Exploring Energy Behaviors of I/O Management Approaches for Exascale Systems

Orcun YILDIZ

Dissertação para obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Orientador: Doutor José Carlos Campos Costa

Júri

Presidente: Doutor Luis Rodrigues
Orientador: Doutor José Carlos Campos Costa
Vogal: Doutor José Carlos Monteiro

Julho de 2014
Acknowledgments

Firstly, I would like to thank my supervisor Prof. José Costa for the continuous support and encouragement he has provided all throughout the project.

Secondly, I would like to thank my internship advisor Dr. Shadi Ibrahim for his support throughout the project. Assisted with his invaluable comments, I was able to successfully conclude this study.

I also acknowledge Matthieu Dorier for his great help regarding the I/O approaches in HPC. Special thanks goes to Dr. A. Cecile Orgerie for her insights into designing the energy model.

I would like to present my special appreciation to KerData team leader Dr. Gabriel Antoniu for his support and trust in my work.

Lastly, I would like to thank my family for their valuable support all throughout my life.

This work was done in the KerData team INRIA (Rennes, France). The experiments presented in this thesis were carried out using the Grid’5000/ALADDIN-G5K experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see http://www.grid5000.fr/ for details)
Abstract

The advent of fast, unprecedentedly scalable, yet energy-hungry exascale supercomputers poses a major challenge consisting in sustaining a high performance per watt ratio. While much recent work has explored new approaches to I/O management, aiming to reduce the I/O performance bottleneck exhibited by HPC applications (and hence to improve application performance), there is comparatively little work investigating the impact of I/O management approaches on energy consumption.

In this work, we explore how much energy a supercomputer consumes while running scientific simulations when adopting various I/O management approaches. We closely examine three radically different I/O schemes: time partitioning, dedicated cores, and dedicated nodes. We investigate the energy/performance trade-offs of the three approaches within the Damaris I/O middleware and perform extensive experiments with one of the target HPC applications of the Blue Waters sustained-petaflop supercomputer project: the CM1 atmospheric model. Our experimental results obtained on the French Grid'5000 platform highlight the differences between these three approaches and illustrate in which way various configurations of the application and of the system can impact performance and energy consumption. Based on our experimental results, we designed an energy model which can estimate the energy consumptions of different I/O management approaches and help users in choosing the optimal I/O approach prior to running their application. We have experimentally demonstrated that our model can select the best I/O approach in terms of energy efficiency regardless of the system’s architecture or application specific parameters.

Keywords

Exascale, Data Management, Damaris, Energy, Performance, Dedicated Resources
Resumo

O advento de supercomputadores, com uma escalabilidade sem precedentes, mas com grandes requisitos de energia, representa um grande desafio que consiste na manutenção de um alto desempenho por energia gasta. Enquanto o trabalho mais recente tem explorado novas abordagens para a gestão de E/S, com o objetivo de aumentar o seu desempenho nas aplicações de HPC (e, consequentemente, para melhorar o desempenho da aplicação), há relativamente pouco trabalho que investigue o impacto das abordagens da gestão de E/S no consumo de energia.

Neste trabalho, exploramos quanta energia um supercomputador consome durante a execução de simulações científicas ao adotar várias abordagens na gestão das E/S. São examinados três soluções de E/S radicalmente diferentes: particionamento no tempo, núcleos dedicados, e nós dedicados. Investigamos as relações energia/desempenho das três aproximações usando o middleware de E/S Damaris e realizamos extensas experiências usando uma das aplicações HPC do "Blue Waters sustained-petaflop supercomputer project": o modelo atmosférico CM1. Os nossos resultados experimentais obtidos na plataforma francesa Grid’5000 mostram as diferenças entre essas três abordagens e ilustram de que forma várias configurações da aplicação e do sistema podem afetar o desempenho e consumo de energia. Com base nos nossos resultados experimentais foi elaborado um modelo energético que pode estimar os consumos de energia usando diferentes abordagens na gestão de E/S e ajudar os utilizadores na escolha da melhor abordagem de E/S antes de executar a sua aplicação. Demonstramos ainda experimentalmente que com o nosso modelo é possível selecionar a melhor abordagem de E/S em termos de eficiência energética, independentemente da arquitetura ou de parâmetros específicos do sistema.

Palavras Chave

Exaescala,Gestão de Dados,Damaris,Energia,Performance,Recursos Dedicados
## List of Figures

2.1 Approaches to I/O management in HPC applications ........................................... 7

3.1 Damaris approach [8] ........................................................................................................ 14

3.2 Dedicated nodes approach .......................................................................................... 20

4.1 Energy vs completion time on Nancy site with output at every 10 iteration .................. 24

4.2 Average power usage under different I/O approaches ............................................... 24

4.3 CDFs for different I/O approaches .............................................................................. 26

4.4 Measured energy consumption under different approaches to I/O management in HPC
    applications with different output frequencies: different number of iterations between each
    output .................................................................................................................................. 27

4.5 Energy vs completion time on Rennes site with output at every 10 iteration ............... 28

4.6 Power behavior of different I/O Approaches ................................................................ 28

4.7 Energy consumption under different I/O management approaches and system architectures
    ........................................................................................................................................ 29

4.8 Average Power Consumption and Throughput under different data management approaches
    and system configurations ............................................................................................... 30

5.1 Scalability results for the core level (within the node) .................................................. 34

5.2 Scalability results for the node level (within the subset of a cluster) ............................ 34

5.3 Idle Power Measurements for set of nodes in the parapluie cluster ............................... 35

5.4 Average Power Values during the simulation ................................................................ 35

6.1 Observed and estimated energy consumption under different I/O scenarios on the para-
    pluie cluster ..................................................................................................................... 38

6.2 Observed and estimated energy consumption under different I/O scenarios on the graphene
    cluster ................................................................................................................................ 39

6.3 Accuracy of the model ................................................................................................. 40

6.4 Energy Model and its estimation framework .................................................................. 40
List of Tables

2.1 Summary of the I/O approaches in HPC systems ....................................................... 9

4.1 Cluster-wide energy statistics under different approaches ........................................ 25
1. Introduction

Power has become an essential issue in the design of modern computing systems. Power bills become a substantial part of the total cost of ownership (TCO) of supercomputers: a typical supercomputer of thousands of cores consumes several megawatt of power \(^2\) which in turn represents almost 40% of the total cost \(^3\). Performance has long been the major focus of the HPC community. Today’s supercomputers are therefore equipped with millions of processing cores that run parallel programs and consume a large amount of energy. For example, Tianhe-2, No.1 in the top 500 supercomputers list, is a 3,120,000 processor supercomputer with a Linpack performance of 33.8 petaflop/s \(^1\) but with a 17 megawatt of power consumption \(^2\). This amount of energy will even increase as we reach the era of exascale systems.

Scientific simulations running on HPC systems have been traditionally designed to write their data in the form of many files stored in a parallel file system. Yet, the increasing computational power of new supercomputers largely overcomes the performance of storage systems, which substantially impacts the performance of these simulations. The traditional approach of periodically checkpointing the simulation’s data into files and processing it offline to retrieve scientific results does not scale anymore \(^9\). Therefore, a number of new approaches to large-scale data management have been proposed that make use of dedicated resources such as dedicated cores \(^9\), dedicated nodes \(^36\), accelerators (such as GPUs) \(^10\) or SSDs \(^34\) to process data as it is being generated by the simulation.

As energy is becoming an increasingly important concern for large-scale systems, a major challenge of exascale computing is how to sustain a high performance per watt ratio. While most studies have been focusing on profiling and characterizing power usage in supercomputers, modeling and exploring data-related energy/performance tradeoffs \(^13\), and exploiting dynamic voltage frequency scaling (DVFS) techniques to reduce power consumption \(^18\), there is comparatively little work on investigating the impact of I/O management on energy consumption (i.e., how much energy a supercomputer consumes while running a scientific simulation when adopting different data management approaches).

We investigate three different approaches to I/O management (that is, (1) periodically stopping the simulation to write data, (2) using dedicated I/O cores and (3) using dedicated I/O nodes) within the Damaris middleware \(^9\). We perform extensive experiments that involve one of the target HPC applications for the Blue Waters petascale supercomputer project –the CM1 atmospheric model– on Grid’5000 \(^2\). Our experimental results bring out the differences between these three approaches and show that they are only sub-optimal for different configurations of the application and of the system: the energy consumption under different approaches significantly varies with the frequency of output and the system’s architecture. Specifically, we show evidence that a shorter execution time doesn’t always imply better energy consumption, especially when comparing data management using dedicated cores against dedicated nodes.

After demonstrating that data management approaches differ greatly in terms of their energy consumption and performance, we design an energy model which can estimate the energy requirements of different I/O approaches with regard to the targeted application. We propose a profiling method which integrates the system’s architecture information into our model. Therefore, our model is able to select the

\(^1\) http://www.top500.org
\(^2\) http://www.green500.org
best I/O approach in terms of the energy efficiency regardless of the system's architecture or application specific parameters. Thanks to these estimations, our energy model can help users of HPC applications in choosing the optimal I/O approach prior to running their applications. Moreover, our model is able to answer the question of what is the optimal ratio of dedicated resources compared to the computational resources in the system for the I/O management. We have experimentally demonstrated that our model is accurate in its estimations for different management approaches by running a set of experiments with the CM1 atmospheric model on French Grid'5000 testbed.

1.1 Motivation

With the exascale on the horizon, tight power budgets for the supercomputers are inevitable. There are several proposed methods for reducing the energy usage of HPC applications which only investigate hardware specific solutions. In order to overcome these energy constraints in HPC systems, we also need novel software approaches. Therefore, we have seen the shift in the data management approaches in HPC from post-processing to in-situ processing such as dedicated cores or nodes to reduce the energy costs while handling the large volumes of the data. However, there is little work on investigating the energy requirements of these data management approaches. Therefore, it is necessary to seek a framework which can estimate the related costs of each I/O approach and can help users to choose the optimal one according to their priorities. Moreover, the HPC community shows an interest in alternative reservation frameworks for HPC systems which consider power usages of jobs rather than their running time [37]. This emphasizes the need for the knowledge of the energy impact of the different I/O approaches on targeted HPC applications.

1.2 Objectives

Our proposed work first aims at providing a clearer understanding of the interplay between current data management approaches in exascale supercomputers and energy consumption, and offers a preliminary insight into designing energy-efficient approaches for exascale supercomputers. Then, we aim to investigate an energy model based on the I/O management approach, HPC application's characteristics and HPC system's architecture and use this model to pre-select an optimal I/O management approach prior to running the simulation.

1.3 Main contributions

The primary contributions of this paper are as follows:

1. It introduces our implementation of the dedicated nodes approach on the Damaris middleware which allows us to compare different I/O management approaches in HPC on the fair ground of a common implementation.
1. Introduction

2. It experimentally demonstrates that HPC applications experience variations in performance and power consumption under different I/O management approaches and provides a deep analysis to explain this variation and its cause.

3. It illustrates in practice how the differences in system’s architecture influence the execution of HPC applications and how they shape the energy consumption of the entire system.

4. Based on the findings from the experimental study, it presents an energy model that can estimate the best I/O management approach under a given configuration.

5. It experimentally demonstrates that our energy model is accurate in its estimations by running a set of experiments with different configurations on the French Grid’5000 testbed.

Our preliminary results which present the performance and energy trade-offs of different I/O approaches appeared in the Data Intensive Distributed Computing (DIDC’14) Workshop held in conjunction with a premier computer science conference in high performance parallel and distributed systems, the ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC’14).

1.4 Dissertation outline

This paper is organized as follows: Section 2 presents the background of our study as well as the related work. Section 3 introduces the Damaris middleware and presents our implementation of dedicated nodes approach on Damaris. Section 4 describes an overview of our methodologies, followed by the experimental results which investigate energy and performance trade-offs of different data management approaches in HPC. Section 5 introduces our energy model and Section 6 presents a validation of this model. Finally, Section 7 concludes this paper and discusses the future work.
2

Background

Contents

2.1 I/O Landscape in HPC Systems ........................................ 6
2.2 Energy in HPC Systems .................................................. 9
In this section, we first give a detailed overview of data management approaches in the field of high performance computing (HPC). We then present the relevant works on energy in HPC.

2. Background

In this section, we first give a detailed overview of data management approaches in the field of high performance computing (HPC). We then present the relevant works on energy in HPC.

2.1 I/O Landscape in HPC Systems

As we approach the exascale era in HPC, the mismatch between computation and I/O performance has become a limiting factor for the performance and scalability of HPC applications. The traditional approach to perform I/O requires simulation to stop periodically and output data for various purposes such as analysis, visualization and checkpointing [15]. This results in a large number of processes competing for the access to the file system which prevents scaling well to thousands of processes. Therefore, alternative approaches for performing I/O have been investigated recently by the HPC community [9, 36]. These approaches perform I/O while the simulation keeps running, in an asynchronous manner, taking advantage of dedicated resources such as cores or nodes. These approaches eliminate the additional machine time needed for performing data management tasks by overlapping I/O with the simulation. In this section, we will present three different data management approaches which can be used for performing I/O tasks in HPC systems.

2.1.1 Time Partitioning

Time partitioning is the traditional approach to I/O in HPC applications. In this approach, I/O operations are performed in a synchronous manner: the simulation stops periodically, performs I/O operations and continues its computation where it left it off. This I/O activity is presented in Figure 2.1(a). While this approach takes advantage of direct access to simulation data, it impacts the simulation run time since it stops the simulation periodically to perform I/O.

Two different approaches are generally used: the file per process and the collective I/O approaches. In the file per process approach, each process generates output as a separate file which removes the need for synchronization between processes. However, managing big numbers of output files for analysis or visualization purposes is challenging when scaling to thousands of processes in HPC systems. The latter approach takes quite the opposite side compared to the former one, where each process needs to synchronize for writing to specified regions of an output file that is shared among all processes. By using collective I/O, it is easier to manage single shared file for data management tasks. However, this approach comes with the cost of the synchronization that can lead to big degradation in the I/O throughput of the system.

Both aforementioned traditional I/O approaches suffer from the competition of thousands of processes to access HPC resources such as network and file servers [8]. Therefore, in the following sections we will present other approaches which use a small portion of resources (dedicated resources) for performing data management tasks to address aforementioned challenges.
2.1 I/O Landscape in HPC Systems

2.1.2 Dedicated cores

This approach takes advantage of the hierarchical nature of HPC systems and in particular the increasing use of multicore nodes, where many cores share a common memory. It dedicates one or multiple cores in each SMP nodes for performing I/O asynchronously while the simulation runs [9]. Since dedicated cores share the same resources as the cores which run the simulation, they can leverage the already available simulation data in the shared memory. By assigning asynchronous I/O tasks to dedicated cores, one can hide the I/O costs. Figure 2.1(b) presents this approach. In this approach, if I/O tasks do not scale with the simulation, they can start to impact the overall performance. What is more, memory space is limited and there is a need for buffer management coordination between simulation and dedicated cores.

A closely related work on space partitioning is presented by Li et al [20]. They dedicate cores for
2. Background

non-computational tasks in which they rely on FUSE interface for the data transfer. Also, they make use of SSDs to increase the I/O throughput in the system.

2.1.3 Dedicated nodes

With this approach, separate nodes are used for performing I/O operations while the simulation is running [36]. Data is transmitted from computing nodes to dedicated nodes throughout the network. Once the data transfer is completed, dedicated nodes interact with the file system to write the output files. This approach is demonstrated in Figure 2.1(c).

One of the challenges with this approach is the memory management on dedicated nodes. As we come closer to exascale, the memory per core tends to shrink. When applying a dedicated nodes approach, one should be aware of the limited space in the dedicated nodes. While computing nodes send simulation data to dedicated nodes, they are not aware of the available space. Therefore, dedicated nodes should handle the incoming transfers according to their memory availability. The ratio between dedicated nodes and computing nodes is thus an important factor. A mismatch between capacity of dedicated nodes and computing nodes can lead to inefficiency. Another drawback of this approach is the requirement for data transfer through the network, less efficient than a transfer through shared memory. However, asynchronous data transfers can have less impact on the running simulation if one can overlap the cost of the data movement with the simulation.

Zheng et al. [36] present the PreDatA framework to handle the large volumes of output data which is generated by scientific simulations. They propose decoupling of I/O tasks from the simulation by using staging nodes which perform processing operations on scientific data. They define the nodes which run the simulation as computing nodes. They perform asynchronous remote direct memory access (RDMA) operations for the data transfer between staging and computing nodes. Moreover, they schedule data transfer to have better overlapping between I/O tasks and computation. Therefore, in communication intensive applications they are not able to fully hide the I/O costs of the HPC system. For the buffer management, PreDatA framework should be aware of the available memory in the staging nodes. Since staging nodes may not have sufficient memory to hold all of the generated output data, they employ stream processing on staging nodes. With data streaming, staging nodes process the data in smaller chunks which are streamed from computing nodes.

Another interesting work which performs I/O tasks in a hybrid way by combining both dedicated cores and dedicated nodes is introduced by Bennett et al [1]. As we mentioned earlier, the cost of data transfers can be a bottleneck for some HPC applications. Therefore, they aim to minimize the amount of the data transferred between staging and compute nodes to reduce the data transfer cost by leveraging data locality. They introduce the HybridDART communication layer which selects the data transfer method dynamically between shared memory and RDMA depending on the location of the communicating tasks. They also introduce CoDS, co-located DataSpace, a virtual distributed shared space abstraction which provides data lookup and data sharing services similar to put and get operations.
2.1.4 Summary

These three approaches have inherent advantages and disadvantages. While the traditional time partitioning approach can still be viable for some non-data-intensive applications, scientists would not be able to tolerate its large impact on the simulation execution time for applications that output massive amounts of data. On the other hand, the other two approaches minimize this impact by performing I/O in an asynchronous manner while they differ in the location where the actual I/O tasks are actually performed. Dedicated cores approach uses the same resources as the simulation and thus may impact the simulation more than an approach based on dedicated nodes that opts for the cost of data network transfers rather than taking the risk of impacting the simulation. Main characteristics of the aforementioned approaches are summarized in the table 2.1.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time partitioning</td>
<td>Stops simulation periodically to perform I/O</td>
<td>Large impact on simulation</td>
</tr>
<tr>
<td>Dedicated cores</td>
<td>I/O is performed on dedicated cores with direct access to data through shared memory</td>
<td>Uses same resources as simulation</td>
</tr>
<tr>
<td>Dedicated nodes</td>
<td>I/O is performed on dedicated nodes with access to data via network</td>
<td>Cost of the data transfer through network</td>
</tr>
</tbody>
</table>

2.2 Energy in HPC Systems

There exists a growing interest on energy efficiency in the high performance computing field in recent years. With the exascale on horizon, energy requirements of the supercomputers will be challenging to achieve. Therefore, many research groups have tried to address this issue in HPC by analyzing the energy trends of various HPC applications. Gamell et al. [12] present their power model for in-situ analysis, in particular for the analysis workflow integrated with S3D [6], a turbulent combustion code. They investigated the roles of system architecture, algorithm design and deployment options for the energy/performance profiles. In their power model, they use the Byfl compiler analysis tool [26] to obtain the application specific parameters with respect to the hardware on which it runs. While they explore the power behaviors of the S3D code under different scenarios, they have not addressed the impact of these different scenarios to in-situ analysis in terms of performance.

Kamil et al. [16] introduce a power model which does not require full shutdown of a machine by extrapolating single rack power consumption to all racks with AC to DC conversion accuracy in mind. For power measurement, they use the High Performance Linpack (HPL) benchmark, which they claim that it has similar power behaviors with other compute-intensive scientific workloads. However, they also indicate that HPL cannot serve as an ideal workload for performance measurements. Therefore, while they target power efficiency in their work, their contribution was limited to the exploration of power behaviors of computation intensive scientific workloads. Also, they run the benchmarks for three minutes which may not be sufficient for representing HPC applications’ power behaviors.
Song et al. [30] present a power performance profiling framework, PowerPack, to study power behaviors of HPC Challenge benchmarks. They focus on applications’ memory access patterns and the impact of the system size on the energy efficiency. They find out that workloads which have high temporal and spatial locality spend little time waiting for data and consume more processor power compared to other workloads. In their HPCC tests, they observe that memory is the second largest power consumer after CPU. For the energy profiling, they find that embarrassingly parallel codes achieve better energy efficiency as system size increases. However, for the codes that are not embarrassingly parallel, energy increases much faster than the performance of the workload. Therefore, they indicate that the size of the system is an important factor to consider along with application characteristics when trying to achieve the energy efficiency. However, they use a cluster which consists of 9 nodes and 4 cores per node which may not be representative for HPC systems.

Apart from profiling energy behaviors of HPC applications, some research works consider methods for saving energy. Recently, Orgerie et al. [25] conducted a survey on the methods for improving the energy efficiency in large scale systems. They discuss the methods to evaluate and model the energy consumed by computing and networking resources. They indicate that system energy consists of two parts: static and dynamic. The former one depends on the system size and type of the components while the latter one results from the usage of the resources such as computing and network. They mention that we can improve the energy efficiency by minimizing the static part and obtaining more performance in proportion to the dynamic part of the system. After discussing modeling of the energy consumption of computing resources, they propose several techniques to have energy savings such as Dynamic Voltage Frequency Scaling (DVFS) [17, 21, 28, 35], software improvements [11, 19, 33] and hardware capabilities [5]. For the networking resources, they find that switch fabrics are an important part of the power consumption, e.g. 90% for IBM Infiniband router. Similarly to computing resources, the energy consumption of networking devices is not proportional to their usage. Therefore, they indicate that energy models may not be realistic since they represent the ideal scenario.

Laros et al. [18] present the impact of CPU frequency and network bandwidth scaling on energy efficiency. They apply static changes in the CPU frequency to save energy at the cost of performance degradation. They find that impact of the CPU frequency scaling depends on the workload type. While for computation intensive applications there is a big degradation in the performance, there can be great improvements in the energy efficiency for communication intensive applications.

Another similar work has been carried out by Springer et al. [32] in which they demonstrate that significant potential exists for energy savings in HPC applications without sacrificing the performance. They apply dynamic frequency scaling by shifting the gears which represent the different levels of CPU frequencies. They observe that well-tuned programs such as NAS benchmarks can benefit from their approach especially during their idle time resulting from communication. The biggest contribution of their work is to be able to apply multiple energy gears dynamically by observing the memory pressure and MPI call locations in the program to obtain better energy efficiency. However, their experiment was on a small scale which only consisted of 9 nodes.

A closely related work on energy efficiency in HPC, which focuses on fault tolerance protocols, is con-
ducted by Diouri et al. [7]. They estimate the energy usage of different fault tolerance protocols based on protocol parameters (checkpointing interval), application parameters (number of processes, message size and number) and hardware parameters (number of cores per node, disk type, memory etc.). They use a calibration approach, that inspired our profiling approach, in order to take into consideration the specific hardware used in their energy estimations. Besides energy estimations, they also apply power saving techniques to improve energy efficiency. They observe that their approach performs better than Linux’s governor since it does not change the CPU frequency unless the system’s load decreases below a certain threshold. They also find that the CPU load generally remains very high for memory intensive workloads which do not require full computational power.

Most of the studies on energy efficiency in HPC target entire system power. Gamell et al. [12] separate the network component in their model, however they make assumptions for the related component type since power information for every component is not available, particularly the NIC in their work. Son et al. [29] present a different work than aforementioned works just by targeting the disk power. They use SPEC2000 floating benchmark suite [31] and generate statistical data for performance and energy consumption via a simulator similar to DiskSim. Considering the data access patterns of the applications and combining this information with the disk layout they extract the disk access patterns. They apply proactive disk power management and also make use of code restructuring which result in promising results, i.e. energy savings up to 43% compared to Traditional Power Management (TPM). This work has inspired our work by showing us that software-driven approaches can be more efficient than existing hardware solutions for energy efficiency.

The aforementioned works also differ in the methodology that they apply for power measurements. Kamil et al. [16] investigate various power measurement methodologies such as line meter, clamp meter, integrated meter and power panels, and opt for power panels in their work. What is more, we can observe voltage regulator models to provide current and voltage readings in the node level [18], cluster specifications [12], simulators [29] and wattmeters [7] for the power measurements. While there is a wide range of options for measuring the power, most of them are subject to measurement errors. Therefore, power measurements are generally multiplied to reduce the impact of the measurement error. We also follow the same trend in our work.
2. Background
3 Implementation

Contents

3.1 Damaris ......................................................... 14
3.2 Dedicated Nodes for Damaris ................................. 17
In this section, we will present Damaris middleware which we use to compare the different I/O approaches on a fair ground of a common implementation. Damaris is a middleware for I/O and data management (including data processing and in situ visualization) [8]. Initially developed to provide data management capabilities through dedicated I/O cores on multicore nodes, it has later been extended to support the more classical time partitioning approach. We further extended it to support dedicated nodes approach which we will present in detail after introducing Damaris.

3.1 Damaris

Damaris middleware applies the space partitioning approach to address the challenges in data management in HPC applications. One of these challenges is the contention which stems from the attempt of the large number of writers to access the system resources (network, file system) at the same time for data management purposes. Damaris approach addresses this issue by using dedicating core(s) for only performing data management tasks which reduces number of writers in the system. Moreover, this requires gathering simulation data into larger output files which can prevent metadata servers from being bottleneck to the HPC system. Another common problem which is observed via common I/O approaches is the I/O variability [9]. I/O variability can occur due to various reasons such as synchronization between processes and resource contention. Damaris hides these I/O costs by overlapping I/O with the simulation. We describe its architecture and main functionality below.

![Damaris approach diagram](image)
3.1 Damaris

3.1.1 Architecture

The design of the Damaris approach is presented in Figure 3.1 [8]. While compute cores run the simulation, a Damaris core is dedicated for performing data management tasks. With the large number of cores in HPC systems, dedicating core(s) has a negligible impact on simulation as experimentally proven by Dorier et al [9].

One of the unique design choices of the Damaris approach is to make use of the shared memory for the communication between compute cores and dedicated cores. Shared memory segment is created by the Damaris core in the beginning of the simulation. In every output interval of the simulation, compute cores write the simulation data to the shared memory via memcpy. While sending data to Damaris cores, compute cores first gain ownership of the region that it requires depending on the size of the data. After compute core finishes writing data to that region, it notifies the Damaris core. Ownership of this buffer region will be taken by the Damaris core by obtaining the relevant pointer from the compute core. When dedicated (Damaris) core gets the ownership, it can employ the required data management tasks such as writing simulation data to the parallel file system.

Another important design component of the Damaris is the configuration file which is inspired by ADIOS framework. Loftstead et al. [22] designed the ADIOS framework to provide the simple and flexible approach for I/O by using XML configuration files to specify the interaction between simulation code and data management mechanisms. Example configuration file for simulating the mesh which is used by Damaris can be found in Listing 3.1.

With this external configuration file, users will have the full control over the simulation. For example, this specific XML file represents the configuration for the mesh simulation. In the architecture component, the user can select how many resources he wants to dedicate for performing I/O. In this example, core count equals to 1 which indicates Damaris will employ the space partitioning approach by dedicating 1 cores. If both numbers (core and nodes) are set to zero, Damaris will employ the time partitioning approach which represents the classical I/O approach that does not involve any dedicated resources for I/O tasks. Moreover, user can define the size of the shared memory segment which is used for communication between computing resources and the dedicated ones. In the data entity, parameter holds any type of data and associate it with a name. We have w,h and d parameters which represent the width, height and depth of the simulation data and their values can change the layout dynamically as we observe for the cmesh layout. Layout describes the shape of the data in the memory and they define the dimensions of the variables which are used in the simulation.

Listing 3.1: Sample XML configuration file

```xml
<simulation name="mesh" language="c" xmlns="http://damaris.gforge.inria.fr/damaris/model">
  <architecture>
    <domains count="1" />
    <core count="1" nodes="0" />
    <buffer name="buffer" size="67108864" />
  </architecture>
  <data>
```

3. Implementation

<parameter name="w" type="int" value="2" />
<parameter name="h" type="int" value="3" />
<parameter name="d" type="int" value="4" />

<layout name="rmeshX" type="float" dimensions="4" />
<layout name="rmeshY" type="float" dimensions="5" />

<layout name="cmesh" type="float" dimensions="w,h,d" />

<mesh name="mesh2d" type="rectilinear" topology="2">
  <coord name="coordinates/x2d" unit="cm" label="Width"/>
  <coord name="coordinates/y2d" unit="cm" label="Height"/>
</mesh>

<group name="coordinates">
  <variable name="x2d" layout="rmeshX" visualizable="false" time-varying="false" />
  <variable name="y2d" layout="rmeshY" visualizable="false" time-varying="false" />
  
  <variable name="x3d" layout="cmesh" visualizable="false" time-varying="false" />
  <variable name="y3d" layout="cmesh" visualizable="false" time-varying="false" />
  <variable name="z3d" layout="cmesh" visualizable="false" time-varying="false" />
</group>
</data>
</simulation>

3.1.2 Main functionality

Damaris is implemented in C++ and uses the Boost library for interprocess communications and Xerces-C for XML configuration. Two important classes which are related with the data management tasks are the Client and Server classes. While Server object represents the dedicated resources (cores/nodes) Client object represents a single core running the simulation and it exposes functions allowing the simulation to write data and send signals to dedicated resources. This client object can be in the same or separate node depending on the employed data management approach. Moreover, with the time partitioning approach there are no dedicated resources, therefore simulation runs as without server (standalone client mode). The main functions that the client object exposes are:

- **Connect**: Sends a message to the dedicated resource to make this core known to it. When the server receives this message, it adds this client to the its known clients list and sets a communication channel between them. This method is called first before all other methods which interact with the server.
3.2 Dedicated Nodes for Damaris

- **Write**: Takes the name of the variable and the pointer to the simulation data to write as arguments. This method simply writes the simulation data to shared memory in the space partitioning mode. For dedicated nodes approach, we override this method to send the data to the dedicated nodes via network which we will explain in detail in the next section.

- **Signal**: This method simply sends signals to the server. There are two types of signals: header and body. Header signal carries the information about the source, size and type of the body signal which will be sent after sending the header signal. For body signals, we can have different types depending on the methods that client wants to employ. For instance, write message in dedicated cores mode which provides information to be able to copy data to shared memory or remoteWrite message for dedicated nodes approach which gives information to the servers for the transfer of the data through network.

Moreover, we have an additional functionality for the space partitioning approach such as alloc and commit methods which are used by the clients to allocate a buffer in shared memory so that the data can be written by the simulation. Commit should follow this method to notify the dedicated core that simulation data has written and it will set the variable as read-only for the simulation and pass the pointer to the dedicated core. Therefore, dedicated cores get the ownership of the relevant data and can perform further data management tasks. These two functions are different than write function since they allow direct access to some specific parts of the shared memory.

For the server side, there are callback functions such as OnConnect and OnWrite which are associated to the events sent by clients as we described above. Two other main Damaris functions are initialize and finalize which will initialize the environment according to the given configuration file such as number of dedicated resources and the size of the shared memory segment and the latter one will free the resources used by Damaris. Excerpt from the mesh simulation and its integration with Damaris can be found in Listing 3.2. Moreover, its associated XML file can be found in Listing 3.1.

### 3.2 Dedicated Nodes for Damaris

To be able to compare different I/O approaches on a common framework, we extended Damaris to provide dedicated nodes approach. Dedicated cores and dedicated nodes approaches have the same main goal which is to overlap I/O with the simulation in order to hide the I/O costs. In both approaches, there are Client and Server objects which represent the simulation and dedicated resources respectively. The difference between two different approaches takes place in the communication pattern where the former one uses shared memory while the latter one transfers the data through network. To accomplish this, we override the write method which we will declare as remoteWrite from now on. Also, we need an associated callback function on the server side which we define as OnRemoteWrite method.

In the initialization of the environment for Damaris, we obtain the number of dedicated nodes from the external configuration file. This number should divide the number of client nodes to be able to map each client to a server in a uniform way. After that, we perform client to server mappings according to
the rank of the MPI processes. These mappings are followed by setting the relevant communication channels and connection of clients to the servers. Moreover, process with rank 0 creates the shared memory segment for each node with a given size in the XML file.

Listing 3.2: **Sample mesh simulation with Damaris**

```c
#include "damaris.h"

int main(int argc, char **argv) {
    MPI_Init(&argc,&argv);
    /* Initialize Damaris client. */
    damaris_initialize("mesh.xml",MPI_COMM_WORLD);

    /* Set up some simulation data. */
    simulation_data sim;

    int i;
    for(i=0;i<60;i++)
    {
        simulate_one_timestep(&sim);
        exposeDataToDamaris(&sim);
    }

    /* Finalize Damaris. */
    damaris_finalize();
    MPI_Finalize();

    return 0;
}

void exposeDataToDamaris(simulation_data* sim) {
    damaris_write("coordinates/x2d",rmesh_x);
    damaris_write("coordinates/y2d",rmesh_y);
    damaris_write("coordinates/x3d",cmesh_x);
    damaris_write("coordinates/y3d",cmesh_y);
    damaris_write("coordinates/z3d",cmesh_z);

    damaris_end_iteration();
}
```

We inherited from *Client* class as *RemoteClient* to provide necessary functionality for the dedicated nodes approach. In this scope, we override the existing *write* method. *RemoteClient* object first prepares the header signal message that indicates the source of the message and the body of the message
which in this case is `remoteWrite` message. After sending the header message, client prepares the `remoteWrite` message which consists of the name of the variable, size of the data, the current iteration number regarding the simulation and the source id. Here, the size of the data is the most important information for the server side. After receiving this message, `OnRemoteWrite` callback will be triggered on the dedicated nodes. With the given size information, servers allocate a portion of the shared memory segment to able to retrieve the data which will be sent by the clients. As we mentioned earlier, buffer management is crucial for the dedicated nodes approach. Since clients are in a separate node they do not have the knowledge of the available memory space in the dedicated nodes. We opt for asynchronous data transfer in order to hide the cost of the data transfer. However, due to buffer coordination between client and servers we have implemented the allocation method on the server side as blocking. Blocking allocation means that process on the dedicated node has to wait until there is enough memory for the size of the data which the client wants to write. Dedicated nodes free the memory after they complete writing this simulation data to the parallel file system which requires a good balance between the write time of the dedicated nodes and the interval between two outputs generated by the simulation. Therefore, if the output frequency is too high, servers might not able to find the required time to write the current output to the file system before the write request of the client for the new generated output. This situation can result in degradation in the performance of the simulation since dedicated nodes will not able to overlap I/O with the simulation. Hence, ratio between the client and dedicated nodes should be chosen wisely to prevent the mismatch between the capacity of client and dedicated nodes.

Listing 3.3: **Write function with dedicated nodes approach**

### Client side:

1. SET Header(size, body type)
2. ASYNC_SEND Header message
3. SET remoteWrite(source, varname, size, iteration)
4. ASYNC_SEND remoteWrite message
5. SEND simulation data

### Server side:

1. ASYNC_RECV Header(remoteWrite)
2. ASYNC_RECV remoteWrite(source, varname, size, iteration)
3. IF availableMemory > size
   1. ALLOC dataSpace(size)
4. ELSE
   1. wait UNTIL availableMemory > size
5. RECV simulation data
6. WRITE data to parallel file system
7. FREE dataSpace(size)
3. Implementation

Figure 3.2: Dedicated nodes approach

It is worthwhile to note that the size of the simulation data can change dynamically during the simulation via the parameters as we explained in the previous section. Due to this behavior, allocation method on the dedicated nodes differs also in obtaining the size information from the clients. For the space partitioning approach, dedicated cores use the layout information of the simulation data to achieve the required size that they need to allocate. However, with dedicated resources on the separate node for the dedicated nodes approach, they will not be aware of the changes in the layout which are occurred on the client side. Therefore, dedicated nodes also receive the size information for the related simulation data in addition to its name and layout information. Dedicated nodes allocate fixed size for the write request of the clients according to this information. After the preparation to data transfer on client and server sides, we use Send and Recv functions to transfer the data which are simply $MPI_{Send}$ and $MPI_{Recv}$ methods. After that, dedicated nodes write the simulation data to the parallel file system for the current iteration. Listing 3.3 represents the write function with dedicated nodes approach on Damaris while Figure 3.2 illustrates the main principle of the dedicated nodes approach.
4

Zoom on Energy/Performance Trade-offs of I/O Approaches in HPC

Contents

4.1 Methodology ................................................................. 22
4.2 Results ................................................................. 23
In this section, we performed an experimental study to have a clearer understanding of the interplay between performance of the current data management approaches in exascale supercomputers and related energy consumption. First, we will present the overview of methodology and then we will illustrate the key results and provide the detailed analysis in order to obtain insights into designing energy-efficient framework.

4.1 Methodology

We conducted a series of experiments in order to assess the impact of the three data management approaches on both energy consumption and application performance. We further describe the experimental environment: the platform, the HPC application, the tools used, and the deployment setup.

4.1.1 Platform

The experiments were carried out on the French Grid’5000 [2] testbed. Contrary to a real HPC platform, where end-users can only submit jobs, we have a root access on all the reserved resources during the time of this reservation, which in particular allows us to deploy and configure our own parallel file system and our own implementation of MPI. The MPI implementation used here is MPICH-2 [23]. Particularly, we employed nodes belonging to its Nancy and Rennes sites:

On the Nancy site: each node is a 4-core Intel Xeon 2.53 GHz CPU with 16 GB of RAM. Intra-cluster communication is done through a 1G Ethernet network while 20G Infiniband network is used between the parallel file system and nodes that run the simulation. 40 nodes of the Nancy site are equipped with power monitoring hardware consisting of 2 Power Distribution Units (EATON PDUs), each hosting 20 outlets which are mapped to a specific node.

On the Rennes site: each node has two 12-core AMD 1.7 GHz CPU with 48 GB of RAM. The nodes communicate through a 1G Ethernet network and they communicate with the parallel file system through 20G Infiniband 4x QDR link connected to a common Voltaire switch. 40 nodes of the Rennes site are equipped with power monitoring hardware consisting of 4 Power Distribution Units (EATON PDUs), each hosting 10 outlets which are mapped to a specific node.

For PDUs, since each node is mapped to a specific outlet, we are able to acquire coarse and fine-grained power monitoring information using the Simple Network Management Protocol (SNMP). We measure the energy consumption with a resolution of one second. Grid’5000 allows us to create an isolated environment in order to have full control over the experiments and the obtained results.

4.1.2 Real life application

For our analysis, we chose one of the target HPC applications of the Blue Waters petascale supercomputer project [24]: the CM1 atmospheric model [3]. CM1 is used for atmospheric research to model small-scale atmosphere phenomena such as thunderstorms and tornadoes. It follows a typical behavior
of scientific simulations by alternating computation phases and I/O phases. The simulated domain is a fixed 3D array. The number of points along the $x$, $y$ and $z$ axes is given by the parameters $n_x$, $n_y$ and $n_z$. Each point in this domain is characterized by a set of variables such as local temperature or wind speed.

CM1 is written in Fortran 95. Parallelization is done using MPI, by splitting the 3D array along a 2D grid. Each process simulates a $n_{x} \times n_{y} \times n_{z}$ point subdomain. In the current release (r16), the I/O phase uses HDF5 to write one file per process at every output.

4.1.3 Experimental Deployment

We used 32 nodes (128 cores) to run CM1 on the Nancy site, 6 additional nodes are used by a PVFS [4] file system. These PVFS nodes are accessed by computation nodes through a 20G Infiniband network. On the Rennes site, we run CM1 on 16 nodes (384 cores) and 3 nodes are used by PVFS, also accessed from compute nodes through a 20G InfiniBand network. In addition, in both sites, we configure CM1 to complete 2520 time steps and vary the frequency of output, using 10, 20 or 30 time steps between outputs. Damaris is configured to run with CM1 in five different scenarios which cover the three different I/O approaches: time partitioning (i.e., TP), dedicating core(s) employing one core per node (i.e., DC(ONE)) and employing two cores per node (i.e., DC(TWO)), and dedicating node(s) using a ratio of 15:1 for compute nodes to dedicated nodes (i.e., DN(15:1)) and using a ratio of 7:1 for compute nodes to dedicated nodes (i.e., DN(7:1)).

In our first experiment, the I/O output frequency is performed at every 10 time steps. In order to understand the impact of the output frequency in the second set of experiments (Section 4.2.1), we vary the output frequency, using 10, 20 and 30 time steps between outputs. Finally, to illustrate the impacts of system’s architectures, we complement the first set of experiments (i.e., Nancy site) by running the CM1 simulation on Rennes site. Here, we fixed the number of I/O output frequency to every 10 time steps.

4.2 Results

In this section, we provide an analysis of the experimental results. Our goal is to study the impact of various I/O approaches on the energy and performance of HPC applications, particularly CM1.

In terms of performance, Figure 4.1 shows that the time partitioning approach largely impacts the simulation. This behavior results from the fact that it performs I/O from all processes and stops the simulation periodically. On the other hand, approaches that perform I/O asynchronously and from a reduced number of writers obtained a much better run time. Among these approaches, dedicating nodes with a ratio of 7:1 (i.e., DN(7:1)) outperforms the other configurations. We can argue that the approach based on dedicated cores can have a larger impact on the simulation because there are only four cores per node, i.e., dedicating even one core already removes 25% of the computation resources that could have been used by the simulation. Additionally, it is worthwhile to note the importance of the ratio between dedicated and compute nodes in the dedicated nodes approach. As we mentioned before, the mismatch between capacity of dedicated and computing nodes can result in inefficient simulation
4. Zoom on Energy/Performance Trade-offs of I/O Approaches in HPC

Figure 4.1: Energy vs completion time on Nancy site with output at every 10 iteration

In terms of energy consumption, Figure 4.1 shows a strong correlation between completion time and energy consumption. However, we can also see that although dedicating nodes approach with a ratio of 15:1 (i.e., DN(15:1)) finishes the simulation earlier than with one dedicated cores (i.e., DC(ONE)), the former one consumes more energy than the latter. This implies that short execution times do not always lead to less energy consumption. Additionally, we can see that completion time is almost tripled with the time partitioning approach while energy consumption stayed at two times of other approaches. This stems from stopping the simulation periodically to perform I/O which leads to less utilization of CPU and less power consumption with the penalty of much longer execution times compare to other approaches. Figure 4.2 illustrates the power behavior of different I/O approaches by showing the average power usage within the cluster during the simulation. These power behaviors confirm our observation where we can see that there is a significant decrease for the average power usage of time partitioning approach which mainly stems from having both I/O and computation phases alternatively. On the other hand, I/O

Figure 4.2: Average power usage under different I/O approaches
approaches with dedicated resources aim to overlap I/O with the simulation. Therefore, we observe similar power behaviors among these approaches. However, we can observe similar energy/performance trade-offs in approaches which employ dedicated resources as in the time partitioning approach. For example, in the dedicated nodes approach all cores only perform computation tasks whereas in the dedicated cores approach we employed one or two cores to perform I/O, which leads to less CPU usage on average. We can observe slightly decrease in the power usage for dedicated cores approach compared to the dedicated nodes as a response to less CPU usage on average. Since the average CPU utilization affects the completion time, we observe longer execution times for lower CPU utilization. However, this can lead to less energy consumption as we described for the \((i.e., DC(ONE))\) approach, at the expense of performance compared to \((i.e., DN(15:1))\). These energy/performance trade-offs can be favored according to the users’ priority and, of course, largely depends on the application considered.

Table 4.1 shows the cluster-wide energy consumption statistics of the different data management approaches. In terms of energy consumption of the nodes within the cluster, we can observe greater variability \((i.e., stdDev(4.5))\) under dedicated nodes approach compare to other approaches. This behavior results from the fact that there are two different nodes as computing nodes and dedicated nodes in the dedicated nodes approach. Computing nodes run the simulation where dedicated nodes only perform data management tasks which result in less energy consumption compare to computing nodes. Moreover, we can see these different power behaviors of computing nodes and dedicated nodes in Figure 4.3. Figure 4.3(a) and 4.3(b) present the cumulative distribution function (CDF) of the average power consumption under three different I/O approaches. We obtained these CDFs from two different nodes and they represent the computing nodes for the time partitioning \((i.e., TP)\) and dedicated cores \((i.e., DC)\) approaches. However, Figure 4.3(b) represents the dedicated node for the dedicated nodes approach \((i.e., DN)\). We selected two different kinds of nodes for the dedicated nodes approach to show the significant difference between dedicated and computing nodes. We can easily observe the difference of the power behaviors of the dedicated and computing nodes within the dedicated nodes approach. What is more, by looking at the cumulative distribution functions of three different I/O approaches, we can see the greater variability with time partitioning approach since it performs I/O and computation tasks alternatively during the simulation. When we look at the dedicated node for the dedicated nodes approach, it shows a similar trend to time partitioning approach in terms of its power behavior. However, computing nodes (in the dedicated nodes approach) do not have a variability in their power usage during the simulation since they only perform computation tasks. For the dedicated cores approach, we can observe that its power behavior is in between the time partitioning and dedicated nodes approach in Figure 4.3(a). Moreover, its trend is similar to the one in dedicated nodes approach which does not show variability in

**Table 4.1: Cluster-wide energy statistics under different approaches**

<table>
<thead>
<tr>
<th>Approach</th>
<th>Total(kJ)</th>
<th>Avg(kJ)</th>
<th>Min(kJ)</th>
<th>Max(kJ)</th>
<th>StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time partitioning</td>
<td>3324</td>
<td>101</td>
<td>94</td>
<td>108</td>
<td>3.4</td>
</tr>
<tr>
<td>Dedicated Core (ONE)</td>
<td>1777</td>
<td>54</td>
<td>38</td>
<td>56</td>
<td>2.9</td>
</tr>
<tr>
<td>Dedicated Cores (TWO)</td>
<td>2211</td>
<td>67</td>
<td>52</td>
<td>70</td>
<td>3</td>
</tr>
<tr>
<td>Dedicated Nodes (15:1)</td>
<td>1736</td>
<td>53</td>
<td>35</td>
<td>56</td>
<td>4.2</td>
</tr>
<tr>
<td>Dedicated Nodes (7:1)</td>
<td>1340</td>
<td>41</td>
<td>28</td>
<td>44</td>
<td>4.5</td>
</tr>
</tbody>
</table>
terms of power consumption. However, there is a slight decrease in the power usage due to dedicated cores that perform I/O tasks which result in lower CPU usage and also power consumption.

Another thing that worth discussing would be the question that can we apply any energy savings method by considering our deep analysis of the interplay between energy and performance of the different I/O approaches. In terms of possible energy savings, it would be interesting to investigate Dynamic Voltage Frequency Scaling (DVFS) method on dedicated resources. As we observe lower power consumption with dedicated nodes, we can explore the trade-offs between reducing the CPU frequencies of the dedicated resources which perform I/O tasks in order to reduce the overall energy consumption.

### 4.2.1 Impact of the Output Frequency

Figure 4.4 shows the resulting energy consumption of the different data management approaches under different output frequencies (i.e., different number of iterations between each output).

These results show that there is a correlation between the output frequency and energy consump-
4.2 Results

Figure 4.4: Measured energy consumption under different approaches to I/O management in HPC applications with different output frequencies: different number of iterations between each output

ation for the time partitioning approach. However, for the other approaches there is no clear difference between last two configurations (i.e., output every 20 and 30 iterations) and there is an increase in the energy consumption only under the highest I/O frequency where outputs are generated at every 10 iterations. For the time partitioning approach, this correlation is expected since increasing the output frequency leads to higher energy consumption which stems from longer execution times. However, I/O approaches which employ dedicated resources to perform I/O asynchronously were able to overlap I/O with the simulation for the output frequencies at every 30 and 20 iterations. Interestingly, we observe higher energy consumption under the output frequency of every 10 iterations. This shows us that only configuration of dedicated nodes approach (i.e., DN(7:1)) is able to hide the costs of I/O operations where in other scenarios we observe a significant increase in the energy consumption. This increase is due to longer execution times which stems from the fact that dedicated resources are not able to finish their I/O tasks within the cycle of 10 iterations. Therefore, computation cores/nodes have to wait until dedicated cores or nodes are able to receive data from them which will happen after dedicated cores/nodes finish writing the output files to the PVFS. Hence, it is worthwhile to mention again the importance of the ratio between compute nodes and dedicated nodes. We were only able to fully hide I/O costs of the simulation with the ratio of 7:1 in the dedicated nodes approach.

4.2.2 Impact of the System Architecture

In this section, we first provide an analysis of the experimental results we obtained on the Rennes site. Then, we give a comparative study of the results for both sites to illustrate the impact of the system architecture on the energy and performance trade-offs of different data management approaches.

Figure 4.5 illustrates the energy consumption and performance of the different I/O scenarios. Again, time partitioning approach performed worse compare to other approaches due to its synchronous I/O behavior. Among the approaches with dedicated resources, dedicating cores with one dedicated core (i.e., DC(ONE)) outperforms the other configurations. Here, we can argue that impact of the dedicating
cores on the simulation is negligible \(\text{i.e., DC(ONE) takes only } 4\% \text{ of the computational power} \) due to large number of cores per node.

Figure 4.6 presents the power behaviors of different I/O approaches. In terms of average power usage, again there is a significant decrease with time partitioning approach which stems from having I/O and computation phases alternatively. In addition, within dedicated nodes approach we can observe a power drop when we have more dedicated nodes \(\text{i.e., DN(7:1)}\). Since this figure presents the average power usage within the cluster, more nodes with the I/O behavior (dedicated nodes) would lead to decrease in the average power usage.

Figure 4.7 shows the energy consumption of each I/O approach and Figure 4.8 depicts the throughput and average power usage during the simulation using different system architectures. The results indicate that the comparative behavior of the different approaches with respect to performance and energy efficiency depends on the system on which they run. With a larger number of cores per node, dedicating one core outperforms the other configurations while the dedicated nodes approach was the optimal choice with a smaller number of cores per node. The different behaviors in dedicated cores
approaches stem from reducing the number of cores that could otherwise be used by the simulation. Since CM1 is computation-intensive, dedicating cores for performing I/O tasks reduces the computing resources dramatically in an architecture with a small number of cores per node. However, with a larger number of cores per node, dedicating cores removes only a small amount of computing resources (e.g., 4% when dedicating one core out of 24 cores) and therefore its impact on the simulation stays minimal.

Interestingly, we also observe that average throughput under different I/O scenarios is higher on the Nancy site (i.e., 4 cores/node) compare to the Rennes site (i.e., 24 cores/node). This mainly stems from the difference in the number of the nodes that we deploy CM1 (i.e., 32 nodes for the Nancy site and 16 nodes for the Rennes site). Also, we can partially attribute this behavior to the different system architectures (i.e., Intel 2.53 GHz CPU for the Nancy site and AMD 1.7 GHz CPU for the Rennes site).

These different behaviors highlight the need for an energy efficient framework which can select the best I/O approach in terms of energy efficiency by considering different parameters such as the system’s architecture or the output size and frequency.
Figure 4.8: Average Power Consumption and Throughput under different data management approaches and system configurations
5

Energy Model

Contents

5.1 Model Formulation .................................................. 32
5.2 Profiling Approach .................................................. 33
Data management approaches for exascale systems have different energy behaviors and they are only sub-optimal for different applications and system configurations. We have experimentally demonstrated that the energy consumption under different I/O approaches significantly varies with the application specific parameters and system’s configurations. This highlights the need for an energy efficient framework which will select the best data management approach by considering the relevant configuration. Therefore, we have developed an energy model for computation intensive HPC applications under different data management approaches. In order to estimate the energy consumption, parameters of our model consists of the specific hardware used, number of nodes, cores per node used for the computation and their related efficiency values in scaling. To be able to assess the impact of the specific hardware used during the simulation we employed a profiling method. We also measured the related scalability values during the profiling period. After employing the profiling approach, we can estimate the related energy/performance trade-offs of different data management approaches to achieve energy savings by selecting the best I/O approach. Finally, we validated our model under different set of experiments to obtain its accuracy.

5.1 Model Formulation

In our model, we assume that we can hide the I/O costs during simulation with dedicated cores/nodes approaches. Therefore, I/O time is negligible from the point of view of the simulation [9] and our energy model as following:

\[ E = T_{sim} \times P_{sim} \]  \hspace{1cm} (5.1)

where \( T_{sim} \) is the total execution time of the simulation and \( P_{sim} \) is the average power spent during the simulation. The two separate components of our model are discussed below.

\( T_{sim} \): To estimate the total execution time, we need to know the number of nodes and cores per node used for the computation. After taking the number of nodes and cores as parameters to our model, we integrate the related scalability values which we obtained during the profiling period into our model to determine the effect of the number of nodes/cores involved in the computation. While estimating the total execution time, we apply two scalability levels. The first one is the core level scalability which depends on the number of cores per node used in the simulation and the other one is the node level scalability which changes according to number of nodes used in the simulation. The reason behind this choice of formulation is also to be able to answer to the question [9, 36] of how many cores or nodes should we dedicate for performing I/O with respect to the data management approach. Putting all together,

\[ T_{sim} = (T_{base}/(n_{core} \times s_{core})/(n_{node} \times s_{node})) \times n_{iter} \]  \hspace{1cm} (5.2)

where \( T_{base} \) is the time per iteration with only one core involved in the simulation; \( n_{core}, n_{node} \) and \( n_{iter} \) are the number of cores, nodes and iterations respectively. \( s_{core} \) and \( s_{node} \) are the related overall efficiency values for the aforementioned scalability levels that are determined experimentally during the
5.2 Profiling Approach

This section describes our profiling approach as we mentioned in the previous section. The goal of the profiling approach is to take into consideration the specific hardware used during the simulation. As energy equals power over time, we need to be able to obtain power and time behaviors of the application with respect to the specific hardware used. Therefore, we perform a set of micro benchmarks to extract relevant values for our energy model. We provide the profiling results of the parapluie cluster which features 40 nodes of 2 AMD 1.7 Ghz CPUs, 12 cores/CPU, 48 GB RAM and 10 Gigabit Ethernet network. We run CM1 atmospheric simulation without any I/O operations enabled in order to obtain its power and time behavior.

To be able to extract the time behavior of the simulation with respect to the specific hardware used, we need scalability values of the application depending on the numbers of cores and nodes used for the simulation. First, we run the CM1 application with different number of cores (Figure 5.1) which will determine the impact of the number of the dedicated cores on the total execution time. Then, we run the CM1 application with different number of nodes to assess the impact of the number of dedicated nodes on simulation (Figure 5.2).

We extracted the base computation time ($T_{base}$) as the time with only 1 core involved in computation.
For the scalability values for the core and node levels in our model \( (s_{\text{core}}, s_{\text{node}}) \), we obtain the ratio between two values: perfect scaling and the speedup that we observed under this specific cluster. For instance, we observed perfect scaling within the node until 16 cores involved in the computation. However, it follows weak scaling after 16 cores for this specific application (i.e., CM1). Similarly, we can see the performance drop after some number of nodes for the scalability values within the cluster. Moreover, similar strong scaling tests of CM1 on the distributed memory supercomputers are conducted by Bryan et al.  

For the power behavior of different I/O approaches, we need to extract the two power values which are \( P_{\text{idle}} \) and \( P_{\text{max}} \). First, we measure the idle power consumption \( (P_{\text{idle}}) \) of a set of nodes (Figure 5.3), then we measure their average power usage \( (P_{\text{max}}) \) during the simulation (Figure 5.4). For power measurements, we used Power Distribution Units (4 EATON PDUS) which host 10 outlets that are attached to...

\[\text{http://ww2.mmm.ucar.edu/people/bryan/cm1/pp.html}\]
5.2 Profiling Approach

to specific nodes and we retrieved the power measurements by using the Simple Network Management Protocol (SNMP) with the resolution of 1s. Both of the figures show that idle and full power usages vary significantly even within the nodes of the same cluster. After performing these measurements, we take their average in order to calculate the $P_{idle}$ and $P_{max}$ values. The significant variance in the nodes of the same cluster for their related power consumption ($P_{max}$, $P_{idle}$) shows the importance of the profiling approach. What is more, scientific simulations are used by the scientific community for a long period of time. Therefore, we can easily say that long-life cycles of HPC applications can amortize the cost of our profiling approach which is performed only once for the specific hardware.
6. Validation

6.1 Model Validation

In order to validate our model, we have performed a set of experiments in the parapluie cluster as the one described in the previous section. We configure CM1 application to run as 2520 iterations with output frequency at every 30 iterations. We run CM1 on 16 nodes (384 cores) with additional 3 nodes as PVFS file system. Damaris is configured to run CM1 using four different data management scenarios: dedicating core(s) employing one core per node (i.e., DC(ONE)) and employing two cores per node (i.e., DC(TWO)), and dedicating node(s) using a ratio of 15:1 for compute nodes to dedicated nodes (i.e., DN(15:1)) and using a ratio of 7:1 for compute nodes to dedicated nodes (i.e., DN(7:1)). We left out the time partitioning approach (i.e., TP) comparison to the experimental study that we have performed. We have experimentally demonstrated that time partitioning approach performs worse when compared to other data management approaches regardless of the configuration(system architecture, application parameters) that it runs. Therefore, in our energy model we focused on the I/O approaches with dedicated resources that can hold the assumption of the model by being able to overlap I/O with the simulation. We measured the total execution time and the power usage five times for each scenario to reduce the impact of the measurement error.

![Figure 6.1: Observed and estimated energy consumption under different I/O scenarios on the parapluie cluster](image)

Figure 6.1 shows the observed average energy consumption over five executions and our model's estimation. We can see that our model is accurate since relative differences are low. The worst relative difference we observed is 4% under the I/O approach with dedicating core(s) employing one core per node (i.e., DC(ONE)). We can partially attribute these difference to the measurement error. Also, it is worthwhile to mention that our model represent the ideal scenario with the assumption of overlapping I/O with the simulation. When we look at the measurements, we can also observe greater variability for the DN(7:1) configuration which is due to few iterations during the simulation where output rate was higher than the network bandwidth in the system. What is more, our model is able to predict the best I/O management scenario among different ones, in this case dedicating core(s) employing one core per
6.1 Model Validation

Node (i.e., DC(ONE)), which fulfills the main motivation behind our model.

To further test our model's accuracy, we have conducted a set of experiments in a different cluster in Grid'5000: graphene cluster which features 40 nodes of 2.53 Ghz CPU, 4 cores/CPU, 16 GB RAM and 10 Gigabit Ethernet network. We run CM1 on 32 nodes (128 cores) with additional 6 nodes as PVFS file system. We used the same configuration for CM1 as we described for the previous set of experiments. Damaris is configured to run CM1 using three different data management scenarios: dedicating core(s) employing one core per node (i.e., DC(ONE)), and dedicating node(s) using a ratio of 15:1 for compute nodes to dedicated nodes (i.e., DN(15:1)) and using a ratio of 7:1 for compute nodes to dedicated nodes (i.e., DN(7:1)). We left out dedicating two cores per node (i.e., DC(TWO)) compare to previous configuration since each node has only 4 cores and dedicating two of them can impact the simulation significantly due decreasing the node's computational power by half.

![Figure 6.2: Observed and estimated energy consumption under different I/O scenarios on the graphene cluster](image)

We employed a profiling approach on graphene cluster to extract the necessary values for our model as we described in the previous section. After the profiling approach, we measured the energy consumption five times for each different scenario. Figure 6.2 shows the observed average energy consumption over five executions and our model's estimation. The worst relative difference we observed is 5.7% under the I/O scenario with dedicating node(s) employing a ratio of 15:1 (i.e., DN(15:1)) which indicates our estimations are accurate. What is more, our model is again able to predict the best I/O scenario, in this case dedicating node(s) with a ratio of 7:1 (i.e., DN(7:1)), which shows that we can easily apply our model to different set of clusters after a profiling approach.

Figure 6.3 displays the accuracy of our energy model with the configurations for the different set of experiments that described above. We can observe that our model's estimations are accurate regardless of the configuration (data management approach, system's architecture, application parameters) employed. Therefore, users of the HPC applications can decide which data management approach to employ before running their HPC application by looking at our model's estimations. Moreover, within each data management approach, we are able to answer the question of what is the optimal ratio of the
dedicated resources compare to the computational resources in the system.

---

6.2 Usability

In this section, we summarize our energy model and discuss its usability. Figure 6.4 illustrates our energy model and its interactions with users and HPC systems. We first perform micro-benchmark executions on the specific hardware with the targeted HPC application. Then, we integrate output of these executions into our model. This output (i.e., profiling output) consists of the power behaviors of the HPC system and scalability values of the HPC application on this specific hardware. As an another input to
our estimation framework, we collect the application specific parameters from the user. These parameters can consist of number of processes, number of nodes and number of iterations. Our energy model relies on these inputs and estimates the energy consumption of different I/O management approaches. After that, it provides the optimal I/O management approach among all possible I/O scenarios (i.e., I/O scenarios with different number of dedicated resources) according to users’ priorities.

We have demonstrated our model’s accuracy on different clusters with one of the target HPC applications of the Blue Waters sustained-petaflop supercomputer project: the CM1 atmospheric model. We can easily apply our energy model to any compute-intensive HPC application. Our model only requires profiling approach to be able to integrate application and hardware specific parameters into our estimation framework. After applying the profiling period, our model can predict the optimal data management approach according to user’s priorities.

HPC applications can have different characteristics in terms of their energy and performance trade-offs with respect to employed I/O management approach. Also, we observed the different power behaviors of same cluster nodes. These factors show the importance of the profiling method. We also know that HPC applications, particularly scientific simulations, are used for a long period of time. This usage would amortize the cost of the micro-benchmark executions for our profiling approach which should be employed only once for the specific hardware with the targeted HPC application.
Conclusion

Contents

7.1 Future work .......................................................... 44
7. Conclusion

Power consumption has started to severely constrain the design and the way HPC systems are operated. As HPC systems and HPC application's data size rise, their power efficiency calls for empirical evaluations and technical innovations. In this study, we investigated the performance and energy efficiency of three I/O approaches in HPC systems: time partitioning, dedicated cores, and dedicated nodes. We investigated the energy and performance trade-offs of three approaches within the Damaris I/O middleware and performed extensive experiments using the CM1 atmospheric model on the French Grid'5000 platform.

Our detailed study reveals a significant variation in the performance of CM1 and the energy consumption of the HPC system. Three factors at least contribute to such variations. First, the adopted I/O approach. Second, the output frequency (number of iterations between each output). Third, the architecture of the system on which we run the HPC application.

Our experimental results have motivated us to seek an energy efficient framework which can help users in their decisions regarding the data management approach. In order to make better decisions in terms of energy efficiency, we present a model that estimates the energy consumption of data management approaches. In our model, we consider the I/O approaches which employ dedicated resources (cores, nodes). We apply a profiling method in order to assess the impact of the HPC system's architecture and user defined application parameters (number of nodes, number of processes) on our model’s energy estimations. We observed a significant variability in the power values of the same cluster nodes when they are idle or running the simulation which shows the importance of our profiling approach. Moreover, we can apply our energy model to any HPC system after applying the profiling method. We have experimentally validated our energy model's accuracy on French Grid'5000 testbed for two different clusters (i.e., our model's accuracy is 96.1% on average). By providing accurate energy consumption estimations, we can help users in selecting the optimal data management approach according to their priorities before running their simulations.

7.1 Future work

As a future work, we plan to investigate the energy saving methods such as reducing the CPU frequencies of the dedicated resources (which perform I/O) within each I/O management approach. We also plan to explore full utilization of the dedicated resources (e.g., performing compression on the generated simulation data) to reduce the I/O time and storage space which can reduce the total energy consumption. As a more long-term agenda, we are going to investigate the feasibility of building a hybrid I/O management approach that not only incorporates all the aforementioned I/O management approaches but also can adaptively select the optimal I/O approach at run time.
Bibliography


