EXPLORING POWER-PERFORMANCE-QUALITY TRADE-OFFS FOR EXASCALE COMBUSTION SIMULATION

BY YUBO QIN

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Ivan Rodero

and approved by

_____

_____

_____

New Brunswick, New Jersey

May, 2016

ABSTRACT OF THE THESIS

Exploring Power-Performance-Quality trade-offs for Exascale Combustion Simulation

by Yubo Qin

Thesis Director: Ivan Rodero

The computational demand of high-performance computing (HPC) applications has brought major changes to the HPC system architecture. As a result, it is now possible to run simulations faster and get more accurate results. But behind this, power and energy are becoming critical concerns for HPC systems, e.g. Titans electric cost is about $9 million per year[1]. Power efficiency has become a critical challenge for the exascale research challenges, and U.S. Department of Energy (DOE) has set the goal to achieve exascale performance with a power budget of 20MW[2].

Current research efforts have studied power and performance tradeoffs, and how to balance these, e.g., using DVFS to meet power constraints, which significantly impacts performance. However, scientific applications may not tolerate degradation in performance and other tradeoffs need to be explored to meet power budgets, e.g., involving the application in making energy-performance tradeoff decisions.

This research focuses on studying the properties and exploring the performance and powerenergy tradeoffs of Low-Mach-Number Combustion (LMC) application which is an Adaptive Mesh Refinement (AMR) algorithm. Our experimental evaluation provides an empirical evaluation of different application configurations that gives insights into the power-performance tradeoffs space for this LMC or AMR-based application workflows.

The key contribution of this work is a better understanding of the running behavior of this AMR-based application and the power-performance tradeoffs for this application under power constraints, which can be used to better schedule power budgets across HPC systems.

Acknowledgements

I would like to start by acknowledging my family in China who has always been very understanding and encouraging during my studies. I would like to thank my advisor Prof. Ivan Rodero and Prof. Manish Parashar for their invaluable guidance, support and encouragement during this research and throughout my graduate studies. I am thankful to Jim Housell and Prof. Sandro Rigo for their excellent support and guidance. Also am thankful to Prof. Deborah Silver for being on my thesis committee. I also thank all my colleagues in RDI^2.

This presented work is supported in part by the US National Science Foundation (NSF) via grant numbers ACI 1339036, ACI 1310283, DMS 1228203, and IIP 0758566; by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the US Department of Energy through the Scientific Discovery through Advanced Computing (SciDAC) Institute of Scalable Data Management, Analysis and Visualization (SDAV) under award number DE-SC0007455; by the Advanced Scientific Computing Research and Fusion Energy Sciences Partnership for Edge Physics Simulations (EPSI) under award number DE-FG02-06ER54857; by the ExaCT Combustion Co-Design Center via subcontract number 4000110839 from UT Battelle; by the RSVP grant via subcontract number 4000126989 from UT Battelle; and by an IBM Faculty Award. The research was conducted as part of the NSF Cloud and Autonomic Computing (CAC) Center at Rutgers University and the Rutgers Discovery Informatics Institute (RDI^2).

Dedication

To my parents, dear brothers and sisters and all the prayers.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

High performance computing (HPC) was introduced in the 1960s, they have been played an important role in the field of computational science. From design perspective, HPC were built to maximize performance while irrespective of power and energy consumption, though their energy consumption has already occupied a large part of cost (i.e., operation cost). However, as we are approaching the exascale era, power is turning from an optimization goal to a critical operation constraint. U.S Department of Energy (DOE) has currently set a bound of 20MW for an exascale system.[3] This strict power constraint poses a hard research challenge with current hardware and software. Tianhe-2, the top one supercomputer as of 2015, has a peak performance of 54.9 PetaFLOPS at 17.8 MW, which is 1.9 GigaFLOPS per watt. However, achieving the goal of exascale computing at 20 MW, it requires 50 GigaFLOPS per watt. So that, current HPC system still need at least a 26 times power efficiency improvement towards exascale.

In order to achieve this exascale system power constraint, current research efforts have studied power and performance tradeoffs, and how to balance these.[4, 5, 6, 7, 8, 9] Many power management strategies have been proposed[10, 11, 12, 13, 14], but most of the work is tend to choose a performance loss for the application and then attempt to constraint the power consumption under that performance loss. Even if this can meet power constraints, it significantly impacts performance. Since scientific applications may not tolerate degradation in performance, other tradeoffs need to be explored to meet power budgets. At the same time, it is clear that future HPC system will have a whole-system power constraint that will be filtered down to job-level power constraint. This indicates that the application should be involved in making tradeoffs decisions.

This research targets a Center for Exascale simulation of Combustion in Turbulence (ExaCT) Low-Mach-number Combustion simulation code (LMC). [15] This combustion simulation is developed based on Adaptive-Mesh-Refinement (AMR) algorithm. The AMR method can be customized to resolve problems at different resolution levels. This property motivates us to find the potential tradeoffs to run applications in power-constrained environments without impacting performance by tolerating lower resolution/quality levels.

1.2 Problem description

Whole-system power constraints will be filtered down to job-level power constraints, i.e., power budgets need to be managed at application or workflow level. This research use power capping techniques to meet the job-level power constraints while maintaining the performance.

Nowadays, many research efforts have studied power and performance tradeoffs, and most energy models or strategies are based on runtime (e.g., leveraging MPI slack) for power clamping or power capping techniques, like Dynamic Voltage and Frequency Scaling (DVFS) to constraint the power. However, power and performance are in the two sides of a balance scale, that it is hard to improve one side without scarifying the other one. Therefore, one of the key problems addressed in this research is keeping the power bound (or budget) without losing performance, which is challenging for real world application targeting exascale.

To address this challenge, we believe that the applications should be involved in making tradeoff decisions. For this specific application, LMC combustion simulation code, we propose to exploit AMR unique properties (e.g., dynamically resolution adjustment) together with power capping to address this issue.

1.3 Overview of the approach

In this work, AMR codes consider a hierarchy of grids of differing resolution ranging from the coarsest to the finest. It can focus computational resources in regions of interest but decrease computing resolution in regions with less interest. Less resolution means lighter workload and less power consumption. Therefore, this flexible resolution property gives us a potential opportunity to extract power budget for other usage. In order the take best advantage of it, this work first studies the mechanisms and policies to control AMR properties, and then, it characterizes LMC power performance running on different number of cores with different levels of resolution (e.g., levels of refinement in AMR). Finally, the characterization is complemented with power capping techniques (i.e., RAPL). The overarching goal of this thesis is to understand the tradeoffs between power-performance and quality, and building models taking into account AMR properties for managing power budgets and workflow at scale.

1.4 Contribution

The contributions of this work are summarized below:

1. It presents an empirical evaluation of different configurations of application that gives insights into the energy-performance-quality tradeoff for scientific data-driven workflows.

2. It provides a comprehensive study of this LMC simulation performance, quality, and power and energy behavior.

3. It presents a proof-of-concept study of potential of power capping and power management to balance power-performance-quality tradeoffs.

1.5 Thesis organization

The rest of this thesis is organized as follows. Chapter 2 summarizes the related work. Chapter 3 presents background information related to the techniques used in this research. Chapter 4 describes the evaluation methodology and presents the results of the

experimental evaluation. Chapter 5 concludes the chapter and outlines ongoing and future research.

Chapter 2

Related work

2.1 Energy efficiency

Energy-efficiency has become a critical concern for HPC applications. There are many approaches have been proposed to obtaining energy savings during HPC application execution. Some of them are to focus on identifying stalls during the execution by measuring architectural parameters from performance counters as proposed in[4, 5, 6]. In addition to using performance counters, Rountree et al.[10] developed a runtime system called Adagio, by doing the critical path analysis, it can determine which tasks may be slowed down and also suitable opportunities to apply DVFS to minimize the performance loss in the parallel execution. This achieves significant energy saving in scientific application with negligible performance lose. However, this approach appears beneficial when applications have computation or communication imbalances among participating processes, which is typically not the case for a highly efficient parallel application nor suitable for the LMC program targeted in this work. Some approaches in[7, 11] are proposing to determine the communication phases to apply DVFS. Kandalla et al.[16] give algorithms to save energy in the collectives such as MPI_Alltoall and MPI_Bcase. Moreover, Ioannou et al.[12] describe a runtime system for the Intel Single-chip Cloud Computer (SCC) processor to detect repeatable communication phases followed by an application of frequency scaling. Donofrio et al.[8] have studied energy efficiency for extreme-scale science and developed Green Flash, an application-driven design to improve the kernels computational efficiency. Li et al.[13] developed a new power-aware performance prediction model of hybrid MPI/OpenMP programming model and combine memory and disk management techniques to provide performance guarantees for control algorithms. Rodero et al.[14] explored the potential

of application-centric aggressive power management of HPC scientific workloads while considering power management mechanisms and controls available at different levels and different subsystems. Gmall et al.[9] explored data-related energy/performance trade-offs for end-to-end co-design simulation workflows on current and on-going high-end computing systems. Lively et al.[17] explored and investigated energy consumption and execution time of different parallel implementation of scientific applications on multi-cluster systems.

Most of the work described above are from system level and are based on tolerating performance loss and constrain the power consumption under that performance loss. The work presented here discusses the possibility to filter down power constraint to job-level, and then take advantage of applications specific properties, such as resolution, to keep the power budget while maintaining the level of performance.

2.2 Power management

The cost of provisioning power in data centers is a very large fraction of the total cost of operating a data center.[18, 19] Therefore the ability to cap peak power consumption is a desirable feature in modern data centers.[20] Many power management approaches have been proposed to provide performance guarantees while constraining power budget. Sartori et al.[21] describe a peak power management technique for multi-core systems by choosing the power state for each core that meets the power constraints. Cebrian et al.[22] develop a power balancing strategy that through borrowing power budgets from cores that consume lower power to dynamically adapts the per-core power budgets. While, Gandhi et al.[23] give a power capping strategy to meet the power budget by inserting idle cycles during execution. This approach aims at controlling the average power consumption, but cannot guarantee the peak power. So a number of other approaches are proposed through reconfiguring hardware to meet the power budget. Meng et al.[24] provide a power management strategy through dynamic reconfiguration of cores by cache resizing. Konotorinis et al.[25] propose a table-driven adaptive core reconfiguration technique that configures core resources such as load-store

queues and floating point units to meet peak power budget. Most of the power management strategies are using DVFS. Since Intel SandyBridge family processors, Intel provide Running Average Power Limit (RAPL) for controlling the power constraint on processors and memory. Several studies have begun to evaluate the RAPL power management system. Rountree et al.[26] explore RAPL as a replacement for DVFS in HPC systems by evaluating power consumption for package and memory subsystem. Zhang et al.[27] give a systematic evaluation of RAPL behavior such as stability, setting time, overshoot and, etc. RAPL also been used to study application runtime variability and power optimization for exascale computing in work of Allan et al.[28][21] Sarood et al.[29] use RAPL to set power bounds on across an over-provisioned cluster running homogeneous application processes. This work also takes advantage of RAPL power controlling ability to dynamically manage power consumption.

2.3 Power measurement systems

Measuring the power/energy consumption of software components is the key for energy-aware scheduling, accounting and budgeting. RAPL measurement mechanism is described in[28] and Marcus et al.[30] has investigated the RAPL measurements performance and discussed the practical obstacles that existed in performing these measurements on complex modern CPUs. Vignesh et al.[31] studied the greenness of the in-situ and the post-processing visualization pipelines using RAPL to measure the CPUs and RAMs power consumption and with an average error rate of less than 1%. Venkatesh et al.[32] use RAPL to measure energy consumption in large message-passing applications.

Chapter 3

Background

3.1 Overview of combustion simulation software

Nowadays, more then 85% energy in the US is provided by combustion. Meanwhile, the internal combustion engine is still widely used in the world. Those place enormous pressure to improve the combustion efficiency in both power generation devices and transportation. For this purpose, combustion simulation is important for scientists to gain insight of turbulent flame combustion process. But simulating practical-scale combustion is a challenging work. Because typically, combustion process often involves hundreds of species and thousands of chemical reactions, and the problem is inherently multi-scale both in time and space.[15] At same time, from an engineering point of view, practical flows in combustion simulation are mostly turbulent.[33] Turbulent combustion models use systematic mathematical derivations based on the Navier-Stokes equations.[34] Traditionally, people use direct numerical simulation (DNS) approaches to solve Navier-Stokes equations. This approach is based on explicit numerical methods for the compressible flow equations on uniform grids. This DNS is good to be used in small idealized combustion simulation problems geared at the fundamental nature of turbulence/chemistry interactions. However, this approach is not ideal for large-scale combustion simulation. On the one hand, it requires very fine spatial grids to resolve the local flame structure. On the other hand, it requires small time steps to resolve the acoustic and chemical time scales inherent in the model.

In LMC combustion code, the Center for Computational Sciences and Engineering (CCSE) has taken a different research approach that explicitly targeted both the temporal and spatial multi-scale aspects of combustion modeling. First, instead of the traditional compressible equations, a low Mach number formulation is used. Second,

adaptive mesh refinement (AMR) is used to increase computational utilization that focus computational resources in regions of interest but decrease computing resolution in regions with less interest. Third, robust integration methods are employed to allow reasonable solution behavior with a minimum of computational resolution. By those three features, the computational requirements of this combustion simulation has been reduced by a factor of 10,000 relatives to traditional approaches.[35]

This combustion code is embodied in a hybrid C++/FORTRAN software system. In this framework, memory management and control flow are expressed in the C++ portions of the program and the numerically intensive portions of the computation are handled in FORTRAN. The software is written using a layered approach, with a foundation library, BoxLib, that is responsible for the basic algorithm domain abstractions at the bottom, and a framework library, IAMR, that marshals the components of the AMR algorithm, at the top. Support libraries built on BoxLib are used as necessary to implement application components such as interpolation of data between levels, the coarse/fine interface synchronization routines, and linear solvers used in the projections and diffusion solves.[36]

3.2 BoxLib

BoxLib was developed at the Center for Computational Sciences and Engineering (CCSE) at Lawrence Berkeley National Laboratory. It is a software library containing all the functionality for building massively parallel, block-structured adaptive mesh refinement (AMR) applications for solving time-dependent PDEs in two and three dimensions.[37] Following sections give a high-level overview of Boxlib features.

3.2.1 Parallel Programming Model

MultiFab is the fundamental parallel abstraction in BoxLib, which holds the data on the union of grids at a level of refinement. A MultiFab is composed of multiple Fortran array boxes and each Fab is a multidimensional array of data on a single grid. Fabs at each level of refinement are distributed independently, thus each processor can operate

independently on its local data. But for operations that require data owned by other processors, the MultiFab operations are preceded by a data exchange between processors to fill ghost cells. Each processor contains meta-data that is needed to full specify the data locality and processor assignments of the Fabs.

3.2.2 Hybrid MPI-OpenMP

The basic parallelization strategy uses a hierarchical programming approach for multicore architectures based on both MPI and OpenMP. In the pure-MPI instantiation, each Fab is assigned to a core, and each core communicates with every other core using only MPI. In the hybrid approach, where on each socket/node there are n cores that all access the same memory, we can divide our domain into fewer, larger grids, and assign each Fab to a socket/node, with the work associated with that grid distributed among the n cores using OpenMP.

3.2.3 Checkpoint and Plotfile

Data for checkpoints and analysis are written in a self-describing format that consists of a directory for each time step written. Checkpoint directories contain all necessary data to restart the calculation from that time step. Plotfile directories contain data for post-processing, visualization, and analytics, which can be read using VisIt. Checkpoint and Plotfiles are portable to machines with a different byte ordering and precision from the machine that wrote the files.

3.3 Adaptive Mesh Refinement

Variety of approaches have been used to reduce the computational costs of simulating combustion flows and Adaptive Mesh Refinement (AMR) turn out to be a very effective one. Berger and Oliger developed AMR for computing time-dependent solutions to hyperbolic partial differential equations in multiple space dimensions.[38] Later on, this AMR approach has been developed for a variety of engineering problems.[39, 40, 41] The beauty of AMR is that it focus computational resources in regions of interest

but decrease computing resolution in regions with less interest. This largely increase computational utilization and therefore minimizing memory and storage requirements. It is powerful especially in treating problems with multiple scales.

AMR is based on a sequence of nested grids with successively finer resolution in both time and space. In this approach, fine grids are formed by dividing coarse cells by even integer factor R_l in each direction. R_l is called refinement factor, it is the ratio in resolution between consecutive levels, and is uniform in all spatial directions. R_l is typically 2 or 4. Increasingly finer grids are recursively embedded in coarse grids until the solution is adequately resolved with each level contained in the next coarser level. An error estimation procedure based on user-specified criteria evaluates where additional refinement is needed and grid generation procedures dynamically create or remove rectangular fine grid patches as resolution requirements change.

3.3.1 AMR algorithm

A general AMR formulation is presented below.

Advance (l,t)

If (time to regrid) **then**

Regrid (l)

FillPatch (l,t)

Integrate (l,t, Δt_l)

If (l ¡ l_{finest}) **then**

For i_{sub} = 1, , r_l

Advance (l + 1, t + (i_{sub} - 1) Δt_{l+1})

Average down (l, t + Δt_l)

Reflux (l, t + Δt_l)

End if

Regrid (l): generate new grids at levels l + l and higher

FillPatch (l, t): fill patch of data at level l and time t

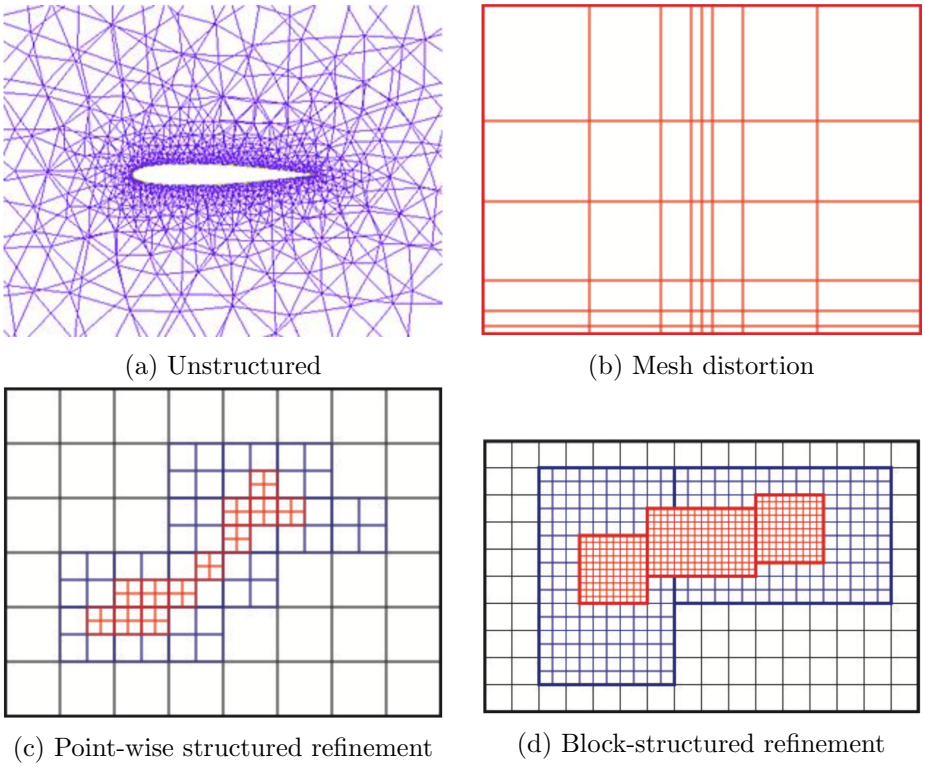
Integrate (l, t, Δt): Advance data at level l from t to t + Δt, averaging and storing fluxes at boundaries of level l grids if l ¿ 0 and level l cells at boundary of $l + 1$

Average down (l,t): average (in space) level $l + 1$ data at time t to level l

Reflux (l,t): Add (time- and space-averaged) refluxing corrections to level l cells at time t adjacent to level $l + 1$ grids

3.3.2 Types of Refinement

Adaptive mesh refinement has several type of refinement grids. It includes the following main four types as illustrated in Figure 3.1.



(a) Unstructured (b) Mesh distortion

(c) Point-wise structured refinement (d) Block-structured refinement

Figure 3.1: Type of refinement

3.3.3 Grid Hierarchy and time-stepping procedure

The adaptive mesh refinement (AMR) algorithm uses a hierarchical grid structure, in Figure 3.2, which composed of rectangular, uniform grids of varying resolution. Each refinement level is represented as a union of rectangular grid patches of a specific resolution contained within the computational domain. By definition, level 0 covers the

entire problem domain. The widths of the cell in the level l grids differ from those at $l+1$ by refinement factor R_l. The grids are properly nested, i.e. the union of grids at level $l+1$ is contained in the union of grids at level l. The containment is strict in the sense that, except at physical boundaries, the level l grids are large enough to ensure that there is a border at least one level l cell surrounding each level $l+1$ grid.

During the adaptive mesh process, the AMR timestep consists of separate timesteps on each of the levels. Those including synchronization operations that insure correct behavior at the coarse-fine interfaces, and regridding operations which permit the refined grids to track complex and/or interesting regions of the flow. The ratio of the level l and the level $l+1$ time steps is R_l. Figure 3.3 shows a space-time diagram of a single level 0 timestep, during which a regridding operation moves the interface between levels 1 and 2.

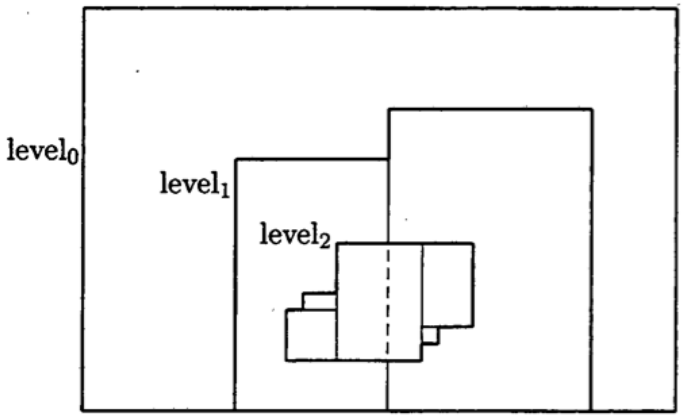


Figure 3.2: AMR hierarchical grid structure

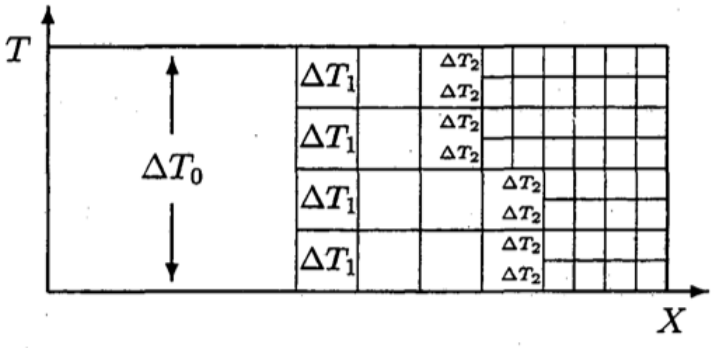


Figure 3.3: AMR space-time diagram

3.4 Combustion simulation mathematical model

The fundamental mathematical model of combustion simulation is the compressible Navier-Stokes equations for a reacting flow.[36] Navier-Stokes equations, the formulation below , is named after Claude-Louis Navier and George Gabriel Stokes, describe the motion of viscous fluid substances.[42] It governs the motion of fluids and can be seen as Newton's second law of motion for fluids.

$$\rho(\frac{\partial u}{\partial t} + u \cdot \nabla u) = -\nabla p + \nabla \cdot (\mu(\nabla u + (\nabla u)^T)) - \frac{2}{3}\mu(\nabla \cdot u)\mathbf{I}) + \mathbf{F} \tag{3.1}$$

where u is the fluid velocity, p is the fluid pressure, ρ is the fluid density, and μ is the fluid dynamic viscosity. These equations are always solved together with the continuity equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0 \tag{3.2}$$

In the LMC, it has considered a multicomponent gaseous mixture ignoring Soret and Dufour effects, body forces and radiative heat transfer, and assume a mixture model for species diffusion.[43, 44] The governing equations, which express conservation of mass, momentum and energy, augmented by species transport, are:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho U = 0 \tag{3.3}$$

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot \rho UU + \nabla p = \nabla \cdot \tau \tag{3.4}$$

$$\frac{\partial \rho E}{\partial t} + \nabla \cdot \rho UE + pU = \nabla \cdot \lambda \nabla T + \sum_m \nabla \cdot h_m \rho D_m \nabla Y_m + \nabla \cdot \tau U \tag{3.5}$$

$$\frac{\partial \rho Y_m}{\partial t} + \nabla \cdot \rho U Y_m = \nabla \cdot \rho D_m \nabla Y_m - \dot{\omega}_m \tag{3.6}$$

where ρ is the density, U is the velocity, E is the total energy, Y_m is the mass fraction

of species m, T is the temperature, and $\dot{\omega}_m$ is the net destruction rate for species m due to chemical reactions. Also, λ is the thermal conductivity, τ is the stress tensor, c_p is the specific heat of the mixture, and $e_m(T)$, $h_m(T)$ and D_m are the internal energy, enthalpy and species mixture-averaged diffusion coefficients of species m, respectively. We note that for these equations, we have $\sum Y_m = 1$, $\sum \rho D_m \nabla Y_m = 0$, and $\sum \dot{\omega}_m = 0$ so that the sum of the species transport equations gives the conservation of total mass. These evolution equations are supplemented by an equation of state for a perfect gas mixture:

$$p_0 = \rho R_m i x T = \rho \mathcal{R} T \sum_m \frac{Y_m}{W_m} \tag{3.7}$$

where W_m is the molecular weight of species m, and R is the universal gas constant.

To simulate reacting flow, one possible approach is to simply discretize this system of equations above. However, the compressible flow equations provide a general description of essentially continuum fluid dynamic phenomena. Therefore, an alternative approach to directly discretize the compressible equations is to exploit the separation of scales between the acoustic waves and the fluid motion by adopting a low Mach number formulation

The low Mach number combustion formulation was first introduced by Rehm and Baum [45] and was later derived rigorously from low Mach number asymptotic analysis by Majda and Sethian [46]. It is a classical solution of the compressible Navier-Stokes or Euler equations for non-isentropic fluids. The Mach number, is denoted by M, is a fundamental dimensionless number. By definition, it is the ratio of a characteristic velocity in the flow to the sound speed in the fluid.[47] The basic steps of the analysis are first to normalize the problem by rescaling and then to expand the terms in the compressible Navier-Stokes equations in M. Equating terms in Mach number and examining the behavior as M $\rightarrow$ 0 one can show that in an unconfined domain, the pressure can be decomposed as:

$$p(x,t) = p_0 + \pi(x,t) \tag{3.8}$$

where p_0 is the ambient thermodynamic pressure and is a perturbational pressure field that satisfies $\pi/p_0 \sim \mathcal{O}(M^2)$. (In a more general setting, p_0 is a function of t.) With this decomposition, p_0 defines the thermodynamic state; all thermodynamic quantities are independent of π.

With this decomposition, the low Mach number equations for an open domain are

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho U = 0 \tag{3.9}$$

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot \rho UU + \nabla \pi = \nabla \cdot \tau \tag{3.10}$$

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot \rho Uh = \nabla \cdot \lambda \nabla T + \sum_m \nabla \cdot h_m \rho D_m \nabla Y_m \tag{3.11}$$

$$\frac{\partial \rho Y_m}{\partial t} + \nabla \cdot \rho U Y_m = \nabla \cdot \rho D_m \nabla Y_m - \dot{\omega}_m \tag{3.12}$$

Thought these still similar to the compressible equations, the low Mach number model expresses the energy equation in terms of enthalpy, $h(T, Y_m) = \sum_m h_m(T) Y_m$. More importantly, acoustic waves are instantaneously equilibrated and the equation of state, $p_0 = \rho R_m ix T$, now constrains the evolution. As a result, this description retains compressibility effect due to heat release but removes the time scale associated with acoustic wave propagation from the dynamics of the system. In this form, the perturbation pressure, π, plays the role of a Lagrange multiplier to constrain the evolution so that this constraint is satisfied.

These equations are supplemented by an equation of state for a perfect gas mixture:

$$\rho c_p (\frac{\partial T}{\partial t} + U \cdot \nabla T) = \nabla \cdot \mu \nabla T + \sum_m (\rho D_m \nabla Y_m \cdot \nabla h_m + h_m \dot{\omega}_m) \tag{3.13}$$

3.5 RAPL

Since Intel Sandy Bridge processors, Intel has introduced capabilities for both onboard power meters and power clamping. The Intel Running Average Power Limit (RAPL) provides a standard interface for measuring and limiting processor and memory power by HW, OS, applications and, etc. The power limit and the time window (Power, TimeWindow) form essential parameters of RAPL interface.[48, 49]

Users can measure and control processor power consumption by using several model-specific registers, or MSR using two basic privileged instructions, *readmsr* and *writemsr* through the MSR kernel module. This module exports a file interface at **/dev/cpu/N/msr** (with N being the CPU number) that, given suitable file permission, can be used to read and write any MSR on the node.

Intel has separated the Sandy Bridge family into two classes namely client and server. The two architectures share a subset of RAPL features where the server class provides additional features such as DRAM power measurement. The Sandy Bridge architecture supports three power domains on both server and client architectures. In Figure 3.4, both architectures support package (PKG) and Power Plane 0 (PP0) domains, while the server adds a separate DRAM domain and the client adds a second power plane (PP1).

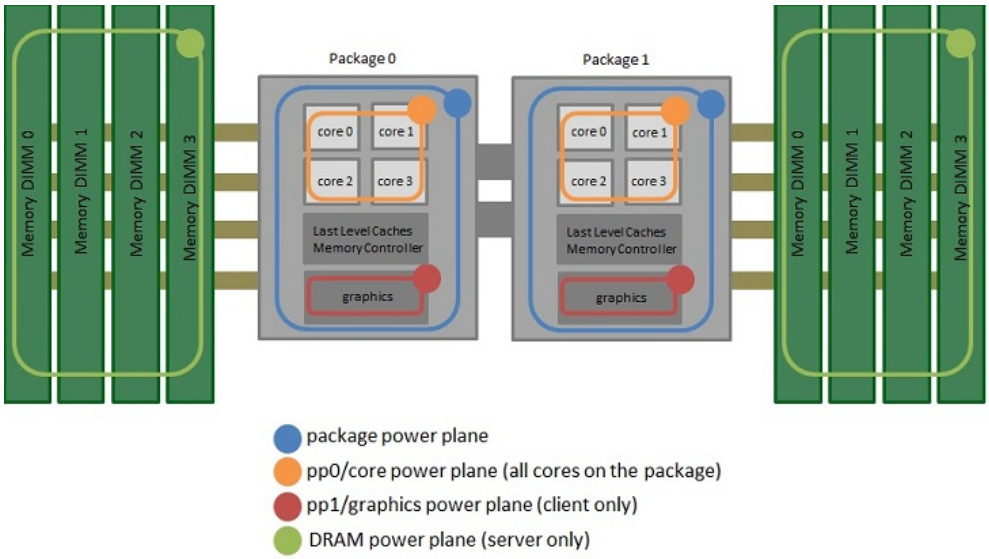


Figure 3.4: Power domains for which power monitoring/control is available

3.5.1 Power capping

Exascale computing will be power limited. To satisfy this requirement, either to limit the number and speed of processors or, more likely, using power capping to enforce execution under the power budgets. Power capping techniques can provide the possibilities of energy savings for computing devices. As the name implies, power capping limits the instantaneous power used by a computing device. It is widely used to fulfill power constraints and save energy. More over, it is also implemented for power budget management and thermal control for data centers and computing devices.

In order to keep processor power consumption under the user-defined threshold, RAPL has combined automatic DVFS and clock throttling techniques. But unlike most proposed mechanisms that maintain instantaneous power limits, RAPL employs an internal model of energy consumption, to estimate and maintain an average power limit over a sliding time window. The power limit and the time window form essential parameters of RAPL memory interface. Multiple limits can be set simultaneously for different power and thermal constraints. RAPL can control and measure power in three distinct domains, including the cores and caches, the entire CPU, or the DRAM.[50]

3.5.2 RAPL measurement

RAPL provides capabilities to measure energy and power usage of different parts of the silicon chip. RAPL supports approximately 1ms resolution measurements. Its sensor data is read from model-specific registers (MSR) and exported by the Linux kernel via devfs, which makes these measurements readily available to profiling tools.[51]

3.5.3 RAPL algorithm

RAPL algorithm determines the power budget for the next interval based on memory bandwidth, specified power limit, time window and a history of power consumption. This algorithm aims to deterministically maintain a power limit while maximizing memory bandwidth and performance. RAPL algorithm including three formulas. The first equation calculates the hard limits and the second one computes the soft limits. The

third equation is used for updating the M states in Table 3.1.

The hard limit is computed at beginning of each interval and it is calculated by subtracting the power consumed over the last N-1 intervals from the available power budget. However, only use hard limit to determine M-state from Table 1 and therefore control the memory bandwidth and limits might lead to several drawbacks. Firstly, M-state in Table 1 represent worst case power based on worst case memory bandwidth and access pattern allowed in this state, thus it is unlikely to represent any realistic workload or its true memory power consumption. Secondly, the algorithm uses the entire window in computing the power budget for the next interval, which cannot represent the most recent workload phase behavior. Also, it tends to allocate the entire available budget in single intervals which has largely bad effect for multithreaded server workloads in multi-core systems. To address these issue, RAPL also computes the soft limits that shift the time window by M intervals to capture the most recent workload behavior. And by predicting average bandwidth demand over the next M time intervals, it can smooth the effects of power limit. Moreover, instead of using wore case power in Table 1, it updates them at the end of each time interval using the Weighted Running Average (WRA) formula. And use the updated M-state table to select a power state for next interval.

$$PwrBudget_{hard} = N \cdot PwrLimit - \sum_{i=1}^{N-1} MemoryPwr_i \tag{3.14}$$

$$PwrBudget_{soft} = \frac{N \cdot PwrLimit - \sum_{i=M}^{N-1} MemoryPwr_i}{M} \tag{3.15}$$

$$WRA^{new}[M] = a \cdot MemoryPwr + (1 - a) \cdot WRA^{curr}[M] \tag{3.16}$$

MPL STATES	**BANDWIDTH** (%)	**POWER (W)**
MPL0	100	11.9
MPL1	80	10.5
MPL2	66.67	9.46
MPL3	57.14	8.70
MPL4	50.0	8.12
MPL5	44.44	7.67
MPL6	40.0	7.30
MPL7	36.16	6.31
MPL8	33.33	6.01
MPL9	30.25	5.69
MPL10	27.07	5.35
MPL11	23.58	4.99
MPL12	20.0	4.61
MPL13	16.34	4.23
MPL14	12.33	3.80
MPL15	8.24	3.37

Table 3.1: M-state, represent worst case power based on worst case memory bandwidth and access pattern allowed in this state

Chapter 4

Experimental evaluation

4.1 Infrastructure overview

4.1.1 CAPER Cluster

CAPER (Computational And data-enabled Platform for Energy efficiency Research) is a unique and flexible instrument funded by The National Science Foundation that combines high performance Intel Xeon processors with a complete deep memory hierarchy, latest generation co-processors, high performance network interconnect and powerful system power instrumentation.

It is currently an eight-node cluster, which is capable of housing concurrently, in one node up to eight general-purpose graphical processing units (GPGPU), or eight Intel many-integrated-core (MIC) coprocessors - or any eight-card combination of the two; and up to 48 hard disk drives (HDD), or solid-state drives (SSD). A single-node configuration features a theoretical peak performance of 20TF/s (single precisions) or 10TF/s (double precision) and supports up to 32TB of storage and 24TB of flash-based non-volatile memory. A separate node serves as a login and storage server.

This hardware configuration is unprecedented in its flexibility and adaptability as it can combine multiple components into a smaller set of nodes to reproduce specific configurations. This platform also mirrors key architectural characteristics of high-end system, such as XSEDE's Stampede system at TACC, and provides several unique features to support critical research goals such as software/hardware co-design. CAPER provides a platform to validate models and investigate important aspects of data-centric and energy efficiency research.

4.1.2 DELL Cluster

DELL platform includes a 32 node Dell M610 blade cluster, consisting of two Dell M1000E Modular Blade Enclosures, necessary interconnect/management infrastructure, and a supervisory node. Each enclosure is maximally configured with sixteen blades, each blade having two Intel Xeon E5504 Nehalem family quad-core processors at 2.0 GHz, forming an eight core node. Each node has 24 GB RAM and 73 GB of local disk storage (10,000 RPM), twelve nodes have an additional 1 TB of local storage. The network infrastructure is comprised of an integrated 16-port Mellanox InfiniBand switch within each blade chassis, each switch linked to the switch in the other chassis. All blades have Mellanox Quad-Data-Rate (QDR) InfiniBand interface cards. There is also an integrated (redundant) 1 Gigabit Ethernet within each chassis, with two pairs of 10 Gigabit uplink capabilities in each chassis. In the aggregate, the cluster system consists of 32 nodes, 256 cores, 768 GB memory and 14.5 TB disk capacity, with a 20 Gigabit InfiniBand network and two 1 Gigabit Ethernet networks.

4.1.3 Power measurement methodology

Our experiment platform CAPER is instrumented with both coarse- and fine-grained power metering at server level. On the one hand, an instrumented Raritan iPDU PX2-4527X2U-K2 provides power measurements at 1 Hz. While, on the other hand, a Yokogawa DL850E ScopeCorder provides voltage and current measurement for all nodes through 1 Ms/s modules, and can sample power data at rate of up to 10 KHz. In addition to those server level measurement, we have RAPL meter to provide power measurement at a up to 20Hz sampling rate in processor level.

Use of RAPL power metering

RAPL meter measuring power through reading Machine-Specific Registers (MSRs). Intel released Power Gadget API for using RAPL. This API is a framework that provides very comprehensive information including reading current estimated processor power, current processor frequency, base frequency, thermal design power (TDP), current temperature, timestamps and etc. Also, Intel gives a ready-to-use software-based power

monitoring tool called Intel Power Gadget, this has a completed interface for using RAPL. We are using this to measure the processors power usage.

RAPL meter sampling frequency considerations

Thought, according to Intels manual,[48] RAPL MSRs would update at rate of once every 1 ms, using very lower frequency (e.g. less than 20 ms) may result in significant overhead and might also increase the power consumption, which would make the data less meaningful.[52] Also, since the instantaneous processor frequency would change very frequently, sampling data may be more useful if you sample often and average the samples overtime. So, Intel recommend sampling frequency of 50 ms or upper. For our experiment, we dont require high resolution power data, so we choose to sample twice per second.

Reading power measurement data from PDU

Raritan PDU keep measuring the whole servers power consumption and writing the power measurement data to power log files with timestamp. CAPER has eight nodes, so we extract real-time power measurement data and the timestamp from each nodes power log file after simulation started using Simple Network Management Protocol (SNMP) queries to the PDU from a side script running in an independent node.

4.2 Exploration of LMC load imbalance exploration

LMC is a massively parallel distributed program. Running on thousands processors are very likely have load imbalance problem. Especially in AMR, the number of grids also changes and is seldom an integer multiple of the number of processors, it is therefore inefficient to assign the grids sequentially to the processors, and it is very likely to be load imbalanced. Therefore, this experiment is aimed at exploring LMC program load balancing. There are many studies on AMR load imbalance issue,[53, 54, 55] also many strategies and balancers have been introduced to address it.[56]

In order to understand the application characteristics, we first measure LMC run-time power consumption. Then we use profiling tools to trace its running behavior, trying to find out how much time does it spend on MPI operations such as MPI_Wait.

The goal is to evaluate the possibility to cap power during imbalanced regions.

To trace LMC running behavior, we first tried out some profiling tools, such as PAPI, Perf and TAU. However, we concentrated in the built-in profiling mechanisms already implemented in BoxLib, i.e., we used AMRprofParser to analyze the profiling data.

Measuring power consumption

The command provided below runs LMC on 64 cores under resolution from level 3 to level 1. Intel *power_gadget* is used to measure real-time power consumption.

```
mpirun -machinefile hostfile.txt -np 64 .LMC2d.Linux.g++.gfortran.SDC.MPI.ex
inputfile amr.max_lev=#level
```

Profiling LMC running behavior

To enable LMC profiling function, we need to recompile it with following configuration setting:

```
USE_MPI=TRUE
PROFILE=TRUE
TRACE_PROFILE=TRUE
COMM_PROFILE=TRUE
```

Then we can get this executable file:

```
LMC2d.Linux.g++.gfortran.COMTR_PROF.SDC.MPI.ex
```

Running this program, in addition to plotfile and checkpoint file, we can get profiling file bl_prof_D_00XX in folder bl_prof.

To analyze the resulting data, we need to use AMRProfParser. This tool is a standalone profiler program existed in BoxLib library.

To use AMRProfParser, we need to compile the code with following configuration setting:

```
USE_MPI=FALSE
PROFILE=TRUE
TRACE_PROFILE=TRUE
COMM_PROFILE=FALSE
```

Then we get:

amrprofparser2d.Linux.g++.gfortran.DEBUG.TRACE_PROF.ex

To use it, we can type appropriate options and profiling data directory to it as the following format.

./amrprofparser2d.Linux.g++.gfortran.DEBUG.TRACE_PROF.ex [options] profd-dirname

For example, with option [-ws] it is possible to get a profile summary.

Discussion of results

Figure 4.1 shows the power consumption curve of running LMC on 64 cores. The curve is very flat, which means LMC keep the processors in the maximum performance for the whole time. From this point of view, LMC is running in a balanced way. Although it looks good from the power consumption curve, there still exist possibility that processors are running on highest power without doing actual work. Therefore, we continue to trace LMC function call and function execution time.

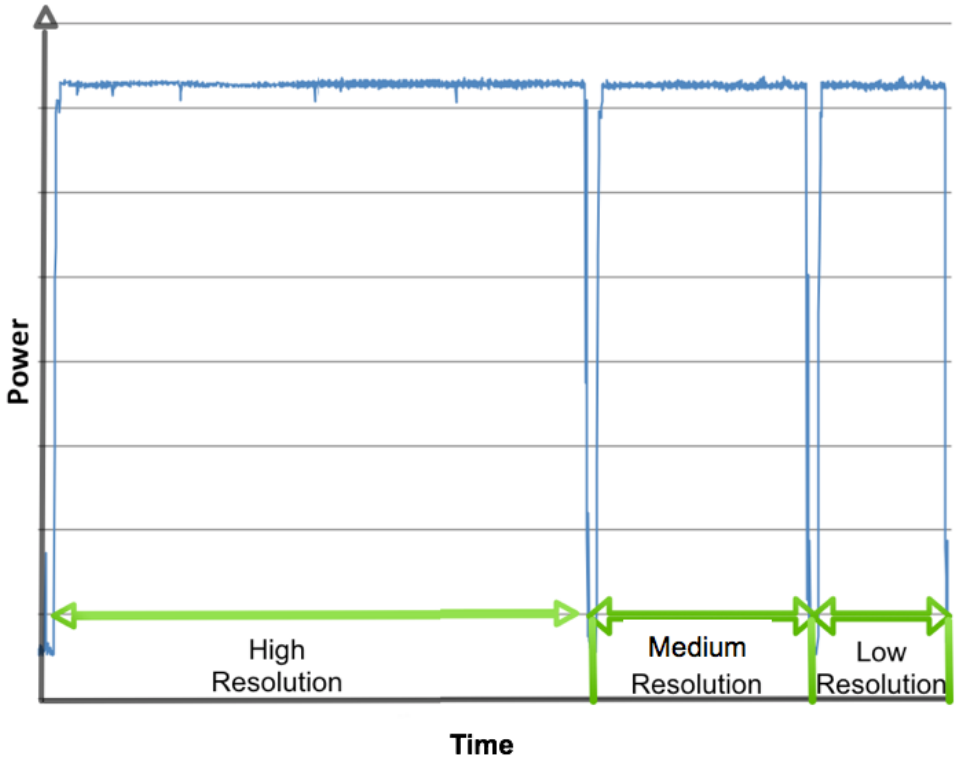


Figure 4.1: LMC power consumption curve with different (qualitative) resolution levels

Table 4.1 show the top 10 function calls when running LMC with level 4 on 128 cores.

Function name	**Number of calls**
Waitall	277,328,482
AsendTsii	146,652,558
ArecvTsii	146,652,558
AllReduceR	46,855,296
Alltoall	11,975,168
Alltoallv	11,975,168
ReduceR	502,784
Allgather	315,648
BCastTsi	84,992
AllReduceI	84,096
NameTag	59,136

Table 4.1: Top 10 number of call functions

Figure 4.2 shows the percentage of selected funcion calls using level 1 to level 3 on profiled LMC.

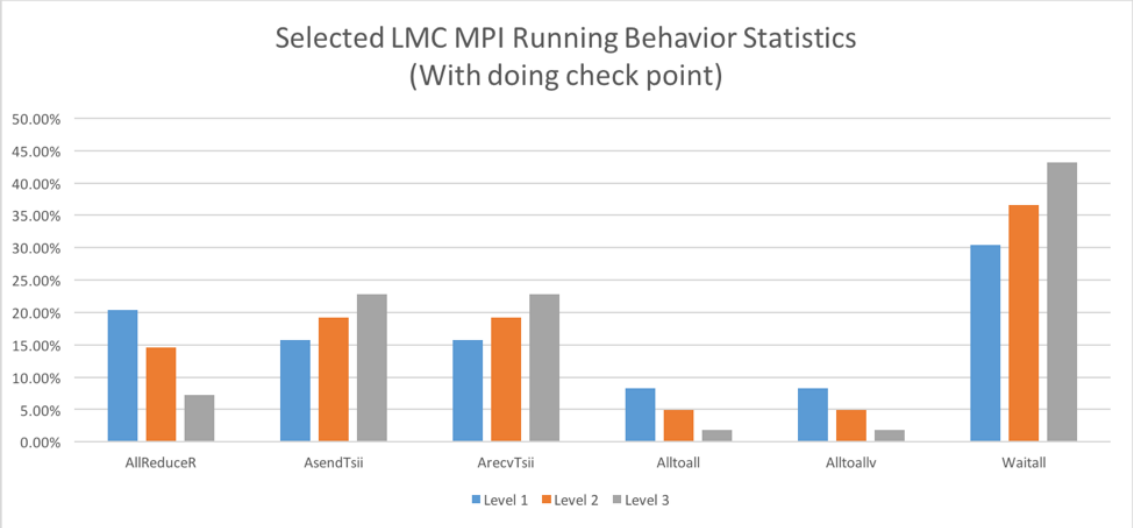


Figure 4.2: Selected LMC MPI Running Behavior Statistics

We can see that function Waitall occupied most function call, and the higher level of refinement, the more Waitall function will be called. However, the number of function calls does not equal to the time program spend on it. So, we continue to profile the time spend on each function, which is shown in Table 4.2. This table shows that the most time consuming function *Amr::coarseTimeStep()* occupied 76.94% execution time but only been called 20 time. While function *LinOp::prepareForLevel()* has been called 8297520 times but only occupies 1.75% execution time.

According to [36], LMC code has embedded load balance scheme which described in Crutchfiled[57] and in Rendleman et al.[58] This load balance scheme is based on a dynamic programming approach for solving the knapsack problem: the computational work in the irregularly sized grids of the AMR data structures is equalized among the available processors. After the initial allocation of grids some additional changes to the grid distribution are performed to reduce communications between processors. Also, **BoxLib** fundamental parallel abstraction is the **MultiFab**, and **MultiFab** operations are performed with an *owner computes* rule with each processor operating independently on its local data. For operations that require data owned by other processors, the **MultiFab** operations are preceded by a data exchange between processors. Each processor contains *meta-data* that is needed to fully specify the geometry and processor assignments of the **MultiFabs**. The results obtained clearly supports that LMC has well address the load-balancing issue, therefore there is not much space for power budgeting management within the application at function call level. The following addresses the characterization of LMC performance-power-quality tradeoffs and power management at a coarser grain level (i.e., workload level).

Function name	Number of calls	Minus	Percent %
Amr::coarseTimeStep	20	539.3	76.94
CollectData_Alltoall	22421	18.4	4.99
StateData::checkPoint	54	33.8	4.83
CGSolver::solve_bicgstab	5374	33.8	3.05
LinOp::prepareForLevel	8297520	12.8	1.75
mg_tower_bottom_solve	1869	11.8	0.85
MultiGrid::solve	4598	7.5	1.69
CGSolver::dotxy	77640	5.9	0.72
Amr::writePlotFile	3	3.3	0.48
Projection::doNodalProjection	216	2.5	0.48
FabArray::FillBoundary	424250	0.46	0.40

Table 4.2: Top 10 time consuming functions

4.3 AMR resolution adjustment

LMC simulation program is based on AMR algorithm, which uses a hierarchical grid structure, and fill finer patches on the region of interest. Therefore, the simulations resolution from the computational point of view is mainly directed from the AMR algorithm resolution configuration. There are many parameters controlling AMR solution, but in the work, only those four parameters listed in Table 4.3 have been used to tune the simulation resolution.

Parameter	Definition
amr.n_cell	Number of cells in each direction at the coareset level
amr.max_level	Number of levels of refinement above the coarsest level
amr.ref_ratio	Ratio of coarse to fine grid spacing between subsequent levels
amr.regrid_int	How often to regrid

Table 4.3: AMR parameters which are used to tune simulation resolution

Each parameter has different impact on the resolution, which is illustrated with the example use case shown in Figure 4.3. The parameters of the example case are discussed as follows.

Example case:

- amr.n_cell = 32 32 32

This would define the domain size (at coarsest level) to have 32 cells in the x-direction, 32 cells in the y-direction and 32 cells in the z-direction. (If it is in the 2D input file, the last number will be ignored). As shown in Figure 4.3, there are 32 cells in both x and y direction.

- amr.max_level = 2

This would set a maximum of 2 refinement levels in addition to the coarse level. Within the calculation, the number of refinement level must be $\leqslant$ amr.max_level, but it can be change in time and it is not necessary always be equal to amr.max_level. Because these additional levels will only be created if the tagging criteria are such that cells are flagged as needing refinement. Figure 4.3 shows the mesh grids with maximum of 2 refinement levels.

- amr.ref_ratio = 4 2

Refinement ration means how many individual cells will a cell be divided into. For example in the left-hand side Figure 4.3, Setting amr.ref_ratio = 4 2 means dividing cell into 4 cells from levels 0 and 1, and dividing cell into 2 from levels 1 and 2.

- amr.regrid_int = 2

This would tell the code to regrid every 2 steps. This means level l+1 grids will be created every 2 level l time steps.

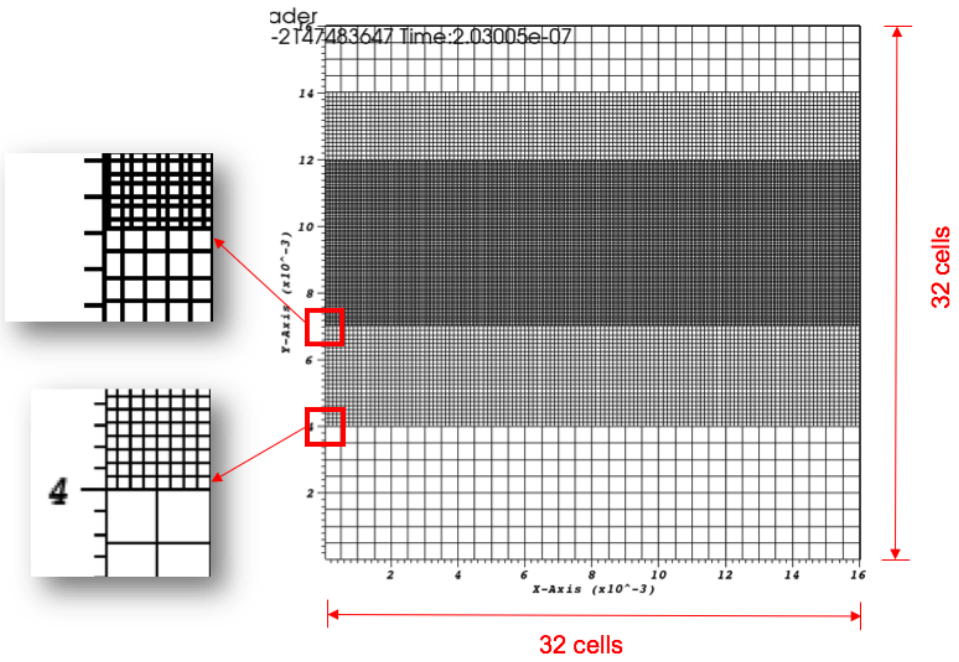


Figure 4.3: AMR mesh grids outcome of the example configuration

Issued found for pushing LMC to higher resolution level

By default, the maximum level of refinement for the LMC program was set to be level 3. When we tried to increase the parameter amr.max_level to 4 or higher, the program will thrown the following error:

BOXLIB ERROR: Multigrid Solve: failed to converge in max_iter iterations

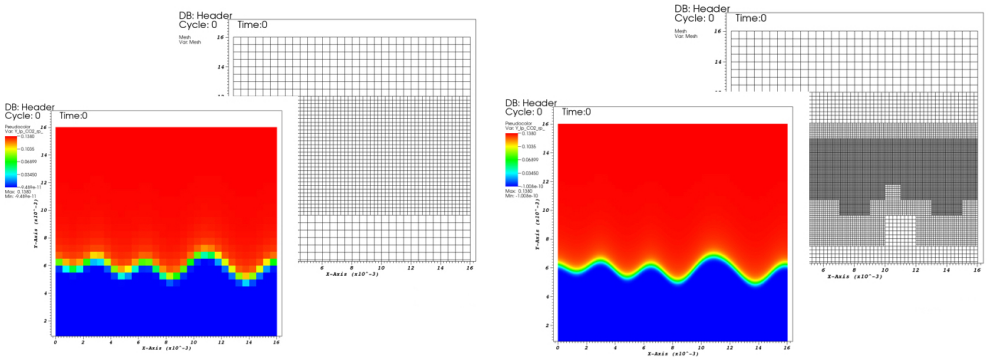
This error arises because when we are trying to solve a linear system with multigrid and the tolerance is below roundoff – that particular solve is the "sync project". Since the norm computed is a function of the number of cells and resolution, when we add refinement levels, eventually roundoff starts to dominate and we have to loosen the tolerances. To confirm this, we run LMC with option "mg.v=4" and we see the norm of the residual not dropping below the tolerance set in the inputs file.

Solution of increase LMC resolution level higher than 3 LMC input including two files, input.2d.* and probin.3d.test. input.2d.* contains all run-time information and probin.3d.test file includes all detailed settings for refinement control. We take refinement level 5 as an example. First, we need to set max_trac_lev = 5 in the probin.3d.test file. This is the maximum refinement level triggered by refinement on

the flame tracer. According to how it is set up, if the mass fraction of the species designated as the flame tracer is larger than flametracval (set in the probin file), the the cell is tagged for refinement. The tracer species is set in the inputs file to atomic hydrogen via the line: "ns.flameTracName = H". Second, we need to change proj.sync_tol to 1.0e-8. This is to increase the tolerance in the inputs and fix the issue weve discussed above. Finally, we can go to input.2d.* file and change amr.max_level to 5. By do these changes, we can successfully running the LMC to higher resolution level.

4.4 AMR mesh and LMC output

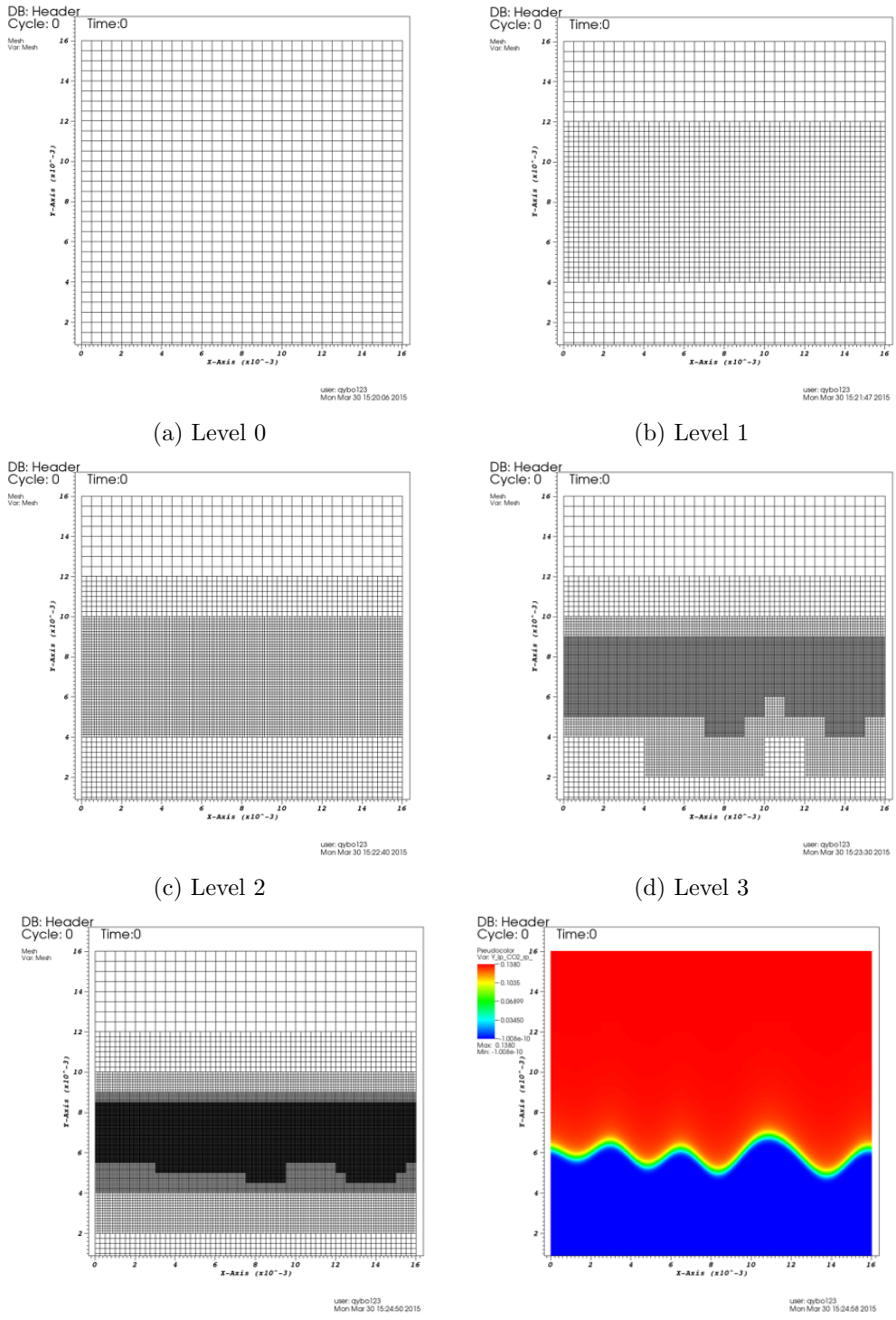
Figure 4.4 compares the LMC output with lower and higher level of refinement. And the output resolution is increase with the level of refinement goes higher. Figure 4.5 shows AMR mesh grids of different levels ranging from coarse to finer. In level 0, AMR generate regular grids that covers the entire computational domain. As the maximum refinement level increase, AMR will tag and refine the cells in the region which need higher resolution, so that it can increase the computational utilization by focusing computational resources in regions of interest but decrease computing resolution in regions with less interest. This also can be seen from Level 2 to Level 4, where finer grids are more concentrate on the center curve region.



(a) Lower resolution

(b) Higher resolution

Figure 4.4: Refinement level directly affects AMR resolution: mesh grid density is increasing while refinement level is increasing and relevant flame plots are getting finer (more accurate)



(a) Level 0 (b) Level 1

(c) Level 2 (d) Level 3

(e) Level 4 (f) Level 4 LMC output image

Figure 4.5: Different level of refinement and LMC output image

4.5 Evaluation of different levels of refinement for different number of cores

This experiment is first run on CAPER cluster, with a maximum of 128 cores, from level 1 to level 4 on 2, 4, 8, 16, 32, 64 and 128 cores. The execution time is recorded simply with the system tool "time".

Configuration

Parameter	**Value**
Platform	CAPER and DELL cluster
Number of cores	2 to 128 quadratic growth
Level of refinement	level 1 to level 4
Time step	10
Time unit	Second

Table 4.4: Configuration for experiment of execution time of different levels of refinement under variety number of cores

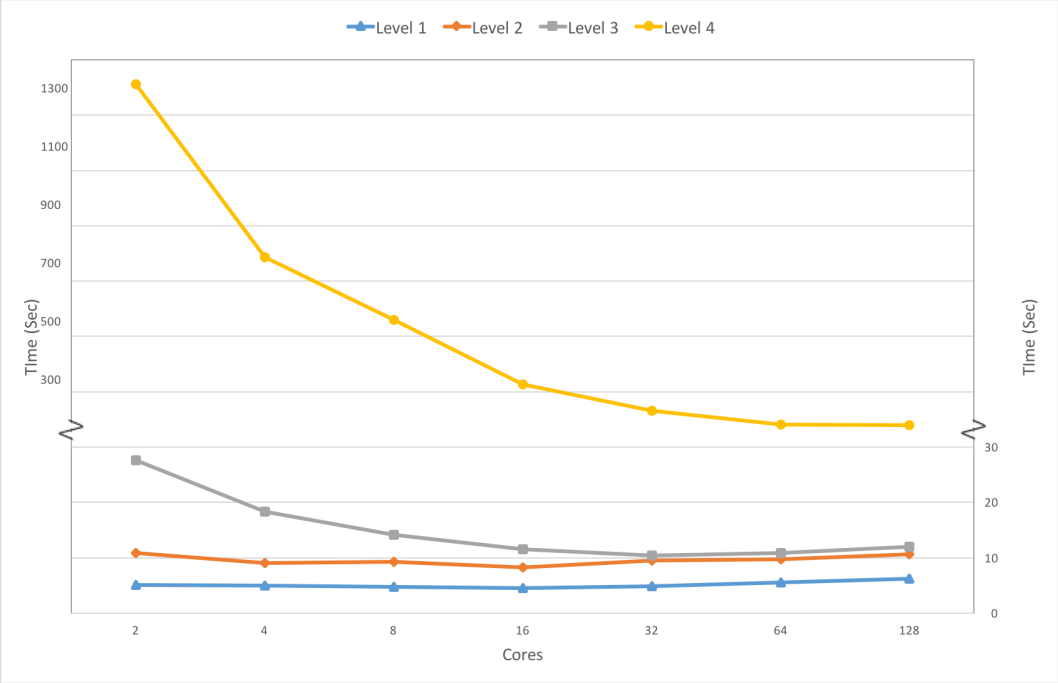


Figure 4.6: Execution time of running different levels LMC on CAPER cluster through 2 to 128 cores.

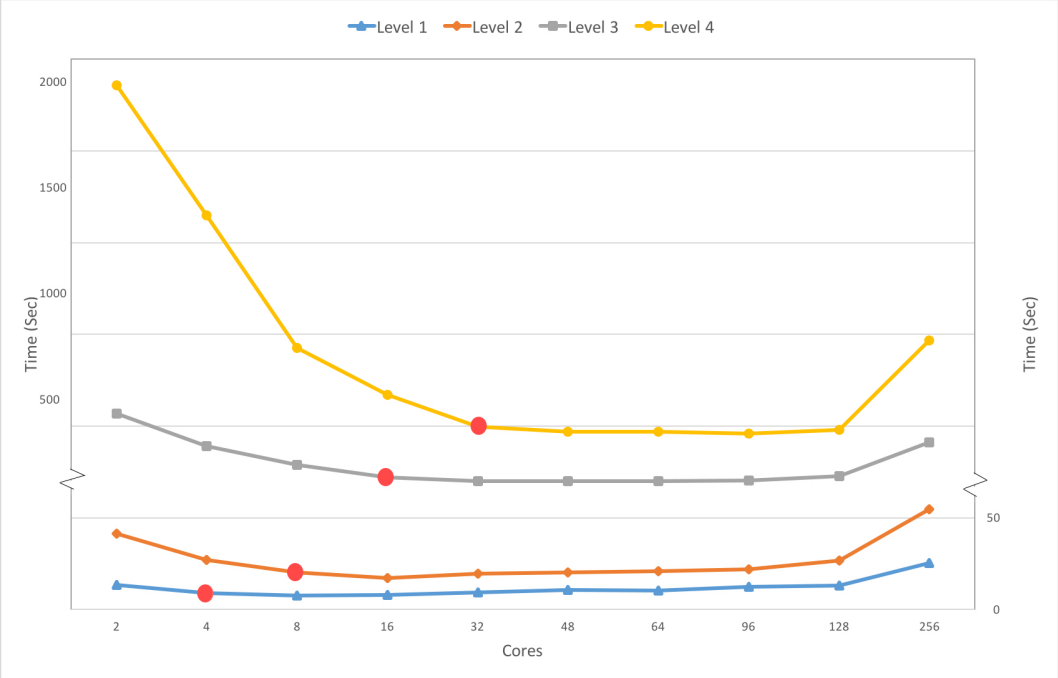


Figure 4.7: Execution time of running different levels LMC on DELL cluster through 2 to 256 cores.

The results of the first experiment using CAPER cluster are shown in Figure 4.6. As can be clearly seen from the level 4 line, which is yellow line with solid dot maker, the more computing resource we use, the faster the execution is. Comparing these four lines, we can also notice that the slope is decreasing from level 4 to level 1, and then the lines go to a relative flat zoom. From this notice, we can see that heavier workload need more computational resources, but provisioning more resources do not bring any performance increase. In Figure 4.6 it can be seen from level 1 to level 3 lines that their tails are tilting a little bit, but level 4 line doesnt have this trend. In order to show its trend more clearly, we ran this experiment on DELL cluster (256 cores cluster). The result is shown in Figure 4.7. All levels' curve is like a "U", that tilt in the head and tile. From level 1 to level 4, the head turning point, which marked by red dot, are respectively 4, 8, 16 and 32 cores. Then they tend to get into a flat zoom, while under the 256 core, the execution time are all increase. From these two experiments we can conclude that, to achieve optimal running time, we need to configure appropriate number of processors to execute certain level of quality for the LMC program and using

more processors does guarantee better performance.

4.6 Evaluation of energy consumption of different levels of refinement under variety number of cores

In this experiment, we are aim at exploring the energy consumption of running LMC with different level of refinement under multiple number of cores.

Energy is the product of power and time, therefore, in this experiment, we need to measure the execution time and power respectively. To measure time, we still use Linux command TIME in the script file. And for the CPU power, we are using RAPL power meter to measure it. RAPL power meter is a sub-function of Intel Power Gadget program, which has been mentioned in previous background section. We are sampling power data every 0.5 second.

Configuration

Parameter	**Value**
Platform	CAPER cluster
Number of cores	2 128 quadratic growth
Level of refinement	level 1 to level 4
Time step	10
Time unit	Second
Power measurement object	8 nodes processors
RAPL sample rate	2 Hz

Table 4.5: Configuration for experiment of power consumption of different levels of refinement under variety number of cores

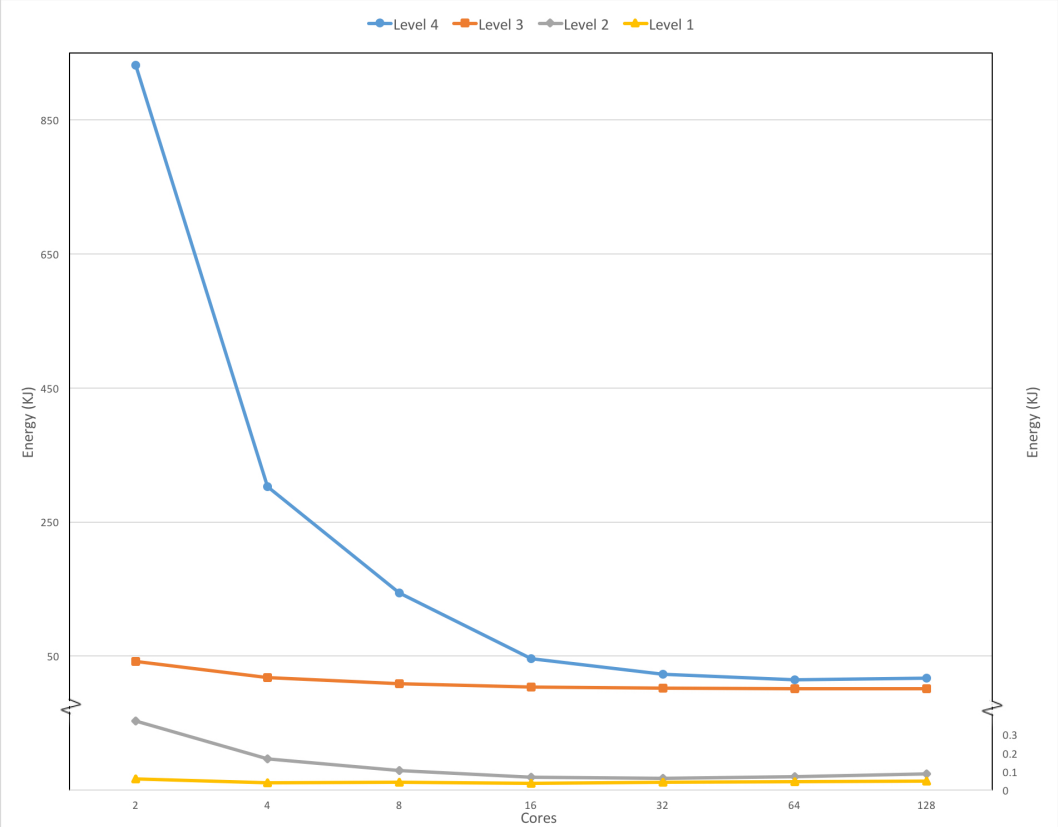


Figure 4.8: Power consumption of different levels of refinement under different number of cores

Figure 4.8 shows that there is a huge energy difference on the beginning of this curve. When running on 2 cores, the highest one (level 4) consumes almost 930 KJ while the lowest one (level 1) only take around 0.6 KJ. Since this experiment measure the all 8 node processors power status, the huge energy consumption of level 4 comes from the long execution time. It takes 1,314 seconds to run level 4 job while only 10 seconds to execute level 1. Most energy consumption comes from processors in idle state. But when assign appropriate processor resources to it, the curve reaches the flat zoom, the energy consumption difference is reasonable and stable. From this point of view, an appropriate number of cores for a certain workload is very crucial for energy consumption.

4.7 Evaluation of the impact of RAPL power capping on LMC performance

This experiment is aimed at studying the impact of power capping on LMC power-performance. And it would also give us a reference about the optimal power capping setting for different resolution levels.

We use level 1 to level 4 in LMC as inputs to measure the time and power consumption. Since CAPER CPU power is 95W (Intel Xeon E5-2650 V2), in this experiment, we cap the CPU power to 25W, 35W, 45W, 55W, 65W, 75W, 85W, and 95W and record the package energy consumption (PKG) and execution time respectively. Package energy consumption is the energy used by the CPU chip itself including cores, caches and graphics, etc.

Configuration

Parameter	**Value**
Platform	CAPER cluster
Number of cores	64 cores
Level of refinement	level 1 to level 4
Time step	10
Time unit	Second
Power measurement object	8 nodes processors
Power capping	95W to 25W decrease by 10
RAPL sample rate	2 Hz

Table 4.6: Configuration for experiment of exploring the affection of RAPL power capping on LMC performance

We use Intel *power_gadget* tool to cap the power. To use it, we assign value to the parameter *MY_POWER_LIMIT*, then compile and run it. We use linux command line "time" to measure the execution time. The execution command line is provided as follows:

time mpirun -machinefile hostfile.txt -np 64 .LMC2d.Linux.g++.gfortran.SDC.MPI.ex

inputfile amr.max_lev=#level

For the power measurement, we also use the Intel *power_gadget* tool. When running it, it will give the real-time power consumption data every 0.5 second.

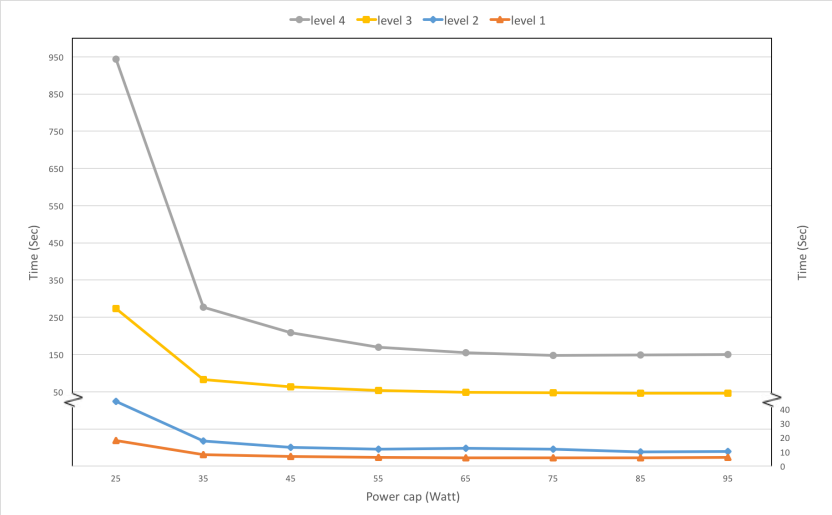


Figure 4.9: Relationship between different resolution of LMC and their execution time under different power capping levels

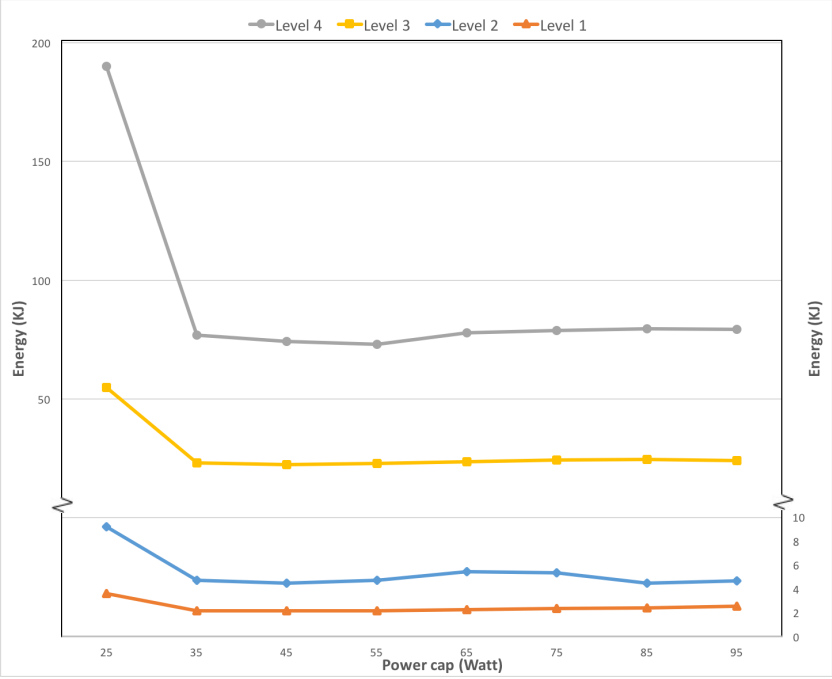


Figure 4.10: Relationship between different resolution of LMC and the energy consumption under different power capping levels

Figure presents the energy consumption result of executing level 1 to level 4 LMC

program on certain number of cores under different power capping configurations. Those curves have the same pattern that sharply decreasing from 25 to 35 and becoming flat between 65 and 95. The lowest energy consumption is appearing when CPU power was capped to 45W to 55W. This can be explaining from execution data that without any CPU power capping, LMC program will boots CPU to 65W to 75W during execution whatever executing on how many cores. This also can be seem from the flat curve on segment 65W to 95W on each chart. Therefore, the advantage of power capping is working when CPU power down below 65W, and optimize between 45W to 55W.

4.8 Evaluation of power budget acquisition through power capping and resolution degradation

The previous experiments have characterized the behavior of LMC. It can be concluded that the execution time will increase as the level of refinement/resolution increases or CPU power is capped down. Also, the energy consumption trend for different power caps and refinement levels is shown in Figure 4.11. The energy consumption presents the same trend as the execution time, i.e., LMC will consume more energy as the levels of refinement/resolution increase or capping down CPU power. The question here is how to use this characterization to create available power budget. We propose to combine these two factors, adjusting resolution and applying appropriate power capping, to get available power budget for running other tasks (e.g., checkpointing).

Configuration

Parameter	**Value**
Platform	CAPER cluster
Number of cores	64 cores
Resolution	level 3 2 1
Time step	10
Time unit	Second
Power measurement	RAPL meter
Power cap	RAPL

Table 4.7: Configuration for experiment of getting available power budget through power capping and resolution degradation

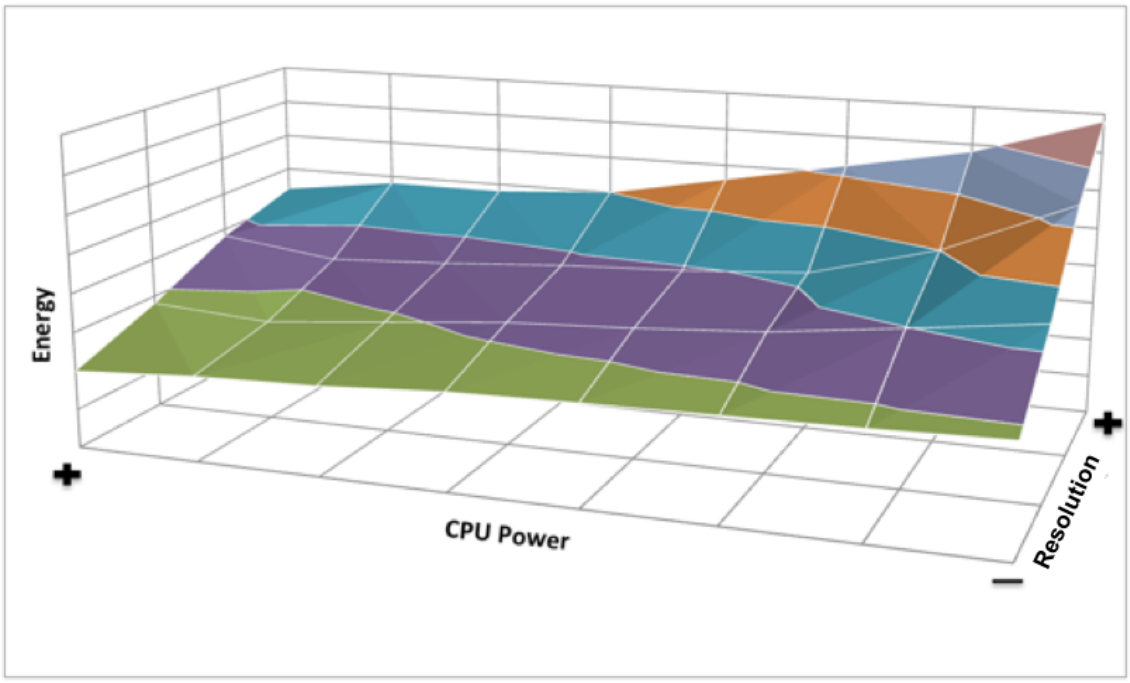


Figure 4.11: Energy consumption trend for different power caps and refinement levels

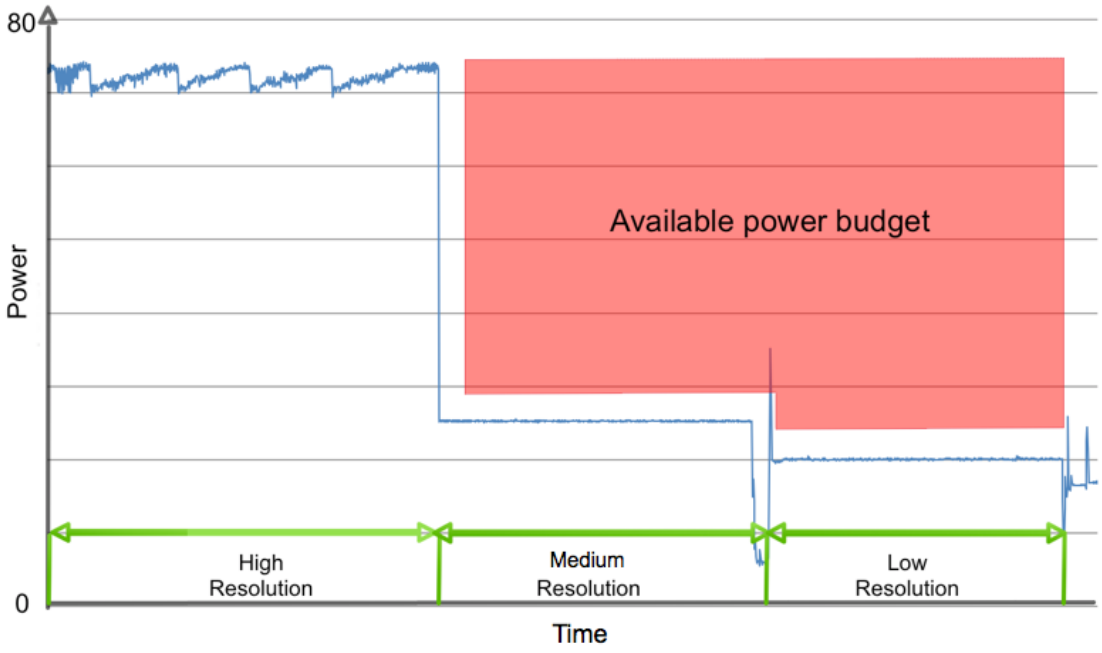


Figure 4.12: Available power budget from applying resolution degradation and appropriate power capping

In previous Figure 4.1, we measured the power consumption of running LMC with different resolution. It shows that the execution time decreases when decreasing the resolution. Therefore, we apply appropriate power capping to the the medium and low resolution execution to make their execution equal the execution time in highest resolution. As shown in Figure 4.12, the red region is the available power budget extracted from the total power budget.

4.9 Evaluation of power budget management

We have tested the impact of power capping and level of refinement of LMC on the performance and energy consumption. We also observed the potential power saving from degrading resolution and implementing power capping. The goal of this experiment is being able to use power budgets opportunistically for running other tasks. In the experiment, we propose to use this power budget to do checkpointing, which will increase the resilience and make the LMC program running more reliably.

In this experiment, we are proposing to degrade LMC resolution and cap its running

power, then use this power budget to do check pointing. However, there are some limitations due to the code characteristics: (i) we can not separate the checkpoint function from simulation program, and (ii) we can not dynamically control LMC resolution.

To address these two issues, we use two LMC executions running alternatively on two set of nodes to emulate one execution dynamically adjusting resolution and doing checkpoint. Each set of nodes contains three nodes for execution. When the first execution start doing checkpoint then the second set start another execution with lower resolution. We focus on system level power consumption.

We implement this using a flag to coordinate these two LMC executions. We've recompiled the first LMC set program to let it write flag when it start to do checkpoint, and the second LMC set keeps reading the flag to start running when the flag turn to be true. As shown in Figure 4.13, when the blue curve is the first instance (set 1) of LMC finishes its execution, it triggers the second instance (set 2), which is the orange curve. Once the second instance is completed the first one is triggered again and so on. By doing this, we can emulate a single LMC execution running and dynamically change the resolution while giving the available power budget for doing checkpoint.

Configuration

Parameter	**Value**
Platform	CAPER cluster
Number of cores	64 cores
Resolution	level 4 3
Time step	10
Time unit	Second
Power measurement object	3 nodes processors
Power cap	RAPL

Table 4.8: Configuration for experiment of exploring the power-performance tradeoffs

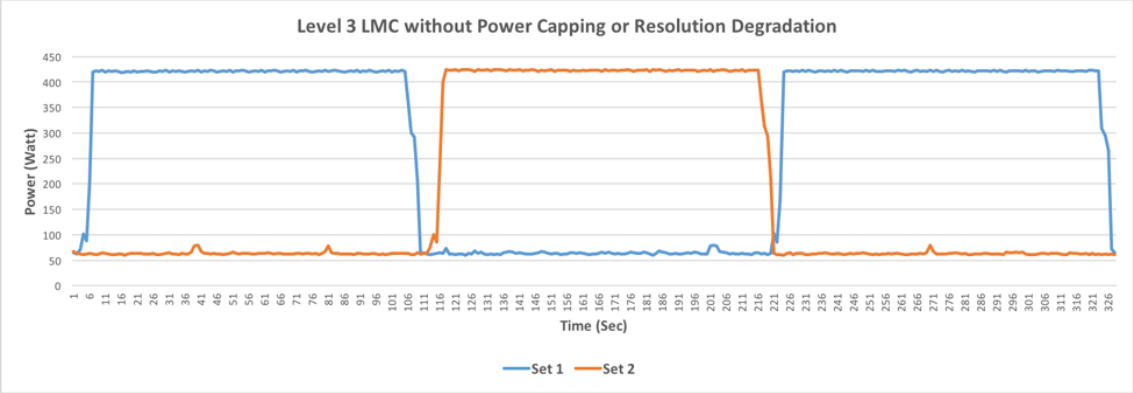


Figure 4.13: Level 3 LMC power consumption without power capping or resolution degradation

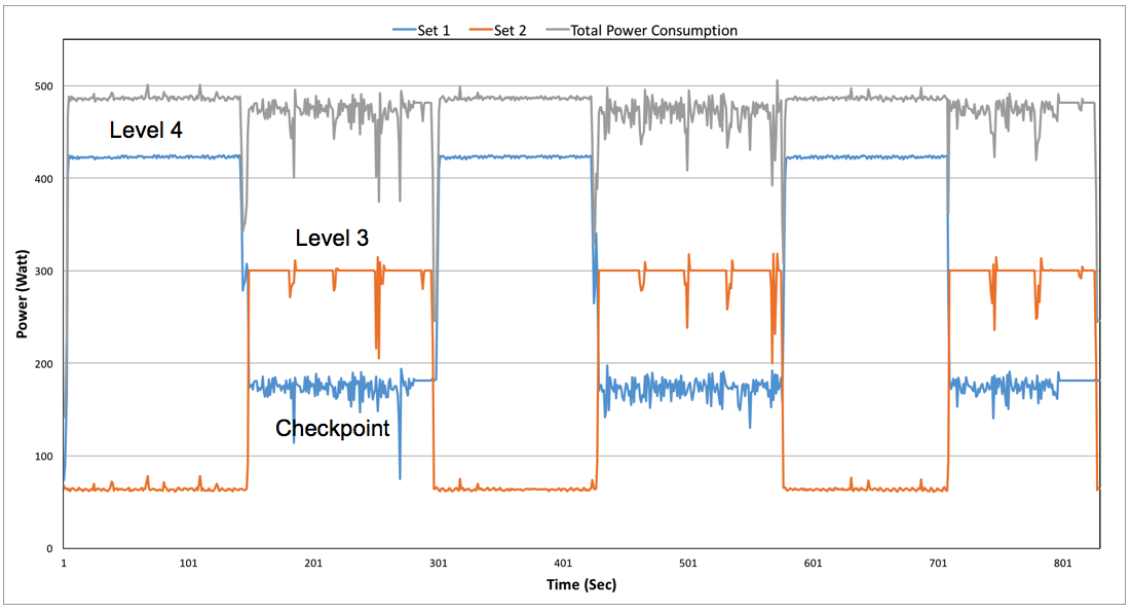


Figure 4.14: Level 4 LMC power consumption with power capping or resolution degradation

In Figure 4.14, the blue curve is the set 1 running the high resolution LMC (level 4). Once it starts doing checkpointing, the set 2 (in orange curve) is triggered to run LMC at level 3. Configuring power capping appropriately, the total power consumption, which is shown in gray curve on the top, is kept constant. This means we successfully constraint the power budget of LMC to perform other tasks (i.e., checkpointing).

Chapter 5

Conclusions and future work

In this work, we have focused on studying the properties and exploring the performance, quality and power/energy tradeoffs of Low-Mach-Number Combustion (LMC) application which is based Adaptive Mesh Refinement (AMR) algorithm. The key contributions of this work are (1) we present an empirical evaluation of different configurations of application that gives insights into the energy-performance-quality tradeoff for this work, (2) we provided a comprehensive study of this LMC simulation performance, power and energy behavior, and (3) we propose a power-performance-quality tradeoff for this application, which can be used to better schedule power budgets across HPC systems.

Our current work investigates how to leverage these insights to implement a runtime to manage power budgets dynamically. Moreover, instead of trading-off with resolution, we also plan to explore the possibility of managing resources dynamically (e.g., cores) to manage power budgets. Future work also includes the management and scheduling of power budgets at whole system scale for assigning power budgets to other applications, not necessarily to the LMC workflow.

References

[1] "Meet the fastest, most powerful science machine in the world: Titan supercomputer." [Online]. Available: http://www.computerworld.com/article/2473620/high-performance-computing/meet-the-fastest–most-powerful-science-machine-in-the-world–titan-super.html

[2] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra *et al.*, "Doe advanced scientific computing advisory subcommittee (ascac) report: Top ten exascale research challenges," USDOE Office of Science (SC)(United States), Tech. Rep., 2014.

[3] M. Tolentino and K. W. Cameron, "The optimist, the pessimist, and the global race to exascale in 20 megawatts," *Computer*, vol. 45, no. 1, pp. 0095–97, 2012.

[4] R. Ge, X. Feng, W.-c. 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.* IEEE, 2007, pp. 18–18.

[5] C.-h. Hsu and W.-c. Feng, "A power-aware run-time system for high-performance computing," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing.* IEEE Computer Society, 2005, p. 1.

[6] S. Huang and W. Feng, "Energy-efficient cluster computing via accurate workload characterization," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid.* IEEE Computer Society, 2009, pp. 68–75.

[7] 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.* ACM, 2005, pp. 164–173.

[8] D. Donofrio, L. Oliker, J. Shalf, M. F. Wehner, C. Rowen, J. Krueger, S. Kamil, and M. Mohiyuddin, "Energy-efficient computing for extreme-scale science." *IEEE Computer*, vol. 42, no. 11, pp. 62–71, 2009.

[9] M. Gamell, I. Rodero, M. Parashar, J. C. Bennett, H. Kolla, J. Chen, P.-T. Bremer, A. G. Landge, A. Gyulassy, P. McCormick *et al.*, "Exploring power behaviors and trade-offs of in-situ data analytics," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.* ACM, 2013, p. 77.

[10] 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.* ACM, 2009, pp. 460–469.

[11] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," in *SC 2006 conference, proceedings of the ACM/IEEE*. IEEE, 2006, pp. 14–14.

[12] 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*. IEEE, 2011, pp. 131–142.

[13] D. Li, B. R. De Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos, "Hybrid mpi/openmp power-aware computing," 2010.

[14] I. Rodero, S. Chandra, M. Parashar, R. Muralidhar, H. Seshadri, and S. Poole, "Investigating the potential of application-centric aggressive power management for hpc workloads," in *High Performance Computing (HiPC), 2010 International Conference on*. IEEE, 2010, pp. 1–10.

[15] "Exact — center for exascale simulation of combustion in turbulence." [Online]. Available: http://exactcodesign.org/

[16] 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*. IEEE, 2010, pp. 218–227.

[17] C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, and K. Cameron, "Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems," *International Journal of High Performance Computing Applications*, vol. 25, no. 3, pp. 342–350, 2011.

[18] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood, "Power routing: dynamic power provisioning in the data center," in *ACM Sigplan Notices*, vol. 45, no. 3. ACM, 2010, pp. 231–242.

[19] J. Hamilton, "Cost of power in large-scale data centers," *Blog entry dated*, vol. 11, p. 28, 2008.

[20] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & cap: adaptive dvfs and thread packing under power caps," in *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*. ACM, 2011, pp. 175–185.

[21] J. Sartori and R. Kumar, "Distributed peak power management for many-core architectures," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.* IEEE, 2009, pp. 1556–1559.

[22] J. M. Cebrián, J. L. Aragon, and S. Kaxiras, "Power token balancing: Adapting cmps to power constraints for parallel multithreaded workloads," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 431–442.

[23] A. Gandhi, M. Harchol-Balter, R. Das, J. O. Kephart, and C. Lefurgy, "Power capping via forced idleness," 2009.

[24] K. Meng, R. Joseph, R. P. Dick, and L. Shang, "Multi-optimization power management for chip multiprocessors," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 177–186.

[25] V. Kontorinis, A. Shayan, D. M. Tullsen, and R. Kumar, "Reducing peak power with a table-driven adaptive processor core," in *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture*. ACM, 2009, pp. 189–200.

[26] B. Rountree, D. H. Ahn, B. R. De Supinski, D. K. Lowenthal, and M. Schulz, "Beyond dvfs: A first look at performance under a hardware-enforced power bound," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 2012, pp. 947–953.

[27] H. Zhang and H. Hoffmann, "A quantitative evaluation of the rapl power control system," *Feedback Computing*, 2015.

[28] A. Porterfield, R. Fowler, S. Bhalachandra, B. Rountree, D. Deb, and R. Lewis, "Application runtime variability and power optimization for exascale computers," in *Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2015, p. 3.

[29] O. Sarood, A. Langer, L. Kalé, B. Rountree, and B. De Supinski, "Optimizing power allocation to cpu and memory subsystems in overprovisioned hpc systems," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–8.

[30] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using rapl," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 3, pp. 13–17, 2012.

[31] V. Adhinarayanan, W.-c. Feng, J. Woodring, D. Rogers, and J. Ahrens, "On the greenness of in-situ and post-processing visualization pipelines," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*. IEEE, 2015, pp. 880–887.

[32] A. Venkatesh, K. Kandalla, and D. K. Panda, "Evaluation of energy characteristics of mpi communication primitives with rapl," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013, pp. 938–945.

[33] B. Aupoix, D. Arnal, H. Bézard, B. Chaouat, F. Chedevergne, S. Deck, V. Gleize, P. Grenard, and E. Laroche, "Transition and turbulence modeling," *AerospaceLab*, no. 2, pp. p–1, 2011.

[34] N. Peters, "Turbulent combustion: The state of the art," in *Turbulent Combustion*. Cambridge University Press, 2000, pp. 1–65, cambridge Books Online. [Online]. Available: http://dx.doi.org/10.1017/CBO9780511612701.002

[35] "Laboratory-scale flames." [Online]. Available: https://ccse.lbl.gov/research/combustion/

[36] J. Bell, M. Day, A. Almgren, M. Lijewski, C. Rendleman, R. Cheng, and I. Shepherd, "Simulation of lean premixed turbulent combustion," in *Journal of Physics: Conference Series*, vol. 46, no. 1. IOP Publishing, 2006, p. 1.

[37] J. Bell, A. Almgren, V. Beckner, M. Day, M. Lijewski, A. Nonaka, and W. Zhang, "Boxlib users guide," Technical Report, CCSE, Lawrence Berkeley National Laboratory. Available at: https://ccse. lbl. gov/BoxLib/BoxLibUsersGuide. pdf, Tech. Rep., 2012.

[38] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of computational Physics*, vol. 53, no. 3, pp. 484–512, 1984.

[39] K. G. Powell, P. L. Roe, and J. Quirk, "Adaptive-mesh algorithms for computational fluid dynamics," in *Algorithmic trends in computational fluid dynamics*. Springer, 1993, pp. 303–337.

[40] J. Bell, M. Berger, J. Saltzman, and M. Welcome, "Three-dimensional adaptive mesh refinement for hyperbolic conservation laws," *SIAM Journal on Scientific Computing*, vol. 15, no. 1, pp. 127–138, 1994.

[41] X. Gao and C. Groth, "Parallel adaptive mesh refinement scheme for three-dimensional turbulent non-premixed combustion," in *46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, AIAA paper*, vol. 1017, 2008, p. 2008.

[42] "Navierstokes equations." [Online]. Available: https://en.wikipedia.org/wiki/navierstokesequations

[43] R. J. Kee, J. Warnatz, and J. Miller, "Fortran computer-code package for the evaluation of gas-phase viscosities, conductivities, and diffusion coefficients." *NTIS, SPRINGFIELD, VA(USA), 1983, 37*, 1983.

[44] J. Warnatz, "Influence of transport models and boundary conditions on flame structure," in *Numerical methods in laminar flame propagation*. Springer, 1982, pp. 87–111.

[45] R. Rehm and H. Baum, "The equations of motion for thermally driven, buoyant flows," *J. RES. NATL. BUR. STAN. Journal of Research of the National Bureau of Standards*, vol. 83, no. 3, p. 297, 1978.

[46] A. MAJDA and J. Sethian, "The derivation and numerical solution of the equations for zero mach number combustion," *Combustion science and technology*, vol. 42, no. 3-4, pp. 185–205, 1985.

[47] T. Alazard, "A minicourse on the low mach number limit," *Discrete and Continuous Dynamical Systems Series S*, vol. 1, no. 3, pp. 365–404, 2008.

[48] I. Intel, "and ia-32 architectures software developer's manual, 2011," *Intel order Number*, 64.

[49] V. Sundriyal and M. Sosonkina, "Runtime power-aware energy-saving scheme for parallel applications," 2015.

[50] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: memory power estimation and capping," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*. IEEE, 2010, pp. 189–194.

[51] "Measuring power on intel xeon phi product family devices." [Online]. Available: https://software.intel.com/en-us/articles/measuring-power-on-intel-xeon-phi-product-family-devices

[52] "Using the intel power gadget api on mac os x." [Online]. Available: https://software.intel.com/en-us/blogs/2012/12/13/using-the-intel-power-gadget-api-on-mac-os-x

[53] Z. Lan, V. E. Taylor, and G. Bryan, "A novel dynamic load balancing scheme for parallel systems," *Journal of Parallel and Distributed Computing*, vol. 62, no. 12, pp. 1763–1781, 2002.

[54] T. Wen, J. Su, P. Colella, K. Yelick, and N. Keen, "An adaptive mesh refinement benchmark for modern parallel programming languages," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. ACM, 2007, p. 40.

[55] A. C. Calder, B. Curts, L. Dursi, B. Fryxell, G. Henry, P. MacNece, K. Olson, P. Ricker, R. Rosner, F. X. Timmes *et al.*, "High-performance reactive fluid flow simulations using adaptive mesh refinement on thousands of processors," in *Supercomputing, ACM/IEEE 2000 Conference*. IEEE, 2000, pp. 56–56.

[56] Q. Meng, J. Luitjens, and M. Berzins, "A comparison of load balancing algorithms for amr in uintah," *Scientific Computing and Imaging Institute, University of Utah, Tech. Rep. UUSCI-2008-006*, 2008.

[57] W. Y. Crutchfield, "Load balancing irregular algorithms," Technical Report UCRL-JC-107679, Lawrence Livermore National Laboratory, Tech. Rep., 1991.

[58] C. A. Rendleman, V. E. Beckner, M. Lijewski, W. Crutchfield, and J. B. Bell, "Parallelization of structured, hierarchical adaptive mesh refinement algorithms," *Computing and Visualization in Science*, vol. 3, no. 3, pp. 147–157, 2000.