INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Scalability under a Power Bound using the GREMLIN Framework

Matthias Maiterth

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Scalability under a Power Bound using the GREMLIN Framework

Matthias Maiterth

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Martin Schulz Dr. Barry L. Rountree
Abgabetermin:	20. Februar 2015

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 20. Februar 2015

. . . .

(Unterschrift des Kandidaten)

Acknowledgment

Foremost, I want to thank Prof. Dr. Dieter Kranzlmüller, not only for introducing me to Martin and establishing the connection that made it possible for me to conduct my research work at the Lawrence Livermore National Laboratory, but also for the excellent support throughout my thesis work.

For the great support and collaboration, I have to thank my advisor Dr. Martin Schulz. His expertise and knowledge helped me advance in major steps with each interaction. The opportunity of working with him at the Lawrence Livermore National Laboratory made it possible to work in a team with research spirit unparalleled. I feel very privileged for having a mentor whose guidance has a lasting impact on me.

Without the insights of Dr. Barry Rountree and his excellent knowledge this work would not have been possible. Having him as an advisor did not only advance my research goals, but also sparked many ideas worth pursuing.

A big thank you to Tapasya Patki, Joachim Protze, Kathleen Shoga, Neha Gholkar, Timmy Meyer and Rogelio Long, the excellent group of students and visiting researchers I was allowed to be part of. The many discussions helped finding solutions and explore the problems at hand.

I'm grateful for the many valuable interactions with the researchers of the Center for Applied Scientific Computing at LLNL, in particular Tanzima Islam, Ignacio Laguna, Todd Gamblin and Greg Bronevetsky. The insights gained by interacting with scientists and visiting scientists from all disciplines at LLNL motivated and encouraged me to advance further in this field of research.

I would also like to thank my home institution, and the MNM-Team at the LMU. A special thank you to Dr. Karl Fürlinger for his great support throughout the whole master's program.

My dearest thank you to my parents, my friends and the people I love for their support.

Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-TH-667535).

Abstract

With the move towards exascale, system and software developers will have to deal with issues of extreme parallelism. The system properties affected most by the increase in node and core count are the shared resources on node and across the system. The increase in parallelism leads to reduced memory and bandwidth when regarding individual cores. Since power is a limiting factor for supercomputers, and power is not fully utilized in current systems, overprovisioning compute resources is a viable approach to maximized power utilization. To maximize system performance in regard to these changing conditions, it is necessary to understand how resource restrictions impact performance and system behavior. For the purpose of understanding anticipated system properties the GREMLIN framework was developed. The framework gives the opportunity to add power restrictions, hinder memory properties and introduce faults to study resilience, among others. These features give the opportunity to use current petascale technology to study problems system designers and software developers will have to face when moving towards exascale and beyond. This work describes the initial release of the GREMLIN framework, developed for this work, and shows how it can be used to study the scaling behavior of proxy applications. These proxy applications represent a selection of HPC workloads important to the scientific community. The proxy applications studied are AMG2013, an algebraic multi-grid linear system solver, CoMD, a classical molecular dynamics proxy application and NEKBONE, an application that uses a high order spectral element method to solve the Navier-Stokes equations. The main interest of these studies lies in analysis regarding their power behavior at scale under a power bound. The objective of this work is to use the GREMLIN framework to study strong and weak scaling using different power bounds on up to 256 nodes. These findings show how the GREMLIN framework can help systems and software designers to attain better application performance and can also be used as basis for CPU power balancing tools to use power more efficiently.

Contents

List of Figures

List of Tables

1 Exascale: Challenges for the Next Generation of HPC

With the initial goal of reaching exascale, i.e., building a HPC (High Performance Computer) system with performance of one exaflop or 10^{18} $FLOPS$ (Floating Point Operations Per Second), by 2018 the DOE (Department of Energy) set the maximum allowable power consumption of the envisioned HPC system to 20 MW [DOE 09]. When looking at todays Top500 list and the Green500 list and performing a projection towards exascale the power values are far from this goal. Figure 1.1 shows the current systems performance plotted against the systems efficiency. Looking at the currently best performing supercomputer, Tianhe-2 [T500 14], both performance and efficiency has to improve by 2 orders of magnitude to reach exascale under 20 MW. The most efficient system, the L-CSC at the GSI Helmholtz Center, achieves 5,271.81 MFLOPS/watt [G500 14]. With this efficiency a system would require 189.688 MW to reach exascale. Thus the energy efficiency of our currently most energy efficient systems still has to improve by more than a factor of 9 to stay under the 20 MW line. Japan's ambitious goal of building their post-K system at the RIKEN AICS with a power consumption of up to 30 MW will still have to solve the same efficiency problem [TSI 14]. Even with increased efficiency energy costs will stay a major concern when acquiring a HPC system.

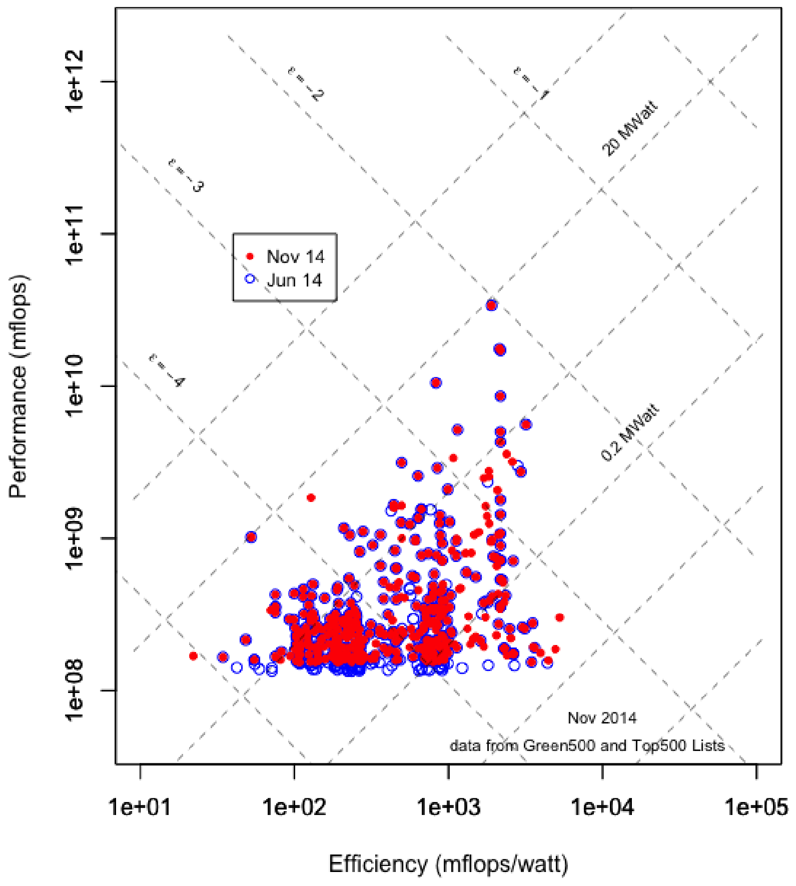


Figure 1.1: Exascale projection from the November 2014 Top500 and Green500 lists. [Saun 14]

Table 1.1: Potential Exascale Computer Design for 2018 and its relationship to current HPC designs. [DOE 10]

	2010	2018	Factor Change
System Peak	2 PFLOPS	1 EFLOPS	500
Power	6 MW	20 MW	3
System Memory	0.3 PB	10 PB	33
Node Performance	0.125 GFLOPS	10 TFLOPS	80
Node Memory BW	25 GB/s	400 GB/s	16
Node Concurrency	12 cpus	$1,000$ cpus	83
Interconnect BW	1.5 GB/s	50 GB/s	33
System Size (nodes)	20 K nodes	1 M nodes	50
Total Concurrency	225 K	1 B	4,444
Storage	15 PB	300 PB	20
Input/Output bandwidth	0.2 TB/s	20 TB/s	100

While improving compute capabilities, researchers and system designers have hit several boundaries. Some of the widely known challenges in serial performance are the ILP (instruction-level parallelism) wall [Corp 99], the power wall [SCL 11] and the memory wall [WuMc 95]. With reaching these scaling barriers for single processors, HPC systems rely on high parallelism. There is a clear trend towards an increase in number of CPU cores per node and regarding the overall system. When looking at the 100 fastest supercomputers of the current Top500 list, there is no system present with less than 10,000 cores. The fastest system in the current list has a core count of 3.12 million cores [T500 14]. With the increase in parallelism issues that could be neglected so far will get more and more significance in the future.

A study form the DOE in 2010 projected how the specifications of an exascale system are likely to look like in 2018. This prediction is shown in Table 1.1. Even though the goal of reaching exascale by 2018 seems unlikely these trends and the associated challenges remain [DOE 14]. Overall node performance will increase by a factor of 80. An issue here is the discrepancy of advances in the different system properties. For example with the slower increase in node memory compared to node concurrency, more nodes will have to share the same memory hierarchy on node. This means either a deeper memory hierarchy is needed or techniques have to be developed to handle less memory per core. With this prediction more and more cores will have to share the same memory hierarchy, thus the per core memory will decrease. This also means memory locality will gain more significance since false sharing and cache misses are likely to increase.

The bandwidth is estimated to increase by a factor of 33, but with more nodes and cores in the system the interconnect has to handle more messages. Changes in how messages are handled by the applications and the programming model used to avoid flooding the system will be an important topic for future systems.

This is the case for scaling to high node counts already, but this is also dependent on the communication pattern of the application running. Application specific topology reconfiguration on chip for minimizing communication cost is already in use [SSC 14]. For HPC systems dynamic interconnects and changeable topologies are not cost efficient yet. To efficiently use an exascale machine handling messages and traffic efficiently will be paramount to achieve good performance on large scale application runs with diverse communication patterns.

The increase in overall concurrency will also increase the chance of system failure. A standard component such as double-bit ECC memory has a mean time of failure of 1,500,000 hours. In a system like the Cray XT5, a 1 petaflop (10^{15} FLOPS) system at $ORNL$ (Oak Ridge National Laboratory) with 75,000 memory modules, this results in a mean time of failure of 20 hours for a single ECC double-bit error for the whole machine [EOS 09]. For the increased probability of component failure introduced by increases in overall concurrency, sophisticated recovery mechanisms are required, but also ways to predict recovery behavior.

The smallest increase seen in Table 1.1 is system power. The number listed in the table is of course the set goal of reaching exascale under 20 MW. Building a more efficient supercomputer involves both using more energy efficient components and increasing their utilization. The components and resources of the system that are the bottleneck for achieving better performance should definitely be at maximum utilization. If we look at the future performance bottleneck, which is power, this is not the case.

For SuperMUC, a 3 petaflop system at the LRZ in Garching, near Munich, a maximum power consumption

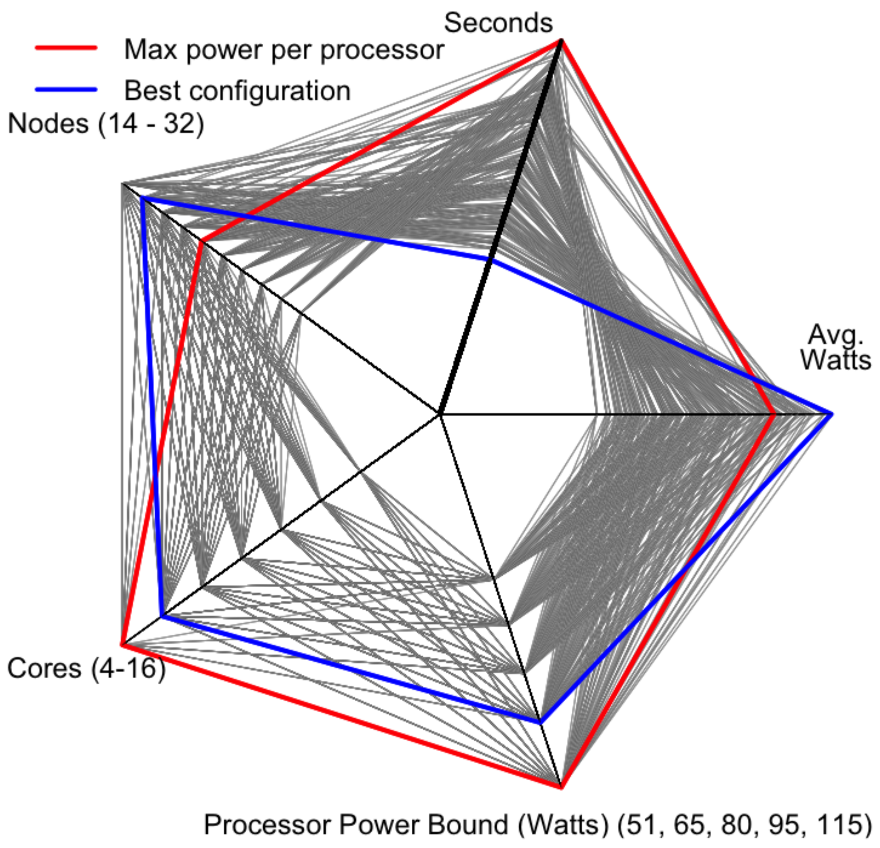


Figure 1.2: Configuration tests of NAS Parallel Benchmark sp-mz. Global power bound of 4500W for each configuration. [Roun 14]

of 3.6 MW was observed during benchmark runs of HPL (High Performance Linpack). This power consumption is rarely achieved with most other applications. During normal operations a system power variation of up to 2.7 MW was observed [SWAB 14].

The power consumption of the system strongly depends on the constraints and characteristics of the application executing. With a fixed maximum power supplied to the system not all of this power can be used, with the percentage usable depending on the application and system properties.

A solution to increasing utilization of underutilized resources is overprovisioning [FeFr 04][PLR$^+$ 13]. By increasing the number of nodes (and thus cores) in the machine the power that was not usable by the machine before due to application characteristics can be used by the added cores. Overprovisioning CPU cores requires mechanisms to prevent the components to draw more power than available in the system as a whole. Limiting the power drawn by the application can be assured by CPU frequency scaling techniques or other tools directly limiting power. The techniques used can be either *DVFS* (Dynamic Voltage and Frequency Scaling) or the more viable option used in this work: Intel's *RAPL* (Running Average Power Limit)[Intel 09]. RAPL is a hardware enforced power limit, which guarantees that a specific power limit is not exceeded within a time frame of a few milliseconds.

There have been studies that show significant performance gains using different system configurations with power bounds applied. Finding an optimal configuration is achieved by searching and evaluating different configurations by varying node count, core count and processor power bound. These configurations are application and system specific and no naive configuration that guarantees improved performance exists. Figure 1.2 shows one of the results obtained by Rountree et al. [Roun 14]. With lower core count and decreased power per processor, the average watts used could be increased and the execution time reduced by over half. The best configuration is outlined in blue, the worst in red [Roun 14]. This can be seen as an overprovisioned system since the whole setup with the limit of 4500W could not run all 32 nodes using 16 cores with maximum power, without violating the global power constraint [PLR+ 13]. To the best of my knowledge, there is no HPC system that uses the overprovisioning approach in a production system to exploit the overprovisioning potential. Therefore these tests are generally conducted on regular hardware by enforcing power limitations artificially.

The goal of this work is to provide the necessary software mechanisms to control such limitations and thereby enable the study of next generation supercomputers by emulating its resource restrictions, with a focus on power. This work contributes to the GREMLN framework, which is introduced to study and predict systems and performance behavior of existing application on future systems. Using the framework gives system and software developers the possibility to add resource restrictions and thus study power, memory, resilience and noise behavior in HPC systems. This capability provides the opportunity to study their respective impact on application performance. The framework offers easy extensibility for other resources the user is interested in. The GREMLIN framework is usable with any C/C++ or Fortran application using MPI (Message Passing Interface [MPI 14]) and does not require changes in the source code of the user application. The GREMLIN framework is used to study the scaling behavior under a power bound.

The rest of this work is structured as follows: Chapter 2 introduces the GREMLIN framework. For the chapters following the introduction to the GREMLINs the focus lies on analysis on power. The proxy applications used in this work are introduced in Chapter 3. The experimental setup is described in Chapter 4, followed by the scaling studies of the proxy applications in Chapter 5. In Chapter 6 a disussion is given on how these results can be used for more power efficient supercomputing. Chapter 7 gives an overview on related work.

2 The GREMLIN Framework

The GREMLIN framework originated in the need to know how current production codes, optimized for petascale supercomputers, will behave on exascale architectures. The project is part of the *Exascale Co-Design Center for Materials in Extreme Environments* (ExMatEx) [ExMatEx 14a] and was developed at *Lawrence Livermore National Laboratory* (LLNL). The co-design center links the design and development of hardware and software. This tight collaboration between developers in hardware, system software, middleware and scientific software allows scientists working on the project to solve today's hardest challenges in multiphysics simulations of materials in extreme mechanical and radiation environments [ExMatEx 14a]. The primary goal is to understand, develop and efficiently use exascale HPC systems.

With performance modeling and architectural simulation it is possible to predict the anticipated properties of future generation HPC systems. These models and simulations build a sophisticated theoretical approach for prediction. The shortcoming of using only these two techniques is also their big advantage: High level of detail. The high level of detail that is required to model a system precisely introduces high complexity. This makes it hard to use in practice. A more practical approach to study anticipated HPC configurations is emulation.

The GREMLIN framework is a tool to emulate the architecture of next generation HPC systems. It focuses on emulating resource restrictions which are introduced by high parallelism and concurrency. With this approach low level details can be studied on real hardware at scale, running on current HPC installations. Using the GREMLIN framework domain scientists can study application behavior with the emulated hardware specifications. This allows them to gain insight on what important issues will occur in an exascale environment. These insights can then be fed back to the system developers for optimization and changes can be introduced to cope with the negative effects observed. The design of hardware, system software and middleware with the goal of achieving the best ecosystem for scientific applications is one of the main ideas of the co-design initiative for exascale.

The combination of architecture emulation, analytic modeling and architectural simulation provide a holistic performance model, as depicted in Figure 2.1. Together these can be used to better understand future generations of HPC systems. Several studies to evaluate the feasibility of such an emulation approach were

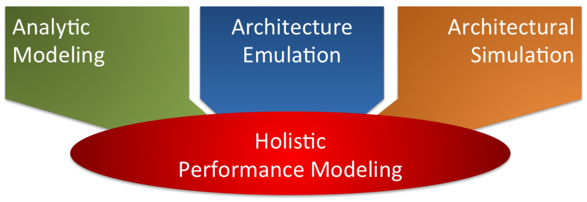


Figure 2.1: Approach to a holistic performance model [GREM 14b]

conducted as part of ExMatEx. The GREMLIN framework combines these proof of concept implementations originating from the studies performed in a single framework. In the following section the GREMLIN framework and its building blocks are introduced.

2.1 GREMLINs Overview

The GREMLIN framework emulates the behavior of anticipated future architectures on current HPC systems. An open source version of the project is available on the GitHub site of the LLNL scalability team [GREM 14a]. The framework focuses on two techniques: First, resource limitations by artificial restriction to emulate either high load or low availability on a per processor basis in a future system. Second, interference by noise and faults. The impact of the system noise and soft and hard faults are emulated by injection techniques. With

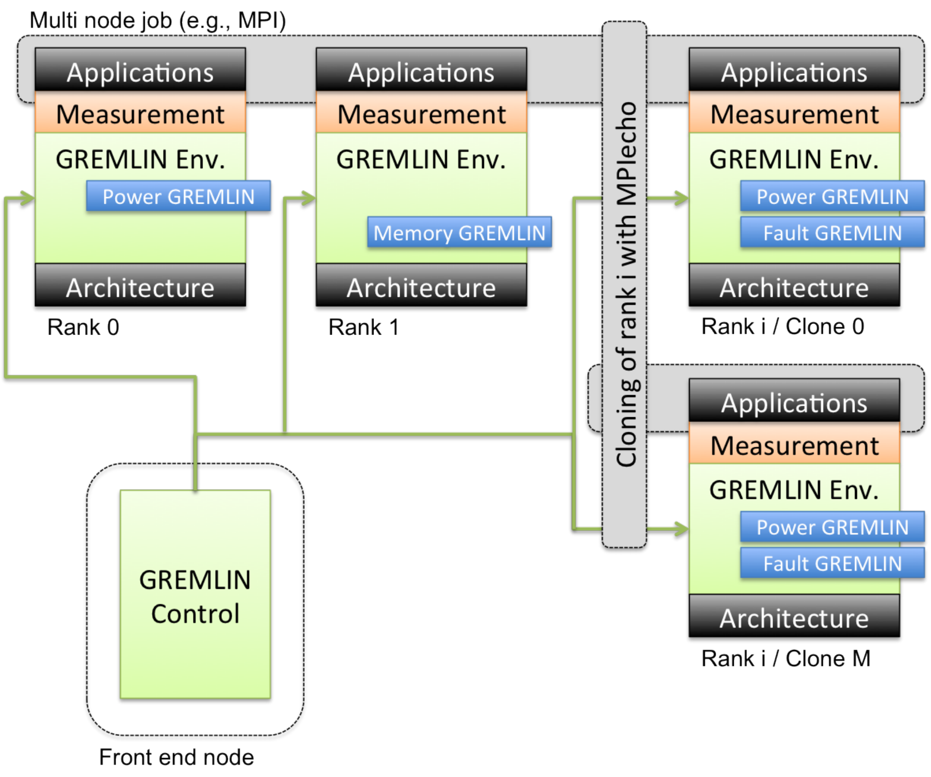


Figure 2.2: The architecture of the GREMLIN framework.[GREM 14b]

these two techniques most challenges anticipated for future HPC systems can be sufficiently emulated to be able to make predictions about their impact on application performance and behavior [SBB+ 13].

The GREMLIN framework consists of four major classes of GREMLINs:

- Power: artificially reduce the operating power of the CPU
- Memory: limit resources in the memory hierarchy, by running interference threads or processes
- Resilience: inject faults into target application
- Noise: inject noise events to emulate system and operating system interference

The framework is built in a modular fashion and is easily extensible. The workflow of the GREMLIN framework is intended as follows: The user is interested in the behavior of a specific application under exascale conditions. The application is setup on a HPC resource with the GREMLIN framework installed. The user decides which system property should be modified and selects the appropriate GREMLINs. The parameters for the selected GREMLINs are set for the anticipated hardware settings or restrictions. The GREMLINs are loaded at program startup, during the initialization phase. The program is then run without additional modifications, or the need to recompile for individual GREMLINs. The user can perform measurements using his performance measurement tool of choice to gain insights on how his code would behave under the emulated conditions. The different GREMLINs can be used individually or by combining multiple GREMLINs to emulate the exact configuration the user wants to observe.

Figure 2.2 shows the high level architecture concept of the GREMLIN framework. The GREMLINs are controlled via environment variables which directly impact how the system resource under alteration behave. The feature of cloning of ranks as seen in Figure 2.2 is not supported yet. Since the target applications are large scale HPC applications the framework relies on MPI, the Message Passing Interface standard [MPI 14], which is the de facto standard for parallel applications running on distributed memory systems.

The GREMLIN framework is provided as a set of libraries intercepting the regular MPI function calls. This is possible thanks to the design of the MPI standard profiling interface PMPI. A single GREMLIN is preloaded using `LD_PRELOAD`. Depending on the function of the GREMLIN different standard MPI calls are intercepted. In general the compulsory functions `MPI_Init()` and `MPI_Finalize()` are intercepted to initialize the GREMLIN functionality. During the program execution the GREMLIN can again intercept MPI function calls or listen for events or signals. Using signals and timers, random or periodic events such as faults and system noise can be emulated. At the time of writing no noise GREMLIN is available, yet. The individual GREMLINs try to be as general as possible and do not use any application specific features. If needed the functionality provided can be used to build an application specific GREMLIN. As mentioned earlier it is not needed to recompile the user application for usage with different GREMLINs. However, initially dynamic linked libraries have to be enabled. This is done by linking the dynamic linker/loader library `ld.so` (see: man 8 ld.so).

P^{n}MPI [ScdS 07] is used to combine multiple GREMLINs. P^{n}MPI is a low-overhead PMPI library, that allows users to chain several PMPI tools together. For this work P^{n}MPI was extended to handle signals and timers, which are redirected only to the tools that either registered or need to know about these events. The extended version of P^{n}MPI can also be found on the GitHub web page of the LLNL scalability team [PnMPI 09].

The GREMLINs use another powerful tool shipped with the P^{n}MPI tool: the wrapper generator `wrap.py` developed for the libra tool [libra 08]. `wrap.py` transforms code written in a domain-specific language as a wrapper file into C/C++ or Fortran, creating source code for PMPI libraries. There are two main advantages of using a wrapper generator for writing PMPI libraries: First, there is no need to write both Fortran and C/C++ while being compatible with both. This is important since there are a lot of production codes using MPI in both languages. Second, in a GREMLIN several of the functions are likely to have the same structure. `wrap.py` allows to write the code structure in a abstracted form. With this abstraction there is no need to rewrite boilerplate code, which helps in developing clean code for the GREMLINs.

For building and installing, the GREMLIN framework relies on CMake [CMake14] to ensure portability across systems. CMake is intended as an out of source build tool for the GREMLIN framework to ensure clean source, build and install directories. For running a single GREMLIN it is sufficient to `LD_PRELOAD` the desired GREMLIN and running the target MPI application. The instructions for building, installing and running the GREMLINs are provided in the `README` file. To use multiple GREMLINs with P^{n}MPI, the desired GREMLINs have to be patched using `pnmpi-patch`. After this the patched GREMLINs libraries can be loaded as modules in the `.pnmpi-conf` file.

In the following sections, the different kinds of GREMLINs and their usage are introduced.

2.2 Power GREMLINs

The intention of the Power GREMLINs is to emulate the power limitations and other power related issues on future systems. For exascale systems it is important for the developer to understand the power characteristics of their applications. This is not only the case for low power scenarios but also since applications have different phase behavior that draw different amounts of power.

Ideally this information can be used on the system side to maximize efficiency and the application developer can be agnostic about this information. Using the Power GREMLINs both application and system developers can better understand application behavior in power restricted environments. The focus of the Power GREMLINs is CPU power consumption.

In HPC there is always a tradeoff between power and performance. Some of today's HPC systems have different energy or frequency modes for the user to choose from. These are mostly static and job bound. For example IBM Loadleveler on SuperMUC offers the option run with different CPU frequencies with the primary objective to save energy [LRZ 13]. The goal for exascale and beyond is not primarily to save energy, but to use all power that is available in the system with a maximum efficiency. The primary goal is to maximize performance. With different application characteristics, phase behavior and their respective power characteristics, controlling CPU frequency and CPU power are of paramount importance for maximizing CPU performance and power utilization.

The best known technique to control clock frequency and thus power consumption is DVFS. Directly controlling CPU clock frequency to balance power consumption while achieving maximum performance is hard from a tools and application perspective. It is not always clear how clock frequency impacts performance and thus the power behavior of different code segments. It can be claimed that DVFS is only loosely correlated with power consumption and thus has only limited applicability compared to directly setting power limits.

With the introduction of Intel's Sandy Bridge processor family a different abstraction for controlling these CPU parameters was introduced: *RAPL* (Running Average Power Limit) [Intel 09]. RAPL provides on board power meters and power clamping. This allows the user to set limits to power consumption over a specific time period on the order of milliseconds. To satisfy this power bound the CPU will be throttled to guarantee that the average power limit is not exceeded. The specifics of RAPL and their utility are discussed by Rountree et al. [RAdS+ 12]. For an introduction to RAPL refer to Howard et al. [DGH+ 10].

The RAPL functionality is exposed via *MSRs* (model-specific registers). These registers are accessible via the privileged instructions `readmsr` and `writemsr`. For the use in the Power GREMLINs these are not used directly since the GREMLINs are a tool intended for the user. The RAPL functionality is exposed via a kernel module that allows safe access to MSRs on Intel processors, `msr-safe` [ShRo 14b]. `libmsr` [ShRo 14a] implements a more user friendly interface to the kernel module `msr-safe`. To use the power GREMLINs, the kernel module `msr-safe` and the libraries of `libmsr` have to be installed and available on the system.

To use `libmsr`, the libraries `msr_core.h` and `msr_rapl.h` have to be included. Access to power settings via the MSRs is controllable on a per package granularity. To avoid confusion the term package will be used over the term CPU socket throughout this work. `libmsr` needs to be initialized using `init_msr()` and to finalize usage `finalize_msr;` is used.

The Power GREMLINs consist of four different GREMLINs, each with a specific purpose:

- `powerBound.w`
- `globalPowerBound.w`
- `powerMeter.w`
- `terseInfo.w`

`powerBound.w` gives the ability to set a package specific limitation to the power consumed by each processor contained in the package, called a power cap or power bound. The environment variable `POWER_CAP` is used to set the power cap for all packages. This guarantees that on average no package will consume more than this value in watts. This average is guaranteed over a time period of 0.001 seconds. To set a different power cap for a specific package `POWER_CAP%d` is used, where `%d` is the n^{th} package of the system. The order of the packages are counted in order of MPI rank.

`globalPowerBound.w` is used to set a global power cap that limits the combined power consumption of all packages in a single MPI program. The environment variable `POWER_CAP_GLOBAL` is set for this global limit. The GREMLIN is implemented in a naive way. This means that every package shares the same per package power cap, which is calculated by dividing the global power cap by the number of packages involved in the program execution. This provides an easy way to emulate overprovisioned systems. The emulated overprovisioned system thus has more CPUs than it can run at full speed obeying the global power cap.

`powerMeter.w` provides an easy way to dump periodic power values. The time interval in between the measurements is specified via the environment variables `INTERVAL_S` and `INVERVAL_US`, in seconds and microseconds respectively. The output is generated per node and the output directory is specified by the environment variable `NODE_OUTPUT_PATH`. If `NODE_OUTPUT_PATH` is not set `stdout` is used to print the output as default. `powerMeter.w` dumps information about the time of measurement in seconds and package power in watts. DRAM power is included, but the values are zero by default since `libmsr` does not provide this information at the time of writing. The power measurement starts with an initial measurement before the user applications returns from `MPI_Init()` and ends with a final measurement for the user application calls `MPI_Finalize()`. Since the information is dumped on a per node basis, the environment variable `PROCS_PER_NODE` needs to be set.

`terseInfo.w` behaves similar to `powerMeter.w` but provides additional information in a terse format which can be read from the MSRs. The functions used for dumping the information are all `dump_X_terse()` functions provided by `libmsr`. These include power information, provided by `dump_rapl_terse()`, but also information about thread frequency via `dump_clocks_terse()` and core temperature provided by `dump_thermal_terse()`.

All Power GREMLINs need the environment variable PROCS_PER_PACKAGE set according to the system specification, to calculate the package number correctly. For compatibility reasons only one GREMLIN to measure and one GREMLIN to control power should be used simultaneously.

2.3 Resilience GREMLINs

The Resilience GREMLINs stress applications by introducing faults and errors. MPI itself is not fault tolerant and does not provide recovery techniques. With current systems the fault rates are too infrequent to successfully test fault tolerance models and recovery strategies on real hardware. The Resilience GREMLINs provide this opportunity by emulating faults. The user can specify fault rates of an emulated fault. The faults try to emulate the characteristics of real world faults. If the user expects a different kind of fault, the GREMLIN can be extended to the expected fault behavior.

The fault injector itself was written by Ignacio Laguna [SBB$^+$ 13]. To inject faults in the user application the signals SIGUSR1 and SIGUSR2 are used. This indicates that a fault occurred and does not really crash the application. The rate at which faults are injected is given in errors/hour. This can be controlled using the environment variable FAULT_RATE. For this GREMLIN, the application has to have a resilience model built in that responds to this kind of fault. In other words the signals SIGUSR1 and SIGUSR2 have to be identified as faults.

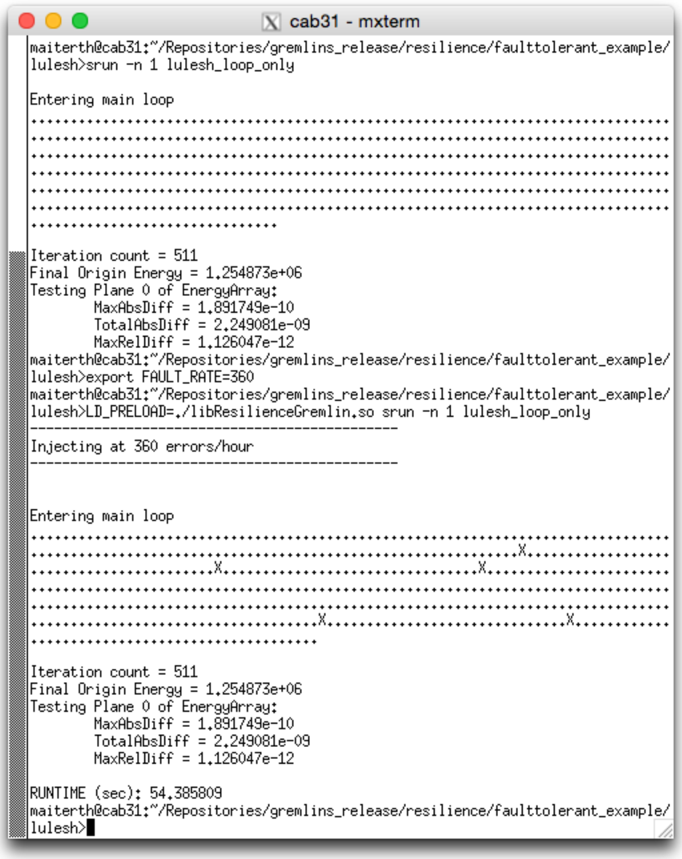
The repository [GREM 14a] provides an example to test the Resilience GREMLINs. For this, *Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics* (LULESH), a well studied proxy application is used. The implementation provided includes two shell scripts create_patch.sh and patch_code.sh. If applied, the resulting patched version of LULESH detects user signals and has a simple try and restart mechanism built in, which restarts if a signal is received during execution.

Figure 2.3 shows the execution of the patched LULESH application. Each iteration prints a dot if executed successfully. After the first run, the environment variable FAULT_RATE is set to 360. This environment variable sets the rate at which faults occur per hour. If the Resilience GREMLIN is preloaded with the above setting, on average every 10 seconds a user signal is sent to the application, indicating an error. The application responds to this by restarting the faulty iteration. This is done until the last iteration is completed successfully and the program exits. The failed iterations are indicated by printing an X to the output as seen in the second part of Figure 2.3.

2.4 Memory GREMLINs

The Memory GREMLINs try to alter memory behavior to emulate memory restrictions that are likely to occur in exascale machines. This includes but is not limited to memory size, memory bandwidth, restrictions in the cache hierarchy and cache sizes. The functionality used by the Memory GREMLINs implemented builds on a modified version of a memory library written by Marc Casas Guix [CGBr 14]. The GREMLIN so far only includes cache capacity interference and memory bandwidth interference.

For stealing cache and bandwidth additional processes are needed that stress these resources. This has to be done in a way transparent to the user application. During the initialization phase a new communicator is created. The original communicator MPI_COMM_WORLD includes all processes. It is split into two groups, one for the contaminating processes and one for the processes that are actually involved in running the application. In subsequent calls to MPI functions that uses the communicator MPI_COMM_WORLD the communicators are switched out. The user application uses the communicator that only includes the processes for running the application while thinking it is using MPI_COMM_WORLD. For this, the MPI program is started with a surplus of processes, since extra processes are needed for the contamination. The GREMLIN is again controlled via environment variables: N_NODES indicates how many nodes are used for the program. N_CONTS is set to the number of contamination processes. N_CORES includes the number of cores used for normal program execution plus contamination processes. BUFFER is used to indicate how much memory a single interference processes should use. Using the environment variable METRIC the type of memory interference can be chosen. The above variables, above all, N_CONTS and BUFFER have to be fine tuned to achieve the desired results. The method used for the interference as used by the Memory GREMLIN is described in [CGBr 14]. This should also give insights on how to configure the variables for the desired interference.



```
maiterth@cab31:~/Repositories/gremlins_release/resilience/faulttolerant_example/
lulesh>srun -n 1 lulesh_loop_only

Entering main loop
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
...............................

Iteration count = 511
Final Origin Energy = 1.254873e+06
Testing Plane 0 of EnergyArray:
        MaxAbsDiff = 1.891749e-10
        TotalAbsDiff = 2.249081e-09
        MaxRelDiff = 1.126047e-12
maiterth@cab31:~/Repositories/gremlins_release/resilience/faulttolerant_example/
lulesh>export FAULT_RATE=360
maiterth@cab31:~/Repositories/gremlins_release/resilience/faulttolerant_example/
lulesh>LD_PRELOAD=./libResilienceGremlin.so srun -n 1 lulesh_loop_only
----------------------------------------------
Injecting at 360 errors/hour
----------------------------------------------


Entering main loop
................................................................................
.............................................................X..................
.......................X................................X.......................
................................................................................
................................................................................
....................................X..............................X............
....................................

Iteration count = 511
Final Origin Energy = 1.254873e+06
Testing Plane 0 of EnergyArray:
        MaxAbsDiff = 1.891749e-10
        TotalAbsDiff = 2.249081e-09
        MaxRelDiff = 1.126047e-12

RUNTIME (sec): 54.385809
maiterth@cab31:~/Repositories/gremlins_release/resilience/faulttolerant_example/
lulesh>
```

Figure 2.3: Running a proxy application with and without the Resilience GREMLIN.

When a program is executed using one of the Memory GREMLINs, on each node the specified amount of contamination processes are created. Depending on the environment variable `METRIC`, restrictions in cache capacity or memory bandwidth are emulated. The GREMLIN for memory bandwidth interference tries to transfer big amounts of data between memory levels. This is achieved by introducing frequent memory misses to keep memory transfers active. The GREMLINs memory access patterns are specified in a way to maximize the bandwidth used by a single interference process. These sustained transfer rates are thus higher than those achieved during normal program execution by user applications over a longer period of time. The cache capacity interference tries to keep the memory in the cache by random access.

With these techniques the user program has either only a smaller amount of bandwidth or smaller cache sizes for itself. This gives the possibility to emulate the increase of cores per node in respect to memory behavior.

3 Proxy Applications

This chapter introduces the proxy applications used to analyze power behavior for large scale scientific applications. A proxy application is a small application consisting of a reduced subset of a production code. It is well-suited to analyze for developers that are non-domain scientists. Proxy applications that capture workloads of production codes help to understand performance bottlenecks on real hardware while staying small and understandable. This gives the possibility to study a close approximation of the real application without having to handle a full production code.

Since proxy applications are open source, they can be studied by a wide audience to help understand and optimize HPC systems for scientific workloads. In contrast to benchmarks, proxy applications capture real scientific workloads. Heroux et al. [HNS 13] give a list of aspects of proxy applications that are of particular interest:

- Algorithms
- Data structures and memory layouts
- Parallelism and I/O models
- Languages and coding styles

These characteristics are important for having a code base that is reliable, performance portable and running with high efficiency on current and future generations of HPC systems. Proxy applications give a testbed for trying new approaches to any of these points. Changing one of those application characteristics may require a total rewrite of the proxy application. Since the code base is relatively small and understandable this can be achieved easily compared to making these changes to the production code itself. The optimized or changed proxy application can then be evaluated against the original proxy application. When changing any of the above mentioned application characteristics in a benchmark, the specific metric captured is not comparable to previous executions anymore. Proxy applications make it easy to evaluate these changes.

In this work none of the above mentioned aspects are directly changed. Thus benchmarks or proxy applications can both be used to see the changes of how the GREMLINs impact the application execution. However the GREMLINs are designed to allow understanding of application runs of production codes on next generation HPC systems. Since the GREMLINs can impact any of the above mentioned aspects proxy applications are the main focus of this work. The information gained from running a GREMLIN that is tampering with the program execution can be fed back into the development cycle. To handle the difficulties introduced by the emulated environment changes have to be made to the proxy application or the system. These changes are then again reflected in one or more of the above mentioned aspects of the proxy application.

There are several kinds of proxy applications: kernels, skeleton apps and mini apps [HNS 13].

- Kernels represent the main workload of a production code. The kernel consists of the most compute intensive function of the application and are often a implementation of a specific mathematical solver. For these the per processor execution time has to be optimized. Kernels provide a good proving ground for algorithmic changes, testing data structure and altering memory layouts. Since proxy applications try to be minimal there is likely no parallelism involved if the mathematical model does not require it. Production code applications may consist of several compute kernels.

- Skeleton apps capture the I/O and memory transactions required to execute the production code. The main target here is network performance and characteristics. The mathematical model is most often not implemented at all or only captured rudimentary. Skeleton apps are suited best to test different distributed memory layouts, parallel language paradigms and I/O models.

- Mini apps combine a single compute kernel of the production code with the memory and network interactions and simple I/O. These stand-alone applications describe a simplified version of a physical phenomenon captured by the production code. Input values are provided by small input files or are hard coded. The focus of a mini app can range from explicit snippets to complete execution paths of a production code.

The proxy applications studied in this work are mini apps using MPI for their parallel execution.

The Department of Energy has several proxy applications of interest to them. The production codes represented in those proxy application codes consume a significant fraction of the execution times at different HPC installations of the DOE. The proxy applications are mostly developed by DOE co-design centers like ExMatEx [ExMatEx 14b] or CESAR [CESAR 13], but also play an important role for acquiring new HPC systems [Ward 14].

This work focuses on three proxy applications: AMG2013, a parallel algebraic multigrid solver for linear systems on unstructured grids [AMG 13]; CoMD, a classical molecular dynamics code, used in the ExMatEx co-design process [CoMD 13]; and NEKBONE, a thermal hydraulics mini-application used in the CESAR co-design center [FiHe 13b].

The proxy applications AMG2013, CoMD and NEKBONE reproduce performance characteristics of 3 different production codes used in scientific computing. These mini applications capture the computational intensive kernel of the application, its communication characteristics and essential parts to mimic the behavior of the production code. The different power levels observed mainly reflect how computational intensive the kernels are. This can be measured by the amount of arithmetic operations performed per memory access. Characteristics that also impact power consumption are what kind of arithmetic operations the compute kernel requires and whether the CPU pipelines are fully utilized. The single node power observations build the baseline for the scaling studies of Chapter 5. The following sections introduce the proxy applications used in this work and their single node power behavior.

3.1 AMG2013

The proxy application AMG2013 [AMG 13] is a mini app derived from the *AMG* (algebraic multigrid) solver used in the hypre library [FaMY 02], BoomerAMG [VEMY 00]. The solver is a scalable AMG implementation that is known to be able to solve problems involving tens of millions of unknowns on several thousands of processors. AMG2013 is a *SPMD* (Single Program, Multiple Data) application written in C using MPI and OpenMP. For simplicity only MPI is used in this work.

Algebraic multigrid solvers are in wide use in scientific applications. Example use cases for AMG solvers are simulations of plastic and elastic deformations of explosive materials at LLNL. Other use cases include the modeling of fluid pressure in the eye, the speed-up of simulations for Maxillo-facial surgeries to correct deformations, sedimentary basin simulations in geosciences or the simulation of magnetohydrodynamics formulation of fusion plasma [BGSY 11].

In contrast to the geometric multigrid method, the algebraic multi grid method does not use the geometric information of the grid and thus is a more general approach. The AMG method uses an iterative approach for convergence towards a solution. The method moves between coarse and fine grids. The operations to get an optimal solution while performing these steps are smoothing and interpolation. Initially the structure is represented as a fine-grained matrix. When moving from fine to coarse, a smoother is applied to eliminate high frequency errors. At the coarsest level a relatively small linear system is solved. To solve the linear system at a coarse level is relatively inexpensive compared to the fine grid. When moving back from coarse to fine, the calculated solution is interpolated, to propagate the estimated solutions over the different grid granularities used for the specific AMG method. At the finest grid level the calculated solution is compared to a estimated solution and the process terminates if the residual error is small enough.

AMG is of particular interest to the scientific community and HPC due to good parallelizability and a better scaling behavior compared to other methods. Algebraic multigrid scales in $O(N)$, where N is the number of unknowns in the linear system to solve. Because of the fact that it uses a generalized representation of the grid, AMG can work on unstructured grids and is applicable where other methods are unusable. A formal introduction to algebraic multigrid solvers is given in [Stüb 99].

AMG2013 implements the algebraic multigrid method according to the hypre library. When running AMG2013 the user can select from several different solvers and preconditioners. A preconditioner is a preliminary step to bring down the conditional number of the linear system. This leads to faster convergence and better stability of the method. AMG2013 uses AMG itself as a preconditioner if selected. For solving the linear system the user can choose PCG, a preconditioned conjugate gradient method, or GMRES, a general minimal residual method. In production codes AMG is often used as a preconditioner for both congugate gradient or GMRES.

The AMG method consists of several phases: the setup phase and the solve phase. In the setup phase the different levels of grid granularity are set up, and how these are connected to the levels above and below. In the solve phase the actual computational iterations are performed.

Since the interest of this paper lies on the power consumption, the power consumption for a baseline run using a single node are measured. Figure 3.1 shows the power profile of of a small run. The average power

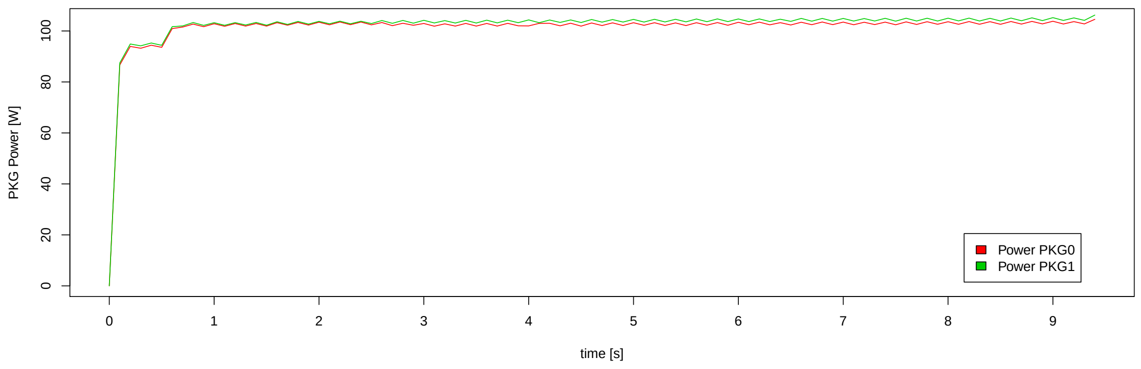


Figure 3.1: Power profile of AMG2013's solve phase. The selected configuration solves a 3D 7-point Laplace problem on a cube using the PCG solver. Time resolution of measurement: 0.1 seconds.

consumption, measured in a 0.1 second interval, is $\sim$103 watts in the solve phase. Initially a lower power consumption of 92 watts is seen. The lower power consumption does not correspond to the setup phase, which is completed after less than 0.1 seconds for this small single node test. The setup phase thus is neglectable here and most time is spend in the solve phase. After 0.5 seconds the power consumption stabilizes in the solve phase and an average of 103 W is observable for the rest of the test run. During the solve phase the power consumption stays relatively constant with a variance of less than 0.7 watts. The two packages show a difference in power consumption with an average difference of 1.1 watts. In this test socket PGK0 always consumes less power than PKG1.

This run was executed on a single node using 16 MPI processes on LLNL's cab system. Regarding the pattern of power consumption the same is observed on both packages with a variance of only 0.2 W. Thus it can be said that both packages observe the same constant power behavior in the compute kernel of the AMG2013 proxy application.

3.2 CoMD

The proxy application CoMD is a mini app describing a classical molecular dynamics code. CoMD is an ExMatEx proxy application [CoMD 13] and written in C. There are three implementations: a serial version, a version using OpenMP only and a version using MPI only. In this work the MPI version is used. Molecular dynamics codes are not only an important case of N-body methods. They are used in many different fields from computational chemistry and biology to fluid dynamics simulation and material science. In molecular dynamics materials are represented as individual atoms or molecules. The interaction between the particles is expressed as force interactions, with each particle having a position and velocity. Molecular dynamics describe the Newtonian equations of motion, while conserving energy. The simulation obeys the conservation of energy with changes in kinetic energy and temperature.

The high level of detail of a molecular or atomic based simulation requires statistical approaches to predict larger structures. Molecular dynamics simulations are a very useful tool to supplement measurements that are hard or nearly impractical for a laboratory setting. The technique allows to study the behavior of molecules that have not been synthesized yet, or particles that are too short lived to measure with current equipment. The simulation environments can be changed to extremes that are not feasible in a laboratory environment. To conduct experiments at a few nanokelvin or near the temperature of stars may not be possible while staying within budget. For molecular dynamics simulations high pressures or extreme temperatures are only a change in parameters.

Examples for production codes using molecular dynamics include ddcMD [GBB+ 12][SGP+ 06], developed at LLNL, and SPaSM [KGL 06], a high energy physics code developed at *LANL* (Los Alamos National Laboratory). Production runs for these applications often run on the complete HPC system and still show good scaling behavior. Thus it is important too predict and identify the most critical factor for good performance in a molecular dynamics simulation in an exascale environment.

CoMD is a general molecular dynamics simulation. Molecular dynamics simulations run in discrete time steps. The spacial domain is distributed across nodes using cells, where each MPI process is responsible for one cell and the atoms the cell contains. In CoMD the initial step of each iteration is to update the velocity of each atom. Using these values, the positions of the atoms are updated. Atoms that moved to a different cell are redistributed and information from neighboring cells is gather for the next step: the force calculation. The force calculation is divided in two separate steps, first the interacting atoms are identified, second the forces of the interacting atoms are computed. The maximum interaction range of the atoms forces under consideration are smaller than the size of a single cell. With this implication only the atoms within the cell itself and its neighboring cell have to be considered for interaction. This reduces the amount of communication needed to neighboring cells. As a final step the forces can be computed to update the velocity for the next iteration.

Figure 3.2 shows the consumption that could be observed on a single node test run using 16 MPI processes on LLNL's cab system. The average power consumption, measured in a 0.1 second interval, is ~ 85 watts.

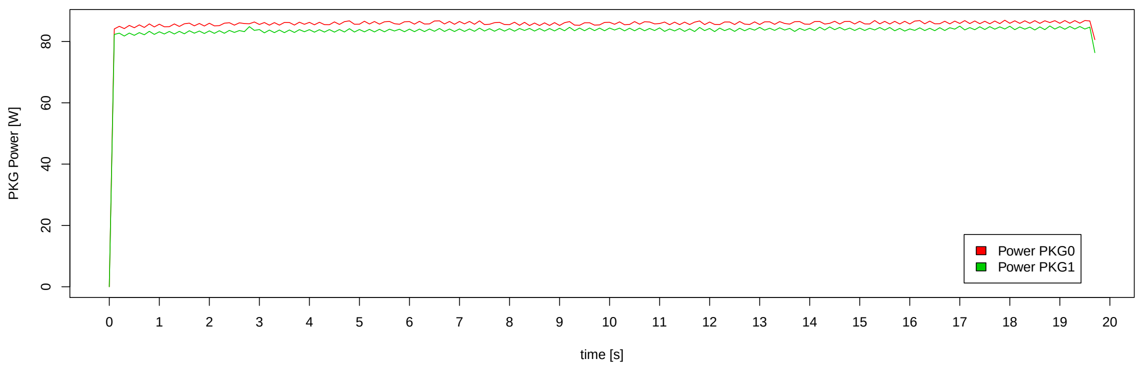


Figure 3.2: Power profile of CoMD execution. The selected configuration solves the Lennard-Jones potential with approx. 110000 atoms. Time resolution of measurement: 0.1 seconds.

In contrast to AMG2013 there is no initial phase, and within 0.1 seconds both packages are in the compute kernel. The power profiles of the packages show a maximum variance of 0.4 watts and are constant over the runs on both packages. Again a difference in power consumption is observable between packages with an average difference of $2.2\ W$. In this test run package PKG0 always consumes more power than PKG1.

The pattern observed for power consumption is again constant and similar for both packages for the CoMD proxy application.

3.3 NEKBONE

NEKBONE [FiHe 13b] is a proxy application from the CESAR exascale co-design center, based on the open source spectral element code NEK5000 developed at *ANL* (Argonne National Laboratory) [FLK 08]. Using the fluid dynamics code of NEK5000 scientists study the unsteady incompressible fluid flow in two and three dimensions. The design goal is to implement scalable turbulence simulations in complex domains using *LES* (Large Eddy Simulation) and *DNS* (Direct Numerical Simulation). Examples for the use of NEK5000 are reactor thermal-hydraulics, astrophysics, combustion, oceanography, or vascular flow modeling. NEK5000 won a Gordon Bell prize for its outstanding scalability for a fluid dynamics simulation and was designed as part of the CESAR co-design effort [CESAR 11].

NEK5000 uses a higher order spectral element method to solve the Navier-Stokes equations. In contrast NEKBONE solves the Helmholtz equation using the spectral element method. The proxy application is a mini app focusing on three dimensional geometries that exposes the main computational kernels to reveal the essential elements of NEK5000. For this NEKBONE uses a standard Poisson equation and conjugate gradient with a simple or spectral element multigrid preconditioner.

NEKBONE is shipped with six example configurations representing different workloads of NEK5000. NEKBONE and NEK5000 show similar scalling behavior and memory requirements. The configuration files for NEKBONE are set in the `SIZE` file and the `data.rea` file. The `SIZE` file sets parameters fixed after compiling. These values include the polynomial order of the simulation, the maximum number of processors as well as the maximum number of elements per processor and the maximum total elements. The `data.rea` file sets runtime parameters.

NEK5000 and NEKBONE are open source and implemented in Fortran77/C, using MPI for parallelism. The `readme.pdf` file [FiHe 13a] gives a more detailed overview including instructions to build and run the application.

Figure 3.3 shows the power profile of a single node run on LLNL's cab system. The average power

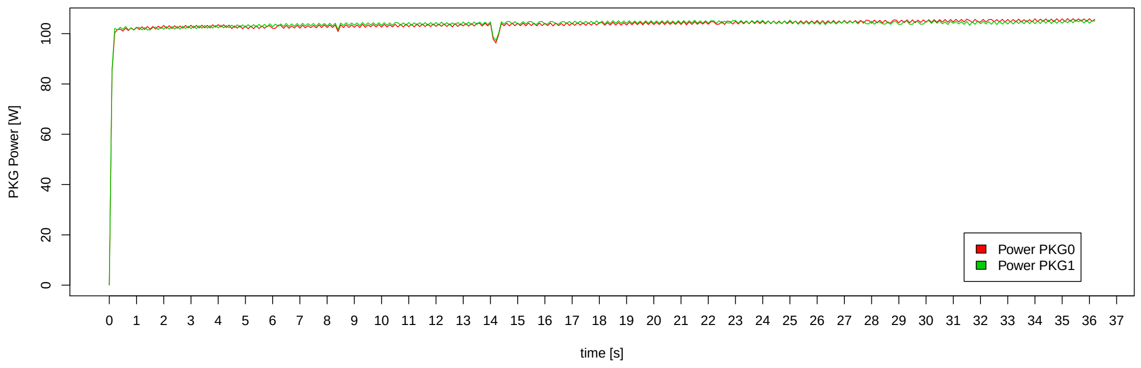


Figure 3.3: Power profile of NEKBONE execution. Time resolution of measurement: 0.1 seconds.

consumption measured in a 0.1 second interval is $\sim$103.5 watts. The power consumption start to increase over the course of the execution. After 0.2 seconds the power consumption starts to increase from $100\ W$ to $104\ W$. After 14 seconds the power consumptions drops below $100\ W$ for 0.3 seconds, and stays above $\sim$103 W for the rest of the execution time. Both packages show similar power behavior, with no package having consistently lower or higher power consumption as seen in the proxy applications introduced above.

4 Experimental Setup and Methodology

The research objective is to analyze application and system behavior for multi node parallel jobs using the proxy applications introduced above. This chapter introduces the experimental setup used to conduct the measurements. All experiments were set up and run on the cab system at LLNL. At the time of writing, the cab system is the largest HPC resource at LLNL with the `msr-safe` kernel module installed and thus is the system of choice for this work. The following sections introduce the hardware and software of cab used to make the measurements possible. The final section of this chapter introduces the experimental setup for the weak and strong scaling studies on up to 256 nodes, which are evaluated in Chapter 5.

4.1 Hardware Setup

Cab consists of a total of 1296 nodes. Each node has 2 Intel Xeon E5-2670 CPUs with 8 cores each, giving a total of 20736 CPU cores. These CPUs are of the "Sandy Bridge EP" microarchitecture using 32 nm technology. The specified CPU speed is 2.6 GHz with a maximum turbo frequency reaching 3.3 GHz. Each node has 32 GB of memory, giving a total system memory of 41.5 TB. The peak CPU memory bandwidth reaches 51.2 GB/s. The interconnect used in the system is InfiniBand QDR by QLogic.

The *TDP* (Thermal Design Power), the average power in watts drawn by the processor under a Intel defined high-complexity workload, is 115 W. The TDP is measured while operating at base frequency with all cores active [Intel 12]. The maximum power clamp applied is 115 W since no application was able to used more power. The measurements have been taken with turbo boost deactivated for better predictability.

The system has two node scheduling pools: 32 nodes designated for debug runs and 1200 nodes for batch job execution. The system restricts the user to a maximum of 258 nodes per job with a maximum runtime of 16 hours for the batch scheduling pool [LC 14]. These usage restrictions build the outline for the experiment setups.

4.2 Software Setup

The operating system used on the majority of Livermore HPC installations is TOSS. Cab uses TOSS 2.2, the Tri-Lab Operating System Stack. TOSS is the operating system formerly developed as CHAOS, the Clustered High Availability Operating System. The operating system is derived from Red Hat Linux. The changes include special customizations for high performance computing hardware, the Lustre parallel file system and system monitoring [LC 10]. TOSS is in wide use across the three *NNSA* (National Nuclear Security Administration) laboratories, LLNL, *LANL* (Los Alamos National Laboratory) and *SNL* (Sandia National Laboratories).

The compiler used for the work is Intel's ICC, version 12.1.5. Regarding MPI, MVAPICH2 [OSU 14] is used, with the compiler version `mvapich2-intel-debug-1.9`.

The tools needed to run the GREMLINs as introduced in Chapter 2 are modified development versions. P^{n}MPI uses a modification of version 1.4. The modified version supports timers and signals and is to be included in the next release on the official GitHub web page [PnMPI 09]. The wrapper tool is provided by P^{n}MPI. `libmsr` and `msr-safe` are used as provided and installed on cab. These versions are the initial releases publicly available via the projects GitHub web page [ShRo 14a][ShRo 14b]. The GREMLIN framework is the initial release provided via GitHub [GREM 14a], developed for this work.

The proxy applications used are the public releases available via the CESAR [CESAR 13], ExMatEx [ExMatEx 14b] and CORAL [Ward 14] Project pages.

The AMG2013 version used is provided via the co-design web page [AMG 13] and can also be found in the CORAL Benchmark Codes [Ward 14]. The configuration used is a slight alteration of the configuration provided with the proxy application. To get comparable runtimes across different node counts from 1 to 256 nodes, the convergence check was disabled resulting in a fixed iteration count of 1000 iterations. The configuration for each run is node dependent. The process layout is 3 dimensional with layout: X processes in x-direction, where X is the number of nodes, 16 processes in y-direction, and 1 process in z-direction. The overall problem sizes for strong scaling is set to $1280 * 16 * 100$. The problem size for weak scaling is set to $400 * 100 * 100$ per node. For the solver `-laplace` is chosen with `pooldist` set to 1.

For CoMD version 1.1 is used, as found on the ExMatEx GitHub web page [McPh 13]. The tests use the pure MPI version of the proxy application. The processor layout is 3 dimensional, with X processes in x-direction, where X is the number of nodes, 1 processes in y-direction, and 16 process in z-direction. The problem size for strong scaling is $12 * x$,$12 * y$ and $12 * z$. For the weak scaling study x is chosen as the maximum number of nodes for each configuration, giving a total overall problem size of $3072 * 12 * 192$. The maximum iteration count is set to 400 for weak scaling and 800 for strong scaling.

The proxy application NEKBONE is used in version 3.1. NEKBONE is shipped with example configurations for testing. In this work the configuration of example3 is used with modified parameters for the scaling tests. In the `SIZE` configuration file the simulation is set for 3 dimensions with a polynomial order of 10. The maximum number of processors is set to 4096. The `data.rea` file is reconfigured for 3000 elements per processor. The remaining parameters are configured as given in the default settings of example3. For strong scaling tests the settings above were used. For the weak scaling tests the elements per processor had to be changed for each run, to get a total number of elements of 2^{15}.

The selected values for problem size and number of elements provide a minimal runtime of 15 s for each of the proxy applications. This makes the measurements more comparable and more resistant to performance variation and noise.

4.3 Initial Weak Scaling Test

An initial weak scaling test was performed on up to 32 nodes. This motivates the experimental setup for the scaling studies. The experiment of Figure 4.1 is split in two individual tests. The left part is executed on the debug nodes 1 to 16. The right part is executed on the debug nodes 17 to 32. The plots on the left and right side of the center plot show scaling behavior. The color indicates the power bound used for the measurement, enforced by RAPL. 50 W are indicated in green, 65 W in blue, 80 W in teal, 95 W in pink and 115 W in yellow. In the center plot each dot indicates a single measurement. If dots are connected via a line, these measurements were taken in a multi node test. The multi node runs were clustered according to the node numbering. This means the two node runs were performed on the sets $\{[1, 2], [3, 4], [5, 6], ...\}$, four node runs on $\{[1, 2, 3, 4], [5, 6, 7, 8], ...\}$, etc. On the left and right plots the lines are visual aids to visualize scaling. The measurements under a power bound of 95 and 115 watts show the same scaling behavior. This is due to the sustained power consumption of $\sim$85 W as discussed in Section 3.2

When looking at the performance under a power bound of 50 watts, on the left or right plot, a big variation is visible. Compared to the other power bounds 50 W shows the strongest variation among all settings. These results agree with the findings observed by Rountree et al. [RAdS$^+$ 12], showing performance variation is strongest using low power bounds.

The center plot reveals the different performance of each single node. This again is strongest using 50 watts. In the plot the multi node runs are always bound by the node with the poorest single node performance participating. Well performing single nodes perform good in a multi node run. An exception to this are the 65 W two node runs on nodes 1 to 16, which experience huge noise levels. However, in general, if a single node with low performance is included in the execution, the multi node execution is slowed down by this node. In other words, the runtime always depends on the slowest processor. This motivates ordering the nodes according to node performance when executing multi node runs.

For each scaling study an initial single node performance test with a reduced problem size is executed under a 50 W power bound. According to the results of this test the nodes for the scaling study are ordered. This means for the scaling studies the order is the following: $\{n, n+1, ..., m-1, m\}$, with n as the fastest and m the slowest node according to the 50 W performance. This gives the clusterings $\{[n, n+1], [n+2, n+3], ..., [m-1, m]\}$ for the two node runs and $\{[n, n+1, n+2, n+3], [n+4, n+5, n+6, n+7], ..., [m-3, m-2, m-1, m]\}$ for the four node runs, etc.

To generate reproducible tests and order the nodes, the Moab option `#MSUB -l hostlist=<hostlist>` for requesting specific nodes was tested (see: man 1 msub). With this option the same nodes of the cluster can be requested for each job submission. This turned out to not be a feasible option, since requesting specific nodes prolonged the queue time of normally hours or few days to several weeks or months. Using

SLURM, the jobs within the Moab submission can be placed on specific nodes of the partition returned by Moab. With the initial single node tests of the nodes obtained by Moab, the nodes can be ordered and clustered. This has to be done for each job submission. The job placement is done using the SLURM option `--nodelist=<nodelist>` (see: man 1 srun). Even if the nodes are not the same, the partition of the machine available for the job gives reproducible results when ordered according to performance. In addition, jobs can be repeated on the exact same nodes within the Moab submission. This gives the possibility to compare measurements and show the inhomogeneity of the CPUs performance for each partition.

The presence of noise in the system can be seen by the two node runs under 65 W of nodes 1 to 16 in Figure 4.1. To mitigate noise present in the system the tests can be repeated. In the scaling studies each test is repeated 5 times. The tests could not be repeated more often or the total runtime would exceed the 16 hour limit for a single job submission on cab. With this repetition performance variation on the same processor and node can also be reduced.

When performing scaling studies on many nodes organized in sets of 2^n nodes, most of the nodes are idle for most of the time. When replicating the test across the reserved partition additional measurements can be taken without prolonging the overall scaling study. For this the tests are organized as follows: 256 single node tests, running in parallel, followed by 128 two node tests, again running in parallel, followed by 64 four node tests, etc. This does not only give the advantage of more measurements and less nodes idling, but also gives the ability to compare performance characteristics of single nodes, as seen in the center plot of Figure 4.1. With the initial results from Figure 4.1 the setup for the scaling studies is constructed.

4.4 Experimental Setup of Scalability Studies

The objective for this work was to conduct scalability studies under a power constraint. These studies consist of strong and weak scaling studies for the applications introduced in Chapter 3. The scaling studies are organized with the following goals:

- Strong scaling and weak scaling
- 3 different proxy applications
- Node count of up to 256 nodes
- 5 power levels from 50 W to 115 W

The exact power levels used are 50, 65, 80, 95 and 115 W. For the strong scaling studies, a fixed total problem size was selected. For the weak scaling studies, a fixed problem size per processor was selected, as described above. The number of nodes selected is set as a power of two. For each scaling study a partition of 256 nodes was requested. For weak scaling, experiments from 1 to 256 nodes were performed. As expected strong scaling tests with low node counts experienced high runtimes or could not be executed because of memory issues. Thus the node counts for strong scaling starts at 8 and 32 nodes. NEKBONE experienced issues with strong scaling and only weak scaling results are available.

The exact layout of the scaling studies looks as follows: Each scaling study, AMG2013 strong and weak scaling, CoMD strong and weak scaling and NEKBONE weak scaling is submitted as a single Moab job. Inside each Moab job the scaling study follows these steps: Initially, the single node performance of all nodes of the partition returned by Moab is measured at 50 watts. The results of these measurements are evaluated and an order is generated, according to which the multi nodes runs are clustered. Each of the tests follows this order, or in other words multi node runs are always performed on nodes with similar performance. For each power level measurements are performed at each node count. The measurements for each node count are replicated across all nodes of the partition to maximize utilization. In addition to that, each of the measurements is repeated 5 times to be more resistant to system noise. The tests themselves are submitted as SLURM jobs within the Moab job.

The runs using 5 different power bounds on up to 256 nodes, replicated across the reserved nodes to avoid idle processors and repeated 5 times, sum up to a total of up to 12775 proxy application measurement per Moab job submission. Most of these can be run in parallel, but with a minimal runtime of 20 seconds for the fastest execution and the 16 hour limit, a hard limit for maximum execution had to be set. Since the performance and runtime of single tests is not precisely predicable the maximum allowable runtime was set to 10 minutes, to guarantee the 16 hour limit. This was critical to perform the strong scaling tests. The results of these scaling studies are discussed in the following chapter.

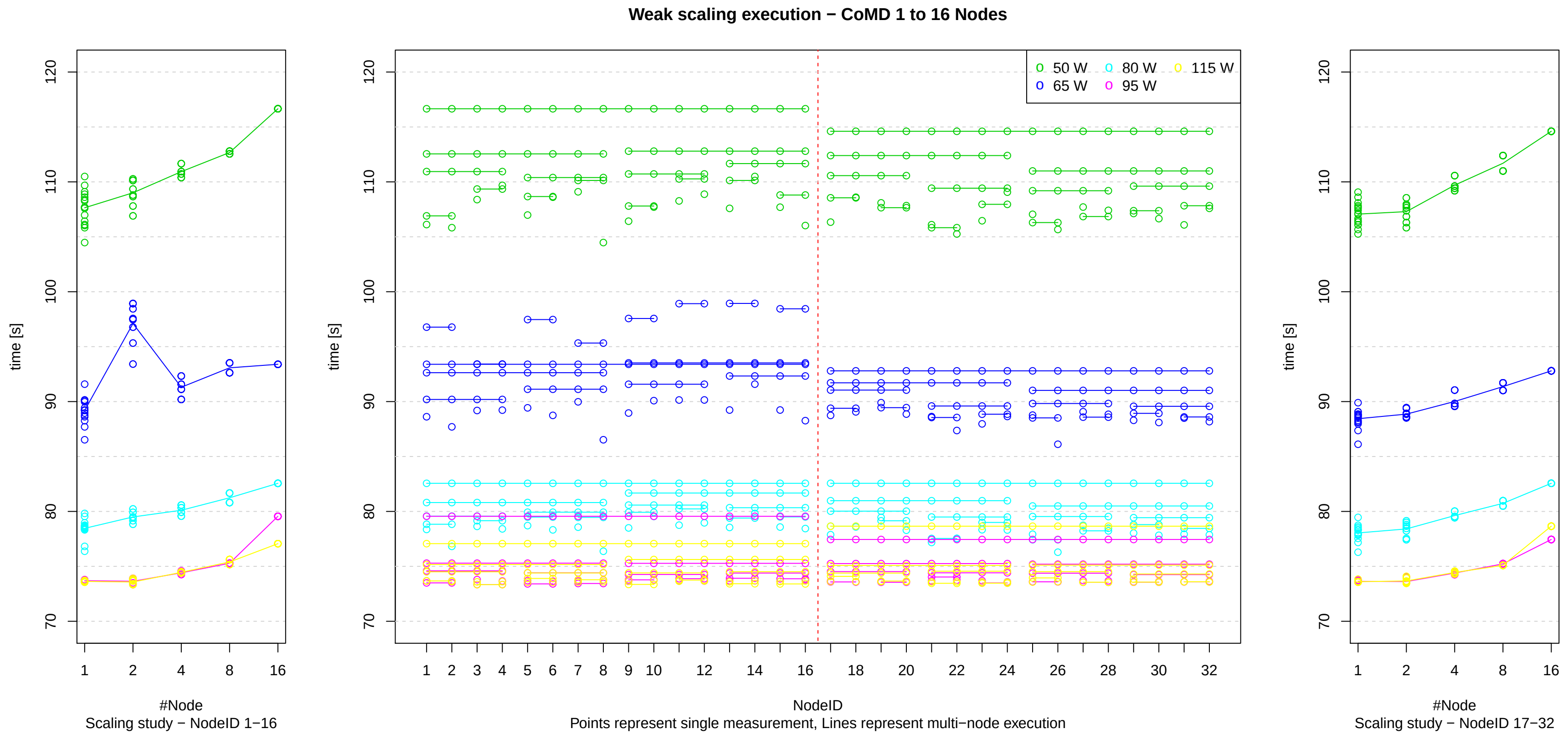


Figure 4.1: Initial CoMD scaling test on 32 debug nodes. The test is split into two individual tests using nodes 1 to 16 and 17 to 32.

5 Scaling Studies

In this chapter the scaling behavior for the previously described proxy applications is discussed. The focus here lies on discussing the different behavior in regards to different CPU power settings. The experimental setup follows the description above.

5.1 Single-node Characteristics

The scaling studies are performed on cab partitions of 256 nodes. Initially the single-node performance under different power bounds is analyzed. These are shown in Figures 5.1, 5.2 and 5.3. The data points are averaged across five measurements and the node order is also provided by average values. This order is set according to the 50 W performance.

To see if the manufacturing variability follows a normal distribution a Shapiro-Wilk normality test is performed [ShWi 65]. On the ordered 50 W values it cannot be concluded, that the performance levels follow a normal distribution. However, this is not true when eliminating the slowest extreme values. For example, when eliminating the two nodes with the extreme values above 41.5 seconds for the AMG2013 single-node performance, the Shapiro-Wilk normality test gives a p-value of 0.806. A value greater than 0.5 allows the assumption of normality. With this information it cannot be assumed that the performance of the nodes follows a normal distribution.

When measuring performance at a power cap above the maximum power drawn by the application nearly no variation is observed. This is the case for AMG2013 at 115 W, CoMD at 95 W and 115 W and NEKBONE at 115 W. As seen in Chapter 3, AMG2013 uses $\sim$103 W, CoMD uses $\sim$85 W and NEKBONE uses $\sim$103.5 W. While above these values, the performance variation is not induced by restricted power, but by noise. This is the case since performance variation is only visible while operating at lower power levels than consumed by the application. The notion of noise is any performance variation induced by changes in the environment not under control of the user. This can be noise generated by the operating system, or on multi-node runs, network traffic slowing down communication, among others. In general noise levels are low, but can occasionally have high impact on job executions.

The runs at lower power caps see additional performance variation introduced by CPU manufacturing variation. The performance impact of manufacturing variation is higher than the noise observed, in this case by a factor of 10. Due to variations in the CPU manufacturing process not every CPU has the same performance to power consumption curve [RAdS$^+$ 12]. The runs at higher power bounds experience a lower performance variation. As seen in Figure 5.1,Figure 5.1 and Figure 5.1,some measurements experience not only performance variation by manufacturing variability, but also by noise. At 65 W the performance still follows the general trend of lower execution time on the left to higher execution time on the right, but this is only a trend and not a strict order anymore. This is very obvious with NEKBONE, where high noise levels could be observed especially with a power cap of 65 W.

The noise levels observed for the single-node runs are relatively small and the variation can be attributed to performance variation. This is emphasized due to the small variation levels when looking at performance variation of the same node compared to variation across all nodes. The maximum performance variation across all nodes is calculated from the averaged values of the single-node results. The maximum performance variation observed on a single node for AMG2013 is 0.017 seconds. The maximum performance variation observed across all nodes for AMG2013 is 0.118 seconds. This indicates that noise levels are small, otherwise these variation values should be similar. For CoMD the maximum performance variation observed on a single node is 0.037 seconds. While a value of 0.150 seconds is observed for maximum performance variation across all nodes.

The measurements for NEKBONE are very noisy. This can be seen in Figure 5.3. The maximum single-node variation observed is at 65 W. One of the 5 iterations encountered high noise levels. This impacts the measurement and a variance of 2.028 seconds is observed at 65 W on a single node. When removing the noisy measurement, the maximum single-node variation drops down to 0.015 s. However, the maximum

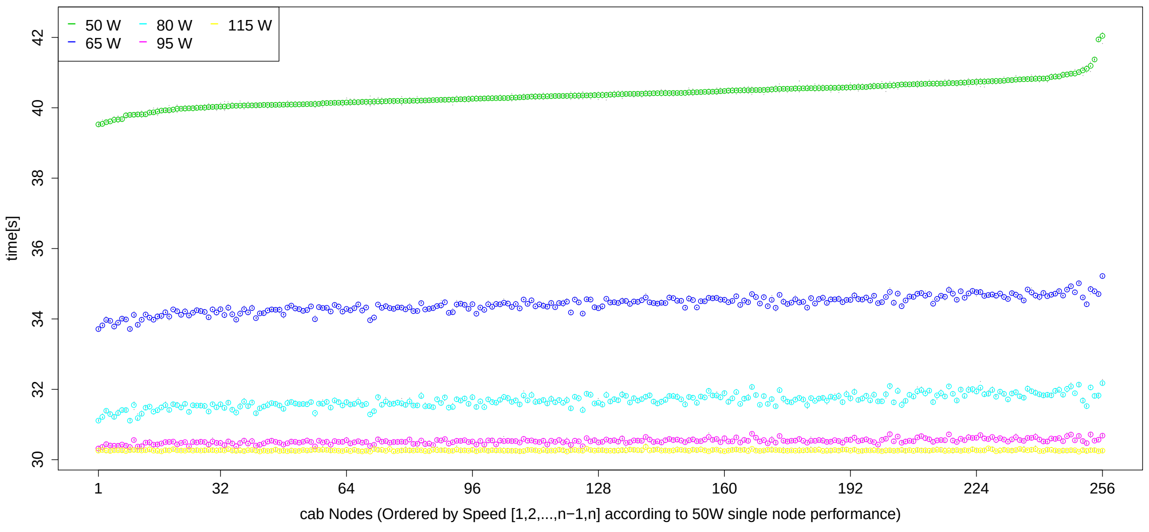


Figure 5.1: AMG2013 single-node performance variation. Average performance over five measurements.

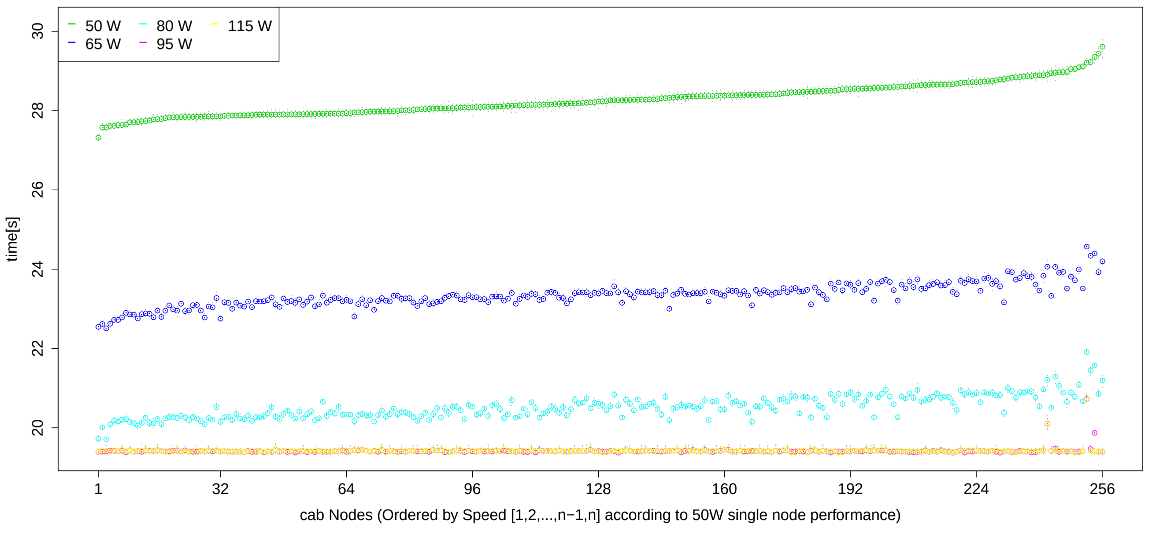


Figure 5.2: CoMD single-node performance variation. Average performance over five measurements.

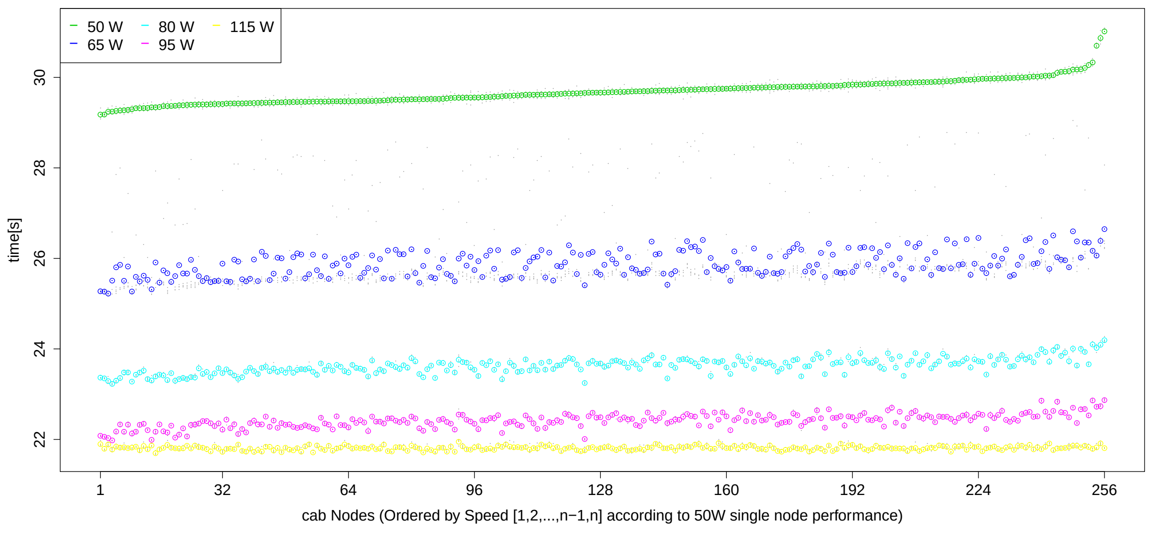


Figure 5.3: NEKBONE single-node performance variation. Average performance over five measurements.

performance variation across all nodes is 0.0803 seconds. This does not change, even when including the averaged performance values of the noisy run.

With the single-node variation discussed for the three proxy applications, the next step is to look at multi-node runs and the weak and strong scaling studies.

5.2 Weak Scaling

This section discusses the weak scaling studies for the multi-node runs. Figures 5.4, 5.5 and 5.6 show a simple weak scaling plot of the proxy applications. Note that the scale of the x-axis is $\log_2$. The color code is kept consistent throughout this work, with 50 W being indicated in green, 65 W in blue, 80 W in teal, 95 W in pink and 115 W in yellow. The yellow line always shows the baseline performance of the proxy application, since 115 W equals no power bound. The general scaling trends of all proxy applications show similar behavior, regardless of a power bound applied. However, some differences in scaling can be observed regarding the different power levels.

The nodes contained within the partition returned by Moab for the measurements are spread across the cab system. This gives other applications running on the system the possibility to interfere by stressing the interconnect. This means that some measurements are noisy despite working with average values across 5 different runs. For the AMG2013 weak scaling run the 4 node measurement at 50 W, as well as the 32 node measurement at 95 appear to be noisy. For CoMD the 32 node run at 65 W and the 128 node run at 95 W are noisy runs. Regarding NEKBONE, the 64 node run at 50 W as well as the 128 node run at 80 W are noisy. All measured values are plotted as gray dots, the averages are plotted in color. For the rest of the measurements, averaging across 5 runs appears to return good values. The general scaling behavior does not change with different power levels, however the slowdown introduced by the power bounds is very noticeable.

A perfectly weak scaling application would show a horizontal line. This is because every processor always has the same amount of work, no matter how many processors are present. The increase in runtime is introduced by an increase in communication per node. Factors contributing to the non perfect scaling can also be load imbalance or missing concurrency.

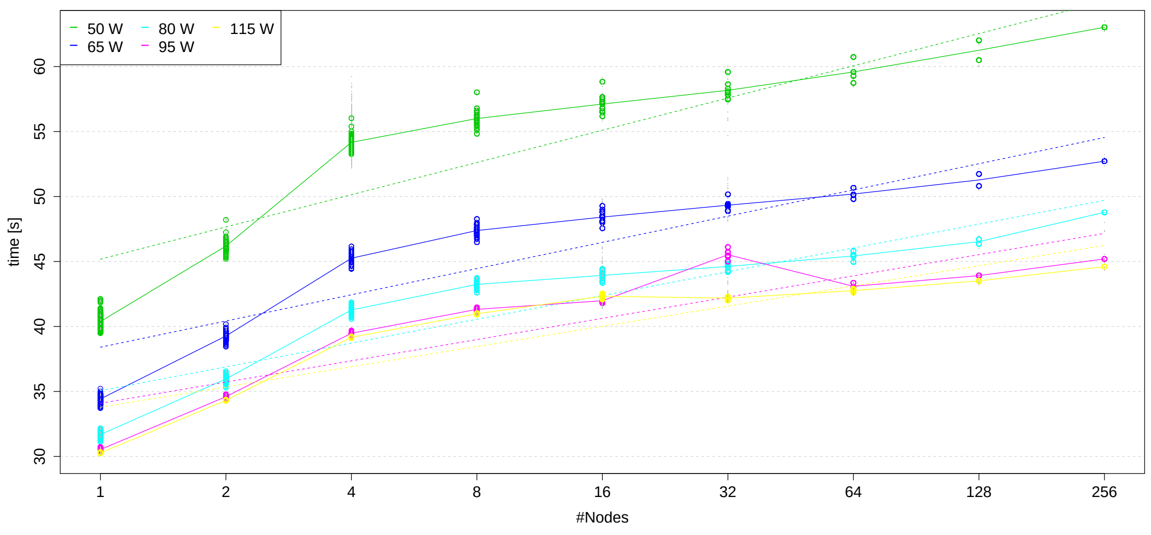


Figure 5.4: AMG2013 weak scaling study. Average performance over five measurements. Each measurement is replicated to utilize all 256 nodes. Dashed lines indicate the simple linear regression curve.

5.2.1 AMG2013

For the weak scaling studies, AMG2013 shows a strong increase in runtime when moving from 1 to 4 nodes. Beyond 4 nodes the steepness of the curve goes down and the increase in runtime appears nearly linear. With the logarithmic x-axis, this growth is logarithmic itself.

The average runtime for the fastest run at the maximum power is 30.267 seconds. The average runtime for the slowest run takes up to 44.603 seconds at maximum power. At 95 W the time difference is less than a second. Starting at 80 W the runtime difference goes from an average low of 31.681 s to an average high of 48.785 seconds. The strongest slowdown is at its maximum at 50 watts with a minimal runtime for a single node averaged at 40.378 seconds and a maximal runtime with an average of 63.026 seconds at 256 nodes.

The maximum variance of the performance increases when lowering power. At 115 W the maximum variance is at only 0.022, At 95 W and 80 W the maximum variance is still below 0.1 with 0.095 and 0.099. At 65 W the variance rises to 0.214 and reaches 0.586 at 50 W. No correlation between variance and node count could be found.

When fitting a logarithmic regression curve, plotted as a dashed line in Figure 5.4, the difference in the slope of the curve is very noticeable for different power levels. For the regression function $a + b * log_2(x)$ the value for b at 50 watts is 2.4. For 115 W the slope b is flatter with a value of 1.5. When looking at values above 80 W, difference in the y-intercept is less than 1.5 seconds with the slope b staying below 1.8.

As seen in Section 3.1 AMG2013 consumes 103 watts. Thus the performance measurement under a power restriction of 95 W already shows a slight different behavior.

5.2.2 CoMD

CoMD shows a smoother scaling curve with no abrupt changes. The plot of Figure 5.5 shows CoMD weak scaling at different power bounds. CoMD shows a good scaling behavior, which is better than linear for all power setting (Note: $\log_2$ x-axis).

The fastest average runtime at maximum power is 19.423 seconds. The slowest average runtime at 115 W reaches 26.371 seconds. At 50 W the runtime is minimal with an average of 28.271 seconds and maximal with an average of 39.901 seconds. As seen in all weak scaling studies performed low node counts always experience faster execution, even though the same amount of work is done per node.

With the power consumption of CoMD of ~85 W as seen in Section 3.2 both 115 W and 95 W show the

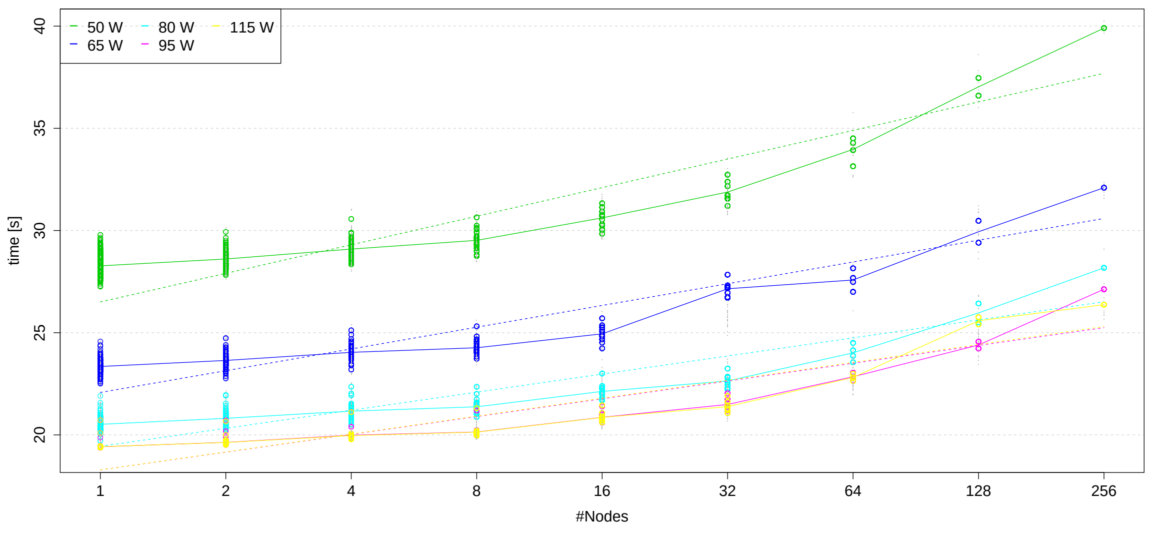


Figure 5.5: CoMD weak scaling study. Average performance over five measurements. Each measurement is replicated to utilize all 256 nodes. Dashed lines indicate the simple linear regression curve.

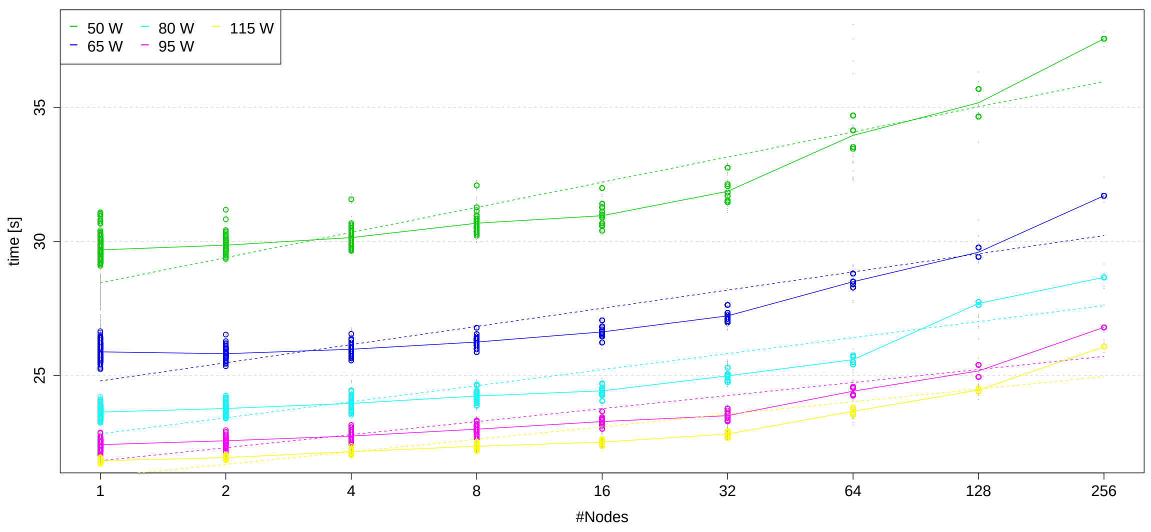


Figure 5.6: NEKBONE weak scaling study. Average performance over five measurements. Each measurement is replicated to utilize all 256 nodes. Dashed lines indicate the simple linear regression curve.

same behavior. The noise observed at 128 and 256 nodes is an exception to this. However, already at 80 W, only 5 watts below the maximum power the application consumes, the execution times slows down by more than 1.5 seconds.

The variance is lowest at 115 W with a maximum variance of 0.063. 50 W, 65 W and 80 W, show increased variance levels with 0.271, 0.288 and 0.215. The variance is highest at 65 W and no correlation between node count and performance variance could be observed.

When fitting a logarithmic regression model, the regression curve has the following parameters: The interesting part of the regression function $a + b * log_2(x)$ for the different power levels is the slope b. For 50 W b equals 1.398, for 65 W b equals 1.064, for 80 W b equals 0.884, for 95 W b equals 0.870 and for 115 W b equals 0.877. The difference in y-intercept between 65 W and 80 W is more than 2 seconds and increases to more than 4.5 seconds when looking at the difference between 50 W and 65 W.

5.2.3 NEKBONE

NEKBONE shows a very similar scaling behavior to CoMD's scaling behavior, as seen in Figure 5.6. The maximum power consumption of NEKBONE is $\sim$103 W as seen in Section 3.3. This already shows one of the differences between NEKBONE's and CoMD's behaviors. For NEKBONE all power settings except for the uncapped at 115 W experience slowdowns introduced by the power bound.

The fastest runtime at maximum power takes on average 21.811 s. The slowest runtime at maximum power is on average 26.079 s. At 50 W the fastest run reaches only a mean of 29.681 s and the slowest run takes 37.559, on average.

The variance is relatively low for power levels above 65 W. Except for a variance of 0.267 at 50 W, the variance is always below 0.081 and is minimal at 115 W with a value of 0.011. The variance does not strictly decrease when lowering the power level. Again no correlation between node count and variance could be observed.

The logarithmic regression model shows the following parameters for the regression function $a+b*log_2(x)$: The slope nearly doubles when moving from unbound at 115 W to 50 W. The values for b are 0.468 at 50 W, 0.486 at 65 W, 0.598 at 80 W, 0.678 at 95 W and 0.938 at 115 W. The y-intercept increases from $a = 21.811$ s at 115 W, to 22.414 s at 95 W, 23.628 s at 80 W, 25.806 s at 65 W and 29.681 at 50 W.

5.3 Strong Scaling

This section discusses the strong scaling studies for the multi-node runs. Figures 5.7 and 5.8 show a simple weak scaling plot of the proxy applications. Figure 5.8 includes a detailed view of the node counts 64, 128 and 256. Again the scale of the x-axis is $\log_2$, and the color code is green for 50 W, blue for 65 W, teal for 80 W, pink for 95 W and yellow representing 115 W.

The yellow line again represents the baseline performance of the proxy application without a power cap. In other words the CPU can consume up to 115 W of power.

A perfectly strong scaling application would show runtime divided in half when doubling the number of processors. For both AMG2013 and CoMD, the scaling behavior degrades when applying lower power caps. However, both show good strong scaling behavior, under all power bounds.

5.3.1 AMG2013

The general scaling trend for AMG2013 is not affected by enforcing a power cap. For AMG2013 the following runtime cuts are observed when doubling the number of processors: When moving from 32 to 64 nodes, only 51.3% are needed at 115 W. At 50 W the scaling behavior is still good with 54.4%. When moving from 64 to 128 nodes at 115 W 54.0% of the execution time is needed. At 50 W the scaling behavior observed is a runtime cut down to 51.3%. Finally when moving from 128 to 256 nodes at 115 W 66.9% of the time is needed. At 50 W the runtime is only decreased by 78.6%.

At 50 W the runtime decreases from 578.34 s at 32 nodes to 126.96 s at 256 nodes, with an ideal target value of 72.29 seconds at 256. At 115 W the runtime decreases from 408.62 s at 32 nodes to 75.77 s at 256 nodes, 51.08 seconds at 256 would mean perfect scaling.

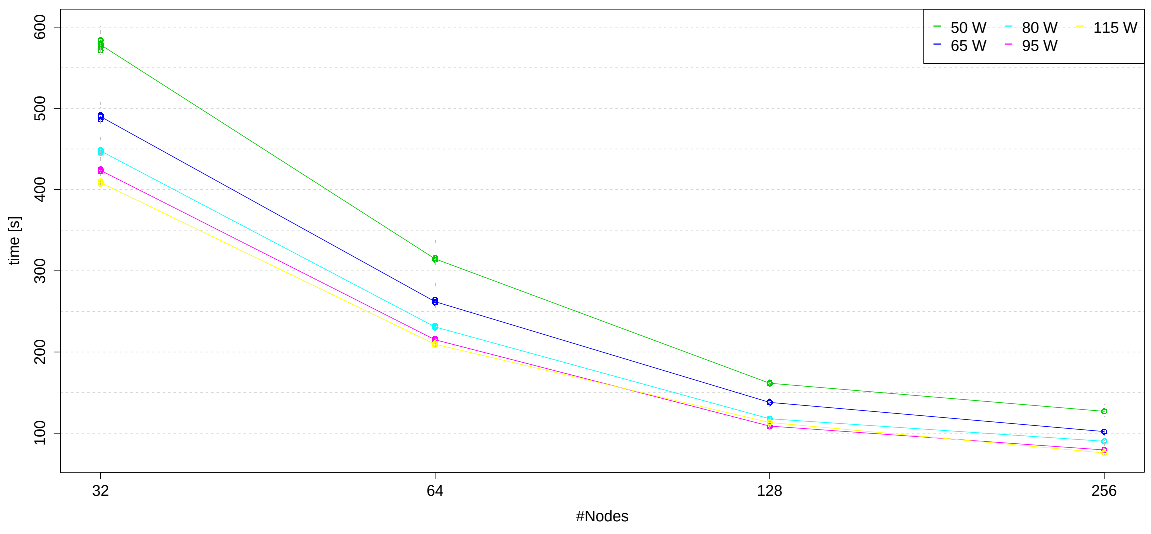


Figure 5.7: AMG2013 strong scaling study. Average performance over five measurements. Each measurement is replicated to utilize all 256 nodes.

Even though the scaling behavior follows the same trend, the scaling behavior degrades faster at lower node counts with a power cap enforced. Most noticeable is the flat performance loss of applying a power cap, which is independent of the node count, as seen in the single-node performance discussion above.

5.3.2 CoMD

When looking at Figure 5.8, a very similar scaling behavior can be observed. CoMD shows the following runtime cuts when doubling the number of processors: At 115 W the runtime observed when moving from 8 to 16 nodes is perfect with runtime decrease of 49.9%. At 50 W the scaling values are still very good with 51.4%. When moving from 16 to 32 nodes the runtime decrease is again near perfect with 50.7% at 115 W. Similarly scaling is achieved at 50 W with a runtime decrease of 50.5%. Moving from 32 to 64 nodes, at 115 W the runtime decreases to 52.1%, at 50 W the decrease is still good with a value of 52.7%. The step 64 nodes to 128 nodes shows a runtime decease of 54.4% at 115 W and 55.3% at 50 W. The last step from 128 to 256 shows the worst runtime cut of 57.2% at 115 W and 57.3%.

A noticeable observation is the scaling behavior is nearly equal, with differences of less than 1.5%. This also means that the relative performance loss observed on a single node is still valid for runs on multiple nodes. The difference in strong scaling behavior under a power bound for AMG2013 and CoMD also show that this is not a general statement but depends on the application.

When comparing the absolute values, the runtime decrease at 50 W moves from 451.65 s at 8 nodes to 19.57 s at 256 nodes. The ideal value at 256 nodes is 14.11 seconds. At 115 W the runtime drops from 318.91 s to 13.05 s, with an ideal value of 9.97 seconds, assuming perfect strong scaling.

5.4 Node Performance Variation

When studying the single-node behavior, performance variation can be seen as discussed in the introduction of Chapter 5. This effect can again be observed when comparing multi-node measurements. The ordering and clustering of the nodes helped when comparing this node performance variation. As discussed in Section 4.4 the experiments are replicated to perform measurements on the whole partition requested for the test, consisting of 256 nodes. The performance variation of the multi-node runs executed on the different nodes reflects the single-node variation results.

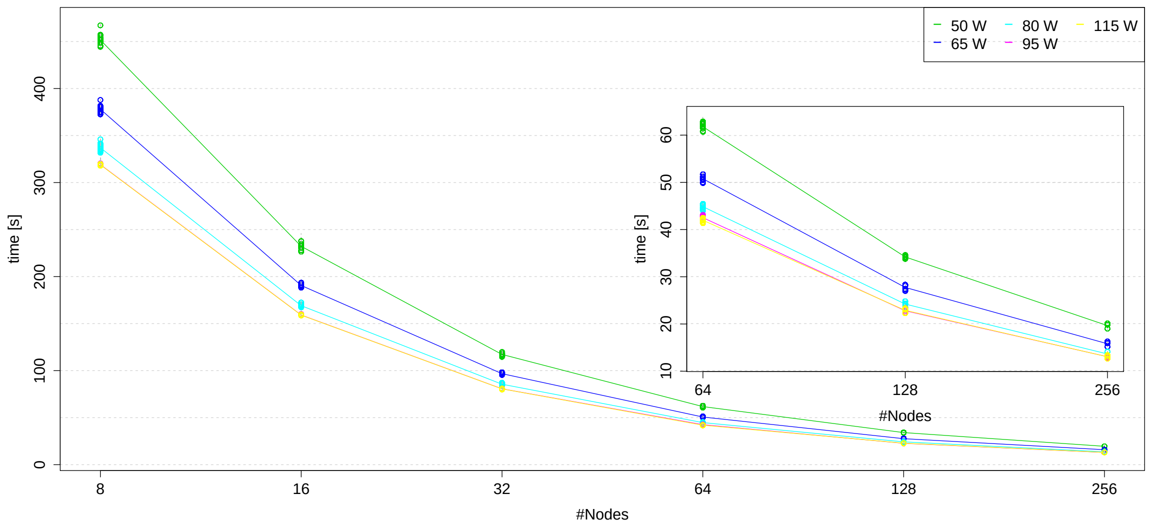


Figure 5.8: CoMD strong scaling study. Average performance over five measurements. Each measurement is replicated to utilize all 256 nodes. Detailed view for 64, 128 and 256 nodes.

All proxy applications show similar multi-node scaling variance. In this section the observed variation is discussed by the example of AMG2013, as seen in Figure 5.9. For illustrative purposes only th 50 W result is shown in the plot. This power cap shows the variation most clearly and distinguishable. The effect however is present in all measurements with power caps below the maximum power consumption of the application.

In Figure 5.9 the node IDs are ordered according to the 50 W single-node performance. The multi-node runs are clustered according to performance as introduced in Section 4.3. The arrangement of the single nodes thus follow a strict order.

The interesting observation here is, that the multi-node runs, where fast nodes are clustered with fast nodes and slow nodes with slow ones, follow a similar order when looking at the performance. The fast multi-node runs are located on the left side and performance degrades for runs located to the right. Because of noise in the interconnect and the fact that the nodes are spread across the machine this order is not strict anymore.

The nodes of the fastest two single-node measurements used for a two node measurement results in the fastest two node measurements. This seems obvious, however, without power restrictions this effect is not noticeable in terms of performance. This general observation of combining fast nodes to a multi-node run resulting in a fast multi-node run is also true for slow nodes resulting in a slow execution.

With this observation a second measurement was conducted with a different node order. For both measurements the exact same set of nodes is used. In Figure 5.10 the nodes are ordered by alternating performance according to the 50 W single-node performance of Figure 5.9. The order for this scaling study is $\{n, m, n+1, m-1, ..., n+m/2, m/2\}$, where n has the best performance and m the worst performance, according to the 50 W single-node performance of the ordered run. In other words the first node of Figure 5.10 has the best performance, the second node has the worst performance, the third node has the second best performance, the fourth node the second worst performance, etc. The order is not strict since this second measurement is ordered according to the previous measurement of Figure 5.9, using the same node set to achieve better comparability.

When performing multi-node measurements using this order for selecting nodes it can be observed that slow nodes hinder fast nodes. The performance of the multi-node runs in Figure 5.10 appear to have a reverse order of the previous results of Figure 5.9. The multi-node runtimes decrease from left to right. Again this decrease is a trend and not strict due to noise. This directly implies that the runtime of a multi-node application is always restricted by the slowest node. With this observation the sets of nodes participating in a multi-node run should have similar performance in a power restricted environment.

Without power restriction there is no or a negligibly low performance variation, as seen in the introduction of

Chapter 5 and thus this is of no real concern. In a power restricted environment this waste of performance can be large when performing measurements on nodes with large performance difference. As mentioned earlier this effect could be observed in any test with a power cap lower than the average power consumption of the proxy application. For large scale runs the likelihood of having large variation increases with the number of nodes used for the program execution. This variation is at a maximum when using the whole HPC system.

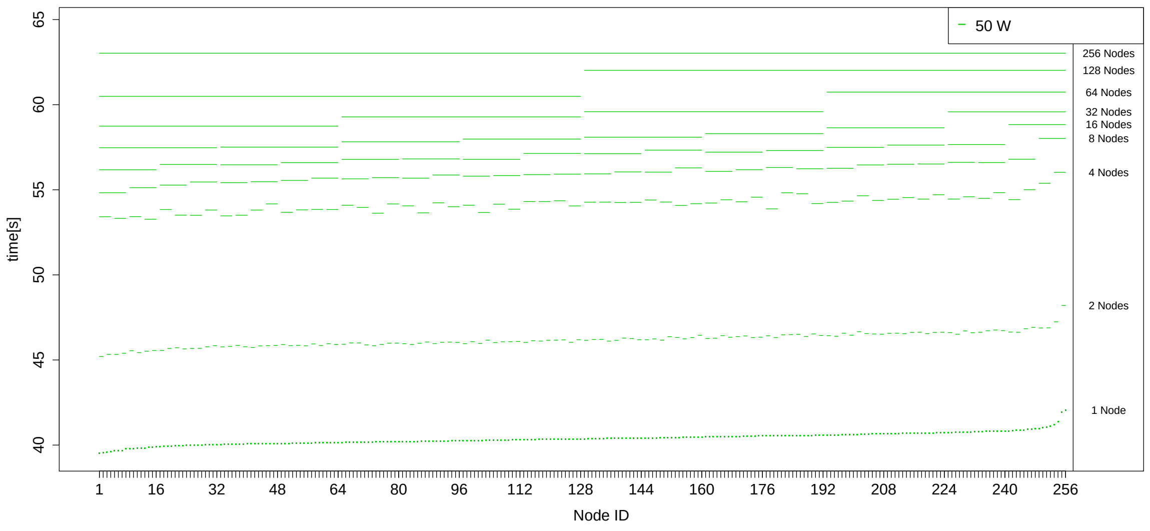


Figure 5.9: AMG2013 weak scaling study, ordered by 50 W single-node performance. Each node is participating in one measurement for each node count. Lines indicate multi-node measurements (see index right). Values are averaged over five measurements.

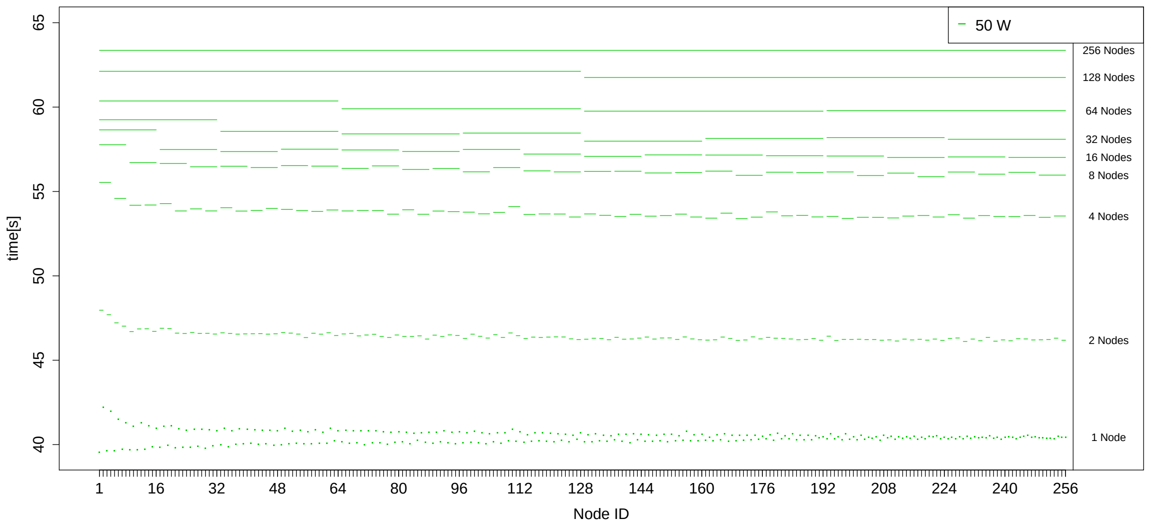


Figure 5.10: Second AMG2013 weak scaling study, ordered by alternating slow and fast nodes of 50 W single-node performance of Figure 5.9. Each node is participating in one measurement for each node count. Lines indicate multi-node measurements (see index right). Values are averaged over five measurements.

6 The Impact of Power and Using Power to Unify Performance Variability

The GREMLIN framework gives the opportunity to study performance of an emulated future HPC system. This is achieved by introducing artificial performance restrictions to mimic the performance bottlenecks of this anticipated machine. With the focus of introducing power restrictions to the HPC system cab at LLNL for scalability studies this work tries to help understand the performance tradeoff of operating at lower power settings.

When not applying a power bound the performance variability may not be observable as time to completion. The manufacturing variability is expressed as varying power consumption for each CPU while achieving comparable performance. When looking at HPC installations and their inability to consume all power procured for the system, as discussed in Chapter 1, a way to control power consumption and maximum power consumption under restricted conditions may give the ability to operate HPC systems more power efficient. This Chapter discusses how to use power bounds to achieve more power efficient system operations.

The weak scaling results in Section 5.2 have two obvious results: First, the strongest impact of running at a lower power bound is the lower CPU performance. This difference depends on the power behavior of each individual CPU and is application dependent. Second, the performance degradation of running under a power bound at larger scales is not deteriorating much faster when running under a power bound, compared to running without power restriction. However, the difference of running unbounded to running under a power bound is noticeable.

Using these results users can optimize the number of resources used for achieving a specific runtime. For a specific application, given a problem size, the independent variables number of nodes, number of cores and power bound can be optimized for the dependent variable runtime. This process requires fine tuning to use the available resources with maximum efficiency while optimizing runtime. Constructing a functional model may not be easy, because of the performance variation observed in the individual CPUs. In addition to that models are application and problem size specific. Finding a general model may be hard or impractical. A first step is a single node power optimization since scaling up does not impact power characteristics in a severe way, as seen above.

Regarding strong scaling behavior Figures 5.7 and 5.8 show an interesting behavior. The scaling behavior, similar to weak scaling, is not impacted by the power bound itself in a severe way. Under the assumption that the power consumption of other components can be neglected, running the same problem size at 50 W power using double the processors is faster and more power saving than operating the original processor count without a power bound. The assumption made is very strong, thus the impact and power consumption of components other than the CPU has to be studied.

In addition to studying the power consumption and behavior of other components it is also interesting how their power consumption can be influenced. Applying power bounds to the memory subsystem and interconnects is a first approach. For studies like these artificial benchmarks stressing CPU, Memory and Interconnect exclusively should be studied first. As of the knowledge of the author, current interconnect technology does not support different power settings. For the memory subsystem power management exists via DVFS [HFG$^+$ 11]. Howard et al. measured memory power in servers at an average of 19% [HFG$^+$ 11]. Altering memory power settings is not included in the GREMLIN framework, yet, even though memory power is a big contributor for system power consumption.

Running a system at lower power gives the ability to install and operate more components than the amount able to run at full power. The notion of overprovisioning reflects this idea [FeFr 04, PLR$^+$ 13]. This work and the strong scaling studies are a step towards showing the feasibility of overprovisioning. Overprovisioning again includes an optimization problem. Open questions regarding overprovisioning are minimal power consumption of idle and low power states and how many nodes can be added and powered up in low energy states. For full system power adaption, studying the impact and correlation of system components' power behavior plays an important role, too.

The performance variability of single node runs is well studied and known [RAdS$^+$ 12]. The variability is introduced as a result of manufacturing variation. This work gives an insight of multi node power behavior. An interesting insight here is, that slow and poorly performing nodes impact time to completion. Figures 5.9 and 5.10 show the variability using the same settings with different node counts, in a strong scaling study. Multi node runs that include nodes with low performance always perform worse than runs excluding low performing nodes. With larger runs slow nodes cannot be excluded because of a finite number of nodes available in the system. A technique to homogenize the execution times of the participating nodes is power balancing. The approach to power balancing is as follows: When operating under a power bound, the faster nodes are provided less power to amortize runtime. This extra power saved, can be either used to speed up the slow nodes, given that the node can consume the extra power, or to power additional nodes using the power surplus. These power balancing considerations include steps to analysis the optimum of power redistribution versus adding new nodes.

Performance variability is observed to be higher using low power bounds. The feasibility of this can be studied in a power bounded environment with low power settings.

With the wide popularity of accelerators power management regarding these kinds of resources is getting of interest, too. NVIDIA's CUDA supports power management and the option to set power limits on supported devices of from the Kepler family [CUDA 12]. As of the knowledge of the author, Intel's Xeon Phi accelerators do not support the RAPL features introduced in the Sandy Bridge microarchitecture, yet. With these technologies the impact on transfer rate between accelerator and host under a power bound are of particular interest.

Development and research using power bounds do not necessarily result in faster execution times, but help to understand power consumption in a HPC environment, with the goal of achieving more power efficiency. With an anticipated power consumption of $\sim$20 MW wasting power is irresponsible. Thus, understanding systems, software and application behavior is paramount.

7 Related Work

The GREMLIN framework is a first of its kind, combining different methods to emulate hardware restrictions. The hardware restrictions themselves are used in other works and different approaches, adaptations and versions of these exist. For the Memory GREMLINs alternative tools are the two works of Cache Pirating [ENBSH 11] and Bandwidth Bandits [ENBSH 12]. These approaches reach a similar goal, but do not provide a holistic framework like the GREMLIN framework does.

To influence power consumption the usage of DVFS is widely studied. Only few of these works deal with a HPC environment. Etinski et al. [ECLV 12] uses DVFS to analyze execution time reduction using the β-metric. The β-metric compares application slowdown to CPU slowdown, where $\beta == 0$ means that the execution time is insensitive to frequency scaling. The range of β is $[0, 1]$. This metric directly depends on the frequency used for the execution. The GREMLINs use RAPL to control power usage. The frequency is thus not set directly and the β-metric is not applicable when using this technique. Etinski et al. analyze the impact of frequency scaling on up to 712 processors, while analyzing the NAS parallel benchmarks. The results show a decrease in β when either increasing number of processors or increasing frequency. The frequency change from low to high frequency has a small impact on β. The change in number of processors shows a bigger impact on β, however stating that only minor impact was observed for the SP benchmark, which observes good scaling behavior.

The proxy applications studied in the work at hand are optimized for parallel execution with good scaling behavior. Thus the results of the NAS benchmark SP and the proxy applications studied here are closely related. The impact of direct power reduction using RAPL does not decrease performance severely when scaling to up to 4096 processors, or 256 nodes. The frequency is set indirectly via the running average power limit, which throttles the CPU frequency, if the average power consumption surpasses the set limit. Both approaches show a similar result for the applications showing good scaling behavior since the power clamp and frequency reduction respectively impacts performance while in the compute kernel. The impact on the communication phase is only minor.

The trade-off analysis used for the work of Etinski et al. show that frequency scaling is not a guaranteed way to save energy. This is the case when CPU power reduction of over 100% is required, as described by the group. When comparing frequency needs to power needs of applications, the power value can be directly read from the power meters. Understanding frequency needs while fully utilizing all resources is an indirection to this approach and harder from a programmers perspective. Application requirements are not obvious and thus DVFS is not used in the work at hand. As shown in Chapter 3, only a custom, artificial workload can achieve full processor load, which consumes all power provided to the CPU. In other words, the energy that is not used by the application can always be saved without impacting performance, under the condition that this power requirement is known. Using RAPL this maximum power consumption of the application could theoretically be set as an upper limit.

Rountree et al. [RAdS$^+$ 12] explores the possibilities of RAPL and shows the impact on the NAS Parallel Benchmark MG via experimental results. The arguments given for using RAPL over DVFS impacted the choice of technology for the GREMLIN framework. His team studied the processor performance using different power bounds and is the first to study and apply this technique in a high-performance computing domain. The work characterizes a large selection of the same processors, pointing out the significance of processor variation within a specific CPU model. The work does not take multi node measurements into account, to get a clear picture of the impact of power bounds on individual processors. All measurements are performed using the NAS Parallel Benchmarks with problem size C and 8 MPI processes.

Scaling and prediction models exist for some of the proxy applications studied, like AMG2013 and NEKBONE. Rosas et al. developed a prediction model using small scale measurements to extrapolate efficiency to large core counts [RJL 14]. The prediction is studied and verified using analytical models, and architectural simulation. To verify the results traces using large core counts are used. The approach used is analytic modeling and architectural simulation. These results do not focus on power consumption of CPUs and have a focus on pure scaling prediction using current hardware. An extension is provided to cope with increasing noise in the system. The GREMLIN framework chooses the approach of emulation. In combination, the analytical model, the simulation and the emulation can provide a very precise prediction of future systems. For exact prognosis the analysis of all resources and their respective behavior is necessary.

The key contribution of this work to the aforementioned are two distinguishable aspects:

- The GREMLIN framework provides an easily extensible emulation tool for hardware restrictions on current high performance computing systems.
- The Framework is applied to study scaling behavior in a power restricted environment.

The results show different aspects and potential use cases of power limits. The main goal is reducing wasted power on large scale systems. This work gives an initial study of how power limits affect application behavior at scale and provides a framework to explore and exploit resource usage. The framework generalizes this and provides the opportunity of extension to different resources and study combined resource restrictions in a unified fashion.

8 Conclusion

The goal of the GREMLIN framework, of providing a holistic emulation tool for studying future high performance computing resources, is a very ambitious goal. The work at hand gives an introduction to the first release of the tool and shows its applicability in a HPC environment. The results gained are insights in power behavior and application performance at scale. The current state of the framework provides a easily extensible tool for all users. This gives the possibility to understand system and application behavior for resource restrictions already implemented as a GREMLIN and other resources of interest. The attempt to make the tool as general and useful for developers in different fields is one main aspect.

The tool is in the direct moral of co-design and supports mutual problem solving in system, software and application design. The GREMLIN framework tries to cover a broad spectrum of applications in HPC, targeting C/C++ and Fortran using MPI. The lack of support for OpenMP and accelerators, GPGPU and co-processors restricts the emulation to hardware architectures similar to the host system of the emulation. With the underlying tools using PMPI extensions for the above mentioned technologies are possible as long as MPI is combined in a hybrid MPI+X approach.

The scaling study conducted shows the impact of power limitations to multi node program executions. The impact of power limits has a direct impact on single node performance, however only degrades scaled behavior in a minor way. One major result of this work is that the application of power bounds can be used on large scale installations without loosing the ability to run highly efficient parallel applications. This gives way to think about how these limits can be used for a more energy efficient systems operation. The more important goal for HPC is combining energy efficiency with maximizing resource utilization. This implies, for a given system, maximize the performance output, compared to minimizing the power spent to achieve a set performance. These goals are similar but system costs are too high to have underutilized resources.

With the GREMLIN framework co-design using the most efficient systems and understanding the application behavior at scale can be studied. This helps to understand application behavior on next generation HPC resources and can help developers identify caveats and explore capabilities to achieve high performance independent of the underlying hardware. Emulating these systems requires adapting to the most efficient paradigms currently in use. This indicates that in a rapidly changing environment relying on standards and evaluating new possibilities of innovation is paramount. In regards to resource usage the proposition of overprovisioning is one of these possibilities. A power balancing approach to save power without giving up runtime has to be evaluated to implement overprovisioned systems without loss of efficiency. This gives way to a first study of actual overprovisioning of compute resources in a non artificial environment.

Bibliography

[AMG 13] CENTER FOR APPLIED SCIENTIFIC COMPUTING (CASC) AT LLNL: *AMG2013*, 2013. `https://codesign.llnl.gov/amg2013.php` (accessed on 2015-01-03).

[BGSY 11] BAKER, A.H., T. GAMBLIN, M. SCHULZ and U.M. YANG: *Challenges of scaling algebraic multigrid across modern multicore architectures*. IEEE, 2011. In IPDPS '11.

[CESAR 11] ARGONNE NATIONAL LABORATORY: *CESAR: Center for Exascale Simulation of Advanced Reactors an Office of Science Co-design Center*, 2011. `https://cesar.mcs.anl.gov/` (accessed on 2014-12-20).

[CESAR 13] ARGONNE NATIONAL LABORATORY: *The CESAR Proxy-apps*, 2013. `https://cesar.mcs.anl.gov/content/software` (accessed on 2015-01-03).

[CGBr 14] CASAS-GUIX, M. and G. BRONEVETSKY: *Active Measurement of Memory Resource Consumption*. IEEE, 2014. In IPDPS '14.

[CMake14] KITWARE: *CMake*, 2014. `www.cmake.org` (accessed on 2014-12-22).

[CoMD 13] EXMATEX TEAM AT LOS ALAMOS NATIONAL LABORATORY AND LAWRENCE LIVERMORE NATIONAL LABORATORY: *CoMD*, 2013. `http://www.exmatex.org/comd.html` (accessed on 2015-01-03).

[Corp 99] CORPORAAL, H.: *TTAs: Missing the ILP Complexity Wall.* J. Syst. Archit., 45(12-13):949–973. Elsevier, 1999.

[CUDA 12] NVIDIA CORPORATION: *nvidia-smi - NVIDIA System Management Interface program*, 2012. `http://developer.download.nvidia.com/compute/cuda/6_0/rel/gdk/nvidia-smi.331.38.pdf` (accessed on 2015-02-05).

[DGH$^+$ 10] DAVID, H., E. GORBATOV, U.R. HANEBUTTE, R. KHANNA and C. LE: *RAPL: Memory Power Estimation and Capping*. ACM, 2010. In ISLPED '10.

[DOE 09] *Scientific Grand Challenges: Cross-Cutting Technologies for Computing at the Exascale Workshop*, San Diego, CA, 2009. Department of Energy Office of Science.

[DOE 10] *ASCAC Subcommittee Report: The Opportunities and Challenges of Exascale Computing*. Department of Energy Office of Science, Fall 2010.

[DOE 14] *The Top Ten Exascale Research Challenges*. Department of Energy Office of Science, February 2014.

[ECLV 12] ETINSKI, M., J. CORBALAN, J. LABARTA and M. VALERO: *Understanding the future of energy-performance trade-off via DVFS in HPC environments*. Journal of Parallel and Distributed Computing, 72(4):579 – 590. Academic Press, 2012.

[ENBSH 11] EKLOV, D., N. NIKOLERIS, D. BLACK-SCHAFFER and E. HAGERSTEN: *Cache Pirating: Measuring the Curse of the Shared Cache*. IEEE, 2011. In ICPP '11.

[ENBSH 12] EKLOV, D., N. NIKOLERIS, D. BLACK-SCHAFFER and E. HAGERSTEN: *Bandwidth Bandit: Quantitative Characterization of Memory Contention*. ACM, 2012. In PACT '12.

[EOS 09] ENGELMANN, C, H.H. ONG and S.L. SCOTT: *The Case for Modular Redundancy in Large-Scale High Performance Computing Systems*. ACTA Press, 2009. In PDCN '09.

[ExMatEx 14a] ExMatEx Team at Los Alamos National Laboratory and Lawrence Livermore National Laboratory: *ExMatEx – DoE Exascale Co-Design Center for Materials in Extreme Environments*, 2014. www.exmatex.org (Accessed on 2014-12-19).

[ExMatEx 14b] ExMatEx Team at Los Alamos National Laboratory and Lawrence Livermore National Laboratory: *ExMatEx - Proxy Apps Overview*, 2014. http://www.exmatex.org/proxy-over.html (Accessed on 2015-01-01).

[FaMY 02] Falgout, R.D. and U. Meier Yang: *hypre: A Library of High Performance Preconditioners*. Springer, 2002. In ICCS '02.

[FeFr 04] Femal, M.E. and V.W. Freeh: *Safe Overprovisioning: Using Power Limits to Increase Aggregate Throughput*. Springer, 2004. In PACS '04.

[FiHe 13a] Fischer, P.F. and K. Heisey: *NEKBONE: Thermal Hydraulics mini-application*, 2013. https://cesar.mcs.anl.gov/sites/cesar.mcs.anl.gov/files/nekbone_3.1_readme.pdf (Accessed on 2015-01-03).

[FiHe 13b] Fischer, P.F. and K. Heisey: *Proxy-Apps for Thermal Hydraulics*, 2013. https://cesar.mcs.anl.gov/content/software/thermal_hydraulics (Accesed on 2015-01-12).

[FLK 08] Fischer, P.F., J.W. Lottes and S.G. Kerkemeier: *nek5000 Web page*, 2008. http://nek5000.mcs.anl.gov (Accessed on 2014-12-20).

[G500 14] Green500: *The Green500 List - November 2014*, 2014. http://www.green500.org/lists/green201411 (Accessed on 2014-12-05).

[GBB+ 12] Graziani, F.R., V.S. Batista, L.X. Benedict, J.I. Castor, H. Chen, S.N. Chen, C.A. Fichtl, J.N. Glosli, P.E. Grabowski, A.T. Graf and others: *Large-scale molecular dynamics simulations of dense plasmas: The Cimarron Project*. High Energy Density Physics, 8(1):105–131. Elsevier, 2012.

[GREM 14a] Schulz, M.: *GitHub scalability-llnl/Gremlins*. https://github.com/scalability-llnl/Gremlins (Accessed on 2014-12-18).

[GREM 14b] Schulz, M.: *GREMLINs: Emulating Exascale Conditions on Today's Platforms*. https:computation.llnl.gov/project/gremlins/ (Accessed on 2014-12-18).

[HFG+ 11] Howard, D., C. Fallin, E. Gorbatov, U.R. Hanebutte and O. Mutlu: *Memory Power Management via Dynamic Voltage/Frequency Scaling*. ACM, 2011. In ICAC '11.

[HNS 13] Heroux, M., R. Neely and S. Swamiarayan: *ASC Co-design Proxy App Strategy*, January 2013. https://codesign.llnl.gov/pdfs/proxyapps_20130106.pdf (Accessed on 2014-12-31).

[Intel 09] Intel: *Intel 64 and IA-32 Architectures Software Develper's Manual, Volume 3B: System Programming Guide*. Intel, 2009. pp. 484ff.

[Intel 12] Intel Corporation: *Intel® Xeon® Processor E5-2670 Specifications*, 2012. http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI (Accessed on 2015-01-14).

[KGL 06] Kadau, K., T.C. Germann and P.S. Lomdahl: *Molecular Dynamics Comes of Age: 320 Billion Atom Simulation on BlueGene/L*. International Journal of Modern Physics C, 17(12):1755–1761. World Scientific, 2006.

[LC 10] Livermore Computing (LC), LLNL: *Linux @ Livermore*, 2010. https://computing.llnl.gov/linux/projects.html (Accessed on 2015-01-14).

[LC 14] LIVERMORE COMPUTING (LC), LLNL: *Open Computing Facility-OCF, Resource overview: Cab*, October 2014. `https://computing.llnl.gov/?set=resources&page=OCF_resources#cab` (Accessed on 2015-01-14).

[libra 08] GAMBLIN, TODD: *GitHub tgamblin/libra*, 2008. `https://github.com/tgamblin/libra` (Accessed on 2014-12-19).

[LRZ 13] LRZ: *SuperMUC Best Practice Guide v0.2*, January 2013. `http://www.lrz.de/services/compute/supermuc/manuals/Best-Practice-Guide-SuperMUC/Best-Practice-Guide-SuperMUC-html/#id-1.5.13` (Accessed on 2014-12-22).

[McPh 13] MCPHERSON, A.: *exmatex/CoMD*, 2013. `https://github.com/exmatex/CoMD` (Accessed on 2015-01-16).

[MPI 14] MPI FORUM: *Message Passing Interface Forum*, May 2014. `www.mpi-forum.org` (Accessed on 2014-21-20).

[OSU 14] THE OHIO STATE UNIVERSITY: *MVAPICH2*, 2014. `mvapich.cse.ohio-state.edu` (Accessed on 2015-01-16).

[PLR+ 13] PATKI, T., D.K. LOWENTHAL, B. ROUNTREE, M. SCHULZ and B.R. DE SUPINSKI: *Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing*. ACM, 2013. In ICS '13.

[PnMPI 09] SCHULZ, M.: *GitHub scalability-llnl/PnMPI*, 2009. `https://github.com/scalability-llnl/PnMPI` (Accessed on 2014-12-18).

[RAdS+ 12] ROUNTREE, B., 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*. IEEE, 2012. In IPDPSW '12.

[RJL 14] ROSAS, C., GIMÉNEZ J. and J. LABARTA: *Scalability prediction for fundamental performance factors*. Supercomputing frontiers and innovations, 1(2). Publishing Center of South Ural State University, 2014.

[Roun 14] ROUNTREE, B.: *Tools and Components for Memory and Power Profiling*, July 2013. Petaflops Tools Workshop.

[Saun 14] SAUNDERS, W.: *Visualizing change: Top500, Green500, and Exascalar*, November 2014. `http://rpubs.com/ww44ss/Nov14ExascalarDelta` (Accessed on 2014-12-06).

[SBB+ 13] SCHULZ, M., J. BELAK, A. BHATELE, P.-T. BREMER, G. BRONEVETSKY, M. CASAS, T. GAMBLIN, K. ISAACS, I. LAGUNA, J. LEVINE, V. PASCUCCI, D. RICHARDS and B. ROUNTREE: *Performance Analysis Techniques for the Exascale Co-Design Process*. September 2013. In PARCO '13.

[ScdS 07] SCHULZ, M. and B.R. DE SUPINSKI: *PNMPI Tools: A Whole Lot Greater Than the Sum of Their Parts*. ACM, 2007. In SC '07.

[SCL 11] STANLEY-MARBELL, P., V. CAPARROS CABEZAS and R. LUIJTEN: *Pinned to the Walls: Impact of Packaging and Application Properties on the Memory and Power Walls*. IEEE, 2011. In ISLPED '11.

[SGP+ 06] STREITZ, F.H., J.N. GLOSLI, M.V. PATEL, B. CHAN, R.K. YATES, B.R. DE SUPINSKI, J. SEXTON and J.A. GUNNELS: *Simulating solidification in metals at high pressure: The drive to petascale computing*. Journal of Physics: Conference Series, 46(1):254–267. IOP Publishing, 2006.

[ShRo 14a] SHOGA, KATHLEEN and BARRY ROUNTREE: *GitHub scalability-llnl/libmsr*, 2014. `https://github.com/scalability-llnl/libmsr` (Accessed on 2014-12-23).

[ShRo 14b] SHOGA, KATHLEEN and BARRY ROUNTREE: *GitHub scalability-llnl/msr-safe*, 2014. `https://github.com/scalability-llnl/msr-safe` (Accessed on 2014-12-23).

[ShWi 65] SHAPIRO, S.S. and WILK M.B.: *An Analysis of Variance Test for Normality (Complete Samples)*. Biometrika, 52(3/4):591–611. Biometrika Trust, 1965.

[SSC 14] SOUMYA, J., A. SHARMA and S. CHATTOPADHYAY: *Multi-Application Network-on-Chip Design Using Global Mapping and Local Reconfiguration*. ACM Trans. Reconfigurable Technol. Syst., 7(2):7:1–7:24. ACM, 2014.

[Stüb 99] STÜBEN, K.: *Algebraic Multigrid (AMG): An Introduction with Applications*, November 1999. German National Research Center for Information Technology (GMD), GMD-Report 70, November 1999.

[SWAB 14] SHOUKOURIAN, H., T. WILDE, A. AUWETER and A. BODE: *Monitoring Power Data: A first step towards a unified energy efficiency evaluation toolset for HPC data centers*. Environmental Modelling and Software, 56:13–26. Elsevier, 2014.

[T500 14] TOP500.ORG: *TOP500 LIST - NOVEMBER 2014*, November 2014. `http://top500.org/list/2014/11/` (Accessed on 2014-12-08).

[TSI 14] TOMITA, H., M. SATO and Y. ISHIKAWA: *Report from Japan*, February 2014. `http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/Talk12-tomita-sato-ishikawa.pdf` (Accessed on 2014-12-11).

[VEMY 00] VAN EMDEN, H. and U. MEIER YANG: *BoomerAMG: a Parallel Algebraic Multigrid Solver and Preconditioner*. Applied Numerical Mathematics, 41:155–177. Elsevier, 2000.

[Ward 14] WARD, G.: *CORAL Benchmark Codes*, 2014. `https://asc.llnl.gov/CORAL-benchmarks/` (Accessed on 2015-01-03).

[WuMc 95] WULF, W.A. and S.A. MCKEE: *Hitting the Memory Wall: Implications of the Obvious*. SIGARCH Comput. Archit. News, 23(1):20–24. ACM, 1995.