Recording	53
5.2.3 Frequency Scaling	53
CHAPTER 6. Experimental Results	61
6.1 Energy Aware Collective Communication Algorithms	63
6.2 Runtime System	69
6.2.1 Phase Characterization of the Applications Tested	69
6.2.2 NAS Benchmarks: Energy Savings with DVFS	70
6.2.3 GAMESS: Energy Savings with DVFS	72
6.2.4 Throttling	76
6.3 Modified Runtime System	76
6.3.1 Characterization of the Applications Tested	76
6.3.2 CPU and Memory Power Consumption	78
6.3.3 Frequency Scaling with DVFS and Throttling	79
CHAPTER 7. Related Work	84

CHAPTER 8. Conclusions and Future Work	88
8.1 Power Consumption Aware Techniques	90
8.2 Frequency Scaling in GPUs	91

LIST OF TABLES

1.1	Given performance loss tolerance $\delta=10\%$, the selection (Case) of core throttling levels (Level) based on the communication type (Type) and message size (Size \mathcal{L}) from 0 to the largest possible $\hat{\mathcal{L}}$	9
4.1	Input set of molecules.	32
4.2	Regression coefficients for determining the average power consumption at a given frequency.	39
4.3	CN and PE power for Saxitoxin direct mode.	43
4.4	On-chip and off-chip times for input molecules in the FScal platform. .	43
4.5	CN energy consumption in the direct mode on three lower frequencies for the input molecules in the FScal platform, normalized with respect to the highest frequency. Frequencies (in GHz) are $f_1 = 3.0$, $f_2 = 2.67$, $f_3 = 2.33$, and $f_4 = 2.0$	44
4.6	Data substituted into the theoretical model to determine the feasibility of the CN energy consumption for different frequencies.	44
6.1	Effective operational frequency (f_{op}) at a combination of the 2 GHz P-state and one of the T-states (Level) used.	64
6.2	Average percentage of the EDP reductions in all the experiments for the three energy saving strategies.	68
6.3	Application characterizations by the runtime procedure. (The average call and phase lengths, CallLen and PhaseLen , are given in microseconds.)	69

6.4	Energy-delay product (EDP) values for the three proposed bindings under the aggressive frequency scaling strategy normalized to the EDP of the Disjoint-I binding operated at the highest frequency.	75
6.5	Characterization of NAS, CPMD, and pARMS tests (column TName) by the proposed runtime system. The number of phases detected is shown in column PhN . The average call and phase lengths, CLen and PhLen , are in microseconds, respectively. The column CTypes gives the MPI call types as observed in the phases.	78

LIST OF FIGURES

1.1	DVFS (2 GHz) with different throttling states affecting idle power consumption of a node.	3
1.2	MPI ping-pong test to determine the effect of the lowest DVFS (2 GHz) on the internode communication (left) and intranode communication (right). The performance loss percentage of the highest DVFS frequency is shown in the y -axis.	6
1.3	MPI ping-pong test to determine the effect of the lowest DVFS (2 GHz) <i>and</i> CPU throttling on intranode communication. Vertical lines indicate the message sizes at which the performance loss of 10% is reached. . .	7
1.4	CPU throttling with DVFS at 2 GHz in internode communications for messages (left) smaller and (right) larger than 8 KB. Vertical line indicates the message size at which the performance loss of 10% is reached.	8
2.1	The first four communication steps of the Bruck Index all-to-all algorithm on three nodes with two sockets (shown as rectangles) and eight cores (ovals) each. Internode communications are shown as straight arrows across the node boundaries.	14
2.2	Energy saving strategy for the <i>all-to-all</i> operations with the throttling level selection based on the message size (per Table 1.1).	15
2.3	Energy saving strategy for the <i>allgather</i> operations with the throttling level selection based on the message size (per Table 1.1).	17
2.4	The RA communication pattern for the block (left) and cyclic (right) rank placements. Arrows indicate the communication direction.	18

3.1	Sequence recognition and phase grouping	20
3.2	State diagram for runtime procedure to apply energy savings efficiently. The transitions are labeled with Lt , where L takes a value of the first 11 letters of the alphabet The transition of a state into itself (At , Et , Ft , It) indicate ongoing state action.	21
3.3	Trace of an MPI application invoking eight MPI.Send and MPI.Recv calls with a phase length of four. (The calls within a phase are ordered lexicographically from <i>a</i> to <i>d</i> .)	23
3.4	An Intel Xeon E5450 processor within one socket, having four physical cores. A physical core has an L1 cache and a shared L2 cache, such that pairs of core processing units (PUs)—(P#0, P#4), (P#2, P#6)—each share an L2 cache.	26
3.5	Overlapping of communication phases in a pair twin-cores that share the L2 cache. The phase boundaries are marked as vertical dashed and dash-dotted for core 0 and core 4, respectively.	27
3.6	Grouping of the communication phases detected for rank 0 of the MG NAS benchmark. The capital letters followed by the double dots rep- resent communication phases followed by interphase gaps, respectively. A single group (shown as solid oval) has been found and the two corre- sponding subsequences are enclosed into the ovals denoting this group.	28
4.1	4x4 configuration: (a) Execution time, (b) Average power consumption, and (c) Energy consumption normalized with respect to the direct mode for the input molecules shown on the <i>x</i> axes in the ascending order of their I/O requirements.	33
4.2	4x1 configuration: (a) Execution time, (b) Average power consumption, and (c) Energy consumption normalized with respect to the direct mode for the input molecules shown on the <i>x</i> axes in the ascending order of their I/O requirements.	35

4.3	Power profiles for some molecules in the 4x4 configuration.	36
4.4	Power profiles for some molecules in the 4x1 configuration.	37
4.5	Average power consumption of input molecules for direct mode on the four frequencies of FScal platform.	39
4.6	Computation times ratio τ for input molecules normalized with τ for Quinine (FScal platform).	39
4.7	Normalized energy consumption of PE and CN for Saxitoxin in direct mode.	45
4.8	Variation of energy consumption with performance loss for Saxitoxin in direct mode.	46
4.9	Three bindings of GAMESS data server (D) and compute (C) processes: (a) Disjoint-I, (b) Disjoint-II, and (c) Slave, for Each compute node has two quad-core processors arranged as two sockets. Twin cores share the L2 cache.	46
5.1	Phase detection and string manipulation for master string with (a) con- secutive and (b) nonconsecutive communication phases.	50
5.2	State diagram for runtime system to apply frequency scaling efficiently. The transitions are labeled with Lt , where L takes a value $A-G$. The transition of a state into itself (At , Bt , Ct) indicate ongoing state action.	51
5.3	Trace of an MPI application invoking eight MPI calls with a phase length of four. (The calls within a phase are ordered lexicographically from a to d .)	54
5.4	Grouping of the communication phases detected for rank 0 of the MG NAS benchmark. The capital letters followed by the double dots rep- resent communication phases followed by interphase gaps, respectively. A single group (shown as solid oval) has been found and the two corre- sponding subsequences are enclosed into the ovals denoting this group.	57

6.1	3D geometrical structure of the TCMS molecule (silicon, carbon, oxygen, nitrogen, chlorine and hydrogen atoms are shown as pink, gray, red, blue, green, and white balls, respectively). The molecular skeletal formula is inserted on the right.	63
6.2	The all-to-all performance degradation on 80 cores (left) for three cases and the power consumption across a compute node (right) for the four cases: Executing at the highest frequency and no throttling (Full power); only frequency scaling without throttling (DVFS only); only CPU throttling without frequency scaling Throttling only ; and using the energy saving strategies proposed for all-to-all (Proposed).	64
6.3	The allgather performance degradation on 80 processes (left) for three cases and the power consumption across a compute node (right) for the four cases: Executing at the highest frequency and no throttling (Full power); only frequency scaling without throttling (DVFS only); only CPU throttling without frequency scaling Throttling only ; and using the energy saving strategies proposed for allgather (Proposed).	66
6.4	Execution time (top) and energy consumption (bottom) of 11 CPMD inputs on 80 cores and of 2 NAS benchmarks on 64 processes for the DVFS only , Throttling only , and Proposed cases normalized to the Full power . The last set of bars (Average) represent the average of the respective y -axis values across all the CPMD and NAS tests.	67
6.5	Execution time (left) and energy consumption (right) of five Elemental algorithms—Cholesky (Chol), Triangular inverse (Inv), LU decomposition (LU), Hermitian eigensolver (Hegst), and LDL^T factorization (LDLT)—on 80 cores for the DVFS only , Throttling only , and Proposed cases normalized to the Full power . The last set of bars (Average) represent the average of the respective y -axis values across the five Elemental algorithms.	68

6.6	Execution time and energy consumption of the NAS MG and CG benchmarks for the different DVFS strategies normalized to the case when all the processes operate at the highest frequency. (Results below 1 are better.)	71
6.7	(a) Execution time and (b) energy consumption of the two Silatrane computations under Disjoint-I process mapping for the different DVFS strategies, normalized to the case when Disjoint-I binding is operated at the highest frequency. (Results below 1 are better.)	73
6.8	(a) Execution time and (b) energy consumption of the two Silatrane computations under Slave process mapping for the different DVFS strategies, normalized to the case when Disjoint-I binding is operated at the highest frequency. (Results below 1 are better.)	74
6.9	(a) Execution time and (b) energy consumption of the two Silatrane computations under Disjoint-II process mapping for the different DVFS strategies, normalized to the case when Disjoint-I binding is operated at the highest frequency. (Results below 1 are better.)	75
6.10	CPU power consumption for CG benchmark.	79
6.11	Normalized execution time of the NAS, pARMS, and CPMD tests for the different frequency scaling strategies. (Results below 1 are better.)	81
6.12	Normalized energy consumption of the NAS, pARMS, and CPMD tests for the different frequency scaling strategies. (Results below 1 are better.)	82
8.1	Frequency scaling range for the K20 Tesla GPU obtained through nvidia-smi	91

ACKNOWLEDGEMENTS

I am very grateful to Dr. Masha Sosonkina for being an understanding guide who is always willing to hear the other person and, for supporting me through ups and downs of my research. I thank her for her numerous hours spent on revising the manuscripts to bring them to the highest standards — providing me an invaluable learning experience in the process. She has always managed to give me time for discussions in spite of her busy schedule, yet, our meetings have never been during the late hours or the weekends. This balanced approach towards work and personal life is one of the most admirable things about her. I am also grateful to Dr. Zhao Zhang for providing me useful inputs in my research through out my stint at Ames Laboratory. His lectures proved to be a major factor in choosing my area of research for this thesis.

I am very thankful to my other committee members Dr. Philip Jones, Dr. Ahmed Kamal and Dr. Mark Gordon for their valuable time and inputs on my research. I am very thankful to my lab members for providing me with a friendly environment and for their support in my research. I thank all the departmental staff for their timely help.

I gratefully acknowledge the support received from in part by Ames Laboratory and Iowa State University under the contract DE-AC02-07CH11358 with the U.S. Department of Energy, by the Air Force Office of Scientific Research under the AFOSR award FA9550-12-1-0476, and by the National Science Foundation grants NSF/OCI—0941434, 0904782, 1047772.

I am very grateful to my parents and family members for supporting me to come here for PhD and, for their love and care in my life.

I am very grateful to God for always arranging the best for me and for all the gifts in life. I am very grateful to my friends in Ames, whose unconditional support has helped me in countless ways in my PhD and in leading a wholesome life.

ABSTRACT

Although high-performance computing traditionally focuses on the efficient execution of large-scale applications, both energy and power have become critical concerns when approaching exascale. Drastic increases in the power consumption of supercomputers affect significantly their operating costs and failure rates. In modern microprocessor architectures, equipped with dynamic voltage and frequency scaling (DVFS) and CPU clock modulation (throttling), the power consumption may be controlled in software. Additionally, network interconnect, such as Infiniband, may be exploited to maximize energy savings while the application performance loss and frequency switching overheads must be carefully balanced. This work first studies two important collective communication operations, all-to-all and allgather and proposes energy saving strategies on the per-call basis. Next, it targets point-to-point communications to group them into phases and apply frequency scaling to them to save energy by exploiting the architectural and communication stalls. Finally, it proposes an automatic runtime system which combines both collective and point-to-point communications into phases, and applies throttling to them apart from DVFS to maximize energy savings. The experimental results are presented for NAS parallel benchmark problems as well as for the realistic parallel electronic structure calculations performed by the widely used quantum chemistry package GAMESS. Close to the maximum energy savings were obtained with a substantially low performance loss on the given platform.

CHAPTER 1. INTRODUCTION

The last few decades have witnessed a tremendous rise in the design of scalable applications for various scientific domains. Their computational requirements force system engineers to develop ever more performance-efficient architectures. As a result, power consumption is rapidly becoming a critical design constraint in modern high-end computing systems. For example, according to an U.S. Department of Energy guidelines [48], to sustain an exaflops machine, its power consumption cannot go beyond ten-fold that of the current petaflops machines, meaning that for a 1000-fold increase in performance, the increase in power consumption may not accede ten-fold. Moreover, if the focus of the high-performance computing (HPC) community is only to maximize application performance, the computing system operating costs and failure rates can reach prohibitive levels.

A wide range of HPC applications rely on the communication libraries implementing the Message Passing Interface¹ (MPI), which has become a *de facto* standard for the design of parallel applications. It defines both point-to-point and collective communication primitives widely used in parallel applications. Researchers in the past have proposed energy-aware techniques in MPI [35, 17] by identifying communication phases in the execution of parallel application which are not compute intensive and then applying dynamic voltage and frequency scaling (DVFS) during those phases. However, most of these studies do not apply energy saving strategies *within* collective operations. For example, in [35], authors identify the communication phase as several MPI calls, all lasting long enough to apply DVFS without a significant overhead.

Conversely, collective operations are studied in this work on the *per-call* (fine-grain) basis as opposed to a “black-box” approach, which treats communication phase as indivisible operation contributing to the parallel overhead. The appeal of fine-grain *per-call* approach to energy

¹MPI Forum: <http://www.mpi-forum.org>

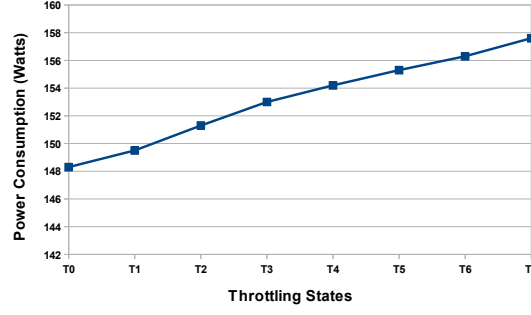


Figure 1.1 DVFS (2 GHz) with different throttling states affecting idle power consumption of a node.

saving grows with the advent of large-scale computing platforms in which the the number of communicating processes is increasing rapidly. In particular, for such platforms a collective operation would amount to a large multitude of point-to-point communications grouped together and essentially present a communication phase in itself. Furthermore, to retain algorithmic efficiency, energy saving strategies are being incorporated within the *existing* algorithms used for executing the collective operations. In this work, the initial steps of which were described in [53], both the type of a collective communication and the amount of data to be transferred are considered to obtain energy saving with the DVFS in conjunction with the CPU throttling. Being widespread in parallel applications, the all-to-all and allgather operations are addressed here. By definition, a collective operation requires the participation of all the processes in a given communicator. Hence, such operations incur a significant amount of the network phase during which there exist excellent opportunities for applying energy saving techniques, such as DVFS and CPU throttling.

CPU Throttling and Idle Power Consumption. It was noticed that idle power consumption of a node increases as a higher level of throttling is applied. In the idle state of the system a loop executes which consists of the HLT assembly language instruction to save power. For example, Fig. 1.1 shows the change in the idle power consumption of a node at 2 GHz with various T-states. Note that, although the idle power of a node refers to the power consumed when the processor is idle, an augmentation in the T-states increases this power. This increase may be attributed to the CPU throttling causing the CPU to reach a higher CPU power mode,

also called C-state², less often and, thus, to draw more power. The operating system kernel installed on the node does not have a tickless scheduler and has frequency of 1000 Hz. Not having a tickless scheduler may also affect this power consumption as suggested in [49], for example. However, contrary to a popular view that throttling is useless for the processors equipped with DVFS and despite this negative effect, it is a viable option when used in conjunction with DVFS.

CPU Throttling and Dynamic Voltage and Frequency Scaling(DVFS) in Intel Architectures. The current generation of Intel processors provides various P-states for DVFS and T-states for throttling. In particular, the Intel “Core” microarchitecture, which provides four P-states and eight T-states from T_0 to T_7 , where state T_j refers to introducing j idle cycles per eight cycles in CPU execution. The delay of switching from one P-state to another can depend on the current and desired P-state and is discussed in [41]. The user may write a specific value to model-specific registers (MSRs) to change the P- and T-states of the system.

The CPU throttling can be viewed as equivalent to dynamic frequency scaling (DFS) [4] because, by inserting a given number of idle cycles in the CPU execution, a particular operating frequency is obtained without changing the operating voltage of the cores. Hence, DFS is less effective than DVFS in terms of power saving but, when used with conjunction with DVFS, it may result in significant reduction of power across a compute node.

Notes on Infiniband. The Infiniband interconnect has become one of most popular standards marking its presence in more than 43% of the systems in the TOP 500³ list. Several network protocol layers are offloaded to the host channel adapters (HCA) in an Infiniband⁴ network. Here, MVAPICH2⁵ implementation of MPI, which is designed for Infiniband networks, is considered. Infiniband offers two modes of message progression: polling and blocking. Even though blocking mode consumes less power than polling, a lower communication overhead is

²Advanced Configuration and Power Interface: <http://www.acpi.info>

³<http://www.top500.org>

⁴<http://www.infinibandta.org>

⁵<http://mvapich.cse.ohio-state.edu>

incurred with the polling mode, in which an MPI process constantly samples for the arrival of a new message rather than the with blocking, which causes CPU to wait for an incoming message. Therefore, MVAPICH2 uses the polling communication mode by default. A detailed comparison of these two modes for the all-to-all operation is done in [27], where the high overhead introduced by blocking mode is demonstrated. Additionally, on the platform used in this work, the average performance loss observed was 19%–340% with power savings of 2%–34% for the blocking mode when compared to polling one.

1.1 Effects of Dynamic Voltage and Frequency Scaling and CPU Throttling on Communication

Since collective operations are developed on top of point-to-point communication operations, it is reasonable to analyze first the DVFS and CPU throttling effects on the latter operations. In particular, this section explores possibilities to apply these frequency scaling techniques with respect to two factors, message size and communication type (inter- or intranode), which are deemed to affect significantly the energy saving potential in any multicore computing environment. Although the investigation was performed with a certain Intel processor architecture using the Infiniband to connect the nodes, the obtained results lead nevertheless to general conclusions since there were no assumptions made on particular values or the number of the C-states (only their relative ordering was used).

According to the theoretical models [59] estimating the time spent in a collective operation, any increase in the point-to-point communication time should proportionally increase the time spent in the collective operation. Therefore, a user may judiciously choose a performance loss for the collectives by analyzing the behavior of point-to-point communications. Then appropriate DVFS and CPU throttling levels may be selected to minimize the energy consumption during a collective operation with the chosen performance loss. In this work, a performance loss threshold of $\delta=10\%$ is considered and the collective communication algorithms are studied in detail to exploit opportunities for energy savings within them on a fine-grain level. Alternatively, one may consider the so-called energy-delay product (EDP) and its variants in order to

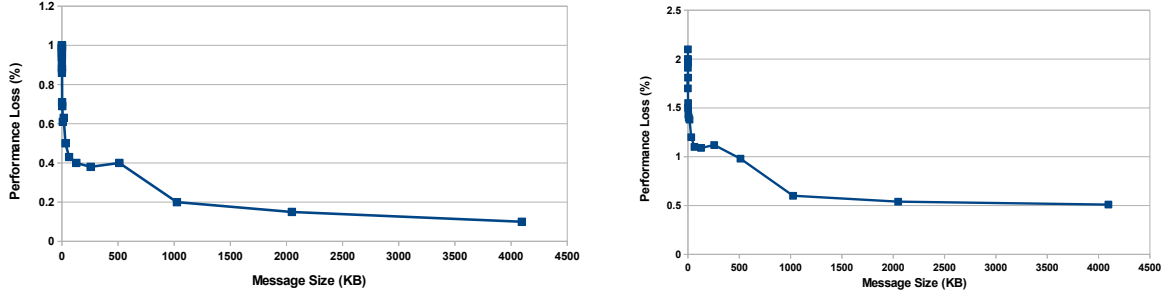


Figure 1.2 MPI ping-pong test to determine the effect of the lowest DVFS (2 GHz) on the internode communication (left) and intranode communication (right). The performance loss percentage of the highest DVFS frequency is shown in the y -axis.

measure the energy savings (see, e.g., [37]). EDP has a clear advantage over measuring “raw” power consumption since it accounts for the execution time as well. So, there is no temptation to run an application as slow as possible to gain in energy efficiency. Although EDP is a preferred general-use metric for high-performance applications, adhering to a performance loss tolerance set by a knowledgeable application user or keeping the performance loss as small as possible may work just as rigorously.

Fig. 1.2(left) and Fig. 1.2(right) depict the degradation in the point-to-point internode and intranode communication time, respectively, when two communicating processes are operated at the the lowest frequency as compared with the highest. A ping-pong communication test is chosen to evaluate the effect of frequency scaling because this is a typical way to use point-to-point communications in MPI. The plots in Fig. 1.2 show the average performance loss of about 1%; thus, the effect of the DVFS on both internode and intranode point-to-point communications is minimal. Therefore, they can safely be operated at the lowest frequency of the processor. It can be noted here that the DVFS and CPU throttling for evaluating point-to-point communications were applied only on the cores on which the processes were executing.

If a DVFS switch is applied for every single communication call, it may result in a significant performance loss because of the DVFS overhead which is present in the majority of the commodity processor types from Intel and AMD processor types. For this reason, the research in [35] focuses on developing a phase-detection framework that dynamically identifies the communications phases in an application and applies DVFS per phase rather than per

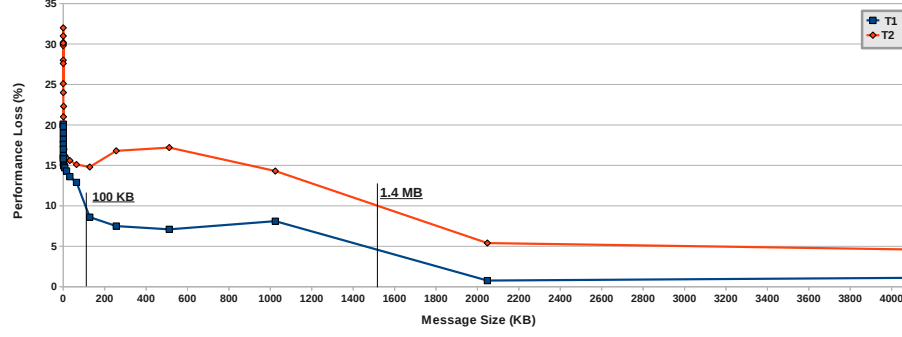


Figure 1.3 MPI ping-pong test to determine the effect of the lowest DVFS (2 GHz) *and* CPU throttling on intranode communication. Vertical lines indicate the message sizes at which the performance loss of 10% is reached.

communication call. For collective communications, however, the impact of DVFS overhead on overall performance decreases rapidly as the number of communicating cores increases because the collectives may be viewed as a bundle of several point-to-point communications from an algorithm implementation viewpoint.

Selecting Appropriate CPU Throttling Levels. Since DVFS reduces both frequency and voltage of the processor cores, it saves more power than CPU throttling, which reduces only the frequency, if both are used separately. Thus, DVFS should be always reduced first to its lowest value and only after that the throttling level should be increased.

The drawback of the CPU throttling is that it may result in significant performance loss for the intranode communication. Fig. 1.3 depicts the performance degradation in intranode point-to-point communication from the case when the communicating cores operate at the highest frequency to the one at the lowest frequency, while in different T-states. The results are shown for the states T_1 and T_2 only because higher T-states produce a performance degradation greater than 10% for all the message sizes. By comparing Fig. 1.2(right) and Fig. 1.3, it is observed that, with just one idle CPU cycle, the average performance loss increases from 1.5% to 15%. Fortunately, as the message size grows, the performance loss decreases when CPU throttling is applied, which also has been noticed in [36].

Fig. 1.4 shows the performance degradation in the *internode* communication time with DVFS and various throttling states. This communication time is much less affected by the

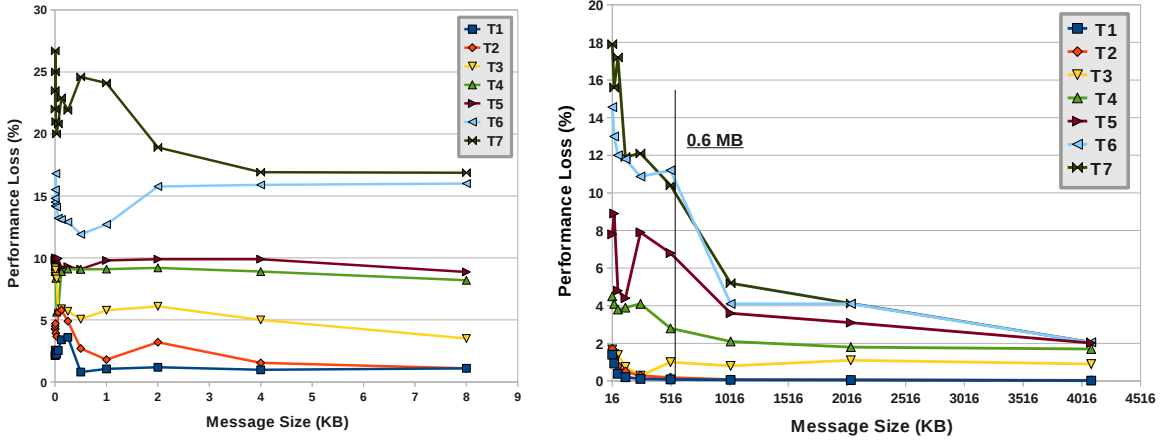


Figure 1.4 CPU throttling with DVFS at 2 GHz in internode communications for messages (left) smaller and (right) larger than 8 KB. Vertical line indicates the message size at which the performance loss of 10% is reached.

CPU throttling as compared with the intranode communication because, for the internode message transfers, the remote dynamic-memory access (RDMA) offloads a large part of the communication processing to the network interface cards (NICs) (see [36]). Thus, the processor involvement in internode message transfers is minimal and higher throttling states may be used safely.

Fig. 1.3 indicates (see vertical lines) that the performance loss is no higher than 10% when the throttling state T_1 is applied to the messages greater than 100 KB and, after the 1.4 MB, the state T_2 may be used during the *intranode communication*. Similarly, from Fig. 1.4, it may be inferred that 0.6 MB is the message size corresponding to the throttling level switch (from T_5 to T_7) so the chosen performance loss is not exceeded for the *internode communication*. If a processing core is not involved in any communication during the collective operation, then it can be operated at the highest throttling level T_7 . It can be noted that performance loss of 10% is the maximum performance loss that an application suffers from since the collective communication may form only a portion of the parallel execution of that application.

For the given computing environment and performance loss of 10%, Table 1.1 summarizes the appropriate CPU throttling levels depending on the type of communication and message size. In a different environment, the message size intervals and the throttling level numbers may differ. However, the deduced general principles of the DVFS and CPU throttling application

Table 1.1 Given performance loss tolerance $\delta=10\%$, the selection (**Case**) of core throttling levels (**Level**) based on the communication type (**Type**) and message size (**Size \mathcal{L}**) from 0 to the largest possible $\hat{\mathcal{L}}$.

Case	Type	Size \mathcal{L}	Level
I	Intranode	[100KB, 1.4MB[T_1
II		[1.4MB, $\hat{\mathcal{L}}$]	T_2
III	Internode	[1B, 0.6MB[T_5
IV		[0.6MB, $\hat{\mathcal{L}}$]	T_7
V	None	0	T_7

will hold. Specifically,

- DVFS may remain at the lowest state during the entire communication regardless of the message size.
- Intranode communications may suffer from high throttling levels. while the internode communications are not affected as much by throttling.
- For the intranode, choose the lowest T-state for small message sizes, while the next level up may be selected for bigger ones.
- For the internode, choose the highest T-state for large message sizes and decrease it for small messages.
- If a core is idle during the communication it may be operated at the highest T-state.

Finally, note that collective communications are blocking in nature and, therefore, reducing the processor frequency during their operation will not slow down the computational portion of an application.

1.1.1 Power Consumption Estimate for Multicore Nodes.

Both DVFS and CPU throttling are employed in this work to obtain energy saving, so there is need to understand the relationship between these two techniques and their relative efficacy. In addition, it would be beneficial to predict how the proposed energy saving strategies work a platform with a large number of cores. Therefore, a theoretical power consumption estimate is presented and experimentally verified here for a multicore node.

Let a multicore compute node have frequencies f_i , ($i = 1, \dots, m$), such that $f_1 < \dots < f_m$, and throttling states T_j , ($j = 0, 1 \dots, n$). When all the c cores of the node execute an application

at the frequency f_i , each core consumes the dynamic power p_i proportional to f_i^3 as has been shown in [9]. Let $P_{i,j}$ be the power consumed by the entire node at the frequency f_i and throttling state T_j ; $\bar{P}_{i,j}$ be the total *idle* power consumption of a node at the frequency f_i and throttling level T_j , and P_d be the dynamic power consumption of the compute node components different from the processor (e.g., memory, disk, and NIC). Then, the power consumption of a node with no idle cycles, i.e., at T_0 , may be defined as

$$P_{i,0} = c \times p_i + \bar{P}_{i,0} + P_d \quad (1.1)$$

to give an idea how the frequency scaling affects the power consumption. This expression may vary with the application characteristics since each application may have a different power consumption pattern depending of its utilization of the compute node components. When the nonzero level of throttling T_j , ($j \neq 0$) is used, equation Equation (1.1) becomes

$$P_{i,j} = \frac{j \times (\bar{P}_{i,j} + P_d) + (n - j)P_{i,0}}{n}. \quad (1.2)$$

To determine the idle power consumption of a node $\bar{P}_{i,j}$, Fig. 1.1 and equation Equation (1.2) may be used. For example, the node power consumptions at the lowest frequency f_1 and no throttling, $P_{1,0}$, and with the T_7 throttling level, $P_{1,7}$, are 148 and 158 watts, respectively, according to Fig. 1.1. Then, the idle power $\bar{P}_{1,7}$ at f_1 and T_7 is found to be around 160 watts by substitution into equation Equation (1.2) and with assumptions of eight T-states and no dynamic power consumption. To predict the entire node power for the proposed energy saving strategies, the P_d of such components as memory and NICs has to be also considered. For the memory dynamic power consumption, an extrapolation method discussed in [15] has been used to obtain 10 watts consumed by memory in the all-to-all operation for 1 MB message size.

1.1.2 Types of Distributed Communication Operations

A collective operation is considered as a multitude of point-to-point communications grouped together, essentially presenting a single communication stream. Since the rank⁶ sequences of

⁶For the sake of brevity throughout this work, the “rank” will denote either the destination or source neighbor rank depending on whether the sending or receiving operation is considered, respectively.

the point-to-point transfers within a collective communication are typically known during the collective algorithm execution, the places to apply DVFS and throttling may be determined in advance for a given message size. For example, an *a priori* algorithmic analysis of the MPI_Alltoall algorithms reveals that, for a few initial iterations, every core undergoes intranode communications after which the communication becomes purely internode. Thus, different throttling states may be preselected while the DVFS is lowered to the maximum at the start of MPI_Alltoall and raised back to the highest level in the end. A single point-to-point communication is, on the other hand, just one call per given processor rank. Therefore, its frequency scaling on the percall basis, as was done for collectives, may easily result in a significant parallel performance degradation due to the switching overhead. Nevertheless, it is desirable to decrease the energy consumption during the point-to-point communications, in addition to collectives, because they may constitute a significant portion of the execution (often more than 10%); and applications communicating heavily in the point-to-point fashion are abundant. Therefore, this work focuses on a class of point-to-point communications as provided by the MPI standard.

By considering test cases from the NAS benchmark suite [5], this work validates a proposed runtime procedure that groups several point-to-point communications, aiming to reduce the overhead from the DVFS and throttling. Next, it applies this procedure to realistic electronic structure calculations performed by the widely-used GAMESS quantum chemistry package [20, 47], which is capable of performing molecular structure and property calculations by a rich variety of *ab initio* methods. An estimated user base of 150,000 comes from more than 100 countries. The GAMESS communication library, which has been custom-built, is based on the partitioned global address space (PGAS) concepts.

1.1.3 Runtime System Encompassing Point-to-point and Collective Communications

The phase-detection mechanism proposed in [56] cannot be simply extended to encompass *both* point-to-point and collective communications and the proposed frequency scaling strategies did not combine DVFS and throttling. A modified runtime system is proposed here that detects communication phases in parallel applications transparently to the application code and com-

munication library and without any prior knowledge of the parallel application communication characteristics. Once the communication phases are detected, a particular frequency is chosen based on the proposed frequency scaling strategies and the library-specific implementations of communication calls. To illustrate the latter, a few examples from an MPI implementation of collective calls are provided. By considering test cases from the NAS benchmark suite [5], as well as real-world applications in molecular dynamics (CPMD⁷) and iterative parallel linear system solver (pARMS [34]), the proposed runtime system is validated.

1.2 Thesis Organization

This thesis is organized as follows: In Chapter 2, the design and implementation of energy aware collective communication algorithms is discussed.

In Chapter 3, the runtime procedure to obtain energy saving in point-to-point communications is proposed.

In chapter 4, the energy characteristics and the application of the runtime procedure to the quantum chemistry software GAMESS is discussed.

In Chapter 5, the modified design of the runtime procedure which applies frequency scaling to both point-to-point and collective communications is discussed.

In Chapter 6, the experimental results are presented.

In Chapter 7, existing related works are reviewed.

In Chapter 8, the conclusions and future work regarding obtaining energy saving in GPUs is discussed.

⁷CPMD Consortium: <http://www.cpmc.org>

CHAPTER 2. Energy-Aware Collective Communication Algorithms

In this chapter, the algorithms for two collective operations, all-to-all and allgather as implemented in MVAPICH2, are studied and strategies are proposed to minimize the energy consumption during their algorithmic steps. For the findings from Section 1.1 to be employed here, the intra- and internode communication types need to be determined within the collective algorithms used as well as different message sizes considered in their implementations. Thus, the proposed strategies will contain stages in accordance with the communication types and each stage will have several cases based on possible message sizes.

Rank placement is directly related to the communication type. MVAPICH2 provides two formats of rank placements on multicores, namely *block* and *cyclic*. In the block strategy, ranks are placed such that any node j ($j = 0, 1, \dots, N - 1$) having c cores contains ranks from $c \times j$ to $c \times (j + 1) - 1$. In the cyclic strategy, all the ranks i belong to j if $(i \bmod N)$ equals j . The block rank placement has been chosen for the rest of the work due to several considerations, outlined in the end of this section.

2.1 All-to-all

The MVAPICH2 implementations of all-to-all are based on three algorithms: 1) Bruck Index, used for small (less than or equal to 8KB) messages with at least eight participating processes; 2) Pairwise Exchange, used for large messages and when the number of processes is a power of two; 3) Send To rank $i + k$ and Receive From rank $i - k$, used for all the other processor numbers and large messages. These algorithms are referred further in text as BIA, PEA, and STRF, respectively.

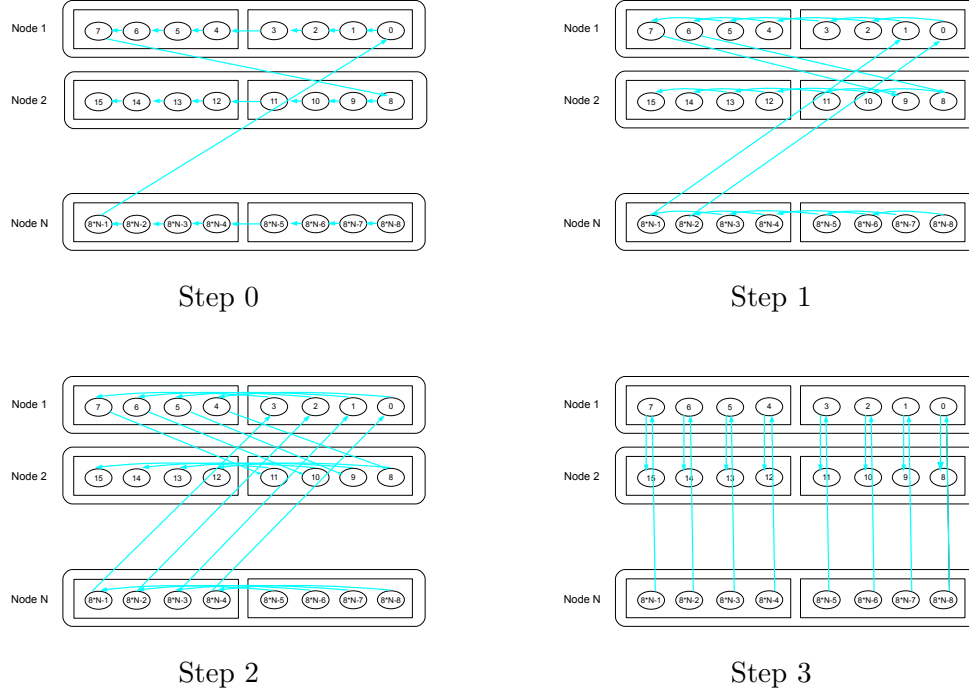


Figure 2.1 The first four communication steps of the Bruck Index all-to-all algorithm on three nodes with two sockets (shown as rectangles) and eight cores (ovals) each. Internode communications are shown as straight arrows across the node boundaries.

Bruck Index Algorithm first makes a local copy and then does an upward shift of the data blocks from the input to output buffer. Specifically, a process with the rank i rotates its data up by i blocks. The communication starts in a way such that, for all the p communicating processes, in each communication step k ($0 \leq k < \lceil \log_2 p \rceil$), process i , ($i = 0, \dots, p-1$), sends to $(i + 2^k) \bmod p$ (with wrap-around) all those data blocks whose k th bit is 1 and who receive from $(i - 2^k) \bmod p$. The incoming data is stored into the blocks whose k th bit is 1. Finally, the local data blocks are shifted downward to place them in the right order. Fig. 2.1 shows N nodes with $c = 8$ cores arranged as two sockets and the total number of $p = 8N$ processors performing the first four steps of the BIA. The rank placement is performed in block manner using consecutive core ordering. It can be observed that, until the k th step where $2^k < c$, the communication is still *intranode* for any core in the cluster. However, after the k th step, the communication becomes purely *internode* for all the participating cores.

If M is the data size to be exchanged in the all-to-all operation, then in each step, a process receives and sends $\mathcal{L} = (M \times p)/2$ amount of data. Therefore, appropriate throttling levels

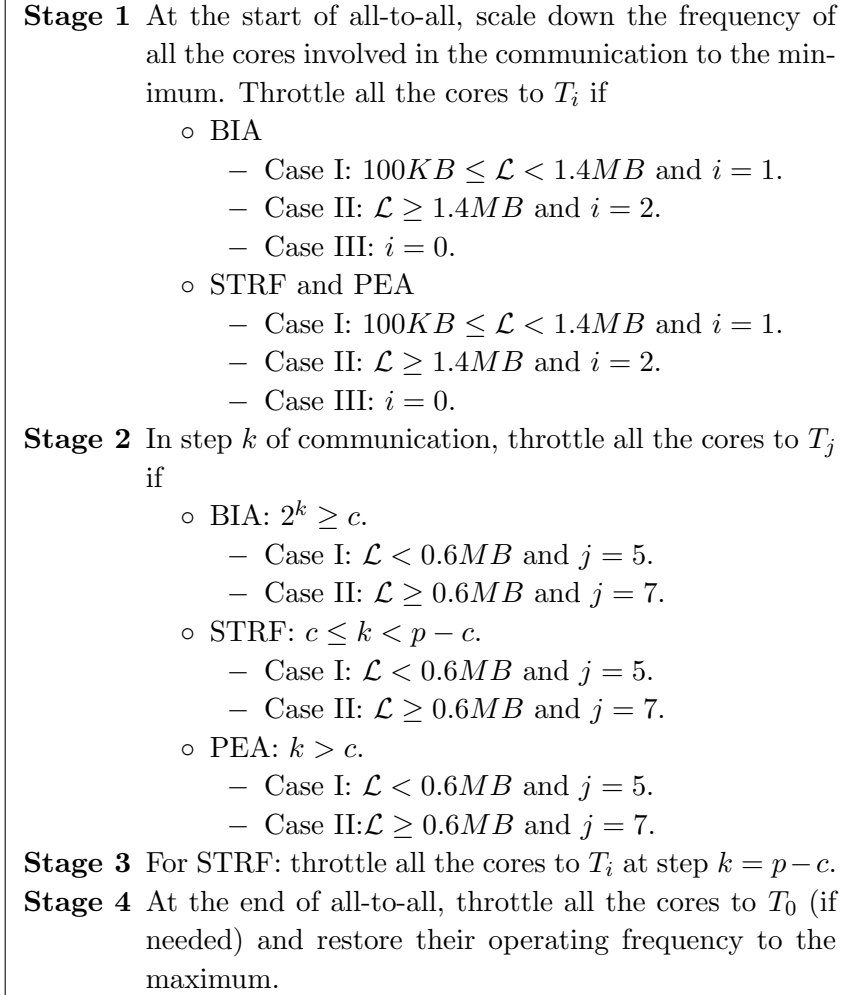


Figure 2.2 Energy saving strategy for the *all-to-all* operations with the throttling level selection based on the message size (per Table 1.1).

can be selected from Table 1.1 according to the communication type, given performance loss tolerance, and the message size.

“Send-To Receive-From” and Pairwise Exchange. For the block placement of ranks, in each step k ($1 \leq k < p$) of STRF, a process with rank i sends data of size $\mathcal{L} = M$ to $(i+k) \bmod p$ and receives from $(i-k+p) \bmod p$. Therefore, for the initial and final $c-1$ steps, the communication is intranode while between these steps the communication is internode for any core in the cluster. The PEA uses *exclusive or* operation to determine the rank of processes for data exchange. It is similar to the BIA in terms of communication phase since after step k where $k = c$, the communication operation remains internode until the end.

Energy Saving Strategy. Because all three algorithms exhibit purely internode communications at a certain step k , the energy saving strategy as shown in Fig. 2.2 may be applied in stages to each of them. Following [38], the DVFS and CPU throttling policies are defined by the specific points within an algorithm where these techniques are applied and by a set of conditions indicating when to apply them. The invocation and the finishing point of the all-to-all operation determine the stages where DVFS is applied and the proposed energy saving strategy defines the set of conditions as to how throttling is applied depending on the type (intra- or internode) of the given collective algorithm.

2.2 Allgather

MVAPICH2 uses three algorithms for performing allgather operation: 1) Bruck Concatenation algorithm for the nonpower of two number of processes and message size smaller than or equal to 1 KB, 2) Recursive Doubling algorithm for the power of two number of processes and message size larger than or equal to 1 KB, and 3) Ring algorithm for the message size greater than 1 KB and any number of processes. These algorithms are referred further in text as BCA, RDA, and RA, respectively.

Bruck Concatenation Algorithm copies the input data in each process to the top of the output buffer. Then, the communication phase proceeds as follows. In each step k , the process of rank i sends to the rank $i - 2^k$ all the data it currently possesses followed by the receive operation from the rank $i + 2^k$. The data from the rank $i + 2^k$ is then appended to the data already residing in the rank i . This procedure continues for $\log_2 p$ steps. If the number of processes is not a power of two, an additional step is needed, such that each process sends the first $(p - 2^{\lfloor \log_2 p \rfloor})$ blocks from the top of its output buffer to the destination and appends the data it receives to the data it already has. For BCA algorithm, in each step, a process receives and sends $\mathcal{L} = (M \times p)/2$ amount of data.

Recursive Doubling Algorithm. In the first step, the processes that are a distance-one apart exchange their data of size M . In the second step, the processes that a distance-two

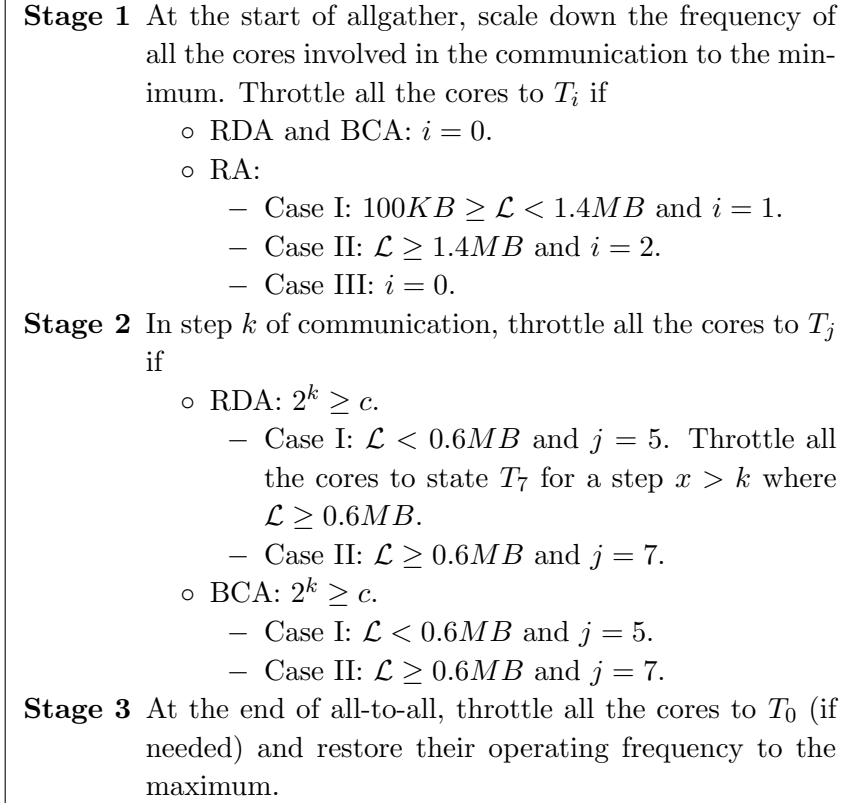


Figure 2.3 Energy saving strategy for the *allgather* operations with the throttling level selection based on the message size (per Table 1.1).

apart exchange their own data along with the data that they have received in the first step. In the third step, the processes that are a distance-four apart exchange their own data and also the data they have received in the previous two steps. Continuing in this manner, for a power of two number of processes p , all the processes receive all the data in $\log_2 p$ steps. For a communication step i ($i \leq \log_2 p$), the message size exchanged by each process is $\mathcal{L} = 2^{i-1}M$. Note that, until the k th step where $2^k < c$, the communication type is intranode for any communicating core while, after this step, the communication becomes purely internode.

Ring Algorithm. The data from each process is sent around a virtual ring of processes. In the first step, each rank i sends its contribution to rank $i + 1$ and receives the contribution from rank $i - 1$ (with wrap-around). From the second step onward each rank i forwards to rank $i + 1$ the data it received from rank $i - 1$ in the previous step. In each of $p - 1$ steps, a message of $\mathcal{L} = M$ size is sent and received by each process.

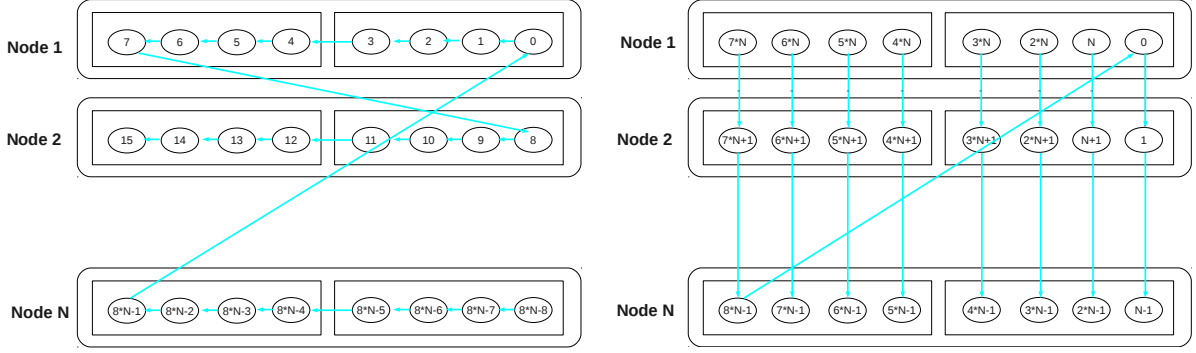


Figure 2.4 The RA communication pattern for the block (left) and cyclic (right) rank placements. Arrows indicate the communication direction.

Energy Saving Strategy. All the algorithms except the RA exhibit internode communication after a certain communication step. Therefore, the energy saving strategy outlined in Fig. 2.3 may be applied to each of them. The RDA and BCA are used for a message size smaller than 1 KB, therefore no throttling is applied for their intranode communication steps. Since the Ring Algorithm does not make purely internode communication at any step (see the left side of Fig. 2.4), the throttling level chosen at its start throughout the collective operation.

Rank Placement Considerations. The block rank placement calls for only two DVFS and up to four throttling switches in the proposed energy saving strategies. Hence, it minimizes the switching overhead. In the cyclic rank placement, however, after a fixed number of steps, the communication would oscillate between intra- and internode, and will require a throttling switch at every such step. Therefore, the block rank placement has been considered for the energy savings application.

In the RA implementation of the allgather operation, the communication is purely intranode for block and internode for cyclic rank placement as shown in Fig. 2.4. Thus, the cyclic rank placement does offer a better opportunity of applying a relatively higher level of throttling. However, it was found experimentally that, for RA, the allgather operation takes much less time to execute for the block rank placement compared with the cyclic one, so the energy saving potential has been foregone here in favor of the overall performance.

CHAPTER 3. Runtime Procedure

In this chapter, a runtime procedure to obtain energy saving in point-to-point communications is discussed which proposes three frequency scaling strategies.

3.1 Runtime Communication Analysis

To apply frequency scaling in point-to-point communications, it is helpful to first categorize them as to the reappearances of rank sequences and message sizes; more generally, to determine the point-to-point *recurring patterns*. Then, by analyzing the obtained recurring patterns, it may be decided whether or not the frequency switching overhead is amortized and thus, whether or not the CPU frequency scaling is warranted. In this section, the ranks recurring during a certain time period are termed *sequence*, which together with its corresponding set of message sizes, is called *phase*. The length of a sequence measured as its number of point-to-point ranks is called the *phase length*. A phase can be uniquely identified by the rank and message size of the call that commences the phase. For each call in the phase, the parameters, such as call duration and time gap between the calls, are recorded to make the frequency scaling decisions.

Once the point-to-point communication phases are determined, they may be efficiently exploited for those applications that are based on iterative computations. Specifically, the frequency scaling is performed in the next iteration containing a given phase. All the chosen test applications exhibit an iterative behavior with point-to-point phases. However, if an application has no iterative communication, the sequence recognition will not be applicable.

The blocking point-to-point communications are selected in this work since reducing the processor frequency during their operation will not slow down the computational portion of the application and will result in a minimal system overhead as has been shown by the au-

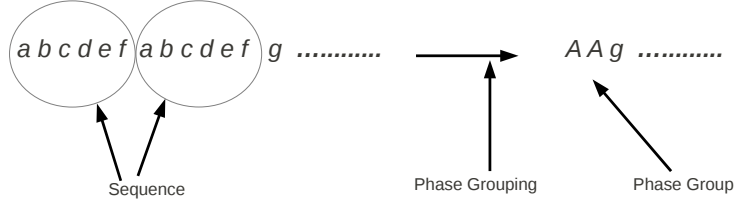


Figure 3.1 Sequence recognition and phase grouping

thors earlier [53]. Specifically, paired communication primitives, MPI_Send and MPI_Recv, are considered here to explore the fundamental design aspects that may be applied to other MPI operations in the future. However, since non-blocking operations do not provide Infini-band CPU offload, this analysis cannot be extended for them. For tapping their energy saving potential, one may be able to employ only performance counters.

A general design of the runtime analysis procedure consists of four major states: *sequence recognition*, *watching*, *recording*, and *application frequency scaling*. Figure 3.2 outlines these four states along with their transitions, which are detailed in sections 3.1.1 to 3.1.4. The idea to use a state machine comes from the work of Freeh *et. al* [17], where a similar one was proposed.

3.1.1 Sequence Recognition

First, the initial recurring pattern of the ranks is to be determined, which may be achieved using a simple string matching algorithm performed during the application execution. Specifically, such an algorithm attempts to find two identical substrings of a certain length, after which a new sequence is declared as starting from the first matching rank and finishing with the rank preceding the start of the second identical substring. Figure 3.1 shows the sequence recognition process along with phase grouping. The point-to-point communication ranks are depicted by lower case letters and their sequences are enclosed into ovals. After recognizing the repeating pattern of the ranks, i.e, the sequence, the phase grouping takes place, which simply records maximal repeating substrings of phases as a group. For example in fig. 3.1, if an upper-case letter is assigned to depict a phase uniquely, then, a substring of the phases,

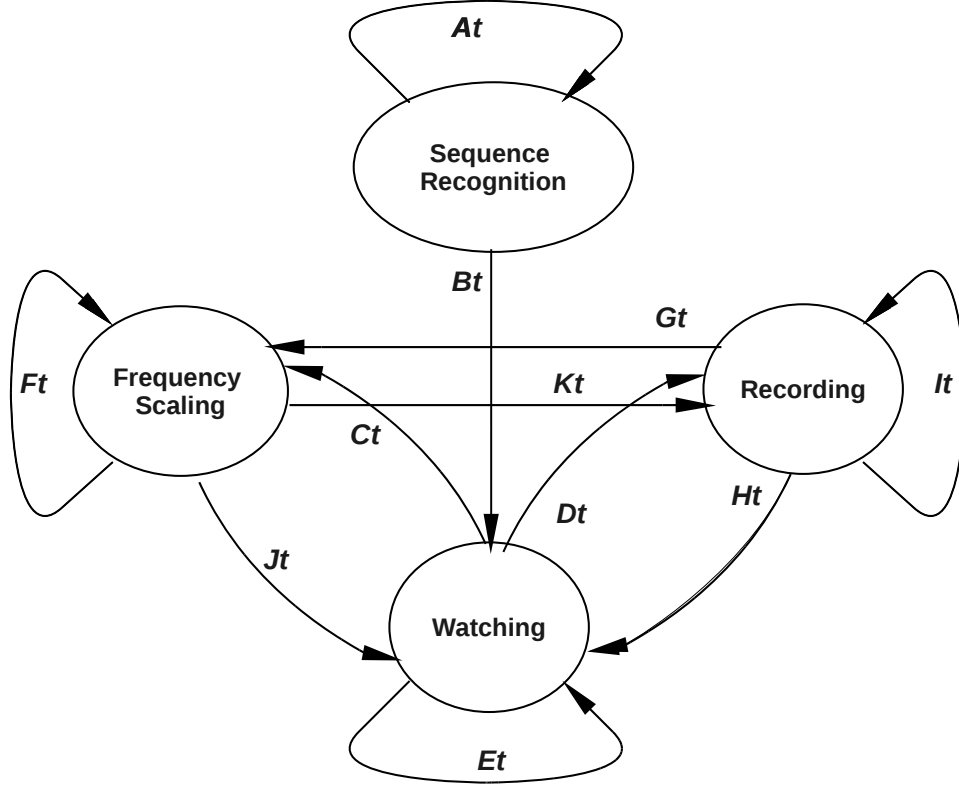


Figure 3.2 State diagram for runtime procedure to apply energy savings efficiently. The transitions are labeled with Lt , where L takes a value of the first 11 letters of the alphabet. The transition of a state into itself (At , Et , Ft , It) indicate ongoing state action.

denoted as AA , forms group. Grouping of phases is important for certain frequency scaling strategies as discussed further in section 3.1.4.4. Alternatively, more sophisticated algorithms, such as the supermaximal repeat string algorithm [21], may be used. However, for simple rank patterns, those with a single maximal repeat string, and a moderate number of communicating processors this is not necessary. Note that a simple rank pattern means here that there is only a single sequence of ranks per each communicating process and that there are no other (out of sequence) calls between the end of one sequence and start of another. The substring length to match may be chosen experimentally, as was done in this work. As soon as the sequence is determined, the state changes to the watching state (transition Bt in fig. 3.2). Since this is the initial sequence, it is deemed to be also the first phase.

3.1.2 Watching

The aim of this state is to monitor call after call and, if certain conditions are met, attribute the current call to either the frequency scaling or the recording states. Specifically, in this state, the rank and the message size of the current call are compared with the information for the *first call* in all the previously recorded phases. Once a match, at least partial, is found the comparison is aborted and a state transition takes place as follows:

- If a complete match exists (i.e., both the rank and message size are matched), then go to the frequency scaling state since an occurrence of an existing phase has been found (*Ct* in fig. 3.2).
- If no complete match exists—only do the communicating ranks match,—then go to the recording state in search for a new phase (*Dt* in fig. 3.2).

Otherwise, the next call is considered in the watching state (*Et* in fig. 3.2).

3.1.3 Recording

If only the rank matches some previously recorded phase but not the message size, then this may indicate the beginning of a new phase. The new phase then starts with the recording of the rank, message size, call duration, and time gap between each pair of the calls. In addition, depending on the chosen frequency scaling strategy (see section 3.1.4), the needed performance-counter values are recorded. Figure 3.3 provides an example of the recorded phases, where a phase is represented by calls *a*, *b*, *c*, and *d*. The time gap between successive phases is termed *interphase gap*, while a time gap within a phase is denoted as *intrapphase*.

Simultaneously, the rank of the current call is checked against the corresponding rank within the initial sequence. If there is a mismatch, the recording of the new phase is aborted with two resulting state transitions:

- frequency scaling, if *both* the rank and message size of the current call are equal to those of the first call in some previously recorded phase (*Gt* in fig. 3.2).
- watching, otherwise (*Ht* in fig. 3.2).

If only the rank matches, then the recording state is re-entered for the next new phase recording

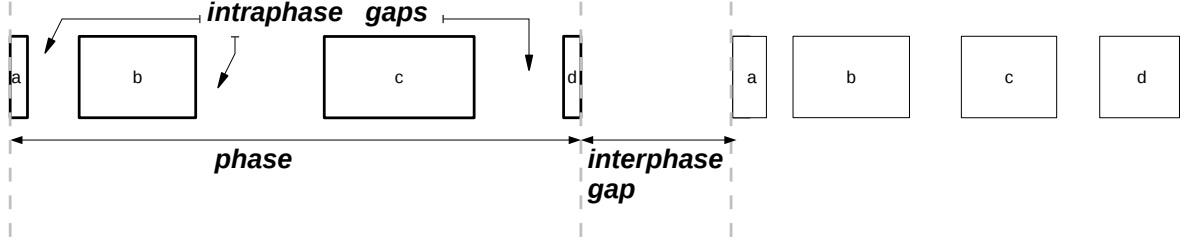


Figure 3.3 Trace of an MPI application invoking eight MPI_Send and MPI_Recv calls with a phase length of four. (The calls within a phase are ordered lexicographically from *a* to *d*.)

(*It* in fig. 3.2). Once a new phase is recorded, the transition to the watching state happens (*Ht* in fig. 3.2).

3.1.4 Frequency Scaling

In the *frequency scaling* state, the calls are continuously checked whether or not their communicating ranks match the *initial* sequence. If there is a mismatch, frequency of the processor is restored to its highest value f^{\max} and the call causing the mismatch either remains in the current state (*Ft* in fig. 3.2), if both its rank and message size match the beginning of any already recorded phase, or enters the recording state (*Kt* in fig. 3.2), if only its rank matches. Otherwise, if both rank and the message size do not match, a transition to the watching state occurs (*Jt* in fig. 3.2).

The frequency scaling operation is performed such that frequency is reduced at the beginning of the phase and generally raised in the end of the phase. Three different strategies are discussed in the rest of section 3.1.4 for applying frequency scaling to the intra- and interphase time gaps.

3.1.4.1 Conservative Strategy

This strategy is applied if the interphase time gaps are conservatively assumed to be entirely compute-intensive, so their durations would scale linearly with the decrease in frequency. A performance loss γ must be defined (possibly by the user) to constrain the amount of the performance degradation that is tolerated when the CPU frequency is scaled down. In [58], an appropriate frequency is selected for a single (collective) communication call by taking into

account just the message size while exploiting the Infiniband CPU offload feature and certain communication characteristics. In the case of a multicall phase, the gaps between call pairs must be examined and evaluated as to their performance losses at a lower frequency.

Consider the execution at the highest frequency f^{\max} . Then, let $T_{call}(f^{\max})$, $T'(f^{\max})$, and T^{\max} denote the total duration of the communication calls, of the intraphase time gaps, and of the entire phase, respectively, at the highest frequency, such that

$$T^{\max} = T_{call}(f^{\max}) + T'(f^{\max}) . \quad (3.1)$$

Let f_{γ}^* be a suitable frequency (as made available by a P-state) when the performance loss is γ ; $O_{call}(f_{\gamma}^*)$ and $O_s(f_{\gamma}^*)$ be the communication call and frequency switching overheads, respectively, when the frequency is changed from f_{γ}^* to f^{\max} . Then the desired available frequency f_{γ}^* is determined as

$$f_{\gamma}^* = \left\lceil \frac{T'(f^{\max}) \times f^{\max}}{\gamma \times T^{\max} - O_{call}(f_{\gamma}^*) - O_s(f_{\gamma}^*)} \right\rceil , \quad (3.2)$$

where the ceiling operation is needed because the CPU provides only a number of P-states, among which the closest upper bound will be selected. This algorithm will work irrespectively of the specific number of P-states. The communication call overhead $O_{call}(f_{\gamma}^*)$ varies with the message size and the inter- and intranode communication type, and its value may be taken as suggested in [53]. The frequency switching overhead $O_s(f_{\gamma}^*)$ may be precomputed in advance for all the P-states in a given platform.

3.1.4.2 Intermediate Strategy

Since the intraphase time gaps may have architectural (resource-related) stalls, such as memory or I/O, an assumption of their high computational intensity may be too conservative in achieving maximum energy savings. Thus, a quantitative analysis is desirable to estimate the CPU usage between communication calls within a phase (i.e., during the intraphase time gaps). To this end, performance counters may be employed to measure the architectural stalls in modern processors. However, the number and applicability of these counters are typically limited. For example, the Intel Xeon E5450 processor, which is used in this work, provides only two general-purpose performance counters, and thus, hinders the construction of sophis-

ticated models, such as the one proposed in [23], which relies on four performance counters. Nevertheless, fewer counters are still useful if the frequency calculation is distilled to the most critical parameters that the available performance counters can measure. In particular, this work proposes to count both the micro-operations retired $\mu\tau$ and memory accesses m mainly because the operation rate may be higher for memory-intensive applications than for compute-intensive applications, which is counterintuitive without considering the memory accesses. By using an *a priori* analysis, the number of micro-operations retired $\mu\tau(f^*)$ at an available CPU frequency f^* $\mu\tau(f^*)$ has been predicted based on both $\mu\tau(f^{\max})$ and $m(f^{\max})$ [31] through the following relation:

$$\mu\tau(f^*) = \frac{f^* \times \mu\tau(f^{\max})}{f^{\max}} + b \times m(f^{\max}) , \quad (3.3)$$

where the b is dependent on the number of memory accesses per second and was determined experimentally. Then, for the intraphase time gap t'_i ($i = 0, \dots, n-1$, where n is the number of calls in a phase), $\mu\tau_i(f^*)$ may be determined and the corresponding performance loss γ_i for $f^* < f^{\max}$ calculated as

$$\gamma_i \approx \frac{\mu\tau_i(f^{\max}) - \mu\tau_i(f^*)}{\mu\tau_i(f^{\max})} \approx \frac{t'_i(f^*) - t'_i(f^{\max})}{t'_i(f^*)} . \quad (3.4)$$

Since the total overhead for executing the intraphase time gaps at a frequency f^* is $O'(f^*) = T'(f^*) - T'(f^{\max})$ and by considering eq. (3.1) the suitable frequency f_γ^* may be calculated from

$$\gamma \geq \frac{O'(f_\gamma^*) + O_{call}(f_\gamma^*) + O_s(f_\gamma^*)}{T^{\max}} . \quad (3.5)$$

3.1.4.3 Shortcomings of Conservative and Intermediate Strategies

The conservative and intermediate strategies do not take into account application of frequency scaling in the interphase time gaps, which they assume to be computationally intensive. Additionally, they derive frequencies on a per-core basis. In practice, however, the DVFS is applied only in pairs of cores, i.e., in any two cores sharing an L2 cache, termed “twin cores” in

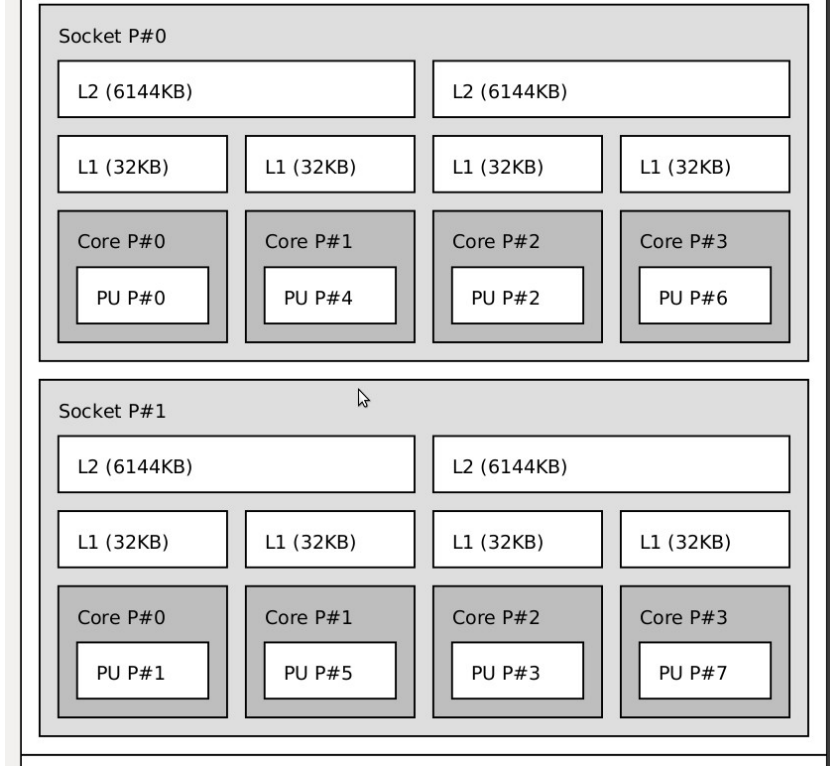


Figure 3.4 An Intel Xeon E5450 processor within one socket, having four physical cores. A physical core has an L1 cache and a shared L2 cache, such that pairs of core processing units (PUs)—(P#0, P#4), (P#2, P#6)—each share an L2 cache.

authors' previous work [54]. DVFS supports only a twin-core granularity on certain processors, meaning that it produces energy savings only when any two cores sharing the same L2 cache (as in fig. 3.4) are scaled to the same P-state. This was expected since the off-chip DVFS regulators are typical in current microprocessors [30]. Since cores are grouped around the L2 cache for the effectiveness of DVFS, the conservative and intermediate strategies are expected to work similarly in the case of three cache levels.

For the point-to-point communications, it is important to detect communication phases sufficiently long to be able to apply the dynamic frequency scaling without much overhead and to first determine a communication *phase overlap* ϕ_o . Figure 3.5 illustrates how ϕ_o is found for twin-cores 0 and 4. On core 0, the phase starts at time t_1 and ends at t_2 . Similarly for core 4, the phase starting and ending times are t_3 and t_4 , respectively. Then, $\phi_o = (t_2 - t_3)/(t_4 - t_1)$. Only if the value of this phase overlap is reasonably large does DVFS provide substantial energy savings for either conservative or intermediate strategies.

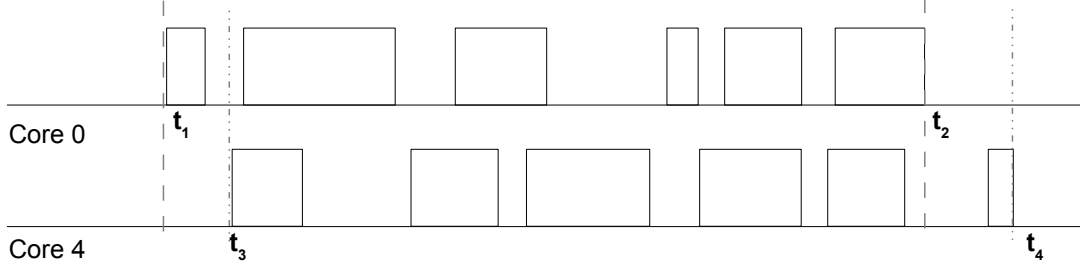


Figure 3.5 Overlapping of communication phases in a pair twin-cores that share the L2 cache. The phase boundaries are marked as vertical dashed and dash-dotted for core 0 and core 4, respectively.

Additionally, when the phase boundaries are assumed, as in [35], to simply be not sensitive to context—rather than compute intensive—meaning that their detection may not be known from the previous iterations, the frequency scaling is applied only for the MPI communication phases. For a single-core processor, such an approach may provide substantial energy savings. In a multicore system, however, if the application communications exhibit very low phase overlap, it may not save any energy, similarly to the conservative or intermediate strategy when used without the phase overlap consideration in twin cores.

3.1.4.4 Aggressive Strategy

To overcome the often tedious provisioning for the phase overlap in multicore platforms, an aggressive strategy is proposed here, so that it targets interphase time gaps for frequency reduction and extends to them the performance-counter usage of the intermediate strategy. Specifically, it calculates the micro-operations retires and memory accesses in the interphase gaps and applies to them an appropriate frequency f_γ^* in a manner similar to that of the intermediate strategy. Then, the performance loss is calculated as in eq. (3.4).

Once a phase ends, the current frequency f^c is compared with f_γ^* and is left unchanged if they are the same. Otherwise, f^c takes the value of f_γ^* if, in addition, the switching overhead $O_s(f_\gamma^*)$ is less than the performance overhead $O''(f^c)$ from executing the interphase gap at the current frequency f^c . By considering the switching and performance overheads during the



Figure 3.6 Grouping of the communication phases detected for rank 0 of the MG NAS benchmark. The capital letters followed by the double dots represent communication phases followed by interphase gaps, respectively. A single group (shown as solid oval) has been found and the two corresponding subsequences are enclosed into the ovals denoting this group.

interphase time gaps, the aggressive strategy avoids unnecessary frequency scaling switches that the other two strategies incur when a mandatory frequency scaling is applied at the phase boundaries.

To account for the cases when the interphase gap is too long or when no other communication phase is observed further in the execution, an additional two-step guard has been incorporated into the aggressive strategy:

Step 1. Record the current interphase gap duration t_j'' .

Step 2. If $j > 0$ and $t_j'' \geq K \times \max_{0 \leq k < j} t_k''$, then restore f^{\max} .

The value of K (in **Step 2**) may be chosen as 2 based on the following two general assumptions. (1) All the interphase gaps are of similar lengths and (2) the length of an interphase gap is comparable to that of a phase. With this guard of the aggressive strategy, potentially compute-intensive tasks are not executed at a lower frequency, and thus, experience no significant performance degradation.

Since the runtime phase characterization procedure processes phases sequentially in a communicating processor, it should apply the frequency scaling to the interphase gap *before* the next phase is detected. Thus, the procedure may have difficulties in recognizing properly this gap, which, by definition, is uniquely identified by the end *and* the start of two adjacent phases. In particular, such a situation occurs when the two phases are different or for complicated rank sequence patterns. For the communications of a repeating nature, however, this difficulty may be alleviated by grouping communication phases in the overall phase sequence. In a sense, each group is akin to a macrophase, as defined in [24]. As an example, fig. 3.6 depicts such a

grouping of the phases detected for rank 0 of the MG NAS benchmark. Observe that there are recurring subsequences of phases, which may be grouped. (In fig. 3.6, the two subsequences are enclosed into identical ovals to emphasize that they belong to the same group.). Once such a grouping is achieved, the interphase gap recognition becomes the matter of determining to which phase group it belongs. Note that the runtime procedure from section 3.1.1 is extended to find the phase grouping in a manner similar to how the initial sequence was determined.

CHAPTER 4. GAMESS

In this chapter, the energy characteristics and the application of runtime system proposed in Chapter 3 to the quantum chemistry application GAMESS is discussed.

4.1 Overview of Quantum Chemistry Package GAMESS

GAMESS [47] is one of the most representative quantum chemistry applications used worldwide to do *ab-initio* electronic structure calculations. GAMESS iteratively approximates the solution to the Schrödinger equation in the form of the Self Consistent Field (SCF) method followed by higher levels of theory, such as Density Functional and Many Body Perturbation. Although GAMESS may be considered as a part of SPECCPU 2006 benchmark suite, studying GAMESS as a stand-alone package yields itself to an investigation of a rich spectrum of quantum chemistry methods and their execution modes. The SCF method is implemented in two forms, namely *direct* and *conventional*, which differ in the handling of the two-electron (2-e) integrals. Specifically, the conventional mode calculates them once at the beginning of the the SCF and stores them on disk for subsequent reuse, whereas the direct mode recalculates the 2-e integrals for each iteration. After the SCF, the gradient and Hessian of the energy may be calculated. In this work, the power and energy characteristics of these two implementations are investigated for a set of molecules.

Data-Server Communications The parallel model used in GAMESS evolves constantly based on the HPC hardware and software advances. Initially, it was based on replicated-data message passing and later moved to MPI-1. Fletcher *et al.* [16] developed the Distributed Data Interface (DDI) in 1999, which has been the parallel communication interface for GAMESS ever since. Later, the DDI has been adapted to symmetric-multiprocessor environments fea-

turing shared memory communications within a node [40], and was generalized in [14] to form groups out of the available nodes and schedule tasks to these groups. DDI ensures that all the processes independently access and modify any element in a logically global but possibly physically distributed data array. In essence, DDI implements the PGAS programming model by employing a data-server concept. Specifically, an additional process, called *data server*, is created for each compute process of GAMESS. While the compute process performs electronic structure calculations, the data server services requests for the data associated with the distributed arrays. Although these requests do not have a structured pattern of array locations to access (i.e., they are irregular), they are, nevertheless, repeating in nature meaning that the same locations accessed in the initial iterations of the quantum chemistry methods, such as SCF-HF, will be accessed again in the subsequent iterations. Thus, the frequency scaling strategies considered here may be applied to GAMESS.

Depending on the installation options, communications between the compute and data server processes occur either via TCP/IP or MPI. A data server responds to data requests initiated by the corresponding compute process, for which it constantly waits. If this waiting is implemented with MPI, then the CPU is polled continuously for the incoming message, thereby being always busy. Therefore, in multicore platforms, it is preferred that a compute process and data server do not share a core to avoid significant performance degradation. This is not a taxing restriction in typical HPC environments with hundreds of cores. Since TCP/IP is hardly ever used for HPC and since this work focuses on MPI communications, the DDI over MPI is considered, such that only one GAMESS process is mapped to a core. Now, assume a multicore platform having $2N$ cores. Then, each core c_i ($i = 0, \dots, N - 1$) with a compute process has a corresponding core d_j ($j = N, \dots, 2N - 1$) with a data server, such that $j = i + N$.

4.2 GAMESS Energy Characteristics

A set of molecules (Table 4.1), is used as inputs to determine the power consumption characteristics of the two SCF implementations. The molecules (column **Molecule**) are listed in the increasing order of their I/O requirements (column **I/O**), as specified in their input files, for the conventional mode. For the experiments, the Ames Lab cluster called “Borges” was

Table 4.1 Input set of molecules.

Molecule	I/O,(GB)
Silatrane	1.0
Luciferin	3.8
Amg221	5.9
cAMP	7.5
Saxitoxin	9.8
Qinghaousu	11.1
Quinine	13.0
Rotenone	17.1
Ergosterol	22.7

used. It consists of four nodes, each having two dual-core 2 GHz Xeon “Woodcrest” CPUs and 8 GB of RAM. The nodes are interconnected with both Gigabit Ethernet and DDR Infiniband. Each processor has a shared 4 MB L2 cache and a 32 KB L1 instruction and data cache per core. Another computing platform (denoted “FScal”) comprises two Dell Optiplex 960 nodes, each of which has an Intel core 2 Duo processor with 2 GB of RAM. This platform was chosen since it allows the CPU frequency scaling for the DVFS optimization. To measure the system power and energy consumption in either platform, a Wattsup power meter [1] was employed.

The power consumption characteristics of GAMESS are explored for a set of input molecules run on Borges. In each node of the four nodes, four processes were executed (denoted as the “4x4” execution configuration), whereas the notation “4x1” stands for executing one process in each node.

4.2.1 The 4x4 Execution Configuration

Fig. 4.1 depicts the execution time, average power consumption, and energy consumption observed for the various input molecules in the 4x4 configuration.

It can be seen in Fig. 4.1(a) that, when the I/O requirement is rather low (see Table 4.1), as in the case of Silatrane, the conventional mode performs better than the direct one. However, as the I/O requirement increases, the direct mode begins to outperform greatly the conventional.

The situation is exacerbated by a slow I/O rate of around 107 MB/s on Borges. For instance, in another computer with the I/O rate of approximately 226MB/s, the execution time for the

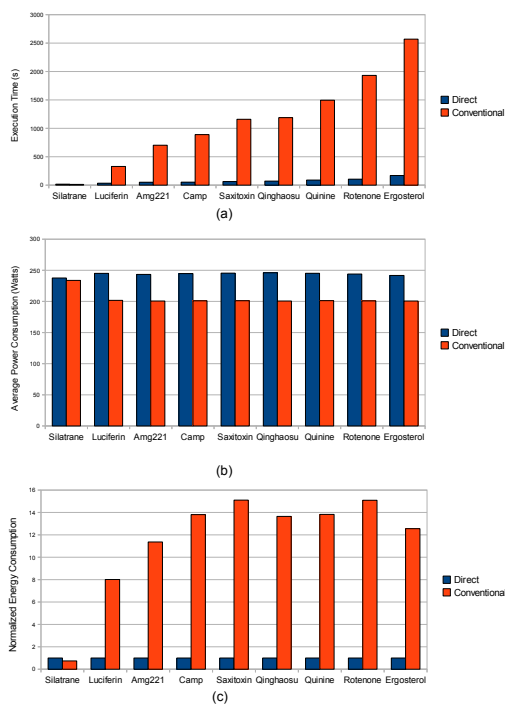


Figure 4.1 4x4 configuration: (a) Execution time, (b) Average power consumption, and (c) Energy consumption normalized with respect to the direct mode for the input molecules shown on the x axes in the ascending order of their I/O requirements.

conventional mode was almost halved. In spite of the direct mode reducing execution time under the high I/O requirements, conventional mode is more preferable from the computational accuracy point of view and may lead to faster method convergence. Fig. 4.1(b) depicts the average power consumption for both conventional and direct modes. It can be observed here that the average power consumption of the conventional mode is less than that of direct. This is so because the conventional mode, being I/O intensive, consumes less power as compared with the direct mode, which is more PE and memory intensive. Moreover, as discussed, the PE goes into the idle state quite frequently for the conventional mode, and thus lowering the power consumption. On average, the conventional mode consumes 16% less power as compared with the direct one.

Fig. 4.1(c) shows that, except for Silatrane, direct mode is more energy efficient than that of conventional in spite of its power consumption being higher. This is mainly due to the fact that the conventional mode suffers from I/O stalls and thus, takes much longer to execute. To summarize, the conventional mode is seemingly more power efficient whereas the direct mode is more energy efficient.

4.2.2 The 4x1 Execution Configuration

Transition to the 4x1 configuration increases the execution time, shown in Fig. 4.2(a), of the direct mode as compared to the 4x4 direct mode since, being more PE intensive, this mode takes longer to execute on a reduced number of cores. On the other hand, the execution time of the conventional mode decreases in this configuration as compared to the 4x4 configuration. This can be attributed to the fact that when only one process is executing on a single node, the I/O contention is also less and, hence, the execution time is reduced. However, for Ergosterol, the conventional 4x1 execution time is greater than that of the 4x4 configuration since the iteration phase is computationally intensive and thus, takes longer to execute on fewer cores.

The 4x1 average power consumption (Fig. 4.2(b)) of the conventional mode is almost the same as that for the 4x4 configuration but is lower for Silatrane, which has low I/O requirements. For the direct mode in the 4x1 configuration, the average power is reduced noticeably compared with that in the 4x4 configuration because only one core is active in each node.

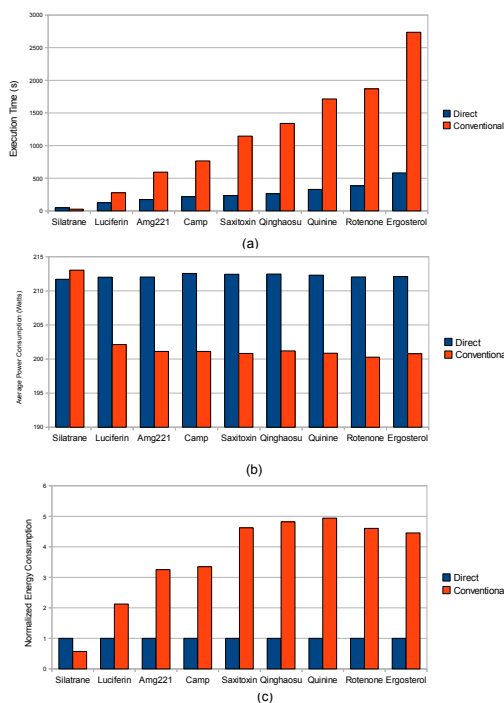


Figure 4.2 4x1 configuration: (a) Execution time, (b) Average power consumption, and (c) Energy consumption normalized with respect to the direct mode for the input molecules shown on the x axes in the ascending order of their I/O requirements.

The energy consumption of the direct mode remains less than that of conventional, as seen in Fig. 4.2(c). However, the gap between them is reduced because the direct mode consumes more energy—due to a longer execution—and the conventional consumes less as compared to 4x4 configuration.

4.2.3 Power profile of Self Consistent Field Phases

For the conventional mode, the *Calculation* phase is where the 2-e integrals are calculated and stored on the disk. The *Iteration* phase represents the SCF iterations and *Gradient* is where the gradient of the energy is computed. Since the direct mode recomputes integrals for each iteration, it only consists of the *Iteration* and *Gradient* phases. Fig. 4.3 shows the power profiles of the phases across a node during the entire execution time in the 4x4 configuration.

Fig. 4.3(a) and Fig. 4.3(b) depict the power profile of Luciferin. It can be observed that the conventional mode for Luciferin has a very smooth power curve as conventional mode is quite

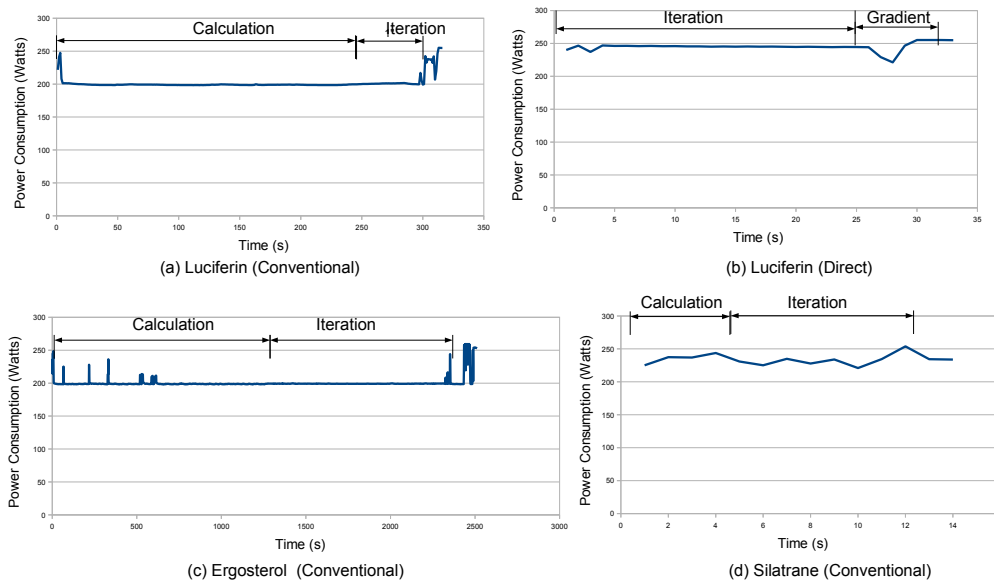


Figure 4.3 Power profiles for some molecules in the 4x4 configuration.

I/O intensive. A molecule executing in the conventional mode suffers from the PE stalls. As a result, PE power consumption is quite low [19] and PE often goes into the idle state. Therefore, the power consumption of the whole CN is reduced considerably. For the *Gradient* phase, the power consumption increases since it is PE intensive. On the other hand, the power curve for the direct mode has spikes and valleys with power varying between 221 and 255 watts. The PE is idle most of the time during the *Calculation* phase in the conventional mode. Therefore, the average power consumptions of PE and CN conventional modes are quite low comparing with those of the direct mode.

Fig. 4.3(c) shows the power profile for only the Ergosterol conventional mode. As the I/O requirements of Ergosterol are quite high, the PE spends more time idling as compared with the molecules having lower I/O requirements. Conversely, for Silatrane (Fig. 4.3(d)), the I/O requirements are low, and the power consumption varies from 225 to 253 watts.

Fig. 4.4 depicts the power profiles for the molecules executed in the 4x1 configuration. Since the three cores remain idle, the *peak* power consumption is considerably less than that in the 4x4 configuration throughout the experiments. The overall reduction is seen in the direct mode (Fig. 4.4(b)) which is compute-intensive and suffers when fewer processes are employed. For the

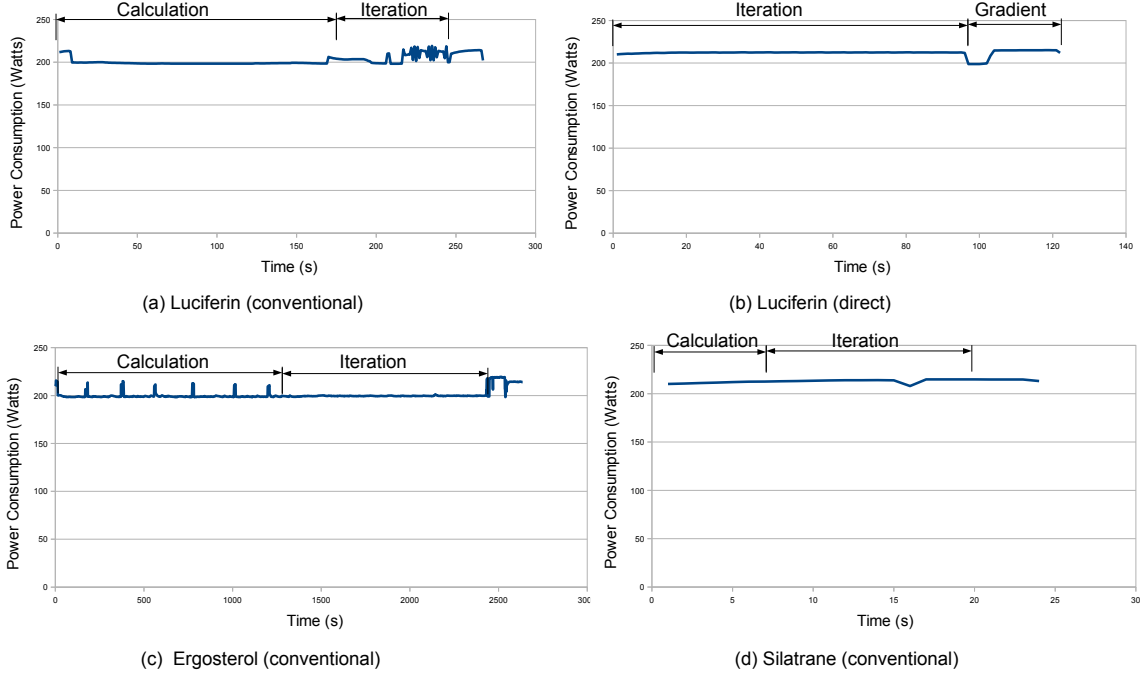


Figure 4.4 Power profiles for some molecules in the 4x1 configuration.

conventional mode to which the molecules with high I/O requirements are input (Fig. 4.4(a) and Fig. 4.4(c)), there is not much difference in the overall power consumption. On the other hand, for Silatrine with low I/O, the power consumption is also reduced for the conventional mode as seen in Fig. 4.4(d) comparing with Fig. 4.3(d).

4.2.4 Energy Consumption Model

The execution time of a program can be divided into two separate parts, on-chip time t_{on} and off-chip time t_{off} , such that t_{on} and t_{off} are non-overlapping [10]. The time t_{off} consists of stall cycles, such as memory, I/O, branch misprediction, and reservation station stalls, during which the PE is not doing any useful work. In an out-of-order processor, the stall cycles can also overlap with the on-chip execution. DVFS affects only t_{on} of the program execution. For example, if the execution time of a program at the highest frequency f_1 is $t_1 = t_{on} + t_{off}$, then, on a frequency f_i ($f_1 > f_i$), the execution time would be

$$t_i = t_{on}(f_1/f_i) + t_{off} . \quad (4.1)$$

Typically, during a DVFS-based optimization, a performance loss tolerance is prescribed by the user for a given application, and the energy savings are maximized under this tolerance. Much research focused on applying such an optimization to the PE-only energy savings. However, an optimization resulting in some PE energy savings may actually have a higher overall energy consumption for the whole CN.

Let a DVFS-based optimization increase t_{on} by a factor of k , so that $t' = kt_{on} + t_{off}$. The total energy saving may appear if

$$P_1 t > \bar{P} t' , \quad (4.2)$$

$$P_1(t_{on} + t_{off}) > \bar{P}(kt_{on} + t_{off}) , \quad (4.3)$$

where P_1 is the average power consumption of the CN at the highest frequency and \bar{P} is the average power consumption when DVFS is applied. The inequality (4.3) may be used to determine the feasibility of total energy saving, being re-written for convenience, as

$$\frac{t_{off}}{t_{on}} > \frac{k\bar{P} - P_1}{P_1 - \bar{P}} . \quad (4.4)$$

Specifically, Fig. 4.5 depicts the average power consumption of the input molecules executed in the direct mode on the FScal platform on four available frequencies. Fig. 4.6 shows the ratio t_{off}/t_{on} (denote it as τ) of the off-chip and on-chip computation times for the input molecules normalized with respect to τ of Quinine. It can be observed that the average power consumption varies inversely with τ , which is understandable. As the off-chip accesses increase, the power consumption goes down. Therefore, the average power consumption P_i at a frequency f_i may be written as

$$P_i = a\tau + b , \quad (4.5)$$

where a and b are some constants, which may be determined using a regression analysis of (4.5). Their values are shown in Table 4.2 along with the correlation coefficient R^2 , which is close to unity. Thus, there is a strong correlation between the ratio τ and the average power consumption at a given frequency.

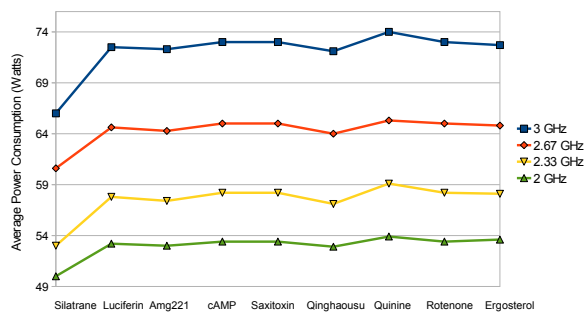


Figure 4.5 Average power consumption of input molecules for direct mode on the four frequencies of FScal platform.

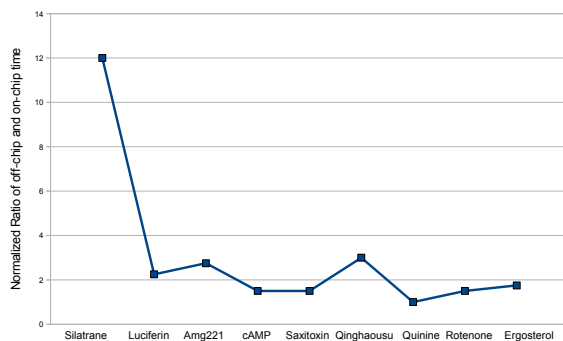


Figure 4.6 Computation times ratio τ for input molecules normalized with τ for Quinine (FScal platform).

Table 4.2 Regression coefficients for determining the average power consumption at a given frequency.

Frequency	b	a	R^2
3.00	74.4	-17.575	0.983
2.67	66	-10.76	0.99
2.33	59	-12.722	0.981
2.00	54	-8.387	0.994

The PE power consumption consists of two components, static and dynamic. The static power is the power consumption of a PE in the idle state. The dynamic power is directly proportional to the product of the operating frequency and the square of the core voltage [41]. In (4.4), the power consumptions P_1 and \bar{P} can be replaced by the respective PE power consumption values [41] to obtain the feasibility of energy savings for the PE under a DVFS-based optimization.

Since the DVFS affects only the on-chip time, an upper bound of energy savings for a DVFS-based optimization, can be determined. For the maximum energy savings, t_{off} should be executed at the lowest frequency. Therefore, the minimum energy consumption E_{off}^* during the off-chip execution is

$$E_{off}^* = P_n t_{off} , \quad (4.6)$$

where P_n is the average CN power consumption at the lowest frequency f_n . The performance loss tolerance δ can be defined as

$$\delta = \frac{t' - t}{t} = \frac{(k - 1)t_{on}}{t_{on} + t_{off}} , \quad (4.7)$$

$$k = 1 + \delta \left(1 + \frac{t_{off}}{t_{on}} \right) . \quad (4.8)$$

For a user-defined performance loss δ , the execution time t_{on} minimizing the energy consumption on the different frequencies may be determined using linear programming (LP). Let P_1, P_2, \dots, P_n be the average power consumptions of the CN or PE and t_1, t_2, \dots, t_n be the on-chip execution times at frequencies f_1, f_2, \dots, f_n , respectively, where $f_1 > f_2 > \dots > f_n$. By defining f_{ij} as the *scaling factor*, which is the ratio f_i/f_j , the LP problem may be formulated as

$$\min_{t_i} E_{on} = \sum_{i=1}^n P_i f_{1i} t_i, \quad (4.9)$$

$$\text{such that } \sum_{i=1}^n t_i = t_{on}, \quad (4.10)$$

$$\sum_{i=1}^n t_i f_{1i} = t_{on}(1 + \delta'), \quad (4.11)$$

$$t_i \geq 0, \quad (4.12)$$

where

$$\delta' = \delta(1 + \frac{t_{off}}{t_{on}}). \quad (4.13)$$

The objective function E_{on} describes the minimum energy consumption of the CN (or PE) during the on-chip work when the DVFS is applied. Note that, during the execution time t_i on the frequency f_i , DVFS changes the average power consumption from P_1 to $P_i f_{1i}$. The formulation (4.9)–(4.12) has n variables and two equality constraints and may be solved by using the two-phase Simplex method [60]. In this method, artificial variables are added to the constraints of the original LP and the solution is obtained in two phases in the form of basic and non basic variables. The non basic variables, forming the non basis set, are equal to zero and the basic variables, forming the basis set, provide the optimal solution. For the FScal platform, there are four frequency levels for which the LP (4.9)–(4.12) is to be solved here.

Case I: If $0 < 1 + \delta' \leq f_{12}$, the basis set $B = (1, 2)$, the optimal solution is obtained when

$$t_1 = t_{on} - \frac{t_{on}\delta'}{f_{12} - 1} \text{ and } t_2 = \frac{t_{on}\delta'}{f_{12} - 1}.$$

Case II: If $f_{12} < 1 + \delta' < f_{14}$ and $B = (2, 4)$, the optimal solution is obtained when

$$t_2 = \frac{t_{on}(f_{14} - 1 - \delta')}{f_{14} - f_{12}} \text{ and } t_4 = \frac{t_{on}(1 + \delta' - f_{12})}{f_{14} - f_{12}}.$$

Case III: If $f_{14} = 1 + \delta'$, and $B = (4)$, the optimal solution is obtained when

$$t_4 = t_{on}.$$

Once the minimum E_{on}^* is calculated in each of the three cases, the minimum power consumption P^* may be expressed as

$$P^* = \frac{E_{on}^* + E_{off}^*}{t_{on}(1 + \delta') + t_{off}} \quad (4.14)$$

for each of the three cases considered. This value of P^* can be used in (4.4) in place of \bar{P} to ascertain the feasibility of energy savings for a DVFS-based optimization since P^* provides a lower bound on \bar{P} (i.e., $\bar{P} \geq P^*$).

4.2.5 Model Verification

Table 4.3 shows the PE and CN power consumptions at different frequencies for Saxitoxin direct mode execution. The PE power consumption is determined by using the analytical model proposed in [41], in which authors have accurately determined the static and dynamic power consumption of an Intel Core 2 duo CPU. CN power consumption is calculated by using (4.3). Using Table 4.3 and the E_{on} expression (4.9), it may be inferred that, for $i > j$, $P_i f_{1i} > P_j f_{1j}$ in the case of the CN power consumption and $P_i f_{1i} < P_j f_{1j}$ in that for PE. The minimum E_{on}^* is suited for both the CN and PE energy consumption minimizations. However, the CN energy consumption increases with the increase in the performance loss tolerance while the PE energy consumption decreases at the same time.

To determine the times t_{off} and t_{on} for the input molecules, a regression analysis was done on the equation

$$t_i = t_{on} f_{1i} + t_{off}, \quad (4.15)$$

where t_i is the execution time on frequency f_i , $i = 1, 2, \dots, n$ and t_{on} and t_{off} are constants. The correlation coefficient R^2 ranged from 0.998 to 0.9996 for this regression analysis. Table 4.4 lists t_{off} and t_{on} for the set of input molecules executed in the direct mode on FScal. From

Table 4.3 CN and PE power for Saxitoxin direct mode.

Frequency (GHz)	CN Power (W)	Voltage per core (V)	PE Power (W)
3.00	73.4	1.23	52
2.67	65.4	1.20	43
2.33	58.2	1.16	36
2.00	53.4	1.12	28

Table 4.4 On-chip and off-chip times for input molecules in the FScal platform.

Molecule	t_{on} (s)	t_{off} (s)	τ
Silatrane	19.2	9.2	0.48
Luciferin	74.2	6.6	0.09
Amg221	83.2	9.5	0.11
cAMP	110.0	6.9	0.06
Saxitoxin	119.0	7.0	0.06
Qinghaousu	128.8	15.1	0.12
Quinine	199.6	7.8	0.04
Rotenone	230.4	12.9	0.06
Ergosterol	283.4	20.4	0.07

the ratio $\tau=t_{off}/t_{on}$, it can be seen that except for Silatrane, direct mode is quite compute intensive. In fact, its execution time scales almost linearly with the change in frequency.

Table 4.5 depicts the CN energy consumptions of all the input molecules on the three lower frequencies (columns f_2 , f_3 , f_4) obtained experimentally in the FScal platform. The energy consumption values are normalized with respect to the energy consumption at the highest frequency f_1 . It can be seen that the variation in energy consumption is non-uniform. The theoretical model proposed in Section 4.2.4 may be used to explain this phenomenon.

Fig. 4.7 provides an example of the variations in the PE and CN energy consumption at different frequencies for Saxitoxin executed in the direct mode. The values on y axis are normalized with respect to the highest frequency operating point for PE and CN energy, respectively. From the PE energy consumption, it is clear that, as the frequency increases, the energy consumption of PE also increases. This can be verified by putting the values of P_1 and \bar{P} from Table 4.3 into inequality (4.4) for the respective frequencies.

From Table 4.4, τ ratio of Saxitoxin equals 0.06. To determine the feasibility of the PE

Table 4.5 CN energy consumption in the direct mode on three lower frequencies for the input molecules in the FScal platform, normalized with respect to the highest frequency. Frequencies (in GHz) are $f_1 = 3.0$, $f_2 = 2.67$, $f_3 = 2.33$, and $f_4 = 2.0$.

Molecule	f_2	f_3	f_4
Silatrane	0.97	0.98	1.01
Luciferin	0.99	1.04	1.09
Amg221	0.99	1.01	1.08
cAMP	0.99	1.04	1.10
Saxitoxin	0.99	1.01	1.03
Qinghaousu	0.98	1.03	1.08
Quinine	1.01	1.04	1.10
Rotenone	1.01	1.04	1.09
Ergosterol	1.01	1.04	1.08

Table 4.6 Data substituted into the theoretical model to determine the feasibility of the CN energy consumption for different frequencies.

Frequency	rhs	k	\bar{P}
2.00	0.335	1.5	65.4
2.33	0.1	1.288	58.2
2.67	0.01	1.1236	53.4

energy savings at some frequency, say 2 GHz, the appropriate values may be substituted into inequality (4.4), such that $P_1 = 52$, $\bar{P} = 28$ (as provided in Table 4.3), and $k = 3/2$. Then, the right-hand side of (4.4) is equal to -0.714 which is smaller than 0.06, and thus, the PE energy saving is obtained for 2 GHz, as seen in Fig. 4.7. The same calculation may be performed for other frequencies leading to the conclusion that, for Saxitoxin, the PE energy consumption decreases as the frequency is decreased.

In contrast, the CN energy consumption exhibits non-monotonic behavior for Saxitoxin. It first decreases at 2.67 GHz followed by an increase at lower frequencies. Such a behavior is supported theoretically. In particular, the right-hand side of (4.4) was calculated for the three lower frequencies, such that their respective \bar{P} values were taken from Table 4.3. The obtained right-hand sides (column **rhs**) as well as the corresponding frequencies (column **Frequency**), average power \bar{P} , and time increase factors k are presented in Table 4.6 while P_1 is fixed at 73.4 for 3 GHz. As shown in Table 4.6, the condition of CN energy saving is met only at 2.67 GHz, when **rhs** is 0.01. At the other two operating points, 2.33 GHz and 2 GHz, the energy

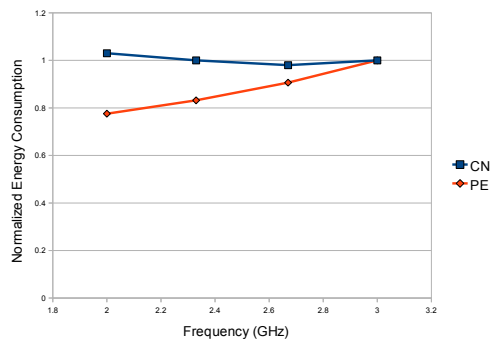


Figure 4.7 Normalized energy consumption of PE and CN for Saxitoxin in direct mode.

consumption of the CN increases.

Fig. 4.8 plots the optimum energy consumption E_{on}^* of Saxitoxin (direct mode), as calculated using the model from Section 4.2.4, for PE and CN versus the performance loss. All the y axis values are normalized with respect to the energy consumption at the highest frequency. The energy consumption for the CN increases with the performance loss increase while the PE is consuming less energy at the same time. Nevertheless, the energy savings are achieved for CN while the performance loss is below 13%, at which point the slope of energy consumption curve increases drastically. These results tally well with the experimental findings for the energy consumption of Saxitoxin shown in Fig. 4.7. For example, the CN energy consumptions at 2.33 GHz and 2 GHz are no less than that at the highest frequency while energy savings are achieved above 2.67 GHz. The values of the performance loss tolerance δ are, respectively, 27.15%, 47.2%, and 11.68%, marked as vertical straight lines in Fig. 4.8.

4.3 GAMESS Process Mapping within a Node

Beyond accessing distributed arrays on remote nodes, GAMESS processes take advantage of the shared arrays on the local nodes. Although such arrays are still accessed using DDI, the communication happens only among the local compute processes without the data server involvement. Furthermore, most of these communications are blocking point-to-point, so they are usually fast and have two compute processes participating in a “handshake” (send-receive)

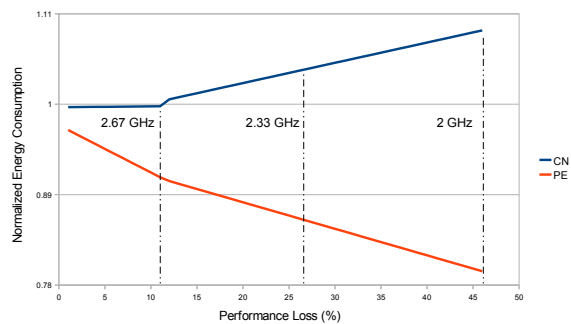


Figure 4.8 Variation of energy consumption with performance loss for Saxitoxin in direct mode.

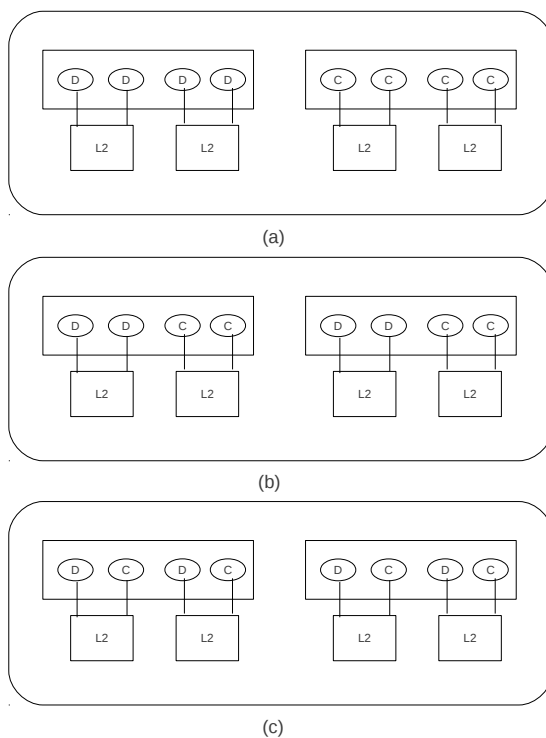


Figure 4.9 Three bindings of GAMESS data server (D) and compute (C) processes: (a) Disjoint-I, (b) Disjoint-II, and (c) Slave, for Each compute node has two quad-core processors arranged as two sockets. Twin cores share the L2 cache.

fashion as opposed to PGAS model.

In GAMESS, only compute processes show the phase behavior, as was determined by the authors previously [55]. Specifically, they present a repeating pattern when the ranks and message sizes for the point-to-point operations are analyzed, regardless of the total size of the GAMESS calculation. Since only half of the GAMESS processes depict phase behavior, it has been determined in [55] that to obtain energy savings under the DVFS twin-core granularity requires careful consideration of process-to-core mappings in addition to the compute-process phase characterization by the runtime procedure. This work builds on the findings of [55] by analyzing an additional GAMESS process-to-core mapping and by proposing novel frequency scaling strategies and applying them to GAMESS. These enhancements deliver more energy savings than what was shown in the authors' previous work with a similar performance loss.

Since data servers used in GAMESS are involved in global communications only, i.e., they do not perform any calculations, reducing the frequency of the cores to which data servers are mapped will not affect much the overall execution. Therefore, it may be tempting to just reduce their frequency to the minimum level. However, this delivers no energy saving for certain mappings of data server relative to compute process within a node because the DVFS switch is efficient on the twin-core basis only. Thus, it is necessary to determine certain relative mappings (also referred to as bindings) of data servers and compute processes for the phase-based frequency scaling to work efficiently in the compute processes.

Disjoint Bindings One way to achieve energy savings is to dedicate each pair of twin cores to either compute or data server processes, as in fig. 4.9(a). Then the compute processes are operated under the phase detection procedure while the frequency of data servers is simply reduced to the maximum through DVFS at runtime. This binding is termed here as Disjoint to stress its independent nature of the frequency scaling in different process types of GAMESS. Furthermore, there are two different variants of the Disjoint binding. (I) Data servers and compute processes are on different sockets, as in fig. 4.9(a). (II) Data servers and compute processes are mixed on the same socket in such a way that pairs of data servers and compute processes operate on pairs of twin cores, as shown in fig. 4.9(b).

Slave Binding Another process binding may be considered, such that it puts one data server and one compute process together on the same pair of twin cores as in fig. 4.9(c). Then, the compute processes are still operated under the phase detection procedure, and the frequency of the twin-core pair is reduced per DVFS application prescribed by this procedure. This strategy is called Slave because the decision regarding frequency scaling is done only through the compute process and data server simply follows it.

CHAPTER 5. Modified Runtime System

In this chapter, a modified design of the runtime system proposed in Chapter 3 which applies frequency scaling to both point-to-point and collective communications is discussed.

5.1 Communication Phase Detection

Many HPC parallel applications exhibit recurring communication phases occurring at the communication-call granularity. The phase detection-mechanism identifies such recurring communication phases by using a string matching algorithm performed during the application execution. In this work, for detecting communication phases per core, each MPI call is intercepted by exploiting PMPI [45], the profiling interface of MPI, and assigned a unique *callid*. The *callid* identifies each communication call by its parameters, such as rank and message size. The call program counter (PC) has been used by others [35] to identify a call. Since the proposed runtime system has to identify different types of communications (i.e., collective and point-to-point) and differentiate among their message sizes, the use of PC becomes inadequate. Therefore, the *callid* assigned to each call for unique identification.

These *callids* are stored in a *master string*; and finding a communication phase is equivalent to finding various maximal repeats [21] in this string because a communication phase, as defined here, is nothing but a sequence of *callids* that repeats itself during the application execution. The time duration of a phase is denoted as *phase length*. The maximal repeats are determined by finding two identical substrings, after which a new phase is declared as starting from the first matching *callid* and lasting while the *callid* continues to match in the two substrings. The substring length to match may be chosen experimentally, as done in this work. After recording the identical substrings as a phase, the master string is modified by purging

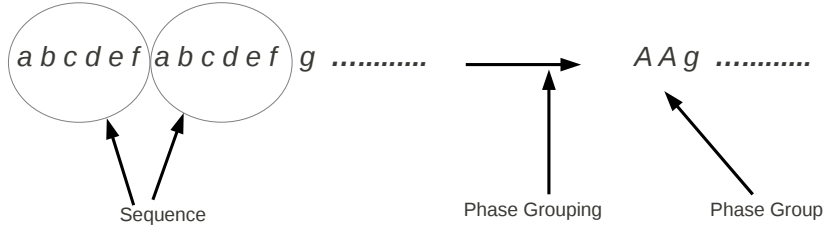


Figure 5.1 Phase detection and string manipulation for master string with (a) consecutive and (b) nonconsecutive communication phases.

their instances and the search for another new phase continues. Such a simple process of determining communication phases obviates the need for a costly construction of a suffix tree and the supermaximal repeat string algorithm [24], which may quickly become unacceptable for a runtime system examining parallel applications with many communication calls.

Figure 5.1 shows an example of the phase detection and master string manipulation for two cases of consecutive (fig. 5.1(a)) and nonconsecutive (fig. 5.1(b)) phase instances. The *calls* are depicted by the lowercase letters and the communication phases are enclosed into ovals. In fig. 5.1(a), after the string manipulation, the recorded phase instances are removed and the subsequent *calls* are shifted to the starting position into the place of removed phase instances to form the *resultant master string*. This deletion of the recorded phase instances from the master string avoids the repeated detection of a phase. In the case of nonconsecutive phase instances (fig. 5.1(b)), there is also a position shift of the *calls* between the two phase instances in addition to the *calls* after the second phase instance to get the resultant master string.

To apply DVFS, reference points within the communication phase are taken as the start and end of the phase, which is similar to the idea of reducible regions [35]. This means that the frequency is lowered to a suitable level at the start of the first call in a phase and is increased to the original value at the end of the last call.

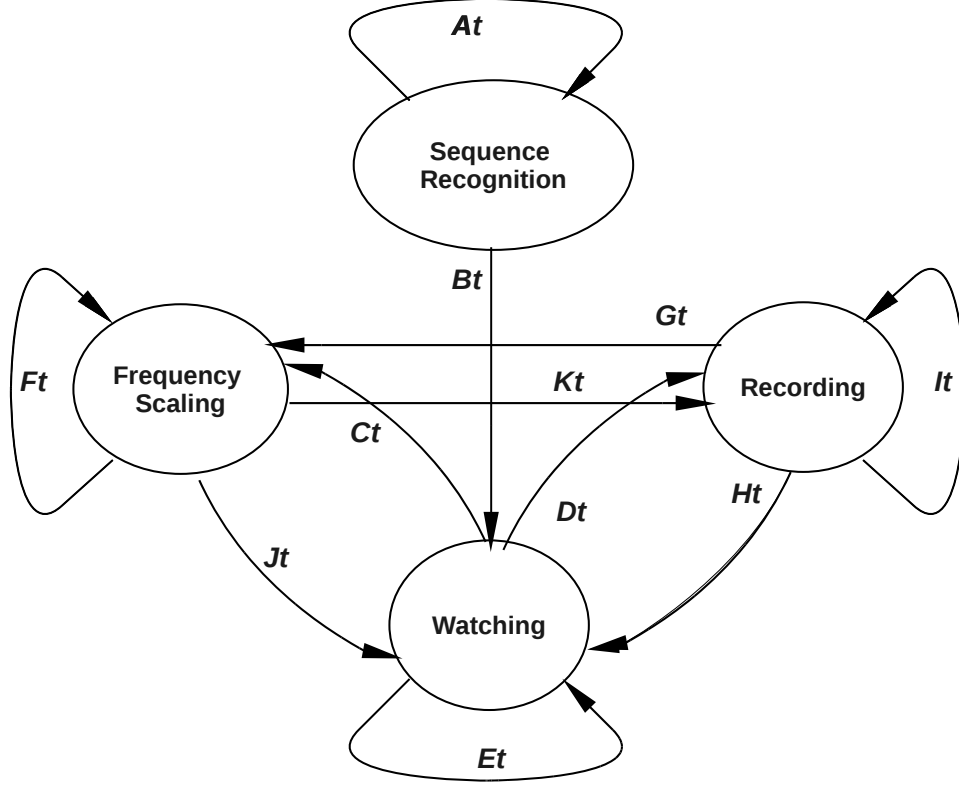


Figure 5.2 State diagram for runtime system to apply frequency scaling efficiently. The transitions are labeled with Lt , where L takes a value $A-G$. The transition of a state into itself (At , Bt , Ct) indicate ongoing state action.

5.2 System Design

A general design of the runtime system consists of the following major states: *phase detection*, *recording*, and *frequency scaling*. Figure 5.2 outlines these three states along with their transitions, which are detailed in sections 5.2.1 to 5.2.3. To use a state machine is proposed here following the work of Freeh *et. al* [17], which considers various states during the phase detection and frequency scaling.

5.2.1 Phase Detection

The communication phase-detection mechanism has been explained in section 5.1. While the first communication phase is undetermined, the runtime system remains in the phase-detection state (transition At in fig. 5.2). As soon as a communication phase is detected, there is a transition (Gt) to the recording state to record different parameters associated with the

communication calls. After that, the phase-detection state is re-entered (transition Ft) to detect next phases while checking the *callid* of every upcoming call. If the current call matches the *callid* of the starting call of an already recorded phase, the system makes a transition (Et) to the frequency-scaling state since this match marks the beginning of some previously recorded phase, otherwise the phase-detection state persists (At) while trying to find new phases.

To amortize the DVFS overhead, the duration of the recorded phase should be rather long. Therefore, after detecting a communication phase, its duration is compared with a certain bound L defined as follows:

$$L = \frac{O(f_p \rightarrow f_P) + O(f_P \rightarrow f_p)}{\gamma}, \quad (5.1)$$

where $O(f_x \rightarrow f_y)$ is the DVFS overhead for the hardware platform when transitioning from level x to y ; p and P are the lowest and highest P-states, available; and γ is the (user-defined) performance loss, which constrains the amount of the performance degradation tolerated when the CPU frequency is scaled down. The rationale behind eq. (5.1) is that the maximum DVFS overhead should not result in a performance loss higher than γ for any phase. If the duration of the found phase is smaller than L , then another instance of this phase is appended to it to form a new phase if these two instances are consecutive, as in fig. 5.1(a), for example. Conversely, when there exist some other calls between the two instances, as calls g, h, i, j, k, l , and m do in the example from fig. 5.1(b), they are appended to the first instance of the phase followed by the second instance to form a new phase. Such a phase enlargement continues until the phase length exceeds the bound L . There may also be the case when an individual call is longer than the bound L but is not part of a phase yet. Then, this single call is recorded as a phase by itself for applying frequency scaling.

In general, to minimize the DVFS switching overhead in parallel applications, the proposed runtime system distinguishes their communication phases inside which the same (typically lower than maximum) frequency is kept while a different frequency may be used on the outside. The aim is to obtain the phases of the longest possible duration.

5.2.2 Recording

After the matching substrings have been determined in the master string, the system transitions (Gt) to the recording state, which continues while the *callids* are matching in the two substrings. The *callids*, message sizes, and other call parameters, such as call duration and time gaps between the calls are recorded in this state. In addition, depending on the chosen frequency scaling strategy, the needed performance-counter values, as explained in section 5.2.3, are recorded.

The recording of a new phase is aborted with two resulting state transitions:

- frequency scaling, if the *callid* of the current call equals to that of the first call in some previous phase (Ht in fig. 5.2).
- phase detection, otherwise (Ft in fig. 5.2).

5.2.3 Frequency Scaling

In the *frequency scaling* state, the calls are continuously checked as to whether or not their *callids* match the ones of a previously recorded phase. If there is a mismatch, frequency of the processor is restored to its highest value f^h and the call causing the mismatch remains in the current state (Bt in fig. 5.2) if its *callid* matches the beginning of any already recorded phase. Otherwise, a transition to the phase-detection state occurs (Dt in fig. 5.2).

The frequency-scaling state changes the processor frequency based on communication phase boundaries, such that the frequency is reduced at the beginning of the phase for the communication operations and generally raised in the end of the phase. Figure 5.3 provides an example of the phase definition for the purpose of frequency scaling, where a phase is represented by calls a , b , c , and d . The time gap between successive phases is termed *interphase gap*, while a time gap within a phase is denoted as *intrapphase*. Three strategies—termed DVFS-Ph, DVFS-PhI, and DVFS-PhIT—are discussed next as distinguished by their application of frequency scaling to the intra- and interphase time gaps.

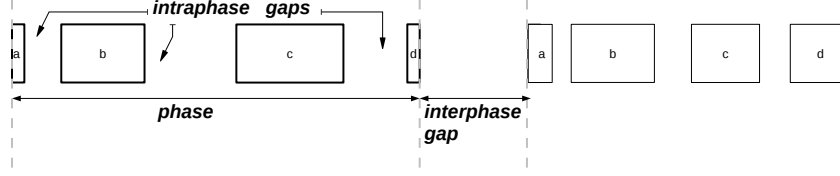


Figure 5.3 Trace of an MPI application invoking eight MPI calls with a phase length of four. (The calls within a phase are ordered lexicographically from *a* to *d*.)

5.2.3.1 DVFS-Ph

The DVFS-Ph strategy applies DVFS only to the communication phases and not to the interphase gaps. When the communication phase finishes, all the *calls* associated with it are removed from the master string. In [58], an appropriate frequency is selected for a single communication call by taking into account just the message size while exploiting the Infiniband CPU offload feature [36] and the communication characteristics. In the case of a multicall phase, the gaps between call pairs must be examined to evaluate their performance losses at a lower frequency.

Denote as $T_{call}(f^h)$ the duration of all the communication calls in a phase and $T'(f^h)$ the total duration of the intraphase time gaps when executed at the maximum frequency of the processor. Then the total phase time T^h at the maximum frequency equals $T_{call}(f^h) + T'(f^h)$. Let f_γ^* be a suitable frequency, selected among the existing P-states, for the phase when the (user-defined) performance loss is γ . Then, let $O_{call}(f_\gamma^*)$ and $O_s(f_\gamma^*)$ be the communication-call and frequency-switching overheads [41], respectively, when f_γ^* is attained from the highest available CPU frequency f^h . Note that, in this work, these overheads have been predetermined in a separate set of experiments for various communication operations for a given range of the available P-states.

Since the intraphase time gaps may have architectural stalls, such as memory, I/O, or other resource-related stalls, during the computations possibly present in these gaps, a quantitative analysis is desirable to estimate the CPU usage in-between communication calls in a phase. In modern processors, performance counters may be employed to measure architectural stalls during the time gaps. However, the number and applicability of these counters may be limited. For example, the Intel Xeon E5450 processor, used in this work, provides only two general-

purpose performance counters, and thus, hinders the utilization of a sophisticated model, such as the one suggested in [23] that relies on four performance counters. A small number of the counters may still be useful if the frequency calculation is distilled to those most critical parameters that the available program counters can measure. In particular, this work proposes to count both the rate of micro-operations retired $\mu\tau$ and the number of memory accesses m . The authors have observed experimentally that, for some memory-intensive applications, the rate $\mu\tau$ was higher than for the compute-intensive ones tested, which is counter intuitive if only the micro-operations retired were considered, as was done in [35].

The rate of micro-operations retired $\mu\tau(f^*)$ at an available CPU frequency f^* can be predicted based on both $\mu\tau(f^h)$ and $m(f^h)$ through the following relation:

$$\mu\tau(f^*) = \frac{f^* \cdot \mu\tau(f^h)}{f^h} + b \cdot m(f^h), \quad (5.2)$$

where the parameter b is dependent on the number of memory accesses per second and can be determined experimentally. Then, for the intraphase time gap t'_i ($i = 0, \dots, N - 1$, where N is the number of calls in a phase), $\mu\tau_i(f^*)$ may be determined and the corresponding performance loss γ_i for $f^* < f^h$ may be calculated as

$$\gamma_i \approx \frac{\mu\tau_i(f^h) - \mu\tau_i(f^*)}{\mu\tau_i(f^h)} \approx \frac{t'_i(f^*) - t'_i(f^h)}{t'_i(f^*)}. \quad (5.3)$$

When the frequency scaling of a particular phase is performed for the first time, the intraphase time gaps are all assumed to be memory-intensive, i.e., $\mu\tau_i(f^h) \approx \mu\tau_i(f^1)$, and the rate of micro-operations retired is recorded to solve eq. (5.2) for b . This value of b is used when the phase is encountered next time to predict the rate of micro-operations retired, which is updated with the actual rate of micro-operations retired once the phase frequency-scaling takes place.

Since the total overhead $O'(f^*)$ for executing the intraphase time gaps at a frequency f^* is $O'(f^*) = T'(f^*) - T'(f^h)$, the suitable frequency for a phase f_γ^* may be calculated from

$$\gamma \geq \frac{O'(f_\gamma^*) + O_{call}(f_\gamma^*) + O_s(f_\gamma^*)}{T^h}. \quad (5.4)$$

5.2.3.2 DVFS-PhI

To overcome the often tedious provisioning for the phase overlap in multicore platforms as discussed in Section 3.1.4.3, the DVFS-PhI (denoting “DVFS in the phase and interphase”) strategy is proposed next. This strategy augments DVFS-Ph with targeting interphase time gaps for frequency reduction and extends for them the performance-counter utilization. Specifically, it calculates the micro-operations retired and memory accesses in the interphase gaps and applies to them an appropriate frequency f_γ^* in a manner similar to that of the DVFS-Ph strategy. Then, the performance loss is calculated as in eq. (5.3).

Once a phase ends, the current frequency f^c is compared with f_γ^* and is left unchanged if they are the same. Otherwise, f^c takes the value of f_γ^* if the switching overhead $O_s(f_\gamma^*)$ is less than the performance overhead $O''(f^c)$ from executing the interphase gap at the current frequency f^c . By considering the switching and performance overheads during the interphase time gaps, the DVFS-PhI strategy avoids unnecessary frequency scaling switches that the DVFS-Ph strategy incurs when a mandatory frequency scaling is applied at the phase boundaries.

To account for the cases when the interphase gap is too long or when no other communication phase is observed further in the execution, an additional two-step guard has been incorporated into the DVFS-PhI strategy:

Step 1. Record the current interphase gap duration t_j'' .

Step 2. If $j > 0$ and $t_j'' \geq K \cdot \max_{0 \leq k < j} t_k''$, then restore f^h .

In this work, the value of $K = 2$ has been chosen (in **Step 2**) based on the assumption that all the interphase gaps are of similar lengths and that the length of the interphase gap is comparable to that of the phase. With this guard of the DVFS-PhI strategy, potentially compute-intensive tasks are not executed at a lower frequency, and thus, experience no significant performance degradation.

Since the proposed runtime system examines each phase sequentially in a communicating processor, it should apply the frequency scaling to the interphase gap *before* the next phase

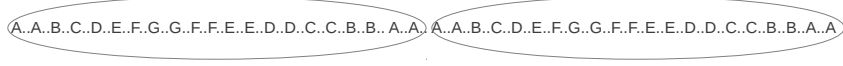


Figure 5.4 Grouping of the communication phases detected for rank 0 of the MG NAS benchmark. The capital letters followed by the double dots represent communication phases followed by interphase gaps, respectively. A single group (shown as solid oval) has been found and the two corresponding subsequences are enclosed into the ovals denoting this group.

is detected. Thus, the runtime system may have difficulties in recognizing properly this gap, which, by definition, is uniquely identified by the start *and* the end of two adjacent phases. In particular, such a situation occurs when the two phases are different or when the complicated rank sequence patterns are present. For the communications of a repeating nature, however, this difficulty may be alleviated by grouping communication phases in the overall phase sequence.

In a sense, finding each group is akin to determining the longest repeated substring (LRS) or the recurring sequence of phases, as defined in [21]. As an example, fig. 5.4 depicts such a grouping of the phases detected for rank 0 of the MG NAS benchmark. Observe that there are recurring subsequences of phases, which may be grouped. In fig. 5.4, the two subsequences are enclosed into identical ovals to emphasize that they belong to the same group. The phase detection mechanism from section 5.1 is used to find the LRS for parallel applications. Once such a grouping is achieved, the interphase gap recognition becomes a matter of determining to which phase group it belongs. While LRS is undefined, the interphase gap is assumed to have the same characteristics as what it had the last time when the current phase was encountered.

5.2.3.3 DVFS-PhIT

Although the DVFS-PhI strategy examines the whole execution of an application (its communication and computation) it uses only DVFS and does not fine-tune the communication frequency scaling further to the (eight) levels of throttling. Many communication operations, such as collective calls, however, present opportunities for throttling as was shown by the authors in [53]. Moreover, superior energy savings may be delivered when both the DVFS and throttling are in use rather than when each is applied separately. Note that throttling can be viewed as equivalent to dynamic frequency scaling (DFS) [4] because, by inserting a given

number of idle cycles in the CPU execution, a particular operating frequency is obtained without changing the operating voltage of the cores. Therefore, by itself, throttling is often less effective than DVFS.

DVFS versus CPU Throttling. Even though the CPU throttling is supported on a per-core granularity as opposed to DVFS, which is supported on a twin-core granularity, its power saving capabilities are inferior to those of DVFS. Throttling does not reduce the actual clock rate but rather inserts the STPCLK (stop clock) signals thereby omitting the duty cycles, and throttling does not change the operating voltage of the cores. Moreover, it results in considerable performance loss for intra-node communications as compared to DVFS [53]. When different frequency scaling strategies were tested using the CPU throttling only, the total energy consumption actually increased at times because the power consumption remained rather high while the performance loss was significant, up to the allowed 10% in certain cases of multiple switches. On the other hand, if an application exhibits poor phase overlap among the twin cores and the phases consist mostly of the inter-node communications—the two conditions under which a simple application of DVFS to communication phases is not effective—throttling may give more energy savings than DVFS.

In the DVFS-PhIT strategy, once a suitable P-state is found, each communication call is then checked as to whether the application of throttling is possible. (A checking procedure aiming to select, for each call, a suitable throttling level T_k is discussed in section 5.2.3.4 that follows.) If τ is the duration of a given call without energy saving techniques, i.e., at the highest frequency and no throttling, then the maximum overhead from throttling should not exceed the performance loss γ . Thus, for the throttling level T_i to be applied to the call, the following relation should hold

$$\gamma \geq \frac{O(T_0 \rightarrow T_i) + O(T_i \rightarrow T_0)}{\tau}, \quad (5.5)$$

where T_i is the highest level that is less or equal to T_k and $O(T_x \rightarrow T_y)$ is the throttling system overhead to switch from level x to y . Equation (5.5) attempts to amortize the total throttling overhead over the duration of the call for the given γ .

5.2.3.4 Estimation of Throttling Level

Since, by design, the proposed runtime system is transparent to both the application code *and* communication library, throttling has to be performed independently from the communication library implementation. On the other hand, as has been shown in [58], different throttling levels must be applied to different *stages* of a collective call algorithm for throttling to be efficient and incur a small switching overhead. It was concluded that *inter-node* communication provides better opportunities for throttling than *intra-node* does so, which owes to the CPU offload by the Infiniband. Aiming for transparency, stage-based throttling is not feasible since it requires changes in the source code of the communication library functions [53]. Therefore, communication calls have to be re-examined with the purpose to *estimate* a throttling level that may be applied to the entire call without much overhead. In a sense, this estimation is based on a cumulative view of intra- or inter-node communication stages in specific communication algorithms.

Next, a few sample MPI operations and their underlying algorithms are presented to walk through the estimation procedure. For other communication libraries and for additional MPI operations similar steps may be taken.

Blocking point-to-point operations, such as MPI_Send and MPI_Recv, have been already examined by the authors in [53]. Since these operations require no complex implementations, the throttling levels were applied based mainly on the message sizes and the initial topology of the ranks. Throttling may be applied to such operations when the communication is inter-node [53]. For the nonblocking point-to-point operations, such as MPI_Isend, an overlap of computation and communication exists, which precludes efficient application of throttling.

MPI collectives. MPI_Alltoall and MPI_Allgather are easily the most communication intensive among all the MPI operations and provide excellent opportunities for applying throttling in their algorithmic stages to maximize energy saving as was shown in [58]. For the runtime system, a lower bound for the possible performance loss is computed here based on the overheads of point-to-point communications that comprise each collective call for a given message size and the number of cores in a node.

For example, consider the Send-To Receive-From (STRF) algorithm used for the MPI_Alltoall operation on n processes. If a compute node has c cores, there are $2 \cdot (c - 1)$ intra-node communication stages and $n - 2 \cdot (c - 1)$ inter-node communication stages [53]. Let $\mathcal{O}'_k(M)$ and $\mathcal{O}''_k(M)$ be the overheads for intra-node and inter-node point-to-point operation, respectively, which use the message size M and the throttling level T_k . Then, T_k is chosen for the entire MPI_Alltoall (implemented with the STRF algorithm) as follows:

$$\gamma \geq \frac{2 \cdot (c - 1) \cdot \mathcal{O}'_k(M) + (n - 2 \cdot (c - 1)) \cdot \mathcal{O}''_k(M)}{\tau}, \quad (5.6)$$

while ignoring the effects of network and memory contention. Throttling levels may be determined for other collective algorithms in a similar fashion. Such an approximation is feasible because all the collective-call implementations comprise several point-to-point for sufficiently large number of processes n . For example, for MPI_Alltoall and MPI_Allgather operations, throttling levels T_1 or T_2 are usually selected using eq. (5.6) except for the Ring algorithm that has mostly intra-node communication and does not provide enough room for throttling.

The MPI_Bcast, MPI_Scatter, and MPI_Gather operations use the leader based algorithm [27] in which the communication happens first between a chosen leader process in each compute node and then the leaders distribute the data to other processes present in the same compute node. For this type of algorithms, inter- and intra-node communication stages are separated and, thus, dealt with in a manner similar to eq. (5.6).

The MPI_Reduce and MPI_Allreduce operations perform computation interleaved with communication using recursive algorithms [59]. Therefore, these operations do not provide much opportunity for throttling, which was also observed through experiments.

CHAPTER 6. Experimental Results

In this chapter, the experimental results are presented for energy aware collective communication algorithms, runtime system for point-to-point communications and modified runtime system for point-to-point and collective communications.

Experimental Setup The experiments were performed on the computing platform Dynamo¹, which comprises 35 Infiniband DDR-connected compute nodes, each of which has 16 GB of main memory and two Intel Xeon E5450 Quad core processors arranged as two sockets with the operating frequency in the interval from 2 to 3 GHz including four P-states in the steps of 0.33 GHz and eight levels of throttling from T_0 to T_7 . Table 6.1 presents the effective operational frequency achieved when the throttling levels (T-states) were applied in this work along with the DVFS that scales CPU to the lowest P-state of 2 GHz. For measuring the node power and energy consumption, a Wattsup² power meter is used with a sampling rate of 1 Hz, having an accuracy of $\pm 1.5\%$ and granularity of Watts. The Wattsup meter was connected to a compute node N_w and its power consumption was measured through a profiling node. Since all the Dynamo nodes are homogeneous in terms of their hardware configuration, the MPI ranks were assigned in turns to the node N_w , so that the power consumption for an entire application is measured. In measuring the power consumption of a compute node, no process variation among the homogeneous nodes is assumed. Due to a low measuring resolution of Wattsup and for consistency, a large number of collective operations were performed. For determining the average power consumption, 100 samples are taken for a particular message size followed by averaging. Specifically, at first the time spent in the collective operation is measured for a given

¹funded and operated jointly by Iowa State University and Ames Laboratory.

²<https://www.wattsupmeters.com>

message size then the number of iterations is determined, so that the collective executes for 100 seconds on Dynamo. A higher resolution meter will be considered in the future.

The MPI implementation is MVAPICH2³ with the default *block* mode for MPI rank placement and *bunch* mode for process mapping. (Refer to the MVAPICH2 User Guide for the explanation of these parameters.) The experiments were executed with 32 MPI processes, one per core of four Dynamo nodes using the MPI profiling layer PMPI [45, 62]. The runtime phase-detection procedure has been first tested on the CG and MG NAS benchmarks [5], which were chosen as representative of widely used efficient parallel applications with a large number of blocking point-to-point communications. Then, the real-world quantum chemistry package GAMESS was considered.

For measuring the CPU power a Fluke i410 current probe connected to an Agilent 34410A digital multimeter (DMM) was used [25]. The current probe measures the total current through the 12V power lines and then the DMM sends the obtained readings to a logging machine via Ethernet. For measuring the memory power consumption, a linear extrapolation method [19] is used.

The input to the GAMESS program was constructed to perform the MP2 energy and gradient calculation—in the direct mode—of substituted silatrane, the 1-trichloromethylsilatrane (TCMS) molecule, shown in fig. 6.1. From the quantum chemistry viewpoint, silatrane and its derivatives belong to an interesting class of silicon-based biologically active substances lacking carbon-based natural analogs. Silatrane and its derivatives have numerous biological and non-biological applications [44]. They have been recently investigated using computational chemistry methods [50], such as MP2, to elucidate reaction mechanisms essential to those applications.

Two sets of MP2 computations were performed on 32 cores of four nodes: with 6-31G(d) basis set (265 basis functions) and 6-31G(d,p) basis set (301 basis functions). These two sets are referred to as Silatrane-265 and Silatrane-301, respectively, in the rest of this work. From the computational point of view, silatrane derivatives represent typical test cases with modest run times. For example, the wallclock times were around 113 and 160 seconds for Silatrane-265 and

³MVAPICH Project: <http://mvapich.cse.ohio-state.edu>

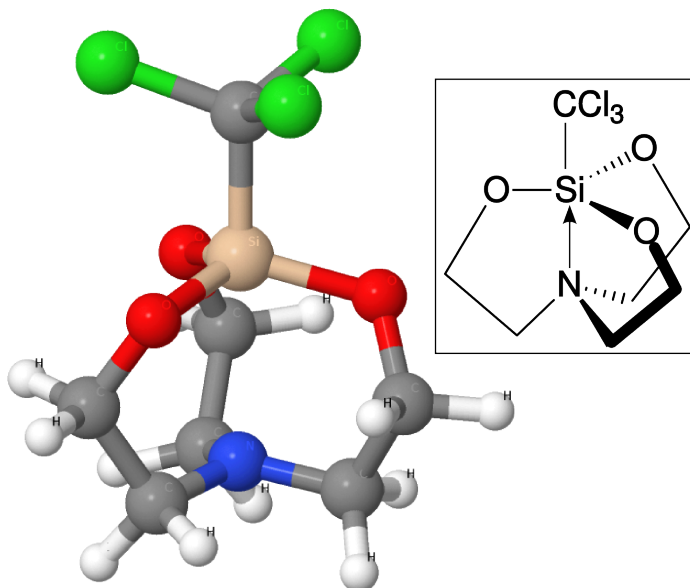


Figure 6.1 3D geometrical structure of the TCMS molecule (silicon, carbon, oxygen, nitrogen, chlorine and hydrogen atoms are shown as pink, gray, red, blue, green, and white balls, respectively). The molecular skeletal formula is inserted on the right.

Silatrane-301, respectively, when run on Dynamo at the highest frequency. In the experiments, since the results were averaged over ten runs for each silatrane computation, and since the relative sizes of the basis sets used do not differ much, some fluctuations may appear in the relative timings of Silatrane-265 and Silatrane-301 when averaging, which, however, do not affect frequency scaling decisions or observed performance loss percentages. Approaches and techniques developed using these test cases may be readily extended to larger computations.

6.1 Energy Aware Collective Communication Algorithms

MPI_Alltoall. OSU MPI Benchmarks⁴ are used here to determine the change in execution time and power consumption of the “stand alone” all-to-all operations. From Fig. 6.2(left), it can be observed that the execution time for all-to-all has very low performance penalty when the proposed energy savings are used. The performance loss and power consumption have been evaluated for four cases of execution:

- at the highest frequency and no throttling (*Full power*),
- only frequency scaling without throttling (*DVFS only*),

⁴OSU MPI Benchmarks: <http://mvapich.cse.ohio-state.edu>

Table 6.1 Effective operational frequency (f_{op}) at a combination of the 2 GHz P-state and one of the T-states (Level) used.

Level	T_1	T_2	T_5	T_7
f_{op} , GHz	1.75	1.50	0.75	0.25

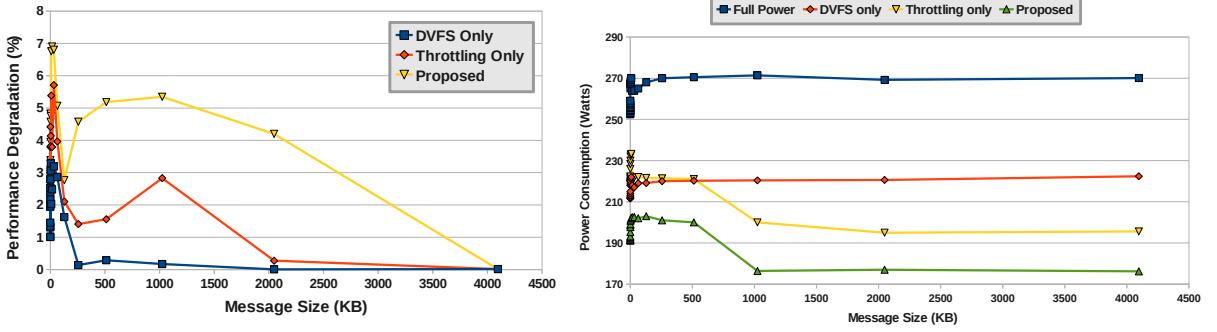


Figure 6.2 The all-to-all performance degradation on 80 cores (left) for three cases and the power consumption across a compute node (right) for the four cases: Executing at the highest frequency and no throttling (**Full power**); only frequency scaling without throttling (**DVFS only**); only CPU throttling without frequency scaling **Throttling only**; and using the energy saving strategies proposed for all-to-all (**Proposed**).

- only CPU throttling without frequency scaling (*Throttling only*),
- using the energy saving strategies proposed for all-to-all (*Proposed*).

The performance loss averaged for various message sizes was just 5% of that for the *Full power* case. While somewhat higher than in the *DVFS only* and *Throttling only* case, which was 2% and 3.2% respectively, it is quite acceptable taking into the consideration large reductions in the power consumption achieved (Fig. 6.2(right)) with the *Proposed* strategy. The *DVFS only* case can be compared to the techniques presented in [13] in which the authors make use of DVFS alone to conserve power during collective operations.

From Fig. 6.2(right), it can be observed that the proposed strategy has the lowest power consumption among all the strategies. The *DVFS only* and *Throttling only* cases have power saving of 17.4% and 16.2% respectively on an average for various message sizes, as compared to the *Full Power* case. The *Throttling only* case saves lesser power than *DVFS only* as it does reduce the clock rate but does not change the working voltage of the cores whereas DVFS does both simultaneously. It is only at for the large messages (> 0.6 MB) when throttling level goes up to to T_7 and at that point *Throttling only* does save more power than *DVFS only* but incurs

a higher performance loss. The *Proposed* case has the highest average power saving of 26% as it makes use of both DVFS and CPU throttling. Note that for message larger than 0.6 MB, the average power saving is 34%.

MPI_Allgather. Intel MPI Benchmarks⁵ are used here to determine the change in execution time and power consumption of the “stand alone” allgather operations. Fig. 6.3(left) presents the percentage of performance degradation of the allgather operation for the three cases of the CPU frequency settings. The maximum performance losses observed were 2%, 4.1%, and 5.9% for *DVFS only*, *Throttling only*, and *Proposed*, respectively, as compared with the *Full power* case. Therefore, the *Proposed for allgather* strategy performance is well below the chosen margin of the 10%. Observe that all the frequency downscaling techniques take less time than the highest frequency execution of allgather for large message sizes (shown as ovals in Fig. 6.3(left)), which may be attributed to the operating system noise effects as suggested in [42].

The *DVFS only* and *Throttling only* cases have maximum power savings of 17% and 7.1%, respectively, on an average for various message sizes whereas the *Proposed* strategy results in the highest power savings of 21%, as seen from Fig. 6.3(right). The significantly lower power savings achieved in the *Throttling only* case are due to the intranode type of communication employed in the ring algorithm, which is the preferred choice for large general message sizes, since it permits only up to T_2 level of throttling (see Fig. 2.3).

Application Testing. CPMD (Car-Parrinello molecular dynamics)⁶ is an ab-initio quantum mechanical molecular dynamics real-world application using pseudopotentials and a plane wave basis set. Eleven input sets from the CPMD application are used here. MPI_Alltoall is the key collective operation in CPMD. Since most messages have the sizes in the range of 128 B to 8 KB, the BIA is used. From the NAS benchmarks [5], the FT and IS Class C benchmarks are chosen because they use the all-to-all operation.

⁵Intel MPI Benchmarks: <http://software.intel.com/en-us/articles/intel-mpi-benchmarks>

⁶CPMD Consortium: <http://www.cpmc.org>

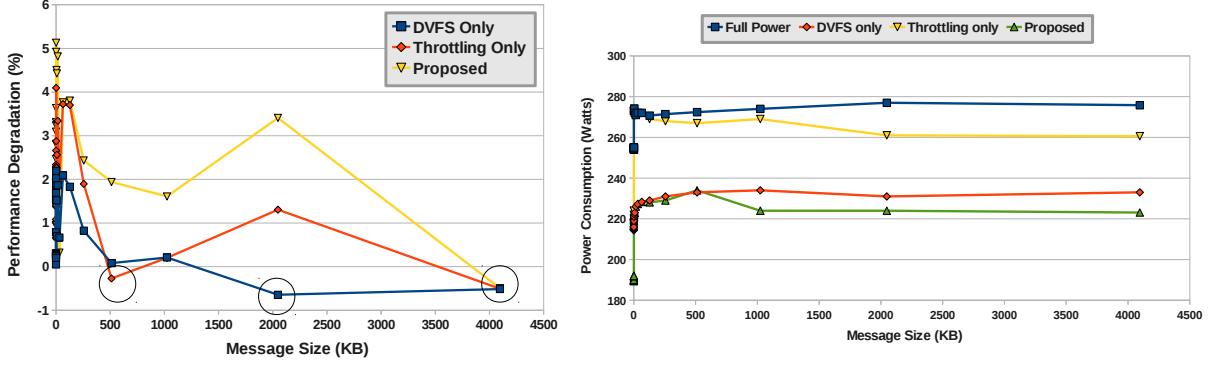


Figure 6.3 The allgather performance degradation on 80 processes (left) for three cases and the power consumption across a compute node (right) for the four cases: Executing at the highest frequency and no throttling (**Full power**); only frequency scaling without throttling (**DVFS only**); only CPU throttling without frequency scaling **Throttling only**; and using the energy saving strategies proposed for allgather (**Proposed**).

Fig. 6.4(top) and Fig. 6.4(bottom) show the execution time and energy consumption, respectively, of CPMD inputs and NAS benchmarks normalized to the *Full power* case. CPMD has been run on 80 cores while NAS benchmarks have been on 64. For the CPMD with the *Proposed* strategies, the performance loss ranges from 0.4% to 4.3% averaging 2.4% leading to the energy savings in the range of 9.8% to 15.7% (13.4% on average). In all the benchmarks tested, the *DVFS only* case has lower performance loss and saves more energy than the *Throttling only* case. The *DVFS Only* case has an average performance loss of 1.2% whereas the *Throttling only* case results in an average performance loss of only 1.8%. As to the energy consumption, the *DVFS Only* case saves 8.4% and *Throttling only* saves 7.2% energy on an average for the CPMD and NAS benchmarks.

For the NAS benchmarks, the performance loss ranges from 1.1% to 4.5% and the average energy savings are about 10%. The NAS IS benchmark uses the MPI_Alltoallv collective in which message size is nonuniform among different processes. Therefore, a naive solution is proposed to deal with this nonuniformity: During the intranode communication, only DVFS is applied once the communication becomes internode, the throttling level T_5 is added to account for both smaller and larger messages. This is similar to the technique proposed and evaluated by authors previously in [53]. A more detailed analysis of the energy savings potential in the MPI_Alltoallv collective with respect to the nonuniform message sizes is left as future work.

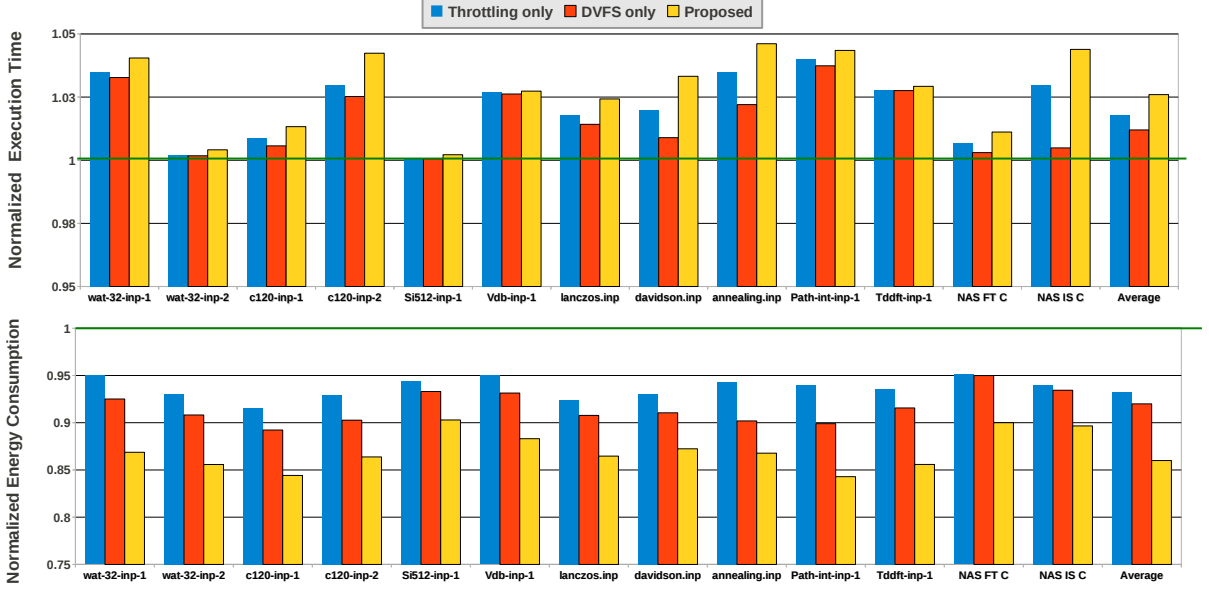


Figure 6.4 Execution time (top) and energy consumption (bottom) of 11 CPMD inputs on 80 cores and of 2 NAS benchmarks on 64 processes for the DVFS only, Throttling only, and Proposed cases normalized to the Full power. The last set of bars (Average) represent the average of the respective y -axis values across all the CPMD and NAS tests.

Elemental [43] is a C++ package for distributed-memory dense linear algebra and may be considered as a generalization of LAPACK [3] to the element-by-element matrix distributions. Elemental relies heavily on the MPI collectives, such as allgather and all-to-all and is, therefore, well-suited to demonstrate the effectiveness of the proposed energy saving strategies. Five different linear algebra algorithms from Elemental were executed on 80 cores for a matrix size of 10,000 with the algorithm block size of 128 and with the local matrix block size of 32 for triangular rank k update. Fig. 6.5 shows the timings and energy consumption of the five chosen Elemental algorithms normalized to the Full power case. The average performance loss and average energy saving for the Proposed strategy are 4.3% and 4.5% respectively. The DVFS only case performs better than the Throttling only one since it has an average performance loss of 3.5% with the energy savings of 2% whereas throttling delivers 1.7% of savings with the performance decreased by 3.1%. On the other hand, the Proposed strategies have the highest energy savings (4.3%) with a little more performance degradation (4.8%).

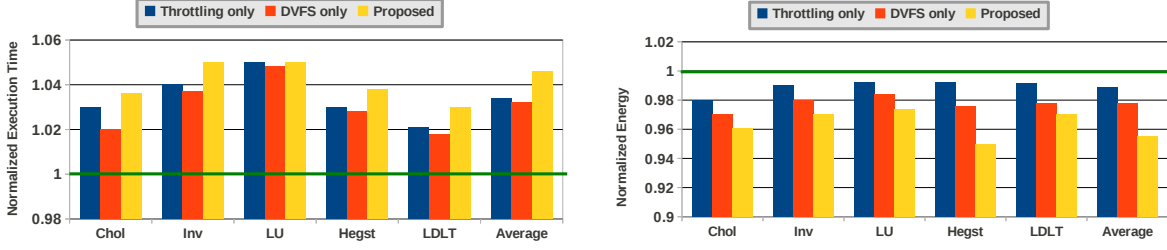


Figure 6.5 Execution time (left) and energy consumption (right) of five Elemental algorithms—Cholesky (Chol), Triangular inverse (Inv), LU decomposition (LU), Hermitian eigensolver (Hegst), and LDL^T factorization (LDLT)—on 80 cores for the DVFS only, Throttling only, and Proposed cases normalized to the Full power. The last set of bars (Average) represent the average of the respective y -axis values across the five Elemental algorithms.

Table 6.2 Average percentage of the EDP reductions in all the experiments for the three energy saving strategies.

Experiment	DVFS Only	Throttling Only	Proposed
MPI.Alltoall	13	10	18.5
MPI.Allgather	14	7	16.0
Applications	7	3	11.0

Energy Delay Product (EDP). Table 6.2 shows the EDP reductions for different energy saving strategies in all the experiments. For the all-to-all and allgather benchmarks, the averages were taken over various message sizes in Fig. 6.2 and 6.3; and they were computed across all the test cases of the three applications (CPMD, NAS, and Elemental) in Fig. 6.4 and 6.5. Observe that the EDP reduction is always higher for the *Proposed* strategy. Thus, it appears the winner among the three strategies and improves considerably the energy efficiency of the system.

Validation of the Proposed Power Consumption Estimate. Since the Infiniband card installed in Dynamo draws no significant power⁷, its consumption may be ignored. Hence, inter- and intranode all-to-all algorithmic steps are considered as consuming approximately equal power. By substituting $\bar{P}_{1,7} = 160W$, the “memory-only” $P_d = 10W$, and $P_{1,0} = 220$ for 1 MB message from Fig. 6.2(right) into Equation (5.6), the power $P_{1,7}$ of the entire node at f_1 and T_7 may be calculated as 176 watts. This value is close to the experimental findings

⁷http://mellanox.com/related-docs/user_manuals/IH3Lx_PCIE4_HCA__user_manual_1_05.pdf

Table 6.3 Application characterizations by the runtime procedure. (The average call and phase lengths, **CallLen** and **PhaseLen**, are given in microseconds.)

Name	CallLen	NumSeq	NumPhase	PhaseGrp	PhaseLen
MG	423.07	1	7	1	5,987
CG	87.32	1	1	1	29,991
Silatrane-265	550.00	1	7	3	175,918
Silatrane-301	700.00	1	12	4	188,206

as evident from Fig. 6.2(right) for the messages of 1 MB in the curve corresponding to the **Proposed** strategy.

6.2 Runtime System

6.2.1 Phase Characterization of the Applications Tested

Table 6.3 depicts the results for the 0th MPI rank obtained when the procedure was executed on Dynamo with 32 MPI processes. (Similar phases were characterized on the other ranks.) Observe that the average blocking point-to-point call duration (column **CallLen**) is much higher for MG, Silatrane-265, and Silatrane-301 than for CG. Thus, it could be inferred that the frequency scaling, if applied separately to each call in CG, will produce a higher performance loss than when the same is done in for the other three test cases. (The performance results shown later (section 6.3.3) confirm this supposition.) All the applications listed have a single initial sequence per communicating process (column **NumSeq**). While CG provides only a single phase throughout its execution, all other applications exhibit multiple phases (column **NumPhase**), which are distinguished by their different message sizes. The number of phase groups is shown in column **PhaseGrp**. The average phase length (column **PhaseLen**) of all the applications is much larger than the DVFS switching overhead for Dynamo, which experimentally was determined to be between 17 and 26 μs and to be larger for upscaling than for downscaling frequency transitions (as suggested also in [41]). Thus, this overhead may be amortized when the DVFS is applied on the per-phase rather than percall basis.

6.2.2 NAS Benchmarks: Energy Savings with DVFS

The effects of applying DVFS through several frequency scaling strategies have been observed for the NAS benchmarks first. In addition to the three frequency scaling strategies described in section 3.1.4, the *percall* one was evaluated. In the latter, the frequency scaling is applied to *each* MPI_Send and MPI_Recv call, such that the frequency is reduced at its start to the level prescribed as suggested in the authors' previous work [58] and restored to its highest level when the call ends. These four frequency scaling strategies are also compared with a scheme known as CPU Miser [18].

The CPU Miser technique divides the execution of an application into intervals (time slices) of a particular duration (typically 250 μ s) and predicts the execution characteristics, such as memory stalls, of the upcoming interval based on recent intervals. The CPU Miser strategy primarily depends on the memory accesses, even though it may use the I/O and idle times (provided by the `/proc/stat` file in Linux) to choose a suitable frequency for a given time slice. CPU Miser provides for a user to define the performance loss, given which it attempts to save energy; and it has been shown to attain significant energy gains [18]. Similarly to the proposed work, CPU Miser requires no modification to the application source code. However, it does not consider the Infiniband CPU offloading.

Figure 6.6(a) shows the execution time for the MG and CG NAS benchmarks, normalized to the case when these benchmarks are executed at the highest frequency of the processor. It can be observed that, among all the strategies, the conservative one has the least (about 1%) and the *percall* has the highest (about 5.5%) average performance degradation for the two benchmarks. The *percall* strategy incurs a huge DVFS switching overhead, which is accumulated over the DVFS applications to every call. On the other hand, the conservative strategy has only two DVFS switches per phase (see section 6.3.1 for the phase counts and call lengths) and, therefore, minimizes the DVFS overhead. The average performance loss for the intermediate, CPU Miser, and aggressive strategies is approximately 2% each, which is well below the chosen threshold of 10%.

Figure 6.6(b) depicts the respective normalized energy consumption. The *percall* strategy

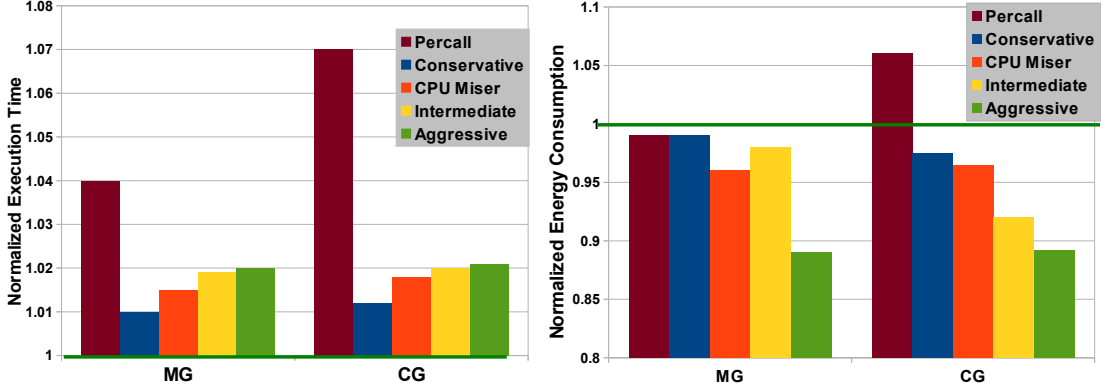


Figure 6.6 Execution time and energy consumption of the NAS MG and CG benchmarks for the different DVFS strategies normalized to the case when all the processes operate at the highest frequency. (Results below 1 are better.)

saves 1% of energy for the MG benchmark but increases the energy consumption of the CG benchmark by 5%. The conservative strategy schedules all the phases at 2.66 GHz and saves 2.5% energy for the CG benchmark but only 1% for MG. This is because CG exhibits an overall large value of the phase overlap ($\phi_o > 90\%$) whereas MG has a lower phase overlap ($\phi_o < 30\%$), as determined experimentally for different twin-core pairs. Since the DVFS is applied only to the phase overlaps in twin cores, it is proved more effective for the CG benchmark.

The CPU Miser schedules the whole execution at 2.66 GHz for both benchmarks, owing to their uniform memory access pattern throughout their execution, and reduces their energy consumption by approximately 3.6%. Even though the conservative and CPU Miser strategies choose the same frequency, CPU Miser saves more energy since it covers the whole execution time, including the interphase gaps. Additionally, the CPU Miser applies DVFS uniformly across the twin cores, which means a near 100% phase overlap.

After determining the architectural stalls in the intraphase time gaps, the intermediate strategy picks the lowest frequency (2 GHz) as the suitable frequency f^* for each phase of the two benchmarks. It saves 8% of energy for the CG benchmark, which is more than what was achieved by CPU Miser, but saves only 1.5% for MG due to poor phase overlap.

The aggressive strategy schedules the whole execution at 2 GHz. Since it picks a suitable

(typically less than the maximum) frequency for the interphase gaps as well—essentially creating a continuous execution coverage by the frequency downscaling—the problem of the low phase overlap is mitigated in applications, such as MG. Specifically, the aggressive strategy saves energy close to 11% for either benchmark and thus, provides the most energy savings among all the strategies on the computing platform considered here. To assess this amount of energy savings in absolute terms, note that the power consumption of a Dynamo node at the highest and lowest P-states of 3 GHz and 2 GHz, respectively, equals to around 270 and 234 watts in the experiments conducted here. Hence, the theoretical peak of energy savings is 13%, which is rather close to the 11% obtained with the aggressive strategy. The latter also results in a minimal performance loss of 2%, hereby confirming that a runtime procedure is possible (in lieu of a static analysis of the application performance) to arrive at an optimal frequency.

6.2.3 GAMESS: Energy Savings with DVFS

Before analyzing the energy efficiency and performing frequency scaling of the three binding types considered here, they were compared with each other in terms of their execution times at the highest frequency. It was observed that the execution time decreased as compute processes were spread more evenly across the two sockets when using the bindings other than Disjoint-I. In particular, there was a decrease in execution time of about 1.8% and 4.5% for the Disjoint-II and Slave, respectively, as compared to the Disjoint-I binding. This can be attributed to the fact that, in the Slave binding, there is little contention to access the L2 cache by the compute processes because they are paired with data servers, which do not use L2 cache significantly. Being the default, Disjoint-I at the highest processor frequency is chosen as the baseline case for comparing other bindings at different frequencies in the rest of the paper.

Disjoint-I Figure 6.7(a) depicts the execution time of the two Silatrane computations for the various frequency scaling strategies in the Disjoint-I process mapping, normalized to the baseline case. It may be observed that the execution time never exceeds the 10% performance threshold for all the frequency scaling strategies: The performance loss varies from 0.3% to 6.6% for the two silatrane computations.

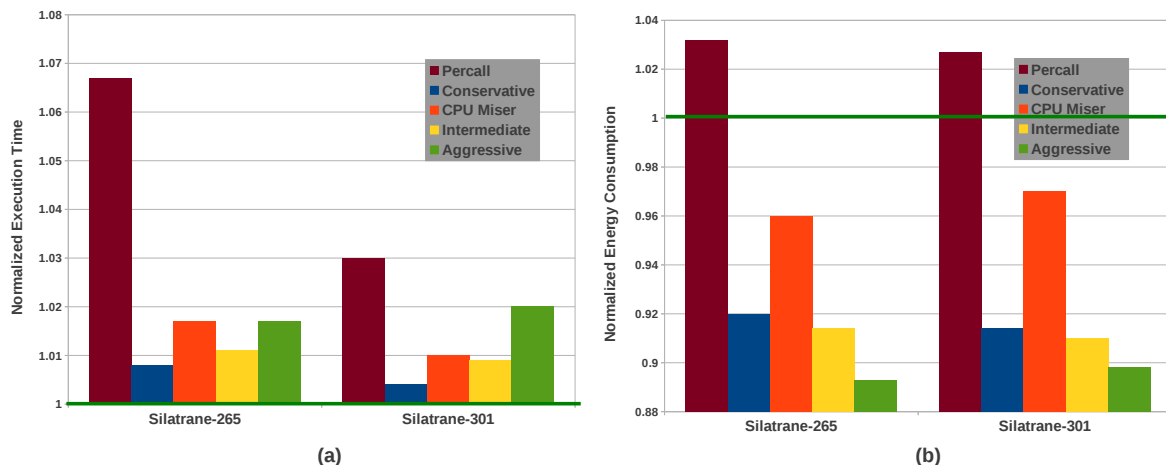


Figure 6.7 (a) Execution time and (b) energy consumption of the two Silatrane computations under **Disjoint-I** process mapping for the different DVFS strategies, normalized to the case when Disjoint-I binding is operated at the highest frequency. (Results below 1 are better.)

Figure 6.7(b) shows the corresponding normalized energy consumption of the two molecule computations. As in the case of the NAS benchmarks, the percall strategy increases the energy consumption of both the molecule computations due to excessive frequency scaling overhead and poor phase overlap associated with the GAMESS execution. The CPU Miser strategy schedules the execution at 2.66 GHz on all the cores except for the compute-intensive gradient calculation, at which it again increases the frequency to the maximum of 3 GHz. Therefore, it ends up saving 4% of energy for either molecule computation. Since the data servers are operated at 2 GHz throughout the execution and compute processes undergo per-phase frequency scaling, all the proposed strategies—conservative, intermediate, and aggressive—save more energy than CPU Miser does. Specifically, the average energy savings are 7%, 7.5%, and 10.2% for conservative, intermediate, and aggressive, respectively.

Slave For the Slave binding, the execution time is reduced with respect to the baseline case for all the frequency scaling strategies but for the percall one, as seen in the results presented in fig. 6.8(a). Recall that this decrease is due to a more even distribution of the compute processes across the node and, thus, less contention for the L2 cache. As far as energy saving is concerned for the Slave binding (fig. 6.8(b)), the CPU Miser saves more energy (7%)

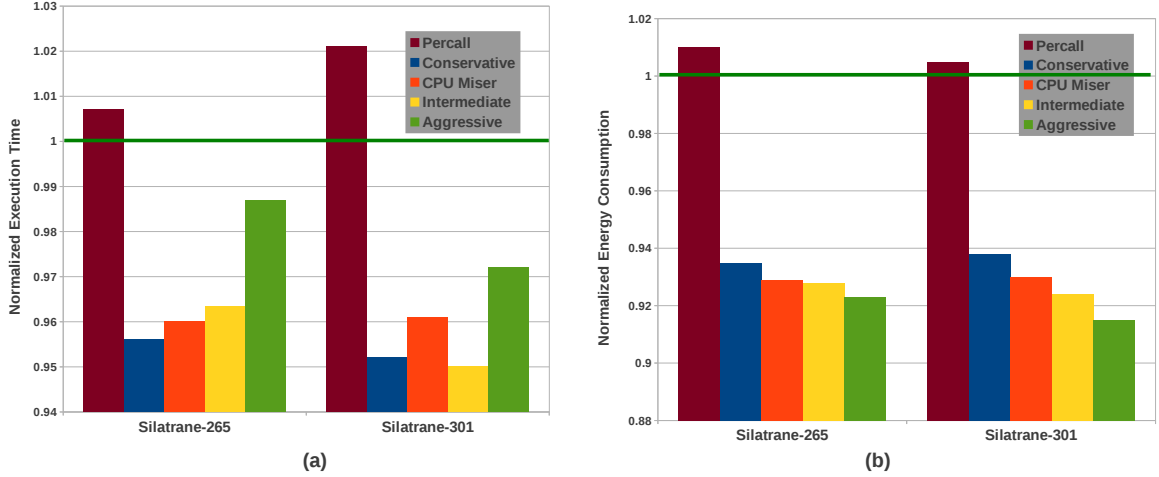


Figure 6.8 (a) Execution time and (b) energy consumption of the two Silatrane computations under **Slave** process mapping for the different DVFS strategies, normalized to the case when Disjoint-I binding is operated at the highest frequency. (Results below 1 are better.)

than the conservative strategy does (6%) because the former also benefits from the decrease in the execution time offered by the Slave binding while its power consumption remains uniformly low among all the cores, irrespective of the binding strategy used. The intermediate and aggressive strategies, however, save more energy (7.2 and 8%, respectively) than the CPU Miser does because they choose the lowest frequency of 2 GHz for the most part of the compute process execution except when the compute-intensive tasks are performed, such as the gradient calculation, which amounts to about 25% of the execution time.

Disjoint-II The normalized execution time under the Disjoint-II process mapping for different frequency scaling strategies can be observed in fig. 6.9(a). Except for the percall strategy, all the other strategies enjoy virtually no performance degradation. The percall strategy increases the energy consumption by 1% whereas the CPU Miser saves 6% energy as seen from fig. 6.9(b). The conservative, intermediate, and aggressive strategies decrease the energy consumption by 8%, 8.6%, and 10%, respectively.

Comparison of the Disjoint Variants and Slave Mappings The energy savings obtained for the three proposed frequency scaling strategies under the Disjoint bindings are higher than those for the Slave because the data servers are always operated at 2 GHz throughout the

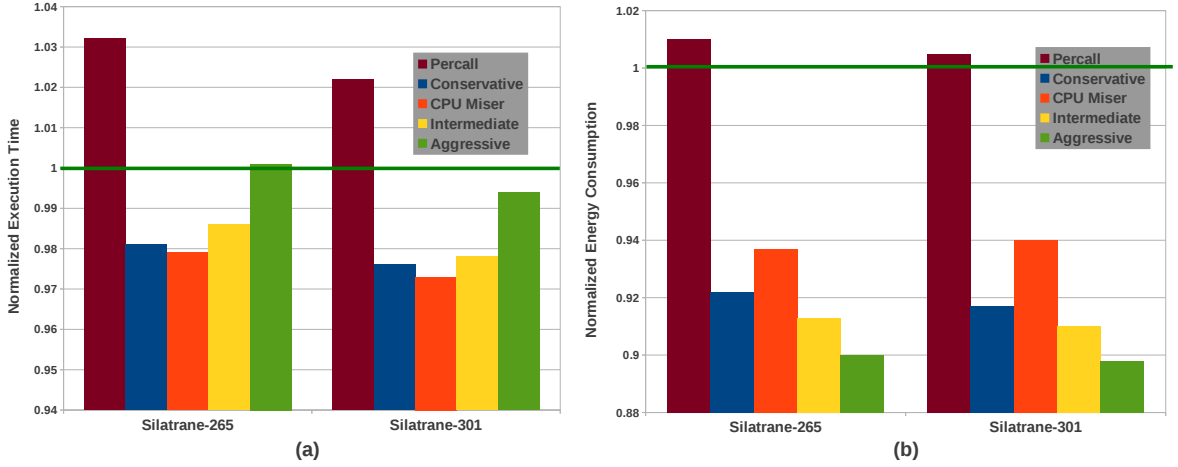


Figure 6.9 (a) Execution time and (b) energy consumption of the two Silatrane computations under **Disjoint-II** process mapping for the different DVFS strategies, normalized to the case when Disjoint-I binding is operated at the highest frequency. (Results below 1 are better.)

Table 6.4 Energy-delay product (EDP) values for the three proposed bindings under the aggressive frequency scaling strategy normalized to the EDP of the Disjoint-I binding operated at the highest frequency.

Name	Disjoint-I	Disjoint-II	Slave
Silatrane-265	0.908	0.900	0.911
Silatrane-301	0.915	0.892	0.895

execution in the Disjoint unlike in the Slave, where they follow the compute processes operating at the highest frequency during the gradient calculation.

Table 6.4 depicts the normalized Energy-delay product (EDP) [37] for the two silatrane computations for aggressive frequency scaling strategy under three different bindings. There is a close competition between Disjoint-I and Disjoint-II as far as energy saving is concerned since both save a maximum of 10% with the aggressive frequency scaling strategy. However, on the basis of the EDP metric, Disjoint-II is marginally better than Disjoint-I since it provides the energy savings with a slight improvement in performance over Disjoint-I.

In MVAPICH2, the default (bunch) process mapping is implemented as Disjoint-I and the other one (scatter) as Slave. The Disjoint-II binding was achieved by giving suitable values to the run-time variable `MV2_CPU_MAPPING` used along with the `mpirun` command in MVAPICH2.

6.2.4 DVFS versus CPU Throttling

Even though CPU throttling is supported on a per-core granularity as opposed to DVFS, which is supported on a twin-core granularity, its power saving capabilities are inferior to those of DVFS. Throttling does not reduce the actual clock rate but rather inserts the STPCLK (stop clock) signals, thereby omitting the duty cycles and does not change the operating voltage of the cores. Moreover, it results in considerable performance loss as compared to DVFS [58] for the intranode communications. For example, any frequency reduction in the GAMESS data-server processes has to be done with DVFS as opposed to throttling because they perform a considerable amount of intranode communications of the globally distributed arrays, so throttling would incur a substantial switching overhead, as has been observed by the authors in [53].

Consider an example of the throttling usage in MG and CG benchmarks. The reduction in power consumption at T-states T_1 and T_2 is 2% and 5%, respectively, across a node for each benchmark with the achieved operational frequency for these states of 2.625 and 2.25 GHz. Observe that these decreases in power consumption are much lower than those provided by DVFS. When different frequency scaling strategies were tested using the CPU throttling only, the total energy consumption actually increased at times because the power consumption remained rather high while the performance loss was significant, up to the allowed 10% in the percall strategy that incurs the largest number of frequency switches. However, if an application exhibits poor phase overlap among the twin cores or the phases consist mostly of the internode communications—the two conditions under which DVFS is not effective—throttling may give more savings than DVFS.

6.3 Modified Runtime System

6.3.1 Characterization of the Applications Tested

The experiments were executed on 32 MPI processes, one per core of four Dynamo nodes, using the lightweight tool MPE [62] with the MPI profiling layer PMPI [45]. (MPE introduced an overhead of less than 1% for the NAS benchmarks.) The NAS class C benchmarks along with

the CPMD and pARMS applications are used to verify the operation of the proposed runtime phase-detection mechanism and the efficacy of the proposed frequency-scaling strategies.

6.3.1.1 pARMS

pARMS is a library of parallel solvers for distributed sparse linear systems of equations. It is based on a preconditioned Krylov subspace approach, using a domain decomposition viewpoint. The plural in "Solvers" is due to the fact that pARMS offers a large selection of preconditioners for distributed sparse linear systems and a few of the best known accelerators. The basic methodology used relies on a Recursive Multi-level ILU factorization which allows to develop many of the standard domain-decomposition type iterative solvers in a single framework. For example, the standard Schwarz procedures are included as are a number of Schur complement techniques.

6.3.1.2 CPMD

CPMD is an ab initio electronic structure and molecular dynamics (MD) program using a plane wave/pseudopotential implementation of density functional theory (DFT). It is mainly targeted at Car-Parrinello MD simulations, but also supports geometry optimizations, Born-Oppenheimer MD, path integral MD, response functions, QM/MM, excited states and calculation of some electronic properties.

6.3.1.3 Application Phase Characterization

Five inputs from NAS (MG, CG, FT, and IS), three inputs from CPMD (wfopt-davidson, wfopt-lanczos, wat32-inp2), and two from pARMS (two 2D regular grids having total 5.12 and 20.48 million nodes each) are used in this work. Table 6.5 depicts the results of the phase-detection process in the 0th MPI rank when the tests were run with 32 MPI processes on Dynamo. (Similar phases were characterized on other ranks.) Note that the input cases for CPMD and pARMS are termed in table 6.5 as CPMD-m1, CPMD-m2, CPMD-p2, and as pARMS-g1 and pARMS-g2, respectively. It was observed that, for all the test cases but CG, the average call duration (column `CLen`) is much larger than the DVFS overhead, which is on

Table 6.5 Characterization of NAS, CPMD, and pARMS tests (column **TName**) by the proposed runtime system. The number of phases detected is shown in column **PhN**. The average call and phase lengths, **CLen** and **PhLen**, are in microseconds, respectively. The column **CTypes** gives the MPI call types as observed in the phases.

TName	CLen	PhN	PhLen	CTypes
MG	298.7	11	8,587	Allreduce, Send, Bcast
CG	75	1	21,240	Send
FT	818,664	1	9,178,124	Alltoall
IS	321	1	595,619.9	Alltoall, Alltoallv, Bcast
CPMD-m1	2,148.5	105	70,000	Alltoall, Recv, Bcast
CPMD-m2	2,417.6	102	110,000	Alltoall, Recv, Bcast
CPMD-p2	2,500	25	270,000	Alltoall, Recv, Bcast
pARMS-g1	497.1	1	404,412	Allreduce
pARMS-g2	3,113	1	1,667,656.1	Allreduce

the order of several microseconds. Thus, the frequency scaling, if applied separately to each call in CG, will produce a higher performance loss than when the same is done for the rest of the test cases. The performance results shown later (section 6.3.3) confirm this supposition.

Except for MG and the three CPMD inputs, a single phase (column **PhN**) was detected for the rest of the inputs and the average duration (column **PhLen**) of a phase was much higher than the amount of the DVFS overhead. Therefore, this overhead is well amortized even when the frequency scaling is applied to the determined phases only, as in the DVFS-Ph strategy. The phase composition from the MPI call types (column **CTypes**) reveals that, in the MG and CG benchmarks, much of the phase communication is due to the MPI_Send calls, which does not allow for efficient application of throttling unless large message sizes are used. Another observation is that all the test cases, but MG, CG, and pARMS, have the collective operations that include no computation. For example, FT, IS, and CPMD use heavily MPI_Alltoall. Conversely, the pARMS phase comprises MPI_Allreduce, which is a compute-intensive collective operation. Therefore, pARMS is expected to be quite sensitive to frequency scaling.

6.3.2 CPU and Memory Power Consumption

The static power consumption values are 66 watts and 22 watts, for the CPU and memory, respectively. Figure 6.10 depicts the CPU power trace of the CG benchmark, in which the CPU power varies from 93 to 150 watts. Among all the NAS benchmarks, the average CPU

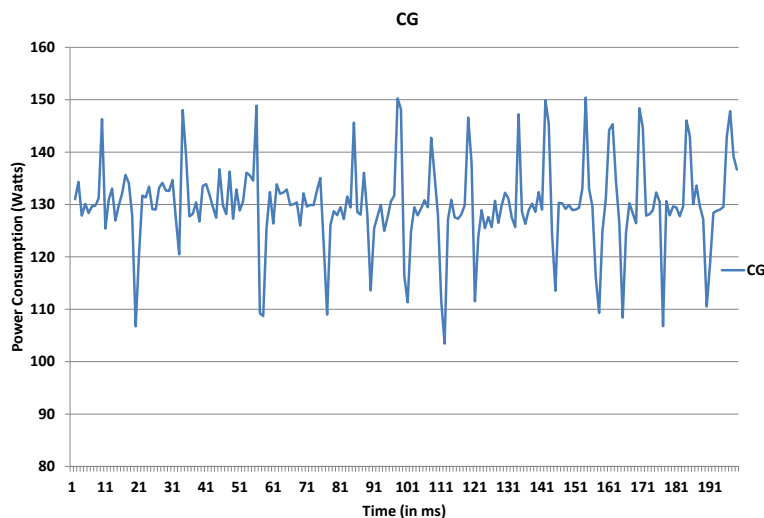


Figure 6.10 CPU power consumption for CG benchmark.

power is ranging from 121 watts (for FT) to 136 watts (for MG) and the memory power is 41 watts. The average CPU power at the various frequency levels for the benchmarks is 135 watts (at 3.0 GHz), 123 watts (at 2.67 GHz), 110 watts (at 2.33 GHz), and 98 watts (at 2.0 GHz). The total power consumption of a compute node in Dynamo is close to 270 watts (at 3.0 GHz). Therefore, the CPU and memory power consumptions comprise about 50% and 15%, respectively, of the total.

6.3.3 Frequency Scaling with DVFS and Throttling

The effects of applying the frequency scaling through DVFS and throttling for different strategies have been observed for the NAS, pARMS, and CPMD test cases.

In addition to the three frequency scaling strategies described in section 5.2.3, the *percall* one was evaluated. In the latter, the frequency scaling is applied to *each* MPI call, such that the frequency is reduced at its start to the level prescribed as suggested in the authors' previous work [58] and restored to its highest level when the call ends. These four frequency scaling strategies are also compared with a scheme known as CPU Miser [18].

CPU Miser The CPU Miser technique divides the execution of an application into intervals of a particular duration (typically 250 μ s) and predicts the execution characteristics, such as memory stalls, of the upcoming interval, called time slice, based on recent intervals. The CPU Miser strategy primarily depends on the memory accesses, even though it may use the I/O and idle times (provided by the `/proc/stat` file in Linux) to choose a suitable frequency for a given time slice. CPU Miser provides for a user to define the performance loss, given which it attempts to save energy; and it has been shown to attain significant energy gains [18]. Note that, similarly to the proposed work, CPU Miser requires no modification to the application source code. However, it does not consider the Infiniband capabilities of the CPU offloading.

Figure 6.11 shows the execution time for the NAS, pARMS, and CPMD test cases normalized to their execution at the highest frequency of the processor. It can be observed that CPU Miser has the least (about 1.5%) and the DVFS-PhIT has the highest (about 4%) average performance degradation for all the inputs. The DVFS-PhIT strategy incurs a relatively higher frequency scaling overhead due to its use of both throttling and DVFS for saving energy. Since the DVFS-Ph strategy applies the same DVFS level to the entire phase, it may minimize the DVFS overhead across the proposed strategies, especially for relatively long phases and for a small number of phases. The DVFS-PhI strategy, while applying DVFS to a larger part of the application execution, has its performance loss offset by the unnecessary DVFS switches when a phase and its next interphase gap may be operated at the same frequency.

The average performance loss for the percall, DVFS-Ph, and DVFS-PhI strategies is approximately 2.5%, 2.8%, and 3.5%, respectively, which is well below the acceptable threshold of 10%.

Figure 6.12 depicts the respective normalized energy consumption. The percall strategy saves on average 3.1% of energy across all inputs with a maximum of about 8% for the IS benchmark. However, for the MG and CG benchmarks, it increases the energy consumption by 1% and 4%, respectively, owing to their relatively smaller call lengths, and provides little savings of 1% for pARMS.

The CPU Miser schedules the whole execution at 2.66 GHz for almost all the inputs, owing to their uniform memory access pattern throughout their execution, and reduces their energy

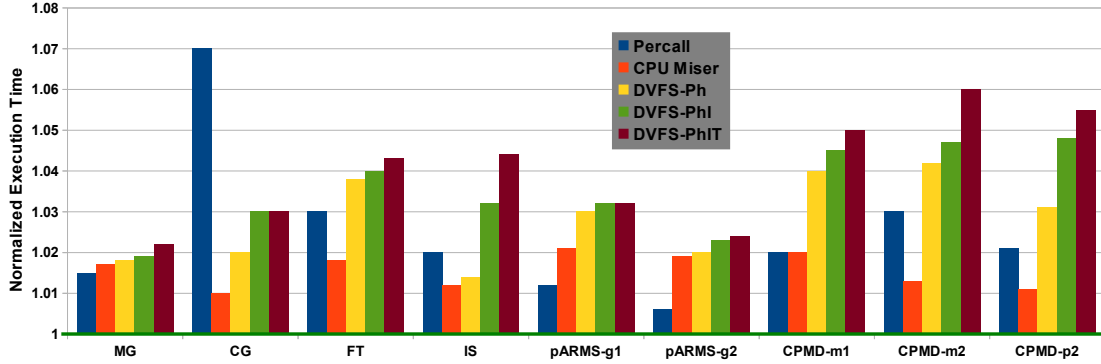


Figure 6.11 Normalized execution time of the NAS, pARMS, and CPMD tests for the different frequency scaling strategies. (Results below 1 are better.)

consumption by approximately 4.5% on average. Even though the percall strategy chooses lower frequency than CPU Miser does so, the latter saves more energy since it covers the whole execution time, including interphase gaps and not just communication calls. Additionally, the CPU Miser applies DVFS uniformly across the twin cores, which means a near 100% phase overlap.

After determining the architectural stalls in the intraphase gaps and considering CPU of-fload of Infiniband, the DVFS-Ph strategy picks the lowest frequency (2 GHz) as the suitable frequency f^* for each phase of all the inputs. It saves 6% of energy on average, which is higher than what was achieved by CPU Miser and the percall frequency application. Specifically, the DVFS-Ph savings range from 3% (for the pARMS-g2 test case) to 9% (for the CPMD-m1).

The DVFS-PhI strategy schedules almost the entire execution at 2 GHz for all the inputs. Since it picks a suitable (typically less than the maximum) frequency for the interphase gaps too, essentially creating a continuous execution cover by the frequency downscaling, the problem of the low phase overlap is mitigated. Specifically, the DVFS-PhI strategy saves close to 9% of energy on average and provides more energy savings than the DVFS-Ph strategy does as the benchmarks are fairly communication and memory intensive.

By carefully applying DVFS and throttling to the inputs, the DVFS-PhIT saves the highest

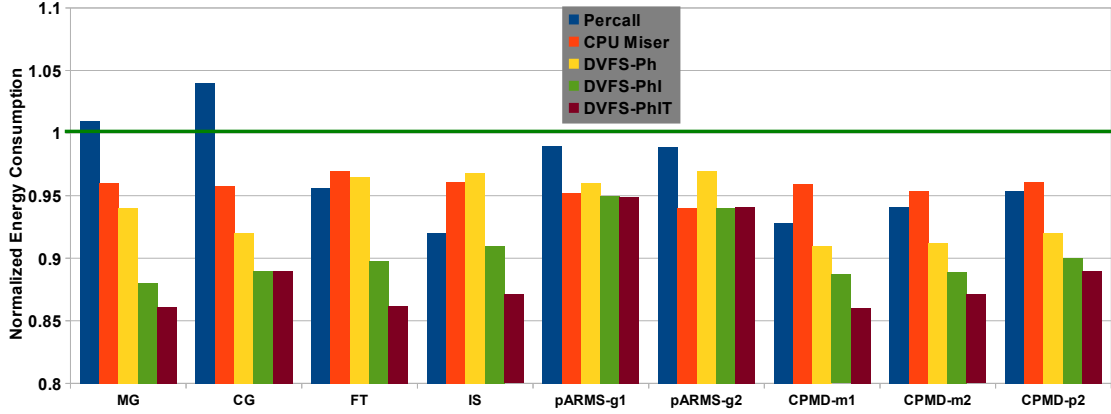


Figure 6.12 Normalized energy consumption of the NAS, pARMS, and CPMD tests for the different frequency scaling strategies. (Results below 1 are better.)

amount of energy (11.2% on average) among all the strategies tested. The CG benchmark undergoes heavy intra-node point-to-point communication. Therefore, CG (as well as pARMS) do not give much opportunity to apply throttling along with DVFS. Note that, for pARMS, only the MPIAllreduce calls were detected (see table 6.5), which contain computations and, thus, are quite sensitive to the CPU throttling. As a result, these three test cases show no significant change in energy saving from the application of the DVFS-PhIT strategy. The MG benchmark exhibits considerable inter-node point-to-point communication and allows for applying throttling level T_1 often, so the DVFS-PhIT strategy saves 14% energy. The minimum and maximum energy savings are 5% and 14% for pARMS-g1 and MG, respectively, with the DVFS-PhIT strategy. Note that the peak of energy savings with DVFS alone is calculated experimentally to be 12.5%, which is rather close to the average of 11.2% and lower than the highest of 14% obtained with the DVFS-PhIT strategy for the MG and CG benchmarks.

In terms of the *CPU energy savings alone*, the lowest and the highest are about 19.5% and 30% for pARMS-g1 and MG, respectively. Even though these numbers seem to suggest that a significant amount of energy has been saved, without considering the *overall* energy saving, they may be misleading. In their previous work [57], the authors have shown that a strategy resulting in some energy saving for the CPU alone can actually lead to an overall increase in

the energy consumption for a compute node. Therefore, the overall energy savings have been emphasized in this work as well.

CHAPTER 7. Related Work

In [27], algorithms to save energy in the collectives, such as MPI_Alltoall and MPI_Bcast, are proposed. They differ significantly with the approach presented in this work. Specifically, [27] views throttling as affecting negatively the internode communication and thus, redesigns the all-to-all operation such that a certain set of sockets does not participate in the communication at some point of time in order to be throttled. However, since the number of cores within a node continues to increase, forcing some sockets to remain idle during the communication can introduce significant performance overheads. The power savings achieved in [27] are equivalent to operating two sockets at the minimum frequency and throttling state T_4 , whereas the approach proposed here achieves power saving by keeping both sockets at the minimum frequency while throttling them to a state T_5 and higher depending on the message size. An experimental comparison of the two algorithms is left as future work.

There are two general approaches to obtaining energy saving during parallel application execution. The first approach is to focus on identifying stalls during the execution by measuring architectural parameters from performance counters as proposed in [18, 22, 23]. Rountree *et. al* [46], apart from using performance counters, do the critical path analysis to determine which tasks may be slowed down to minimize the performance loss in the parallel execution. This analysis appears beneficial when applications have computation or communication imbalances among participating processes, which is typically not the case for a highly efficient parallel application and which was not observed in the NAS benchmarks considered in this work.

The second approach determines the communication phases to apply DVFS as, for example, in [35] and [17]. DVFS is combined in [11] with concurrency throttling on multicore platforms to obtain energy savings. In [27], algorithms to save energy in the collectives, such as MPI_Alltoall

and MPI.Bcast, are proposed. The work in [24] describes a runtime system for the Intel Single-chip Cloud Computer (SCC) processor. This system detects repeatable communication phases followed by an application of frequency scaling. In [36], a detailed comparison of the benefits offered by RDMA versus TCP/IP is given in terms of power efficiency.

The proposed runtime system differs from the related work in several aspects:

- The proposed frequency-scaling strategies are interconnect-aware, i.e., they take into account the Infiniband communication characteristics and its capability to offload the CPU.
- Two computer-architecture parameters are considered by using all the available performance counters rather than just one as done, e.g., in [35].
- Unlike [35], in which a single-core architecture is considered, the present focus is on the multicore architectures that have limited power management in each core and in which core pairing is necessary for the DVFS application. Thus, the communication phase overlap is emphasized here.
- The two frequency-scaling techniques, DVFS and throttling, are clearly distinguished and compared as to their application in various communication calls.
- This work addresses in detail the behavior of various types of communication calls and differentiates among them to apply throttling, rather than considering all communication operations to be the same.
- Communication call parameters, such as rank and message size, along with a unique *callid* are considered to identify the call uniquely instead of using stack addresses.

The energy saving work often aims to reduce the CPU power consumption since the processor consumes more power than other components in the existing computing platforms, as noted in [19, 51], for example. Much research has also focused on exploring system component level power profiling techniques. In [19], authors have developed a tool, PowerPack, which estimates power consumption characteristics of a parallel application in terms of various CPU components. Song *et. al* [51] use PowerPack to perform the energy profiling of the HPCC¹ benchmark. The energy efficiency delivered by the modern interconnects in high performance clusters is discussed in [64]. In [36], a detailed comparison of the benefits offered by RDMA

¹<http://icl.cs.utk.edu/hpcc>

versus TCP/IP is given in terms of their power efficiency. David et. al [12] present a framework for measuring memory power consumption which is featured in the Intel SandyBridge microarchitecture. The power profiling of various components of a processor is done in [7] by making use of the performance counters.

Authors in [61] present a Power Aware One-Sided Communication Library(PASCoL) which detects communication slack, uses Dynamic Voltage and Frequency Scaling (DVFS), and Interrupt driven execution to exploit the detected slack for energy efficiency in one-sided communication primitives. The power profiles of two high performance one-sided dense linear algebra factorizations, namely Cholesky and QR, on distributed memory systems in the context of ScaLAPACK and DPLASMA libraries is studied in [8]. In [33], a novel software-controlled execution scheme that consider the effects of dynamic concurrency throttling (DCT) and dynamic voltage and frequency scaling (DVFS) in the context of hybrid programming models by using predictive models based on statistical analysis is discussed. There are many approaches which use templates and are widespread and used in many existing works which focus basically on the characterization of communication requirements for both application and system exploration. This includes determining commonly used communication patterns [26], creating simplified descriptions for the application [63] etc. By combining many of the principles of templates and using them with energy saving principles the authors in [28] determine energy template for a wavefront algorithm that has a complex processing pattern.

Numerous efforts have been made to introduce energy efficiency in cloud computing as well. In [29], authors proposed a real-time Cloud service framework with power-aware provisioning investigations of virtual machines for real-time Cloud services. Nathuji et. al [39] present a *VirtualPower* approach for online power management which supports the independent operation assumed by guest virtual machines (VMs) executing on the virtualized platforms and controls and globally coordinates the effects of power management policies applied by these VMs to virtualized resources. Kusic et. al [32] have discussed the issue of continuous consolidation as a sequential optimization and have addressed it by proposing a dynamic resource provisioning framework using Limited Lookahead Control (LLC), which requires simulation-based learning for the application specific adjustments. Authors in [52], perform a study which

reveals the energy performance trade-offs for consolidation of applications in cloud computing environment and by modelling the consolidation problem as a heuristical multidimensional bin packing problem. Beloglazov et. al [6] propose a decentralized architecture for energy aware resource management of cloud data centers while minimizing the energy consumption while upholding the QoS requirements.

CHAPTER 8. Conclusions and Future Work

In this work, energy saving strategies are presented for the all-to-all and allgather collective operations implemented in MVAPICH2 without modifying the underlying algorithms. All their MVAPICH2 implementations have been considered. The sensitivity of inter and intranode message transfers to DVFS and CPU throttling has been assessed and the message sizes found such that the chosen performance loss of 10% is not exceeded. It was also observed that applying throttling results in less performance degradation for the internode communications as compared to the intranode ones. These findings lead to the application of DVFS and the appropriate nonzero levels of the CPU throttling within the internode communication algorithmic steps depending on the message size involved while DVFS and no throttling is applied in the case of intranode communication steps. A formula to predict the power consumption of the entire node with DVFS and different levels of CPU throttling has been proposed and then verified experimentally for a particular message size transfer in all-to-all operating on the lowest frequency and the highest throttling level. The experiments have been conducted with NAS benchmarks as well as realistic applications in molecular dynamics (CPMD) and parallel dense linear algebra package (Elemental). The proposed strategies resulted in the highest energy savings with a small performance degradation compared with applying either DVFS or throttling uniformly throughout the application. The energy gains and performance observed in the tests are representative of the potential benefits to scientific applications in general.

Then, a runtime procedure has been proposed to detect communication phases in blocking point-to-point communications. Different strategies have been proposed to select a suitable frequency for the communication phases as well as for the time gaps between the communication calls. For the maximum energy savings, the time gaps are recorded and classified into intranode interphase; and the strategies differ as to their treatment of these time gaps in

point-to-point communications. It has been shown experimentally that to assume these gaps as compute-intensive is too conservative and that there exist good opportunities to lower the frequency during the gaps under a careful investigation of architectural parameters, such as micro-instructions retired and memory accesses. The frequency scaling is performed without any changes to the user application or MPI library. DVFS and CPU throttling were compared as to their hardware-level applicability, energy consumption during communication phases, and switching overheads. Although the DVFS effectiveness depends on the processing core topology within a multicore node, it is much more potent in saving energy than throttling is, especially when used in conjunction with the proposed frequency scaling strategies. Since both the Infiniband offload of the CPU and architectural parameters are analyzed dynamically, more energy savings have been achieved with the more aggressive among the proposed strategies as compared to the existing state-of-the-art techniques, such as CPU Miser, in the test cases considered. For the NAS benchmarks, the aggressive DVFS strategy has resulted in about 11% of savings, which is very close to the maximum achievable energy reduction of 13% in this test-platform combination. For the realistic quantum chemistry package GAMESS, various process-to-core mappings were studied under the proposed three frequency scaling strategies. As a result, up to 10% of energy was saved for the computation of silatrane molecule with as low as 2% of the performance loss.

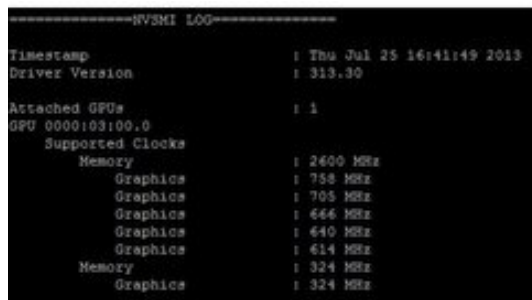
This work also studies the power and energy consumption characteristics of GAMESS when performing SCF calculations in two different ways, direct and conventional. By considering them in stages and using different numbers of processing elements (PEs), it has been observed that the direct mode is more energy efficient although its performance suffers when fewer cores are used and has less potential for the DVFS optimization due to the less time spent off-chip. A general theoretical model for evaluating a DVFS-based optimization is proposed and verified experimentally using GAMESS. The model demonstrates that the energy consumption for the on-chip time, increases with the performance loss increase. Therefore, care must be taken when choosing a performance loss tolerated for energy savings. The ratio of the off-chip and on-chip execution is critical in determining the performance loss. The results emphasize that applying DVFS may actually lead to a higher total energy consumption as compared with always keeping

the highest frequency.

Finally a modified automatic runtime system is proposed which possesses the following novel features: Detects the communication phases transparently in parallel applications; Chooses among three frequency-scaling strategies that differ by their treatment of the communication calls and time gaps between the them; Uses both DVFS and CPU throttling; Calculates the suitable frequency values based on the Infiniband offload and on several architectural parameters obtained from the performance counters; Scales frequency without any changes to the user application or communication (MPI) library. The experiments have shown that up to 14% of energy was saved with a low performance loss of 2% by employing the newly proposed frequency-scaling strategy (called DVFS-PhIT) that calculates a suitable DVFS level within the phases and interphase time gaps as well as selects a proper level of throttling for the communication calls within the detected phases.

8.1 Power Consumption Aware Techniques

Past researches although have dealt with achieving energy efficiency in both CPUs and GPUS, both do so without considering the instantaneous power consumption of these devices. With the result, they choose a particular value of a performance loss and then try to maximize energy saving under that envelope using frequency scaling. This can have adverse results as the energy saving technique becomes completely dependent on the chosen performance loss and the type of workload as shown in a previous work [55]. Modern CPU processors(e.g. SandyBridge) [2] and GPUS provide power draw capabilities which enable users to get the instantaneous power consumption at a high resolution. These capabilities are to be used in the decision making for applying frequency scaling so that the process becomes independent of the chosen performance loss.



```

=====NVSMI LOG=====
Timestamp: Thu Jul 25 16:41:49 2013
Driver Version: 313.30
Attached GPUs: 1
GPU 0000:03:00.0
Supported Clocks
Memory: 2600 MHz
Graphics: 758 MHz
Graphics: 705 MHz
Graphics: 666 MHz
Graphics: 640 MHz
Graphics: 614 MHz
Memory: 324 MHz
Graphics: 324 MHz

```

Figure 8.1 Frequency scaling range for the K20 Tesla GPU obtained through nvidia-smi

8.2 Frequency Scaling in GPUs

Modern GPUs provide capabilities to perform frequency scaling for GPU cores and memory. The NVClock tool¹ enables GPU frequency scaling for GPU series upto GTX 2xx as per the experiments. Therefore, the nvidia-smi utility² is widely used to change the GPU core and memory frequency. Figure 8.1 shows the supported core and memory clocks on a K20 Tesla GPU. It can be observed a very low range of frequencies are supported by the K20. This can actually limit the energy saving capability of a technique e.g in case of a GPU core intensive task, the memory frequency cannot be reduced unless the memory frequency is reduced together with the core frequency. As a result of discussion with NVIDIA experts, it was concluded that a change in BIOS is needed to increase the clock range. Therefore, the current work looks for an option to modify the BIOS of the K20 to enable more frequency levels.

¹NVClock: <http://www.linuxhardware.org/nvclock/>

²nvidia-smi: developer.nvidia.com/nvidia-system-management-interface

BIBLIOGRAPHY

- [1] <https://www.wattsupmeters.com>.
- [2] <http://download.intel.com/products/processor/manual/325384.pdf>.
- [3] P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, and Y.J.J. Wu. Plapack: parallel linear algebra package design overview. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '97, pages 1–16, New York, NY, USA, 1997. ACM.
- [4] M. Annavarami, E. Grochowski, and J. Shen. Mitigating Amdahl’s Law through EPI Throttling. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA’05, pages 298–309, Washington, DC, USA, 2005. IEEE Computer Society.
- [5] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrisnan, and S.K. Weeratunga. The NAS Parallel Benchmarks—Summary and Preliminary Results. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 158–165, 1991.
- [6] A. Beloglazov and R. Buyya. Energy efficient resource management in virtualized cloud data centers. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 826–831, 2010.
- [7] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS’10, pages 147–158, New York, NY, USA, 2010. ACM.

- [8] G. Bosilca, H. Ltaief, and J. Dongarra. Power profiling of cholesky and qr factorizations on distributed memory systems. *Computer Science - Research and Development*, pages 1–9, 2012.
- [9] S. Cho and R. Melhem. Corollaries to amdahl’s law for energy. *IEEE Comput. Archit. Lett.*, 7:25–28, January 2008.
- [10] Kihwan Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(1):18 – 28, 2005.
- [11] M. Curtis-Maury, A. Shah, F. Blagojevic, D.S. Nikolopoulos, B.R. de Supinski, and M. Schulz. Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT ’08, pages 250–259, New York, NY, USA, 2008. ACM.
- [12] H. David, E. Gorbato, U.R. Hanebutte, R. Khannal, and C. Le. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ISLPED’10, pages 189–194, New York, NY, USA, 2010. ACM.
- [13] Y. Dong, J. Chen, X. Yang, C. Yang, and L. Peng. Low power optimization for MPI collective operations. In *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, pages 1047–1052, nov. 2008.
- [14] D.G. Fedorov, R.M. Olson, K. Kitaura, M.S. Gordon, and S. Koseki. A new hierarchical parallelization scheme: Generalized distributed data interface (GDDI), and an application to the fragment molecular orbital method (FMO). *Journal of Computational Chemistry*, 25, Issue 6:872–880, 2004.
- [15] X. Feng, R. Ge, and K.W. Cameron. Power and energy profiling of scientific applications on distributed systems. In *Proceedings of the 19th IEEE International Parallel and Dis-*

- tributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, IPDPS '05, pages 34–, Washington, DC, USA, 2005. IEEE Computer Society.
- [16] G.D. Fletcher, M.W. Schmidt, B.M. Bode, and M.S. Gordon. The Distributed Data Interface in GAMESS. *Computer Physics Communications*, 128(1–2):190–200, 2000.
 - [17] V.W. Freeh and D.K. Lowenthal. Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 164–173, 2005.
 - [18] R. Ge, X. Feng, W. Feng, and K.W. Cameron. CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters. In *Parallel Processing, 2007. ICPP 2007. International Conference on*, page 18, Sep. 2007.
 - [19] R. Ge, X. Feng, S. Song, H.C. Chang, D. Li, and K.W. Cameron. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21:658–671, 2010.
 - [20] M.S. Gordon and M.W. Schmidt. Advances in Electronic Structure Theory: GAMESS a Decade Later. *Theory and Applications of Computational Chemistry: the first forty years*, C.E.Dykstra, G.Frenking, K.S.Kim, G.E.Scuseria (Editors), 2005.
 - [21] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.
 - [22] C.H. Hsu and W. Feng. A Power-Aware Run-Time System for High-Performance Computing. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, page 1, nov 2005.
 - [23] S. Huang and W. Feng. Energy-Efficient Cluster Computing via Accurate Workload Characterization. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 68–75, May 2009.
 - [24] N. Ioannou, M. Kauschke, M. Gries, and M. Cintra. Phase-Based Application-Driven Hierarchical Power Management on the Single-chip Cloud Computer. In *Parallel Architectures*

- and Compilation Techniques (PACT)*, 2011 International Conference on, pages 131–142, oct. 2011.
- [25] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 93–, Washington, DC, USA, 2003. IEEE Computer Society.
 - [26] G. Johnson, D.K. Kerbyson, and M. Lang. Optimization of infiniband for scientific applications. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, 2008.
 - [27] K. Kandalla, E.P. Mancini, S. Sur, and D.K. Panda. Designing Power-Aware Collective Communication Algorithms for InfiniBand Clusters. In *Parallel Processing (ICPP), 2010 39th International Conference on*, pages 218–227, 2010.
 - [28] D. J. Kerbyson and K. J. Barker. Automatic identification of application communication patterns via templates. In *ISCA PDCS'05*, pages 114–121, 2005.
 - [29] K. H. Kim, A. Beloglazov, and R. Buyya. Power-aware provisioning of cloud resources for real-time services. In *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '09, pages 1:1–1:6, New York, NY, USA, 2009. ACM.
 - [30] W. Kim, M.S. Gupta, G. Wei, and D. Brooks. System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators. In *in International Symposium on High-Performance Computer Architecture*, 2008.
 - [31] R. Kotla, S. Ghiasi, T. Keller, and F. Rawson. Scheduling Processor Voltage and Frequency in Server and Cluster Systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 11 - Volume 12*, IPDPS '05, pages 234.2–, Washington, DC, USA, 2005. IEEE Computer Society.
 - [32] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In

- Proceedings of the 2008 International Conference on Autonomic Computing, ICAC '08*, pages 3–12, Washington, DC, USA, 2008. IEEE Computer Society.
- [33] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron. Strategies for energy-efficient resource management of hybrid programming models. *IEEE Trans. Parallel Distrib. Syst.*, pages 144–157, 2013.
 - [34] Z. Li, Y. Saad, and M. Sosonkina. pARMS: A parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications*, 10:485–509, 2003.
 - [35] M.Y. Lim, V.W. Freeh, and D.K. Lowenthal. Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006.
 - [36] J. Liu, D. Poff, and B. Abali. Evaluating High Performance Communication: a Power Perspective. In *Proceedings of the 23rd International Conference on Supercomputing*, pages 326–337, 2009.
 - [37] A.J. Martin, M. Nyström, and P.I. Péntzes. *ET²: A Metric for Time and Energy Efficiency of Computation*, pages 293–315. Kluwer Academic Publishers, Norwell, MA, USA, 2002.
 - [38] M. Martonosi, S. Malik, and F. Xie. Efficient behavior-driven runtime dynamic voltage scaling policies. In *Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on*, pages 105–110, sept. 2005.
 - [39] R. Nathuji and K. Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, SOSP '07*, pages 265–278, New York, NY, USA, 2007. ACM.
 - [40] R.M. Olson, M.W. Schmidt, M.S. Gordon, and A.P. Rendell. Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model. In *SC*, page 41, 2003.
 - [41] J. Park, D. Shin, N. Chang, and M. Pedram. Accurate Modeling and Calculation of Delay and Energy Overheads of Dynamic Voltage Scaling in Modern High-Performance

- Microprocessors. In *2010 International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 419–424, 2010.
- [42] F. Petrini, D.J. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of *asci q*. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, SC '03, pages 55–, New York, NY, USA, 2003. ACM.
- [43] J. Poulson, B. Marker, J.R. Hammond, N. A. Romero, and R. van de Geijn. Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software*, 2011. submitted.
- [44] J.K. Puri, R. Singh, and V.K. Chahal. Silatranes: a review on their synthesis, structure, reactivity and applications. *Chem. Soc. Rev.*, 40:1791–1840, 2011.
- [45] R. Rabenseifner. Automatic Profiling of MPI Applications with Hardware Performance Counters. In *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 35–42, London, UK, 1999. Springer-Verlag.
- [46] B. Rountree, D.K. Lownenthal, B.R. de Supinski, M. Schulz, V.W. Freeh, and T. Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd international conference on Supercomputing*, ICS'09, pages 460–469, New York, NY, USA, 2009. ACM.
- [47] M. W. Schmidt, K.K. Baldridge, J.A. Boatz, S.T. Elbert, M.S. Gordon, J.H. Jensen, S. Koseki, N. Matsunaga, K.A. Nguyen, S. Su, T.L. Windus, M. Dupuis, and Jr. J.A. Montgomery. General Atomic and Molecular Electronic Structure System. *J. Comput. Chem.*, 14:1347–1363, November 1993.
- [48] Scientific grand challenges: Crosscutting technologies for computing at the exascale. In *U.S. Department of Energy sponsored workshop*, Washington, DC, Feb. 2010.

- [49] S. Siddha, V. Pallipadi, and A. Van De Ven. Getting maximum mileage out of tickless. In *Proceedings of the Linux Symposium*. Intel Open Source Technology Center, June 2007.
- [50] S. Sok and M.S. Gordon. A dash of protons: A theoretical study on the hydrolysis mechanism of 1-substituted silatranes and their protonated analogs. *Computational and Theoretical Chemistry*, 987(0):2–15, 2012.
- [51] S. Song, R. Ge, X. Feng, and K.W. Cameron. Energy Profiling and Analysis of the HPC Challenge Benchmarks. *Int. J. High Perform. Comput. Appl.*, 23:265–276, August 2009.
- [52] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower’08, pages 10–10, Berkeley, CA, USA, 2008. USENIX Association.
- [53] V. Sundriyal and M. Sosonkina. Per-call Energy Saving Strategies in All-to-all Communications. In *Proceedings of the 18th European MPI Users’ Group conference on Recent advances in the message passing interface*, EuroMPI’11, pages 188–197, Berlin, Heidelberg, 2011. Springer-Verlag.
- [54] V. Sundriyal, M. Sosonkina, and A. Gaenko. Runtime Procedure for Energy Savings in Applications with Point-to-point Communications. In <http://archives.ece.iastate.edu/archive/00000622/>.
- [55] V. Sundriyal, M. Sosonkina, and A. Gaenko. Energy Efficient Communications in Quantum Chemistry Applications. In *International Conference on Energy-Aware High Performance Computing*, 2012.
- [56] V. Sundriyal, M. Sosonkina, and A. Gaenko. Runtime procedure for energy savings in applications with point-to-point communications. In *to appear in International Symposium on Computer Architecture and High Performance Computing*, 2012,.
- [57] V. Sundriyal, M. Sosonkina, F. Liu, and M.W Schmidt. Dynamic frequency scaling and energy saving in quantum chemistry applications. In *Proceedings of the 2011 IEEE Inter-*

- national Symposium on Parallel and Distributed Processing Workshops and PhD Forum, IPDPSW '11*, pages 837–845, Washington, DC, USA, 2011. IEEE Computer Society.
- [58] V. Sundriyal, M. Sosonkina, and Z. Zhang. Achieving Energy Efficiency during Collective Communications. *Concurrency and Computation-Practice and Experience*, in press.
 - [59] R. Thakur and R. Rabenseifner. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 19:49–66, 2005.
 - [60] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 2001.
 - [61] A. Vishnu, S. Song, A. Marquez, K. Barker, D. Kerbyson, K. Cameron, and P. Balaji. Designing energy efficient communication runtime systems: a view from pgas models. *J. Supercomput.*, 63(3):691–709, March 2013.
 - [62] C. E. Wu, A. Bolmarich, M. Snir, D. Wootton, F. Parpia, A. Chan, E. Lusk, and W. Gropp. From Trace Generation to Visualization: A Performance Framework for Distributed Parallel Systems. In *Proc. of SC2000: High Performance Networking and Computing*, November 2000.
 - [63] Q. Xu, J. Subhlok, Z. Rong, and S. Voss. Logicalization of communication traces from parallel execution. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, IISWC '09, pages 34–43, Washington, DC, USA, 2009. IEEE Computer Society.
 - [64] R. Zamani, A. Afsahi, Y. Qian, and C. Hamacher. A Feasibility Analysis of Power-Awareness and Energy Minimization in Modern Interconnects for High-Performance Computing. In *Proceedings of the 2007 International Conference on Cluster Computing*, pages 118–128, 2007.