The presented work was submitted to the
Chair for High Performance Computing (Computer Science 12), IT Center.

Investigation of Node-Level Performance Variation on the CLAIX2016 and CLAIX2018 Cluster of the RWTH Aachen University

Untersuchung der Node-Level Performance-variation auf dem CLAIX2016 und CLAIX2018 Cluster der RWTH Aachen University

Bachelor Thesis

Lukas Hanneken
Student ID: 367658

Aachen, September 26, 2019

First Examiner:	Prof. Dr. rer. nat. Matthias S. Müller (')
Second Examiner:	Dr. rer. nat. Stefan Lankes (*)
Supervisor:	Daniel Schürhoff, M. Sc. (')

(') Chair for High Performance Computing, RWTH Aachen University
IT Center, RWTH Aachen University
(*) Institute for Automation of Complex Power Systems
E.ON Energy Research Center, RWTH Aachen University

communicated by Prof. Dr. rer. nat. Matthias S. Müller

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden.

Aachen, September 26, 2019

Lukas Hanneken

Abstract

Modern parallel computers are consistently achieving more computing power in recent years. The size of the hardware components decreases continuously and, at the same time, parallel program codes become more complex. As a result, stability and reproducibility and consequently performance variation in particular, are becoming more and more important. While other approaches are already investigating load imbalances at the software-level or architectures at the microprocessor-level for example, an important factor that is present in every measurement on every high-performance computer has not been investigated so far: The node-level performance variation. Therefore, this bachelor thesis first examines the performance of the individual computing nodes of the two clusters at RWTH Aachen University and how large the variations within these measurements are. With the three well-known and widely accepted benchmarks Linpack, HPCG, and STREAM, individual aspects of both systems are strained and tested so that many important components are examined in single-node measurements. After statistically analyzing the collected data, the temporal variations of the individual nodes as well as the spatial variations at cluster-level are quantified.

In the second part of this thesis, the causes of the found variability are systematically investigated. Besides the separate long-term investigation of nodes identified as outliers in repeated measurements over a period of one and a half months, various other factors are systematically examined. As possible causes, the operating system, the position of the nodes within the cluster and its racks, unfinished zombie and orphaned processes, and impairment of neighboring nodes in the form of heat leaks are considered. Additionally, the multi-threaded STREAM benchmark is analyzed with a different amount of OMP threads.

Since the development of performance and its variation over a longer period of time is an exciting research topic, which contributes in particular to our understanding of variations and complex interrelationships within the parallel computer, a toolkit for automated measurement and data acquisition is developed within the scope of this bachelor thesis. This enables the integration of benchmarks into the day-to-day operation on both clusters to be fully automated evaluated with monthly reports. The data obtained in this way form a basis for future research and allow a better insight into the run-to-run variation also with regard to the implementation of state-of-the-art solutions and their evaluation.

Keywords: Performance, Performance Variability, Node-Level, Benchmark Testing, Parallel Computing, High Performance Computing, HPC

Kurzfassung

Moderne Parallelrechner erreichen in den letzten Jahren beständig mehr Rechenleistung. Dabei nimmt die Größe der Hardwarebestandteile durchgehend ab und gleichzeitig werden die parallelen Programmcodes immer komplexer. Dadurch gewinnt die Stabilität und Reproduzierbarkeit und folglich insbesondere auch die Performancevariation immer mehr an Bedeutung. Während andere Ansätze bereits beispielsweise Ungleichgewichte der Lastverteilung auf der Software-Ebene oder die Architekturen auf Mikroprozessor-Ebene untersuchen, wurde ein weitere wichtige Ebene eines Hochleistungsrechner bisher nicht weiter untersucht: Die Node-Level Performancevariation. Daher untersucht diese Bachelorarbeit zunächst, welche Performance die einzelnen Rechenknoten der zwei Cluster der RWTH Aachen University erreichen und wie groß die Variationen innerhalb dieser Messungen sind. Mit den drei bekannten und weit akzeptierten Benchmarks Linpack, HPCG, und STREAM werden jeweils einzelne Aspekte beider Systeme betrachtet und getestet, so dass insgesamt viele wichtige Konponente in single-node Messungen untersucht werden. Durch die statistische Analyse der gesammelten Daten kann die zeitliche Variationen der Knoten sowie die räumlichen Variation auf Cluster-Level quantifiziert werden.

Im zweiten Teil der Arbeit wird systematisch den Ursachen der aufgefundenen Variabilität nachgegangen. Neben der gesonderten Betrachtung von als Ausreißer identifizierten Knoten über einen Zeitraum von anderthalb Monaten werden systematisch weitere Faktoren mit in die Überlegungen hineingezogen. So werden das Betriebssystem, die Position der Knoten innerhalb des Clusters und seiner Racks, nicht beendete verwaiste oder Zombie-Prozesse sowie Einflüsse benachbarter Rechenknoten in der Form von deren Abwärme betrachtet. Zusätzlich wird die Speicherbandbreite mit verschiedenen Anzahlen an OMP Threads untersucht.

Da die Entwicklung der Performance und ihrer Variation über einen längeren Zeitraum ein spannendes Forschungsthema ist, welches insbesondere zu unserem Verständnis der Variation und komplexeren Zusammenhängen innerhalb des Parallelrechners beiträgt, wird im Rahmen dieser Bachelorarbeit ein Toolkit zur automatisierten Messung und Datenerfassung entwickelt. Dieses ermöglicht es, automatisch Benchmarks in den Normalbetrieb beider Cluster zu integrieren, diese vollautomatisiert auszuwerten und in monatliche Berichte zusammenzufassen. Die so gewonnenen Daten können eine Grundlage für zukünftige Forschung bilden und können einen besseren Einblick in die run-to-run Variation auch im Hinblick auf die Implementierung von state-of-the-art Lösungen und deren Evaluation ermöglichen.

Stichwörter: Performance, Performancevariation, Node-Level, Benchmark Tests, Parallelrechner, High Performance Computing, HPC

Contents

List of Figures xi

List of Tables xiii

1. Introduction 1

2. Background and Related Work 5
2.1. Performance . . . 5
2.2. Performance Variation . . . 6
2.2.1. Spatial Variation . . . 7
2.2.2. Temporal Variation . . . 10
2.3. Thesis Goal . . . 12

3. Measuring Performance and Performance Variation 13
3.1. Methodology . . . 13
3.1.1. Methodology of Experiments . . . 14
3.1.2. Statistical Methodology of Evaluation . . . 17
3.2. Results . . . 18
3.2.1. Impact of Operating System Noise . . . 18
3.2.2. Computing Power . . . 21
3.2.3. Memory Bandwidth . . . 29
3.3. Summary . . . 30

4. Identifying Causes of Performance Variation 33
4.1. Studies on CLAIX2016 . . . 33
4.2. Investigations on CLAIX2018 . . . 36
4.2.1. Examining Significantly Low Performing Nodes with HPCG . 36
4.2.2. Observing Outliers with Linpack . . . 39

5. Discussion 41

6. Toolkit for Automatic Measurements and Monthly Reports 47
6.1. Kernels . . . 47
6.2. Mechanics of Usage . . . 48
6.2.1. Mechanics of Building . . . 48
6.2.2. Mechanics of Running and Evaluating . . . 49
6.2.3. Remarks and Recommendations . . . 49

7. Conclusion and Future Work **51**

A. Multi-Threaded STREAM with Different Number of OMP Threads **55**

B. Performance-Placement Heat Maps **57**

Bibliography **61**

List of Figures

3.1. Illustration of Skewness and Kurtosis 17
3.2. Operating System Noise on CLAIX2016 19
3.3. Operating System Noise on Selected Nodes of CLAIX2016 19
3.4. Comparing OS Noise per Core on CLAIX2016 20
3.5. Warm-Up Period of a HPCG Run on CLAIX2018 22
3.6. Node Cooling on CLAIX2016 23
3.7. Linpack Performance on CLAIX2016 25
3.8. Linpack Performance on CLAIX2018 26
3.9. HPCG Performance on CLAIX2016 27
3.10. HPCG Performance on CLAIX2018 28
3.11. Memory Bandwidth on CLAIX2016 29
3.12. Memory Bandwidth on CLAIX2018 30

4.1. Two-Color Heat Map for HPL on CLAIX2016 35
4.2. HPCG Performance Over Longer Time Span on CLAIX2018 37
4.3. HPL Performance Over Longer Time Span on CLAIX2018 40

A.1. Memory Bandwidth with Different Numbers of OMP Threads on CLAIX2016 55
A.2. Memory Bandwidth with Different Numbers of OMP Threads on CLAIX2018 56

B.1. Performance Heat Map with HPL on CLAIX2016 57
B.2. Two-Color Heat Map for HPL on CLAIX2016 57
B.3. Performance Heat Map with HPCG on CLAIX2016 58
B.4. Performance Heat Map with STREAM on CLAIX2016 58
B.5. Performance Heat Map with HPL on CLAIX2018 59
B.6. Performance Heat Map with HPCG on CLAIX2018 59
B.7. Temporal Variation Heat Map for HPCG on CLAIX2018 59
B.8. Performance Heat Map with STREAM on CLAIX2018 60

List of Tables

3.1. Measured Values of the Three Benchmarks and their Statistical Analysis 31
3.2. Globally Considered Ranking of all Nodes Based on their Performance 32

4.1. Performance Differences and Measured Status Data of Selected CLAIX2018 Nodes 39

1. Introduction

In recent years, several trends within modern high-performance computer systems can be recognized. On the one hand, the hardware size is continuously decreasing, while on the other hand, the complexity of the software and hardware is increasing at the same time. With a yearly performance growth of approximately 85% [12], we are on our way to exascale systems that are capable of executing at least one $EFlop/s$, or a billion billion (i.e. a quintillion) calculations per second. Thus, the reliability and stability of these machines and the reproducibility of experiments executed on them are becoming more and more the focus of recent research. Therefore, performance variations should be further investigated as they impact the overall amount of work that can be done on a cluster and how reliable a measurement is, affect the development cycle for applications and might increase energy costs. Several studies show that variation is not only a major problem on all modern parallel computers [4, 7, 28, 34, 41, 46], but also that variation is increasing [1, 26, 27]. A factor influencing every measurement and application run on a highly parallel computer and that has not yet been sufficiently researched is the variation at node-level. Therefore, this thesis will investigate the node-level performance variation.

Variations in performance, which a user typically experiences in different long execution times of their program, can lead to many unwanted negative effects resulting in sub-optimal productivity on high-performance computing (HPC) systems. Since the variations in runtime always deviate downwards from the best-case runtime, the runtime variability has always negative effects on a systems overall performance, i.e. the variation is never better than the optimally achievable performance [41].

Performance variation determines, among other things, how much actual work can be done on a system. For example, consider a highly parallel job that runs simultaneously on 1,440 cores and that has multiple synchronization points within its code, which ensure that the next program section is entered by all processes at the same time. If all cores have to wait for a single minute because one single core executes the code a little slower than the others, a lot of computing time remains unused. In this example, this wastes 24 core-hours, or a whole day of computing time. Therefore, variation at node-level should not be neglected. When static load balancing is used, one slow node not only wastes a lot of computing time, but also deteriorates the overall performance of a multi-node job.

From the perspective of a user of the cluster, it is moreover desirable to get a constant performance. An important task in development is the performance tuning of an application, which is improving the runtime or overall performance by making changes to the program code. However, these modifications can only be evaluated by executing the application on the respective system and compare

the results afterwards. If significant run-to-run performance fluctuations occur now, wrong conclusions could be drawn and actual deteriorations could be identified as improvements or vice versa. In practice, a single change is therefore often evaluated by repeatedly running the program several times. This in turn is not the best use of resources, as otherwise multiple jobs could run and more users could be served, instead of running a single program repeatedly.

Most workload manager like LSF, PBS or SLURM rely on user-provided estimated runtime to schedule work. In the present of variability, a cautious user hence over estimates the walltime of a job, which causes the job scheduler to operate on poor information and might lead to a inefficient scheduling on the system. Any job that overruns the requested batch time is usually terminated and will therefore either loss its progress since the last checkpoint, or in the case the job does not have any check points, the entire run will be lost. These all contribute to the loss of user productivity.

Furthermore, the identification and elimination of performance variation provides consequently more performance "free of charge". By detecting and avoiding the causes, considerably more performance can be achieved without making any changes to or upgrading the hardware. Examples are speedups of 2.4× by mitigating OS interrupts with simultaneous multi-threading [29], 1.9× when optimizing the code to avoids prior load imbalances [13] or 28% more peak performance on a 1,736-node partition after resolving Snoope Filter conflicts [34].

In addition, political decisions and the allocation of research funds are often influenced by the peak performance achieved in the *High-Performance Linpack* (HPL) [11] benchmark and the associated placement on the TOP500 [12], a list which publishes and ranks the fastest 500 computers in the world at that time. The Chinese government, for example, is investing enormously to lead the list with its systems consisting of domestically produced processors [14]. Similarly, Apon *et al.* [3] showed a statistical correlation between appearing in the TOP500 list and the financing by the U.S. National Science Foundation as well as an increase in the number of publications. Therefore, the operators of a computer cluster intend to ensure a stable performance for its users and further to achieve the maximum obtainable performance.

Despite the efforts of modern research and state-of-the-art approaches, performance variation not only still occurs on modern parallel machines [36], but is even increasing [1, 26, 27]. The node-to-node variability is a factor that is present on nearly all multi-node parallel computers, influences conducted measurement and might slow down the whole system, when static load balancing is in use. Our investigation is therefore focused on the node-level. We want to express the run-to-run variation in performance a user is typically experiencing on a multi-node system. Although this variability of the participating nodes is a background noise that is always given and should therefore be investigated, there are no detailed studies on it. Perhaps the closest study to this is McCalpin [34]. He investigated occasionally slow single-node HPL results and identified snoop filter conflicts as a cause, which has been fixed in the benchmark's newest version. This thesis could thus be seen as an extension of these experiments to several aspects.

In Section 2 we first introduce how performance is measured. This includes the definitions of the theoretically achievable compute peak performance and the memory bandwidth. Moreover, related work dealing with performance variation and approaches preventing or mitigating variability are discussed. There have been several previous studies investigating the effects of performance variation, which consider causes at different levels of a parallel computer. These range from the architecture at microprocessor-level over the software and the operating system to the hardware with manufacturing problems or shared resources with competing accesses.

In Section 3, a systematic inventory is made of the achievable performance of our two available clusters with single-node runs. With four different benchmarks, relevant aspects of both compute clusters are strained and tested. For this purpose, the compute performance is considered with the compute-bound Linpack benchmark and *High-Performance Conjugate Gradient* (HPCG) [10, 20] benchmark, which is a memory-bound application. STREAM is used to measure the achievable memory bandwidth. Additionally, the occurrences and length of interrupts by the operation system are observed with the *Fixed Work Quantum* (FWQ) [15] benchmark. We statistically analyze the obtained data in order to quantify and express the occurring variations. With repeating measurements on the same node, we can analyze temporal run-to-run variations, use the mean value to express the spatial variation at cluster-level and identify outliers.

Section 4 investigates possible causes of the revealed variability. The structure is chronological to the order of the examinations carried out by us. Individual results are already discussed as they occur in order to lead to the next steps of the investigations and to reflect the constantly growing understanding of variations and correlations. First the reproducibility of the results is checked and then we separately look at the outliers in a long-term investigation with continuous measurements over one and a half month. As possible causes for node-level variation we consider interrupts of the operating system, non-terminated processes of other users, which still consume resources, and influences of neighboring nodes in the form of heat leaks. In addition, a measurement series with different numbers of threads is performed with STREAM. The gained insight and results are discussed in Section 5.

To carry out repeated evaluations, a toolkit for automated measurement and data acquisition is developed within the scope of this bachelor thesis. This allows us to integrate benchmarks into the normal system operation and evaluate them fully automatically with monthly generated reports. The data obtained in this way can form a basis for future research and contributes to a better insight into the run-to-run variations, also with regard to the implementation of state-of-the-art improvements and increasing age of the two parallel computers. This toolkit is introduced and explained in Section 6.

In Section 7 follows a conclusion with remarks to future work.

2. Background and Related Work

This section contains definitions and explanations of the necessary background knowledge used in this thesis. First, two different ways to define the performance of a computer are presented. The performance of a computer usually refers to its computing power and achievable peak performance, which is presented in this section. As another common performance metric we are using the memory bandwidth of a machine, when it is reading from, or storing to, its memory. Afterwards, variations in this performance are discussed. In Section 2.2, references are made to related work in research of performance variability and state-of-the-art approaches to contain and prevent variations are introduced.

2.1. Performance

In general, the *performance* of a computer is a measure of how many computing operations a machine can perform per second, and is therefore often referred to as *computing power* [19]. For scientific computing, one mainly considers floating-point data with double precision, i.e. numbers occupying 64 bits in memory. The performance at which a system can execute multiplication and addition operations on this data is measured in *floating-point operations per second* ($Flops/s$). Since more complicated arithmetic operations, such as square root or trigonometric functions, often share execution resources with multiplication and addition units and since they are executed so slowly that in practice they do not make significant contribution to overall performance, they are not considered and high-performance applications try to avoid these expensive operations as much as possible.

When all components of a CPU are operating at their maximum speed, this is referred to as *peak performance*. The theoretical peak performance can be obtained by calculating the number of floating-point operations performed in a cycle time of the machine [11]. As an example, we calculate the theoretical peak performance of a single compute node of the CLAIX2016 cluster that is introduced in Section 3.1.1. Each node has two sockets using Intel Broadwell EP (E5-2650v4) CPUs with a maximum all-core Turbo Boost frequency of 2.4 GHz, 12 cores, AVX2 registers (256 $bits$), and two fused-multiply-add (FMA) units per core. During one cycle, the register holds four doubles and the two FMA units perform two operations each. This executes in total 16 $Flops$ per cycle. For this node's peak performance P_{peak}, we therefore calculate:

$$P_{peak} = 24 \cdot \frac{16\ Flops}{1\ \text{cycle}} \cdot 2.4\ GHz = 921.60\ GFlop/s.$$

The peak performance, however, is understood as an upper bound for the actual achievable performance of a machine that will not be exceeded [11]. In practice, the actual performance often deviates significantly from the theoretically calculated peak performance. For example, this can be seen by using the *High-Performance Linpack* (HPL) [11] benchmark, which is explained in detail in Section 3.1.1. We run this performance-analyzing application on the 2-socket machine introduced above and measure on average a performance of 860.39 $GFlop/s$. Thus, our system has a reasonably high efficiency of 93.36%.

To give a statistical overview on high-performance computers, Dongarra *et al.* assemble the *TOP500* [12], a list with the 500 most powerful computer systems worldwide. The list is compiled and published twice a year since 1993 and compares and arranges the machines according to their peak performance achieved in the Linpack benchmark. In the course of time, their ranking has been extended by two more lists. Since June 2013, the fastest 500 computers are sorted according to their energy efficiency in the *GREEN500* list, which is measured in $Flop/watt$. Furthermore, the *High-Performance Conjugate Gradient* (HPCG) [10, 20] benchmark represents an additional test to simulate a more realistic real-life application, which is used since November 2017. To supplement the rankings with more information and statistics, the cluster's operators, their power consumption, the operating system in use, the installed hardware, and more is collected.

In addition to pure compute power, other properties of a computer are often measured and distinct performance metrics are compared. Another often used metric is the *memory bandwidth* [19]. This is the rate at which a processor can read data from or store to its memory. The data path to and from the last-level caches and the main memory is most frequently examined and it is quantified in $Byte/s$. A well-known benchmark to measure the bandwidth of a system is the *STREAM* [35] benchmark, whose results reflect the true capabilities of the hardware [19]. Our reference machine from before achieves an average memory bandwidth of about 92.35 $GByte/s$, with a multi-threaded STREAM using 24 OMP threads.

The consideration of the peak performance and the achievable memory bandwidth over the slowest path covers the respective limitations of the practical achievable performance of a given hardware configuration and a given application according to the *Roofline Model*.

2.2. Performance Variation

When only identical hardware is installed in a cluster and uniform software runs on all nodes, one expects to measure the same homogeneous performance throughout the whole cluster. However, it has already been shown in the past by various authors and papers [1, 4, 7, 26, 28, 34, 41, 46] that this is not the case and that significant *performance variation* occurs. From the user's point of view, this initially manifests itself in differing runtimes with apparently identical execution of their ap-

plication [46]. The causes and influences of these fluctuations in performance are very diverse, have many factors, and often lie in small details. In addition, the ever larger and more complex system architectures make it difficult to identify them and offer more sources for possible errors. There is a reason McCalpin called his work "The Detective Story", when he was investigating performance variation on the Xeon Gold and Platinum processors and Xeon Phi x200 (KNL) [32].

On a multi-user system, where multiple jobs are running on a shared memory processor, memory conflicts, system overload, and other user's priorities can be additional causes for variation in performance [28]. However, it is common on large-scale distributed memory systems for compute-intensive applications to be assigned to their own computing nodes. As a result, many of the factors that normally contribute to variation are not present on these systems, but application runtimes can still vary significantly. Performance variation is a research area of great interest that has already been intensively studied at different levels of the system stack and practical solutions have been developed. Previous work identified problems that cause variations, for example, in the program code of the application due to load imbalances [8]. The constantly occurring interrupts of the operating system and the assignment of hardware are further software-related causes [29]. At the hardware level, hardware variation in manufacturing of semiconductors [18], conflicts over shared resources, such as networks [4, 24, 40, 46], or architectures [7, 9] are proven to cause variability. In order to better classify these causes, we distinguish between spatial variation and temporal variation. However, this is not to be understood as a universally applicable classification of certain phenomena, since a network, for example, can be assigned to both categories under certain circumstances.

2.2.1. Spatial Variation

Spatial variation occurs when the performance measured with the same application at different spatial locations, e.g. different nodes or cores, fluctuates across these locations [26]. These spatial variations between nodes or cores are often due to systematically underlying errors, such as defective hardware or an inhomogeneous cooling, but can also be the result of the design of architectures and networks. Since the influences are usually not limited in time, they can be measured reproducibly and are therefore easier to investigate.

Hardware Variation With ever smaller processors, which are now manufactured at Intel in the 14 *nm* range or at Qualcomm even in the 7 *nm* range, variations in manufacturing are increasing [36]. During the production of semiconductors, several parameters, such as the gate width, device threshold voltage, channel length, and oxide thickness, vary. As a result, among other things, the maximum frequency, energy consumption, and temperature differ between chips of identical models, which in turn leads to performance variation. Not only processors are affected by variation, but also storage devices [17].

Supposedly identical memory flash devices with the same part number have shown to differ significantly in operation energy and bit error rate. Moreover, the International Technology Roadmap for Semiconductors even claims that performance variability and reliability management for design of computing hardware are problems with unknown solution in the next decade [18].

That is the reason why recent approaches, like Gupta *et al.* [18], focus their research on variation-aware software together with underdesigned and opportunistic computing machines. Under the assumption that the hardware does not behave homogeneously with constant specifications, it is continuously monitored. If, for example, changes in the maximum frequency, or failures of individual processor cores are detected, the system reacts accordingly at different levels of the software stack. They also plan to increase energy efficiency and performance in the long term and reduce the costs of a parallel computer at the same time.

Many-Core Architectures The individual cores of a multi-core processor often perform different tasks [7]. In addition to user applications, operating system processes are also executed in the background, which can lead to significant variations in the execution of programs at the core-level. In particular, the first core is commonly over-burdened and causes imbalances, as it executes a major part of the operation system's activity. For example, on a 64-core Intel Xeon Phi KNL 7230 processor significant spatial differences in runtime are measurable with a matrix multiply kernel on the first two cores compared to the remaining 62 cores. The excessive workload of OS activity and user program on certain cores can lead to delayed synchronizations and thus reduce performance with static load balancing. Chunduri *et al.* [7] successfully manage to avoid these effects by specifying core_0 as the last to start executing user applications and by starting with the core of the highest ID when assigning tasks. This core specialization mitigates OS noise and reduces the impact on the user program, which leads to less performance variations.

Similarly, Dighe *et al.* [9] develop a variation-conscious mapping of the workload to the individual cores of a CPU. On different 80-core processor with 65 nm technology, they achieve energy efficient improvements from 6% to 35% with optimal core allocation and from 5% to 10% performance improvements with dynamic thread hopping across a range of compute/communication activity workloads at the same time. Their method also solves the problems caused by hardware variations.

Network Heterogeneity The network of a supercomputer connects the individual nodes with each other and enables communication. By using the network, however, the occurring variation is almost always increased. As already mentioned, temporal and spatial variations can influence a network at the same time. Here, spatial causes, such as the design, are regarded and temporal effects are explained in Section 2.2.2.

In the course of time, versatile ideas and approaches for an optimal network have been developed. However, every design has its advantages and disadvantages and some are superior in different scenarios, i.e. there is not a single best topology [24].

Thus, different networks achieve different throughput and variation with certain communication patterns and workloads.

For example, Prisacari *et al.* [40] investigate Dragonfly networks, which, due to their two-tier topology, achieve particularly high bandwidths with random uniform all-to-all traffic. However, they sustain significant throughput losses in common exchange patterns for scientific computations, such as multi-dimensional nearest neighbor exchanges. They address this problem in a framework, which identifies possible bottlenecks that appear in the network under arbitrary workload. With application decomposing, routing, and mapping, they achieve optimal overall performance and speedups of up to 10× in some applications with Cartesian nearest neighbor exchanges.

Application-Level Imbalance Applications in HPC distribute the workload among the individual processors and their cores, which sometimes results in workload imbalances and cause individual hardware parts to be heavily loaded, while others are less loaded or even remain unused [8]. However, applications must distribute the workload evenly to achieve high scalability and the best performance on highly parallel systems. In addition, imbalances are always potential causes of variations, which should be avoided. Depending on the allocation strategy of the application, this variation can also be classified as temporal. Adams *et al.* [2], who use the Kernighan-Lin algorithm for solving the knapsack problem, show exemplarily that an even distribution of work over all processors is not a trivial task. Since load determination is far too application-specific, they claim that no general-purpose solution is possible and that their approach even relies on input from the user.

A new metric for measuring and expressing load balance by DeRose *et al.* [8] enables to identify sources of imbalances within an application, which they implement in an infrastructure that measures, identifies, and visualizes the application performance imbalances that are insightful for the user.

Programming Model The *bulk-synchronous parallel* (BSP) model [43] has become widely used in parallel applications in recent years, is still the most common programming paradigm used in petascale applications, and is targeted by exascale system architectures [26]. It has been successfully applied in the fields of scientific computing, artificial intelligence, parallel databases, Big Data, and simulations.

A BSP program code can be divided into small partial steps that can in turn be divided into three phases [6, 43]. In the *computation phase*, the data retrieved from the local memory are processed by the processors. In the *communication phase*, the processors exchange data and communicate. Then, the *global barrier synchronization* ensures that the current step has been completed by all components, and that the data exchange is guaranteed to have been performed. Processors wait until every processor has finished the current step, then every processor proceeds to the next step of the algorithm.

One of the main challenges that BSP faces is performance variability [26]. Since BSP uses the single instruction multiple data paradigm and different parallel tasks might require different amounts of time between global synchronizations, workload imbalance can arises and runtimed are determined by the slowest performing processors or tasks. For example, Kocoloski *et al.* [26] measure that in a set of NERSC petascale machines, up to 75% of the aggregated processing time across all processors can be spent waiting for global communication and synchronization. Such critical processes can be identified through Du *et al.*'s metrics and low-overhead implementation [13]. By optimizing the code based on critical imbalances, average speedups of 1.9× are achieved.

2.2.2. Temporal Variation

Variation can also occur for only a limited amount of time [26]. Some factors, such as I/O accesses, interrupts of the operating system, or adjacent nodes, can impact only a single measurement and might no longer occur in the next execution. This *temporal* variation quantifies how a single node varies over time between successive measurements and is therefore more difficult to investigate, as not all conditions are necessarily reproducible.

Snoope Filter Conflicts on Skylake Xeon Chips During acceptance tests, McCalpin [34] notices temporal variation in Linpack performance, which is not limited to a certain subset of nodes. In elaborate studies Snoop Filters are identified as the cause for the significant performance losses. Previous Intel processors use an inclusive L3 cache to track lines held in L1 and L2 caches, which is not anymore the case for Skylake Xeon chips. The new CPUs use an almost completely undocumented cache tracking functionality performed by inclusive Snoop Filters. A workaround to avoid conflicts is to switch to 1 GB huge pages, which allows McCalpin to even upgrade to a bigger problem size for HPL and achieve 28% more peak performance than before on a 1,736-node partition of the investigated cluster.

Network Contention Shared resources always offer great potential for conflicts. When considering the network again, temporal variations can also occur in addition to the spatial variation caused by the architecture mentioned in Section 2.2.1. Competing access slows down the whole system. For example, if several fragmented jobs communicating with each other are executed simultaneously, or, if a single application sends and receives simultaneously data to and from several nodes due to a complex communication behavior, the shared network can be overloaded. Wright *et al.* [46] measure in experiments on three different Teragrid HPC systems running on 256 MPI tasks performance variation of up to 25%, which can be attributed to contention for network resources. They also conclude that the variation for machines with lower performing networks is always greater than for machines with high throughput and that different variability occurs depending on the installed system.

Bhatele *et al.* [4] identify moreover the software that is responsible for the job placement on the individual nodes of a cluster as a cause for variation in applications that communicate a lot and run on multiple nodes. On two high-performance systems that distribute the jobs differently fragmented across the computer, they experiment with communication-heavy applications. When measuring the job of interest, they place other jobs around it that also communicate much and that use the network a lot. They find that fragmentation of jobs has very little impact on variation, but that adjacent jobs have a large impact on other measurements. When a job that communicates a lot is also surrounded with communication-heavy applications, they find that their message passing rates are up to 27.8% slower compared to a noise free measurement.

I/O Variability Another shared resource is an I/O file system like Lustre, Panasas, PCFS, or GPFS. They can be used simultaneously by several thousand nodes to write output or load data [30]. These file systems have significant performance losses especially under heavy load with many simultaneous or several large accesses. On parallel systems, interference typically occurs when too many processes within an application attempt to write to the same target, but also when different jobs simultaneously perform competing accesses to the same shared memory. Lofstead *et al.* [30] implement an adaptive I/O method assigning processes a writing, coordinating or sub-coordinating role for each storage target. When the coordinator process notices busy storage targets, it shifts more work to less loaded targets. They achieve I/O performance improvements for a 16,384 process run with 16 TB output per I/O of up to 4.8× compared to a non-adaptive approach.

Operating System Noise Since November 2017, 100% of the fastest 500 super computers worldwide have exclusively been using the Linux operating system or one of its distributions [12]. Linux allows developers to closely monitor applications, increase productivity and eases porting by offering a rich set of services that other operating systems do not provide [29]. However, the circumstance that there are constantly running software and system processes in the background generates overhead. These software interferences, or *noise*, are not directly controllable by the application and extend its runtime. Work by Petrini *et al.* [39] and Jones *et al.* [25] conduct that noise is an important key factor for the scalability of HPC applications. León *et al.* [29] achieve a performance increase of up to 2.4× by moving system processing off the critical path for their 16,384 tasks high-order finite elements shock hydrodynamics application, which also does not require any changes, neither to the OS, nor to the application.

2.3. Thesis Goal

In this thesis, we use an approach that, to the best of our knowledge, has not been discussed yet. While other authors and papers examine load imbalances at the software-level or architectures at the microprocessor-level, we consider the variation that a user experiences on the different nodes of a highly parallel computer while executing their (single-node) programs. With different benchmarks executed on each node, many important aspects of a modern parallel computer are considered and the performance of each node is measured individually. This allows us to characterize the temporal run-to-run variation of the nodes and the spatial variation of the whole cluster, as experienced by the user. Subsequently, attempts are made to identify possible sources of variation and recommendations are given to reduce or prevent performance variation in the future.

3. Measuring Performance and Performance Variation

First, we get an overview of the achievable performance on our two systems. On each individual compute node, the same sets of benchmarks are executed, each of which is designed to consider and strain a special aspect of a system and to measure a distinct performance. The methodological approach with the used hardware, a detailed description of the benchmarks, and the statistical evaluation are described in Section 3.1. In Section 3.2, the obtained results are then statistically evaluated. By repeating executions of the same benchmarks, a statement can be made about the temporal variations on the respective node. Furthermore, the measurements on each node allow the quantification of the spatial variation of the whole cluster. With the gathered data, we can establish a statistically valid baseline and review the distribution of the results. Additionally, with over 30 executions on each node, we execute a sufficient number of runs to observe several instances of rare events. Finally, the variability of the two clusters is compared.

3.1. Methodology

Since performance variation is frequently caused by the underlying hardware and software architecture [41], and any change to a single component of the hardware [45], or to the software [41] can create or eliminate variability, we examine and compare two different high-performance compute clusters. In order to interpret the results of the measurements correctly, it is important to know the underlying specifications of the machines, which is thus presented in the following Section 3.1.1.

When designing a benchmark, only a particular aspect of the computer is considered and strained [11]. Therefore, a single benchmark can not be used to reflect on and judge the complete performance characteristic of a particular machine. We have thus selected and tested several different benchmarks, which provide a good overview of the performance of the two computer systems and their variations. In Section 3.1.1, we briefly introduce these applications and the detailed configuration for benchmarking is described in the corresponding results section.

The execution of compute-intensive applications generates waste heat. Therefore, after start-up it takes a certain amount of time until the CPUs are fully warmed up and in thermal equilibrium. In several measurements, we determine the duration certain benchmarks need to reach this state, which is described in Section 3.2.2, and exclude this specific warm-up phase in all subsequent evaluations.

3.1.1. Methodology of Experiments

System Architectures

In this thesis, measurements are carried out on two high-performance machines, which differ in their hardware and are also in different phases of their life cycle. Both computers are part of the Cluster Aix-la-Chapelle (CLAIX) located at RWTH Aachen University. In order to distinguish the two systems from each other, they are called CLAIX2016 and CLAIX2018 after the year of their installation.

CLAIX2016 The older system, the CLAIX2016, was installed in November 2016 and uses 2× Intel Broadwell EP (E5-2650v4) processors (12 cores each) with a nominal frequency of 2.2 GHz and a maximum all-core Turbo frequency of 2.4 GHz when running AVX2 code [23]. They have 128 GB DDR4-2400 main memory per node, which are connected with an Intel Omni-Path x16 HPC network. In total, it consists of 600 compute nodes and two separate front-end dialog nodes. CLAIX2016 is placed in the TOP500 list of November 2017 on the 488th place with a peak performance of 558.42 $TFlop/s$ [12].

CLAIX2018 In December 2018, the CLAIX was expanded by 1,032 computing nodes consisting of two Intel Skylake (Platinum 8160) processors with 24 cores each running at a nominal frequency of 2.1 GHz and 192 GB local memory per node [23]. The maximum all-core frequency for AVX2 code with Turbo Boost is at 2.5 GHz. The system uses four dialog front-end nodes and connects all nodes with an Omni-Path x16 HPC network. In the TOP500 list of June 2019, CLAIX2018 is ranked on the 92th place with a measured Linpack peak performance of 2,483.58 $TFlop/s$ [12].

On both clusters, the Intel 19.0.1.144 C++ compiler (20181018) is used for compilation. Further, parallel applications are executed using Intel MPI 2019.1.144 (20181016). Additionally, Intel Turbo Boost is enabled on both machines and each node runs the Linux CentOS 7.6 distribution based on the Linux kernel version 3.10.0-514.

Performance Analysis Tools

We have a very large selection of applications to choose from and try not to overload the systems with measurements. Our selection covers the state-of-the-art performance metrics and considers as many aspects as possible. With the well-known and frequently used benchmarks FWQ, Linpack, HPCG, and STREAM, we have portable applications that are known to give comparable results across systems.

Fixed Work Quantum The *Fixed Work Quantum* (FWQ) [15] benchmark is designed to measure interference caused by the operating system from the application's perspective [29]. The benchmark starts a `pthread` with a fixed amount of work on

each core within a single node and measures repeatedly the time necessary to complete this quantum of work. A single iteration only consists of incrementing a loop variable and therefore has a very short runtime, which causes system interrupts to have a significant effect on the execution time of each iteration. If no threads were interrupted by the operating system, they would all take the same amount of time to complete. In practice, however, applications are regularly interrupted to allow the execution of system processes. This results in some threads taking more time than others to handle the same fixed amount of work.

Due to the fixed workload of the benchmark, the data samples can be used to create useful statistics on the system noise. Since data are collected for each individual core of a CPU, statements can also be made about the load and distribution of OS processes on the different cores.

High-Performance Linpack The *High-Performance Linpack* (HPL) [11] benchmark is part of the LINPACK Package and measures the floating-point execution rate of a parallel computer. As already mentioned before, HPL is used as a metric to compare the performance of computer systems at the TOP500 [12] list. Thus, HPL has gained a lot of recognition and is one of the most widely used and discussed metrics for high-performance computing systems [31].

The benchmark is a heavily optimized and scalable MPI program with a compute-heavy kernel, which almost reaches the theoretical peak performance of a machine [31]. Since the amount of data required per computation is very small, Linpack has a small Byte/Flop ratio and is therefore compute-bound. The kernel solves a system of dense linear equations of the form

$$A \cdot x = b, \text{ with } A \in \mathbb{R}^{n \times n} \text{ and } x, b \in \mathbb{R}^n$$

by performing LU factorization with row partial pivoting and solving the resulting upper triangular system [11]. The dominant calculations are dense matrix-matrix multiplications and related kernels.

It measures the time required to factorize and solve the system, converts this time into a performance rate, and tests the result for accuracy. To verify the result, the scaled residual r is computed with

$$r = \frac{||A \cdot x - b||_\infty}{\epsilon \cdot ||x||_\infty \cdot ||A||_\infty \cdot n}.$$

Here, ϵ is the relative machine precision and the check is passed when r is less than a set threshold (e.g. 16.0 by default).

In order to achieve the best possible result, the user is offered several tuning options allowing the benchmark to be adapted to the individual machine in use. The user can set the following values, among many others: the number of processes and their grid dimensions P by Q and thus the data distribution can be specified, the problem sizes are stated, one from six different broadcast algorithms can be selected, the look-ahead depth is given, and the memory alignment is set.

High-Performance Conjugate Gradient The *High-Performance Conjugate Gradient* (HPCG) [10, 20] benchmark is a metric designed to more closely represent the workload of real-life applications and many important scientific calculations. The metric measures a machine's performance in *Flop/s* accordingly. Real-world applications are usually not strongly characterized by very expensive compute-bound calculations, but have a diverse set of kernels that also contain many communication steps. In addition, many kernels are memory-bound and have a small computation to data-access ratio.

Therefore, the kernel of HPCG is designed to be memory-bound, to simulate the access patterns of real applications through the same irregular access to memory, and to cover major common communication and computation patterns. In a unified stand-alone code, HPCG generates a synthetic discretized three-dimensional PDE model problem and computes preconditioned conjugate-gradient iterations for the resulting sparse linear system. It measures the performance of versatile kernels, which includes sparse matrix-vector multiplications, vector updates, global dot products, local symmetric Gauss-Seidel smoother, and sparse triangular solve.

In November 2017, the TOP500 site started using the HPCG benchmark as an additional metric to complement HPL. However, HPCG is not intended to replace Linpack.

STREAM Processors increase in speed much faster than computer memory systems [33]. In the course of this process, more and more programs are limited in their performance by the memory bandwidth of the system and not by the computing power of the CPU. In practice, they spend a lot of time filling cache misses instead of actually performing arithmetic operations.

The *STREAM* benchmark [35] is typically used to measure memory bandwidth and hence works with data sets that are much larger than the available cache on a particular system. Thus, the results better reflect the performance of very large vector-based applications. The benchmark executes the four long vector operations `copy`, `scale`, `sum`, and `triad`. STREAM always uses the same approach and always counts only the bytes that the user program has requested to be loaded or stored, so results are always directly comparable.

As the name suggests, the `copy` operation calculates the bandwidth with which a vector w can be copied into a vector v. In the operations `scale` and `sum`, their kernels are supplemented by an arithmetical operation. Hence, the vector w is multiplied by a scalar a or added to a vector c respectively and the result is stored again in the vector v. The fourth kernel combines the previous ones in a so called vector `triad` operation by executing the operation $v_i = w_i + a \cdot c_i$ and therefore measures the performance of data transfers between the arithmetic units of a processor and memory.

STREAM also supports multiprocessor systems with OpenMP or MPI, when the respective compiler flags are set and the problem size is adjusted. Since STREAM measures the bandwidth of the main memory and not of the caches, the array size

should be at least four times the size of the sum of all last-level caches.

The twenty shared-memory systems with the highest bandwidth are presented in the STREAM "Top 20" results [33] list, similar to the TOP500.

3.1.2. Statistical Methodology of Evaluation

The repeated execution of the benchmarks results in large quantities of measured values, which are evaluated statistically in order to gain an overview of the variation. Since the individual benchmarks in a measurement series are repeated several times on each node, we first calculate the *arithmetic mean* μ per node. The *empirical standard deviation* σ per node is also considered, which preserves the statistical dispersion behavior [42]. In order to express the variation of the performance, the dispersion of the data is set in relation to the absolute measured values in the form of their mean values. The quotient of empirical standard deviation and arithmetic mean defines the *coefficient of variation* CoV, with $CoV = \frac{\sigma}{\mu}$. This is a measure for statistical dispersion, with which data sets that are measured in different units can be compared without conversions allowing us to directly compare different benchmarks and their variations. Furthermore, this measure can be interpreted very vividly using a normal distribution. Thus, this coefficient specifies exactly the performance window by which the measured values vary in the 1σ range around the mean value. If, for example, the CoV is 2.00%, this means approximately 68.27% of all nodes vary by a maximum of $\pm$2.00% around the expected value.

To describe and characterize the underlying distribution of the measured performance, we use the moments of the distribution. For a better overview, Figure 3.1 shows various distributions with distinct moments. The symmetry of a distribution is measured using a) the *skewness*. A negative skewness usually indicates that the tail is on the left side of the distribution and the distribution is said to be left skewed. Since we histogram performance values later and look at their distribution, a left skewed distribution means in our case that a large part of the measured val-

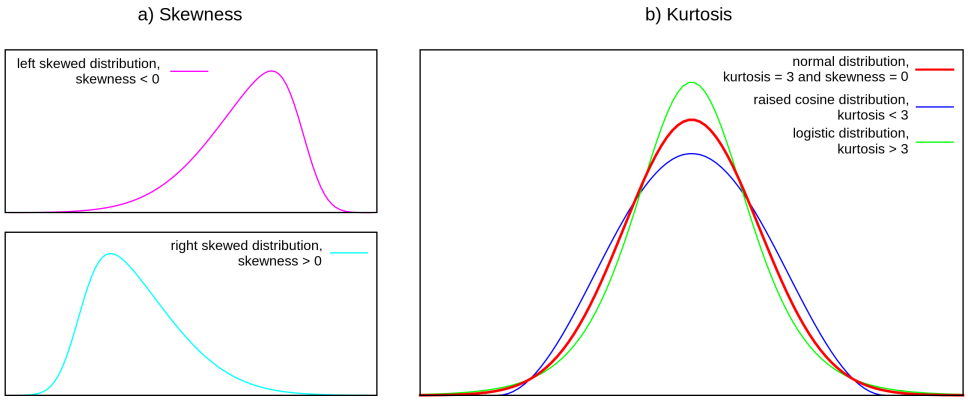

Figure 3.1.: Illustration of the skewness and kurtosis values and how they correlate with the moments of a normal distribution (red).

ues is concentrated on the right side and these nodes deliver a high performance. However, the longer left tail indicates some nodes with strongly downward deviating performance, i.e. slow outliers. A positive skewness suggests that the tail is at the right side. This means the outliers are faster than the average and most nodes are concentrated on the left side of the distribution. The second moment of a distribution is b) the *kurtosis*, which is a measure of the steepness or "tailedness" of a distribution. Distributions with low kurtosis scatter relatively evenly, while distributions with high kurtosis spread more from extreme but rare events, i.e. outliers. These moments of a distribution are typically interpreted in comparison to a normal distribution (red graph in the figure) that has a kurtosis of 3 and a skewness of 0.

3.2. Results

The results of the benchmarks introduced for measuring performance are presented in this section. They are statistically analyzed in order to gain an overview of the existing variability in performance on both clusters. Furthermore, a comparison of the variation in the single-node runs between the individual nodes and between the two clusters is drawn.

3.2.1. Impact of Operating System Noise

In this first series of tests we examine the influence of the operating system on the execution of the programs from a normal user. We simulate an application for about 10 minutes on both clusters with FWQ, as introduced in Section 3.1.1, and measure the occurrence and length of interrupts. The threaded version is used with 24 or 48 threads each that are each bound to a physical core with disabled hyperthreading. There is nearly no communication between the threads, except for a synchronization fo the start time and an aggregation of the sample data in the end. In our setup, FWQ is configured to record 70,000 samples with a nominal execution time of 9.32 ms on CLAIX2016 and 7.96 ms on CLAIX2018. Therefore, we observe the system's activity for a period of about 10.9 and 9.3 minutes on each node and cluster.

CLAIX2016 In order to get an overview of the system noise per node, we first consider the average runtime of all cores together and do not distinguish between the individual cores at this point. Figure 3.2 shows the ascending sorted average duration of the benchmark on 599 nodes. Note that the abscissa has several breaks. We find only a rather small variation in the runtimes of the benchmark, which fluctuates in the 1σ interval with $\pm 1.4\%$, measured by the coefficient of variation. On average one iteration lasts 9.32 ms (standard deviation: 0.13 ms). However, the two nodes `lnm016` and `lnm017` stand out due to an extremely high volume of significant long interrupts with particularly large mean value of about 9.52 ms and 12.46 ms. The later is about 33.70% slower than the average node in this experiment. Here, we

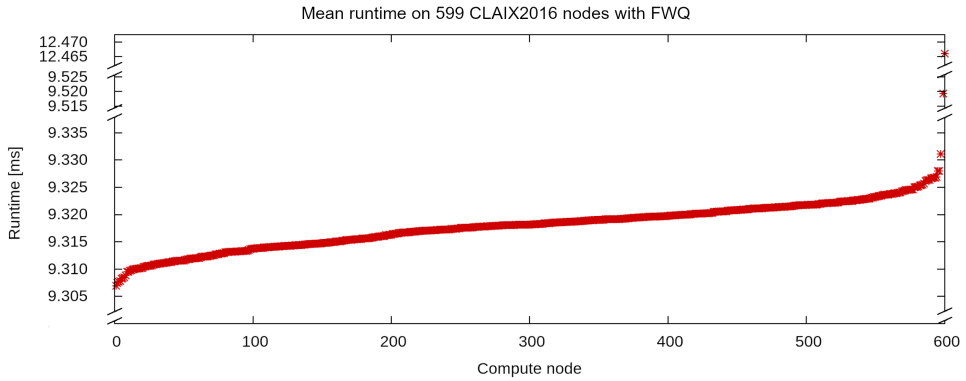

Figure 3.2.: Ascending sorted average runtime per node on the abscissa with FWQ. Measurements are taken on 599 CLAIX2016 nodes, each with 70,000 iterations. Longer runtimes are an indication of increased occurrence of interrupts and operating system noise.

would like to point out again that the workload in the synthetic FWQ benchmark is to count a loop variable and that longer runtime is spent in interrupts. If these two nodes are excluded from the statistics as outliers, the adjusted data set varies only with a CoV of 0.04% ($\mu = 9.32\ ms$, $\sigma = 4.02\ \mu s$).

The differences between these outliers and an average node are particularly evident in Figure 3.3, which shows the FWQ results for different selected nodes of CLAIX2016. The abscissa indicates the sample number, while the ordinate is the runtime of each sample in milliseconds. All cores are recorded in parallel and all cores are displayed individually in the diagram in different colors and point styles. The figure shows in a) the noise signal of a representative compute node chosen at random. On a noise-free system, we would expect a continuous horizontal line at 9.32 ms for each core. Each sample point above this line indicates interference

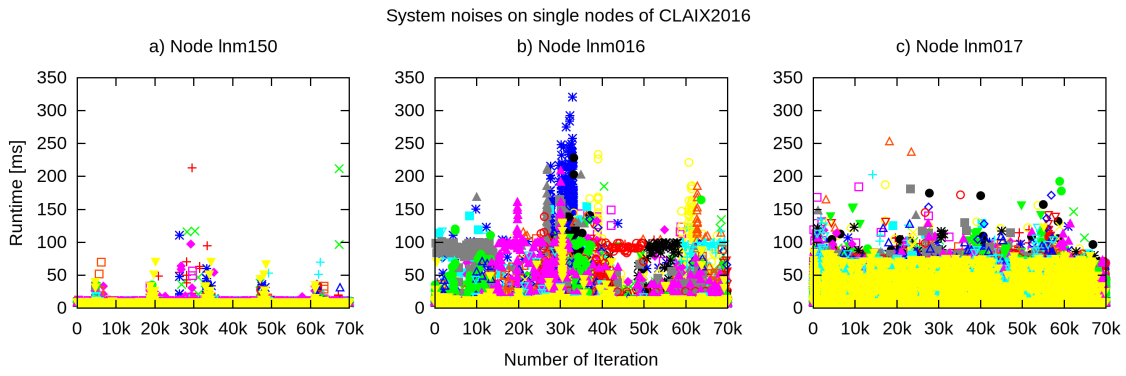

Figure 3.3.: System noise on a) a randomly selected node and b) and c) conspicuous nodes of CLAIX2016. Each color represents the time it takes a single core to complete its iteration. Measured times above 9.32 ms indicate interrupts.

caused by system processes. It is noticeable that at approximately equidistant intervals of 14 thousand iterations, i.e. every 13 seconds, system activity increases.

The next two graphs to the right show the results of the two irregular nodes that have significant long interrupts occurring in many iterations. With FWQ running on these two nodes, we notice a very high amount of interrupts, which influence the program execution very often and for a long amount of time. For example, on `lnm016` the longest iteration is 34.41× longer than the average and about 46.45% of all iterations are interrupted. We were also able to reproduce this measurement over the next few weeks by repeated measurements and again find high system activity on all cores.

This excessive occurrence of interrupts must have a noticeable effect on the execution of other programs and extend their runtimes. However, these nodes were reserved for the integration of a Lustre file system and excluded from normal batch operation, which means they were no longer available for subsequent measurements with other benchmarks. After this maintenance concluded, we are able to remeasure the two nodes with FWQ and other benchmarks. This time, we observe normal levels of interrupts and no significant differences compared to other nodes.

Since we measured the occurrence of interrupts for each of the 14,400 cores, we can also compare them afterwards. The average duration of the FWQ benchmark on the 24 cores of a node is calculated globally. As it can be seen in Figure 3.4, core_0 and core_{12} are interrupted particularly frequently. These cores typically run the operating system of the CPU_0 and CPU_1 of our two-socket machines. Accordingly, user programs on the other cores are interrupted less frequently. Applications are at times familiar with these conditions and therefore dynamically distribute the workload to the individual cores to avoid load imbalances.

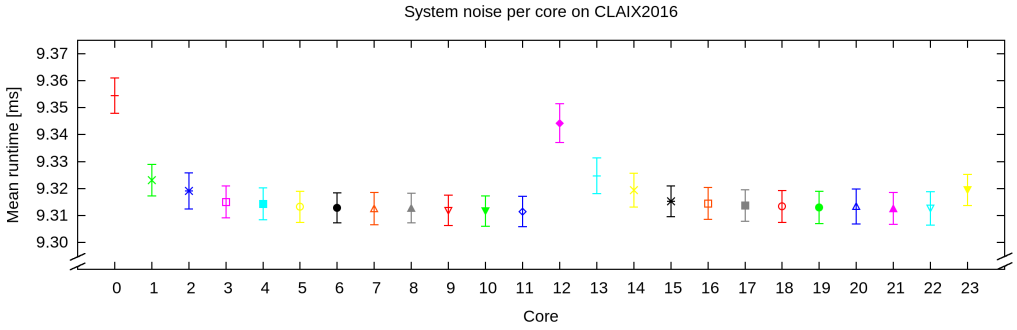

Figure 3.4.: Comparison of system noise on CLAIX2016 per core with FWQ. Longer runtimes suggest that more interrupts occur on the core.

CLAIX2018 We run FWQ with the same binaries on CLAIX2018 without recompiling. The only difference to the execution on CLAIX2016 is the usage of 48 threads to use all physical cores.

Only a small variability between the individual nodes is observed on CLAIX2018. The mean value has a CoV of only 0.07%. In addition, the interrupts on all nodes occur evenly and have approximately the same length, i.e. no outliers are noticed. At core level, core_0 and core_{24}, which are executing most of the Linux kernel's workload, are also more affected by interrupts that the others.

In summary, we find a comparable level of system noise and interrupts on both systems, which only have a very small variation in their occurrence and length (1σ : $\pm 0.04\%$ and $\pm 0.07\%$). Moreover, we only notice in exceptional cases a temporary increase in operating system activity on two out of 1,624 observed nodes.

3.2.2. Computing Power

The user is particularly interested in how much computing power is achieved and in what level of variation to expect. To determine this, we look at the peak performance with Linpack and the imitated real-world workload with the memory-bound benchmark HPCG, which are both explained in Section 3.1.1. Since these computation-intensive applications emit strong heat, we also investigate which influences the temperatures have on our measurements and whether a user has to pay attention to these effects when running their code on a cluster.

Influences of Heat

When a CPU is heavily stressed by complex calculations, it generates a great deal of heat [1]. This warming up of a compute node is not a sudden process, but happens over time and a *warm-up period* can be observed. First, we determine for each benchmark and cluster the time it takes to find and maintain a stable temperature in thermal equilibrium after an application is started. Then we investigate whether neighboring nodes that execute compute intensive kernels have an influence on the observed node by heat transfer.

Characterizing Warm-Up Periods Figure 3.5 illustrates a representative example of the warm-up period on CLAIX2016 for the HPCG benchmark and includes the first five minutes of this measurement. In addition to the temperature in a), the frequency is shown in b). The frequency is considered because Intel Turbo Boost 2.0 and Enhanced SpeedStep Technology dynamically adapt it to the workload, temperature, instruction set, and other factors. Immediately before the application of interest is executed, a shell script is started in the background to monitor the status of the node at equidistant times of approximately one millisecond. The frequency is directly read from the `/proc/cpuinfo` interface, the temperature from the `/sys/devices/platform/coretemp` interface. Since the script generates a low overhead with text processing using `awk` and reading the interfaces, the performance is slightly influenced. The additional workload causes about 5% less performance when benchmarking with HPCG compared to a measurement without the script

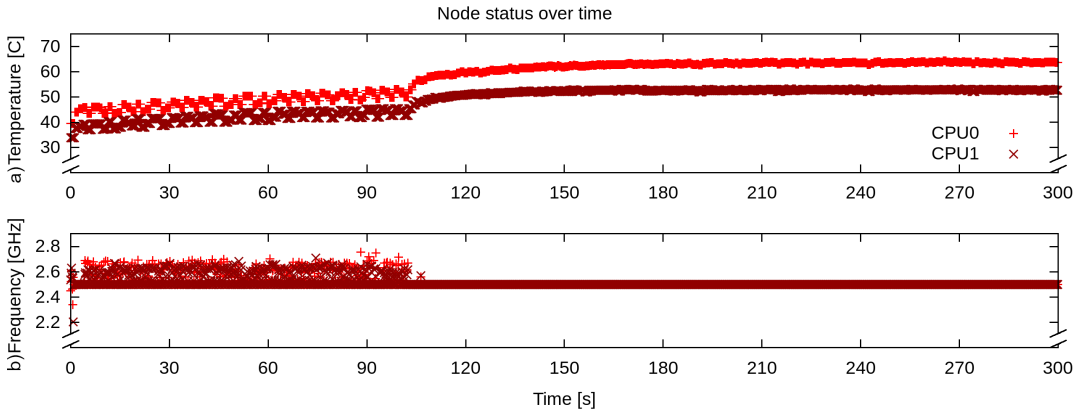

Figure 3.5.: Warm-up period of a HPCG run on CLAIX2016. Temperature and frequency of the first five minutes are observed.

running in the background. Linpack still achieves approximately 99% of its original computing power. In order to be able to reflect on performance without overhead, the following measurements are performed without this type of monitoring, if not stated otherwise.

First, both processors clock between 2.2 GHz and 2.8 GHz. This indicates that not all cores are used in this initialization phase of the benchmark and that the Turbo Boost regulates the frequency. However, after about 110 s the frequencies stabilize at 2.5 GHz on both processors and all cores and then remain at this level for the rest of the run. This adapted frequency of 2.5 GHz is the characteristic Turbo Boost frequency when the AVX2 instruction set is being used on all 12 cores of Intel Broadwell EP processors [21]. The temperatures also rise steadily until constant temperatures of 52 $°C$ and 63 $°C$ respectively are reached after 150 s and the node remains in this thermal equilibrium for the rest of the run.

Since CPU_0 of all observed CLAIX2016 2-socket systems is consistently ten degree Celsius warmer than CPU_1, a systematic cause is suspected. In fact, this difference can be traced back to the hardware, which can be seen in Figure 3.6 that shows a CLAIX2016 node from two perspectives. Underneath the dark gray cover lays CPU_1, CPU_0 is installed under the silver passive cooler. The fresh air first flows through the dark gray cover and through the first passive cooler to cool CPU_1. Only then does the already heated air reach the second processor, which results in a systematic temperature difference between the two CPUs on all observed nodes.

The warm-up period differs from benchmark to benchmark, the used instruction set, and depends just as much on the hardware. Thus, when running Linpack, CLAIX2016 clocks at 2.4 GHz and reaches temperatures of 70 $°C$ and 79 $°C$ after 70 s. The two Xeon Platinum 8160 processors on a CLAIX2018 node are also cooled in a row. However, each node has a total of three active fans and one passive cooler per CPU. The measured differences only amount to about four degrees Celsius. The HPCG code uses the AVX2 units and the CPUs clock with a Turbo frequency of 2.5 GHz after 100 s and temperatures of 69 $°C$ and 73 $°C$ are reached after 210 s.

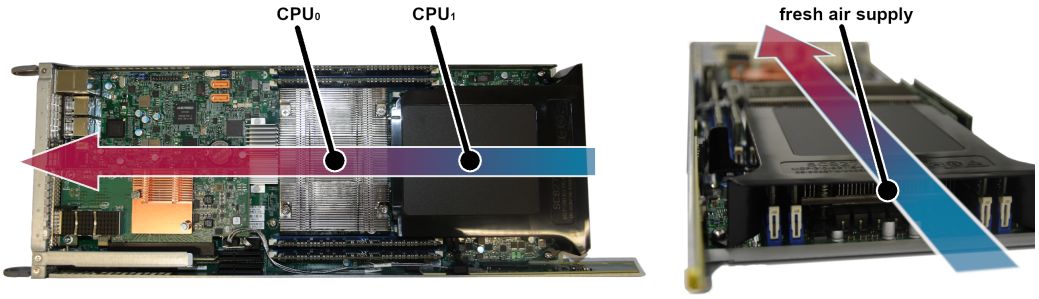

Figure 3.6.: Structure of a CLAIX2016 node from two perspectives. CPU_1 is closer to the fresh air supply than CPU_0 and is therefore better cooled. Pictures provided by the HPC chair and edited by the author of this thesis.

The AVX-512 instruction set is used for HPL and reaches a Turbo frequency of 1.6 GHz after approximately 90 s and a stable temperature of 73 $°C$ and 76 $°C$ respectively after 150 s.

Since significant performance differences of up to 19.43% are measured during the warm-up periods, we exclude the first run in our following reported measurements and use it to get a stable temperature and frequency when necessary.

Impairment by Neighboring Jobs With regards to significant performance differences during warm-up of a node, we also look into influences of emitted heat from adjacent nodes that perform compute-intensive applications, such as Linpack that reached the higher temperature in our measurements compared to HPCG. In related work, we have already seen that neighboring nodes can temporarily influence the performance of our job of interest. As an example, we refer to competing accesses to the network [4, 40, 46]. Therefore, we investigate the impairment from heat leaks caused by adjacent nodes.

For this purpose, we request ten directly adjacent nodes and perform two measurements. In the first test case, all ten nodes perform a Linpack run simultaneously. Initially, we are only interested in the result of the job executed on the middle node. The second test case differs in that Linpack only calculates on the same middle node again, while the remaining nine nodes are idling and do not perform any calculations.

On both clusters, we choose five times ten neighboring nodes on a random basis and therefore perform a total of ten measurements on 100 nodes. When selecting the nodes, we also take into account to request nodes from different racks and locations. We find that in the "solo" measurements with nine idle and one node that executes Linpack performance variations of on average ±0.25% occur on CLAIX2016 and the performance differs on average by ±0.47% on CLAIX2018 compared to the "heat" measurements with all nodes running HPL.

If we continue to compare the measured temperatures, we also find only very small differences. The heat transfer of the neighboring nine nodes causes only a difference of 1 $°C$ to 2 $°C$ on the middle node. In idle measurements, however, the

nine neighbours are 27 $°C$ to 40 $°C$ cooler compared to the measurements in which they perform Linpack.

Since this temporal variation is not practically significant, we conclude that heat leaks are not to be considered any further and that we therefore do not have to pay attention to the workload and emitted heat from other nodes in our evaluation. This allows us to conduct our experiments on several nodes at the same time and independent from any other jobs by the scheduler on adjacent nodes. Therefore, the following section does not deal further with the temperature of a node.

High-Performance Linpack

Both of our machines are equipped with Intel processors, which allows us to use the Intel® Distribution for LINPACK* Benchmark [22], which is an optimized version of the original Netlib HPL benchmark for Intel processors. Intel's distribution can be used for TOP500 runs and is an addition to the HPL benchmark. We use the Intel-provided `xhpl` binary, which is dynamically linked against Intel MPI libraries from the Intel® Math Kernel Library (Intel MKL). For a measurement series, the benchmark is executed 46 times in single-node runs back to back. The first run is used as a warm-up phase for the node and is not considered afterwards (see Section 3.2.2) leaving us with 45 valid measurements to evaluate. These should be a sufficient amount of measurements to consider rare events [32].

CLAIX2016 Linpack is configured to spawn one MPI process and fill the other cores with threads, so the grid dimension P by Q is set to 1×1. The block size NB is set to 192 and the problem size N is selected so that approximately 80 percent of the main memory is filled by the matrix, which are the recommended settings from Inte [22]l. To ensure that the operating system only assigns threads to physical processors, hyper-threading is deactivated. Thus, both processors of a node are used efficiently and a good performance in the benchmark is achieved.

Figure 3.7 a) shows the increasingly sorted mean values of the single-node performance in combination with the standard deviation as error bars on the abscissa in $GFlop/s$. The error bars are a measure for the temporal variation on a node. Each node performed Linpack 45 times and the corresponding arithmetic means are shown in the figure. If a node often achieves a different high performance in the repeated execution of the benchmark due to temporal variation, the measurement will have a larger standard deviation and the associated error bar will be larger. Thus, the relatively small error bars in the figure illustrate that only small temporal variations occur in this experiment. On average, a temporal variation of $\pm 0.07\%$ is measured in the 1σ interval around the mean in these single-node runs.

Spatial variation is better visualized in the course of the curve and the frequency distribution of the performance. A smooth rising curve is noticeable with a global average performance of 860.07 $GFlop/s$ and a standard deviation of 13.87 $GFlop/s$. Especially three nodes stand out due to significantly less performance in the range

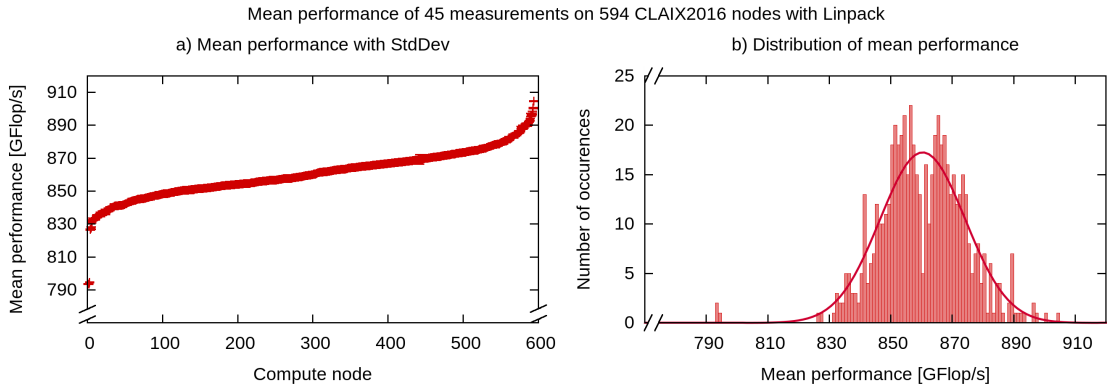

Figure 3.7.: Performance measured with 45 Linpack runs on 594 CLAIX2016 nodes in $GFlop/s$. The graph in a) shows the average performance with standard deviation as error bar. In b) their distribution with an additional distribution density function is illustrated.

of 764 $GFlop/s$ (4.8 standard deviations below the mean value) and the slowest node delivers about 92.23% of mean performance. A better overview is provided by Figure 3.7 b), which shows the distribution in a histogram. With a kurtosis of 4.98, a normal distribution with some outliers is approximately given. With two peaks at 855 $GFlop/s$ and 865 $GFlop/s$, our data is further a bimodal distribution. The outliers can be identified as slower than average by the skewness value of -0.31. Three significantly slow nodes can be seen in the range of 794 $GFlop/s$. For reference, a probability density function of a normal distribution with the same μ and σ values is also added to the figure. Except for these three outliers, we have a relatively stable system with only a small variation in peak performance. Within a 1σ confidence interval, we have a variation of $\pm 1.61\%$ around the mean value, i.e. a CoV of 1.61%.

After removing the three outliers, the spatial variation in the 1σ interval is still at $\pm 1.52\%$ with an average of 860.73 $GFlop/s$ and a standard deviation of 13.07 $GFlop/s$. This adjusted distribution is again almost normally distributed (kurtosis $= 2.50$) but this time slightly right-skewed (skewness $= 0.24$).

CLAIX2018 The experimental setup on CLAIX2018 is very similar. We also run Linpack 46 times on each available node and evaluate 45 runs and discard one warm-up run. In tuning tests, we have achieved better results with one MPI thread per socket, i.e. two per node, than with just one. The problem size is also adapted to the larger RAM of the newer machine and the block size is set to $NB = 384$.

Figure 3.8 a) shows the mean performance in ascending order with the standard deviations as error bars. We find a global average of 1,934.18 $GFlop/s$ with a standard deviation of 40.55 $GFlop/s$. The error bars indicate small temporal variations with an average CoV of 0.25% per node. A single node is particularly noticeable with higher temporal variability ($CoV = 6.51\%$), which has a single run that is much slower than the others. Just like on CLAIX2016, a few individual nodes with

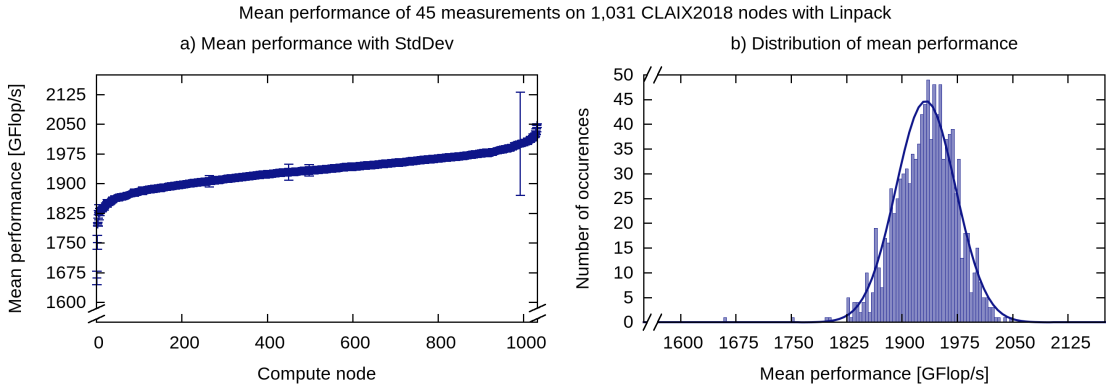

Figure 3.8.: a) Mean performance reported by Linpack in $GFlop/s$ with standard derivation on 1031 nodes of CLAIX2018 with 45 runs.
b) Histogram with distribution density function.

strongly downward deviating performance are striking. The slowest node performs about 86.01% of the mean performance (9.13 standard deviations below the mean).

In the b) of the figure, the results are processed quantitatively in a histogram and a normally distributed density function is added. The data distribution of our measurement corresponds approximately to a normal distribution (kurtosis = 4.85). With a skewness of -0.54, we see single nodes as outliers with lower results of $1,662$ $GFlop/s$ and $1,752$ $GFlop/s$. The variation in peak performance is slightly greater than on CLAIX2016 and varies by $\pm 2.10\%$ in the 1σ interval.

After removing six significantly slow outliers, the mean performance slightly increases to $1,933.38$ $GFlop/s$ with a standard deviation of 38.78 $GFlop/s$. This also reduces the spatial variation to $CoV = 2.01\%$. The distribution is slightly right-skewed, but still approximately normally distributed.

On both clusters the distribution follows approximately a normal distribution. The temporal variations on the nodes are rather small on both clusters and amount to respectively 0.07% and 0.25% measured by the CoV. The spatial variations are higher with $\pm 1.52\%$ and $\pm 2.01\%$ within 1σ, after removing extreme outliers. In summary, both computers have a similar performance variation characteristic in this compute-bound benchmark, with CLAIX2018 having a relatively slower performance of the slowest node compared to the mean.

High-Performance Conjugate Gradient

We also use Intel's optimized binaries for HPCG [22] from the Intel MKL and execute the benchmark 31 times directly back to back. With the first run serving as a warm-up phase, 30 valid measurements are evaluated. For the configuration, we follow Intel's recommendations [22] and use one MPI process per CPU socket and one OpenMP thread per physical core. To ensure that the measurements meet the official criteria, a runtime of 30 minutes per run is set.

CLAIX2016 Figure 3.9 presents the results of a) the mean values in descending order with the standard deviation as error bars, and b) the corresponding histogram of the HPCG benchmark on CLAIX2016. An arithmetic mean of 22.22 $GFlop/s$[1] and a standard deviation of 0.06 $GFlop/s$ are measured. The curve of average values is almost a constant line, which contains only slightly more than two percent of the nodes with downward deviating performance. The individual nodes also vary temporally on average by 0.76% in the single-node runs. Only a single node experiences variation of 14.36% and has a single run that is significantly slower than the others. In addition, three other nodes with significantly less performance, and nine nodes with slightly less performance stand out. With a kurtosis of 37.77 and a skewness of -4.39 of the distribution function, these nodes are also recognized as single, slow outliers by the moment of the distribution. The slowest node, however, only achieves 2.63% less performance than the average. The spatial variation is additionally very low and varies only with $\pm 0.26\%$ in the 1σ interval.

However, if the data set is adjusted for deviating measurements and six slow outliers are removed, this distribution is almost normally distributed (kurtosis = 2.98, skewness = 0.39) and the spatial variation decreases to 0.17%.

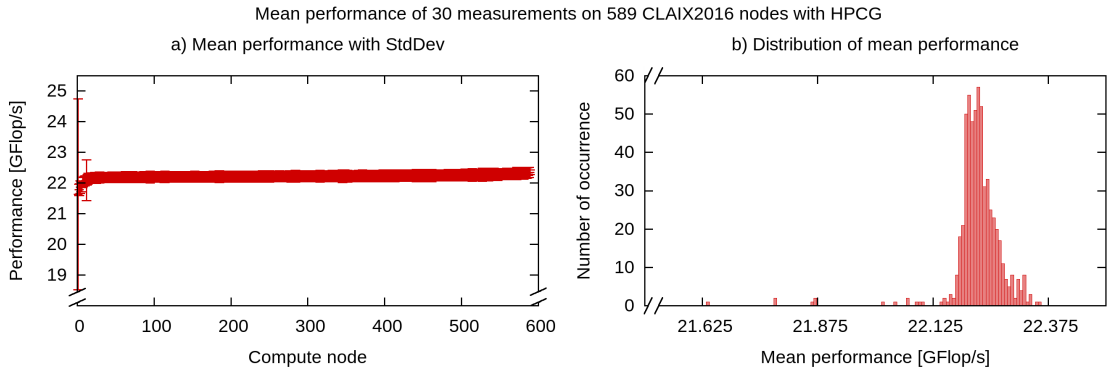

Figure 3.9.: HPCG compute power of 30 measurements on 589 CLAIX2016 nodes. In a), the mean performance in $GFlop/s$ with the standard derivation as error bars is shown. In b), an additional histogram is provided.

CLAIX2018 Again, we measure a greater variation in the achieved performance on CLAIX2018. In Figure 3.10 a), the mean computing power per node is shown in ascending order. On the 1,025 nodes measured, the global mean value is 37.31 $GFlops$[2] with a standard deviation of 1.21 $GFlop/s$. In addition, the curve is again nearly constant with exception to outliers. The error bars show that the temporal variations at node-level decrease with increasing performance. Furthermore, three discrete variation values are recognizable. Thus, the 388 nodes with less performance

[1]The measured performance with HPCG is about 38.71× smaller than the Linpack performance of the same nodes. With regards to TOP500, this is in the expected magnitude. In the list of June 2019, the ratio of the top 10 is 54.33× on average [12].

[2]See footnote 1. Here, the factor is 51.81×.

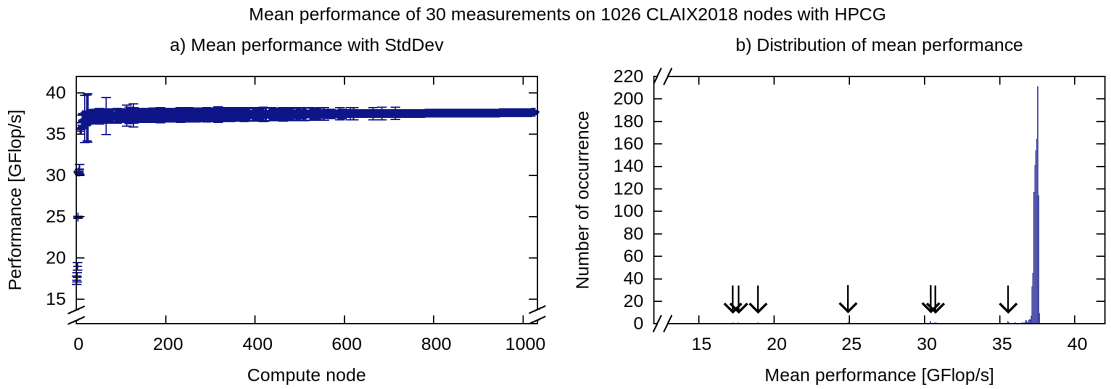

Figure 3.10.: Performance measured with 30 HPCG runs on 1025 CLAIX2018 nodes in $GFlop/s$. a) shows the average performance with standard deviation as error bar. In b), the performance distribution is illustrated in a histogram. Arrows point to the difficult to detect outliers.

vary temporally by more than $\pm 1.5\%$ while the other nodes have variations of less than $\pm 0.25\%$ in the 1σ confidence interval. On average, the CoV is 0.73%.

However, if the spatial variation is considered, it is distinctly greater. The overall performance varies by $\pm 3.24\%$ within the 1σ confidence interval. In particular, the histogram in Figure 3.10 b) illustrates the extent to which the slowest eight nodes deviate from the average. They only perform 18 $GFlop/s$, 25 $GFlop/s$ and 30 $GFlop/s$, whereby the node with the lowest performance only achieves 46.30% of the mean performance (16.62 standard deviations below the mean value). This strong variation with slow outliers can also be recognized by the very large kurtosis value of 207.50 and a skewness of -13.75.

After the removal of eight significantly slow outliers, the spatial variation in the 1σ interval is only $\pm 0.47\%$ with an average performance of 37.41 $GFlop/s$ and a standard deviation of 0.18 $GFlop/s$. Moreover, this adjusted distribution is almost normally distributed (kurtosis = 2.56) and slightly left-skewed (skewness = -0.48).

While most nodes achieve performances at a similarly high level, a few nodes stand out that perform significantly slowly. Compared to the measurement with Linpack, we find much greater and more frequent downward deviations. In addition, different nodes are affected in this measurement than before. The slowest nodes as determined in the measurement with Linpack achieve in the HPCG benchmark an average performance and vice versa. This means that different nodes stand out as outliers in each benchmark.

Compared to the peak performance, both clusters again have a very similar variation characteristic. The only difference is that CLAIX2018 has more extreme outliers, which in the worst case achieve less than half the average performance.

3.2.3. Memory Bandwidth

In order to measure the variation in the memory bandwidth, we perform experiments with the STREAM multi-threaded benchmark version 5.1 on all nodes of the two machines. In each measurement, the benchmark is executed 100 times directly back to back. With an array size larger than four times the sum of all last-level caches, the bandwidth from main memory is measured. To obtain the maximum throughput, we pin one OMP thread to each core.

CLAIX2016 Figure 3.11 a) shows the mean value of the memory bandwidth of the 100 runs of each node in ascending order and additionally the standard deviation as error bars. The temporal variations amount to 1.05% on average, which can be seen in the larger error bars. A mean performance of 90.73 $GByte/s$ and a standard deviation of 0.38 $GByte/s$ is measured, which results in a spatial variation of 0.41% across all 581 nodes. The lowest bandwidth on CLAIX2016 is 1.31% smaller than the mean bandwidth and is 3.16 standard deviations below the arithmetic mean.

The multi-threaded STREAM is approximately normally distributed (kurtosis = 3.93, skewness = 0.17) and is slightly right skewed, which can be seen in Figure 3.11 b) that also includes a probability density function. The few outliers in this run are faster than the average and no slow outliers seem to occur.

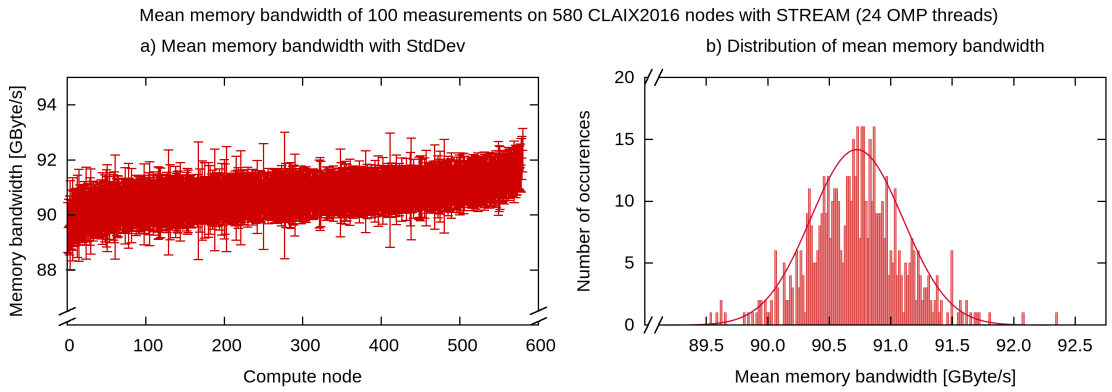

Figure 3.11.: Memory bandwidth measured with 100 STREAM runs (24 OMP threads) on 580 CLAIX2016 nodes in $GByte/s$. a) shows the average performance with standard deviation as error bar. In b) a histogram illustrates the distribution.

CLAIX2018 On the 48-core nodes of CLAIX2018, we run a multi-threaded STREAM with 48 OMP threads, which are also each pinned to a distinct core with hyper-threading disabled. The benchmark runs on 1,029 nodes and its result are displayed in Figure 3.12 a), which includes the mean values in $GByte/s$ with the standard deviations as error bars. The average bandwidth is 164.05 $GByte/s$ with a standard deviation of 3.38 $GByst/s$, which leads to a CoV of 2.06%.

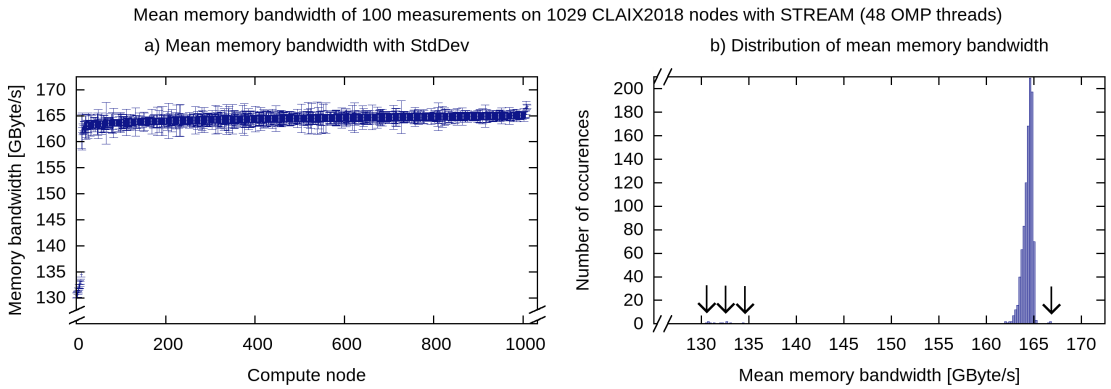

Figure 3.12.: Memory bandwidth measured with 100 STREAM runs on 1,029 CLAIX2018 nodes in $GByte/s$. a) shows the average performance with standard deviation as error bar. In b) their distribution is illustrated in a histogram. Arrows point to the difficult to detect outliers.

Variations of up to 2.45% are observed on single nodes with an average run-to-run variability of ±0.58%. Furthermore, the smallest bandwidth is 79.54% of the mean bandwidth (9.94 standard deviations below the mean), which is also perceptible in the skewness of −9.23. Figure 3.12 b) shows the strongly left skewed histogram of this measurement and, with a kurtosis of 88.14, the distribution is far from being normally distributed.

If we adjust the measurement and remove nine slow outliers, we find a left-skewed distribution whose kurtosis is close to a normal distribution and the spatial variation decreases to only 0.30%.

While normally distributed performance occurs on CLAIX2016, the variation on CLAIX2018 is greater with individual extreme outliers that reach significantly less bandwidth. After filtering outliers, the distribution on CLAIX2018 also is approximately normally distributed. The interested reader is referred to additional experiments with STREAM in Appendix A. There, further measurements with different numbers of OMP threads are conducted.

3.3. Summary

On both systems, we find statistically significant and practically significant performance variability in different single-node benchmarks. The results of our measurements with all relevant data and their statistical analysis are summarized in Table 3.1. In nearly all benchmarks, a small fraction of single-node runs stands out as extreme slow outliers. These outliers deliver performance that deviates significantly downwards from the expected values. Adapted statistics are also listed in the table after practical significant outliers have been excluded. In the analysis, these can be evaluated as probably coming from a different population than the other

System		Linpack	HPCG	STREAM
CLAIX2016	Number of Runs	45	30	100
	Number of Nodes	591 (594)	583 (589)	581
	Mean Performance	860.73 (860.39)	22.23 (22.22)	90.73
	Standard Deviation	13.07 (13.87)	0.04 (0.06)	0.38
	min / mean	96.04% (92.23%)	99.06% (97.37%)	98.69%
	Spatial Variation (1σ)	$\pm$1.52% ($\pm$1.61%)	$\pm$0.17% ($\pm$0.26%)	$\pm$0.41%
	$\varnothing$ Temporal Variation (1σ)	$\pm$0.07% ($\pm$0.07%)	$\pm$0.74% ($\pm$0.76%)	$\pm$1.04%
CLAIX2018	Number of Runs	45	30	100
	Number of Nodes	1,026 (1,032)	1,017 (1,025)	1,018 (1,029)
	Mean Performance	1,933.38 (1,932.59)	37.41 (37.31)	164.40 (164.05)
	Standard Deviation	38.78 (40.55)	0.18 (1.21)	0.50 (3.38)
	min / mean	94.57% (86.01%)	95.02% (46.30%)	98.50% (79.54%)
	Spatial Variation (1σ)	$\pm$2.01% ($\pm$2.10%)	$\pm$0.47% ($\pm$3.24%)	$\pm$0.30% ($\pm$2.06%)
	$\varnothing$ Temporal Variation (1σ)	0.25% (0.25%)	$\pm$1.01% ($\pm$1.02%)	$\pm$0.58% ($\pm$0.58%)

Table 3.1.: Measured values in $GFlop/s$ and $GByte/s$ of the three benchmarks and their statistical analysis. The values are shown after the removal of outliers and the original values of all measurements are shown in brackets.

values [16], since other factors seem to influence the performance on these nodes. The outliers are examined in particular in Section 4.

On CLAIX2016, we find spatial performance variability within 1σ in a range from $\pm$0.26% to $\pm$1.61% depending on the benchmark. The slowest nodes achieve from 1.31% to 7.85% less performance than the expected mean performance. Furthermore, the temporal variations range from $\pm$0.07 to $\pm$1.05%.

On CLAIX2018, spatial variations from $\pm$2.06% to $\pm$3.24% are found within the 1σ interval around the mean value. Depending on the benchmark, the slowest nodes reach only from 41.31% to 80.59% of the average expected performance. If the temporal variation is considered, variations of $\pm 0.25\% - \pm 1.02\%$ are found on average. In exceptional cases, significantly slower runs with temporal variation of up to $\pm$6.51% on individual nodes are observed.

This overview is supplemented by a ranking of five the best and worst performing nodes in Table 3.2. In the comparison, the placement of the nodes per measurement series is determined and an overall ranking is formed. If a node was not available for a single measurement, it is completely excluded from the ranking.

This ranking is created to examine whether there are fast and slow nodes within the clusters that perform well or worse in all benchmarks. We find, however, that it is only possible to create a ranking if the spatial variation is significantly greater than the temporal variation. For example, if Figure 3.7 a) is considered again, we see that the nodes can be compared and ranked using the Linpack performance. The slope of the curve and thus the spatial variation is clearly visible. The error bars are moreover relatively small and smaller than the slope of the curve. Therefore, the temporal variation is noticeably smaller than the spatial variation, which allows us to sort the nodes according to their compute power.

CLAIX2016				CLAIX2018			
Node	**Linpack**	**HPCG**	**STREAM**	**Node**	**Linpack**	**HPCG**	**STREAM**
`lnm274`	38	27	17	`ncm0588`	61	32	7
`lnm166`	3	15	112	`ncm0572`	9	11	165
`lnm363`	44	19	75	`ncm0602`	116	56	19
`lnm320`	32	78	73	`ncm0617`	39	94	98
`lnm187`	11	200	49	`ncm0271`	84	38	111
`[...]`	[...]	[...]	[...]	`[...]`	[...]	[...]	[...]
`lnm208`	570	568	592	`ncm0177`	920	872	1,023
`lnm376`	575	581	588	`ncm0995`	926	943	961
`lnm560`	594	587	593	`ncm0491`	1,008	919	921
`lnm562`	592	586	595	`ncm0792`	987	985	904
`lnm561`	593	588	594	`ncm1010`	980	942	995

Table 3.2.: Section of the globally considered ranking of all nodes based on their placement in the three benchmarks.

However, for measurements with HPCG and STREAM, only very small changes are sufficient to influence the order of the nodes. This can be seen clearly in the plots of the average performance, where the temporal variations (error bars) are significantly greater than the slope of the functions and thus the spatial variation.

With regard to the ranking list, it is noticeable that the nodes placed in the upper area seem to perform chaotically in the individual tests on CLAIX2016. Thus, no nodes can be found that have consistently achieved the best performance in all benchmarks. However, the last places are continuously occupied by the nodes `lnm[560-562]`. With only one exception in HPCG, they are rated with the lowest performance in every single measurement conducted suggesting a systematic problem with these nodes, which is investigated in Section 4.1.

On CLAIX2018, the problem does not seem to be limited to any particular subset of nodes. For example, the first placed node in the three test cases only occupies the 33rd place on average, and the last placed node the 972nd place (out of an average of 1,017 nodes). Since the nodes place widely distributed across the benchmarks, no systematic spatial variation seems to occur either. It is also noticeable that the extreme outliers of the respective benchmarks do not achieve significantly less performance than the mean in the other measurements as well. Therefore, Section 4.2 considers each measurement separately and no correlations between the benchmarks are suspected.

We note that on both clusters we find statistically significant and practically significant performance variations in single-node runs on a small fraction of nodes. The variation on CLAIX2018 appears to be greater than on CLAIX2016 that also shows a very stable performance when running HPCG and STREAM with less outliers. Furthermore, we notice three nodes on CLAIX2016 that perform significant low in all benchmarks. Otherwise, the problem is not limited to any particular subset on both systems. In the next section, we investigate possible sources and discuss possible reasons for these variation characteristics.

4. Identifying Causes of Performance Variation

In this section, we explore possible sources that cause the fluctuations in performance measured and described in Section 3.2. Since the large spatial variation is mainly caused by single but extreme outliers, we closely investigate them in this section and compare them to nodes that perform particularly well. We examine whether we can reproduce the results at later points in time and record status data, such as the temperature, frequency, power usage, and the CPU utilization. Because the temporal variations measured with Linpack are negligible low and only amount from $\pm 0.07\%$ to $\pm 0.25\%$ in the 1σ interval, we do not need to measure repeatedly and form an average value. In the measurements with HPCG, on the other hand, the temporal variations range from $\pm 0.76\%$ to $\pm 1.01\%$ and are therefore larger. However, if these variations are compared with the deviation of the outliers to the mean with 1.99% to 53.70% less performance, it is also sufficient to perform one measurement. Thus, we only carry out a warm-up phase and one valid measurement in each case.

Because CLAIX2016 is longer in operation and is also measurably the more stable system, we start our investigations on this cluster, as described in Section 4.1. Subsequently, in Section 4.2 the new measurements and experiments on CLAIX2018 are depicted. Since continuous scheduling adjustments are made with SLURM and other settings are changed in the installation process, we hope to see improvements in stability through the elapsed time. In addition, an investigation with continuous measurements over a period of one and a half months is carried out on CLAIX2018. Moreover, possible causal relationships between the position of the nodes in the rack and their performance are examined, based on the structure of the clusters.

4.1. Studies on CLAIX2016

By repeating the measurements with HPL on the five slowest and the five fastest nodes, we can verify the results of the first measurement. The nodes `lnm[560-562]` achieve again significantly less performance than the reference nodes. However, no qualitative statement can be made based on the temperature. Again, CPU_0 is between 4.4 $°C$ and 9.9 $°C$ warmer than CPU_1. Furthermore, temperatures between 60.9 $°C$ and 79.0 $°C$ are found on both subsets of nodes. Thus, we exclude the cooling as a possible source and cannot recognize any connection to low performance.

However, the values of the frequency are unusual. The top five nodes work with an average frequency of 2.449 GHz, while the bottom five run at only 2.249 GHz on

average. We also find that the frequency of the nodes `lnm[560-562]` is 2.2 GHz on both CPUs throughout all measurements and appears to not dynamically adapt to the workload, instruction set in use or temperature. Initially, we assumed that Intel Turbo Boost was disabled on these three nodes. However, Turbo Boost is enabled on all observed nodes. In order to verify our measurement on the kernel, we compare the frequencies with Grafana[1]. Grafana accesses a database of the hardware performance counters on all nodes and prepares the data graphically, which enables a retrospective analysis. For the time of our benchmarks, however, a higher, dynamically adapting frequency is found that does not match our measurements that show a static frequency. Since the kernel reports different frequency values than the hardware counter, we suspect problems with the installed drivers. Further research shows that the nodes `lnm[560-562]` have set different boot options than the other nodes. The command `cat /proc/cmdline` reveals that the option `intel_pstate` is disabled, which ensures the use of the CPU frequency scaling driver `acpi-cpufreq` instead of the P-state driver. Since a uniform driver is used on the remaining nodes, we assume that the usage of the ACPI CPUfreq driver was experimental purpose and is not intended in the normal operation. Consultations with the system administration revealed a total of eight nodes with disabled p-states, which were repeatedly reserved for measurements in the past and the P-state drivers were intentionally not used. After having concluded the experiments, it would have been necessary to restore to the default, before making the nodes available to other users. We found that the necessary restoring has not been conducted.

After this configuration error had been fixed and the affected nodes had been restarted, we measure their performance again. The same binaries and configurations as before are used and we measure average performance improvements of +8.55% on these nodes for Linpack. Two nodes achieve with 871 $GFlop/s$ and 873 $GFlop/s$ more performance than the average, one node performs with 840 $GFlop/s$ less than the average. Overall, we reduce the variation measured by the CoV from 1.61% to 1.52% and the new slowest node now reaches about 96.04% of the mean performance (before: 92.23%). This improvement is also reflected in the other benchmarks. We measure on average an improved performance of +2.43% for HPCG and +8.36% with STREAM on these three nodes.

Additionally, the bimodal distribution with two peaks in Figure 3.7 b) indicates that there are two populations of nodes that differ in at least one property, thus they perform differently. In further investigations, we therefore look for correlations between the performance and properties of the nodes. Since in previous experiments we were in all likelihood able to exclude the operating system as a source of error (see Section 3.2.1), the same binaries are executed on all nodes, and the inspected drivers are the same, we suspect hardware related issues. We assign the performance of the three benchmarks to the placement within the rack in so-called heat maps. This mapping is colored in a green-yellow-red color scale indicating where each value

[1] https://grafana.com/grafana

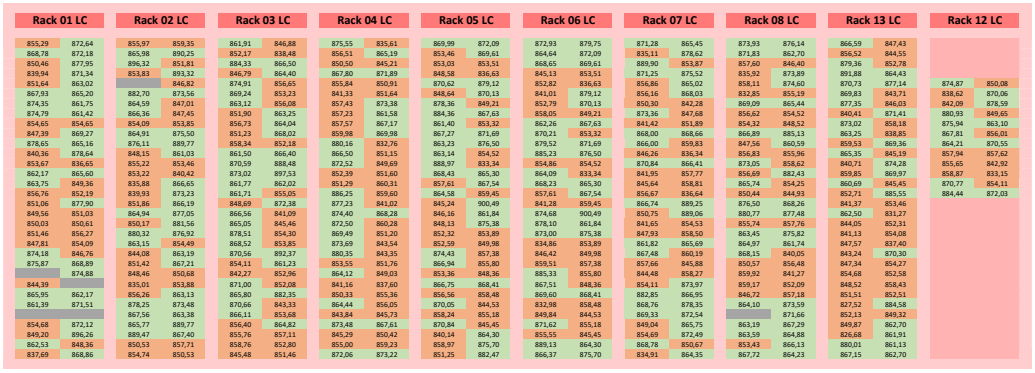

Figure 4.1.: Two-colored heat map for Linpack on CLAIX2016. Nodes of the first peak are colored in orange, nodes of the second peak in green and a gray color indicates that the node was not available during the measurement.

lies within this range of performance. Particularly high values are highlighted in green, performances in the midfield are given a yellow color and red indicates slow nodes. If a node was not available, it is grayed out. This allows us, for example, to identify inhomogeneous cooling systems, when a conspicuous number of nodes with low performance accumulates in one location. In this analysis, the newly obtained values are used for CLAIX2016 after the configuration errors have been corrected on the nodes `lnm[560-562]`. The complete set of generated heat maps can be found in Appendix B and this section only includes selected diagrams.

The larger spatial variation of the Linpack measurements is also reflected in a colorful heat map in Figure B.1. However, no significant pattern is discernible. Neither individual racks perform particularly well, nor patterns can be found with regard to placement in the rack, such as height or relative direction. To reflect on the bimodal distribution in the heat map, an additional two-colored mapping is created. We color the nodes of the first peak in orange and those with more performance in green. The result of this second mapping can be seen in Figure 4.1. However, even this only two-colored mapping cannot explain the bimodal distribution and excludes a node's position within the cluster as a possible cause for variation.

The low spatial variation measured with HPCG leads to a very monotonous color gradient of the heat maps in Figure B.3. Most of the nodes are within the yellow color range, a few nodes are greener, and red outliers are also noticeable. However, no patterns can be detected. STREAM is also further investigated with a bandwidth heat map created from the initial measurements, which can be found in Figure B.4. The mapping is free of patterns and shows no conspicuity as well.

In summary, the outliers of all benchmarks on CLAIX2016 are due to a configuration error. Otherwise, the performance is normally distributed and does not vary much, so no further investigations are carried out on CLAIX2016. Additionally, the position of the nodes within the rack does not seem to have any influence on the performance and offer no definitive explanation for the two peaks with Linpack.

4.2. Investigations on CLAIX2018

The further experiments on CLAIX2018 with HPL and HPCG took place at the same time. We start with the description of the study with HPCG, because both studies depend on the findings of the other measurement, improvements are first achieved with HPCG, and HPCG has more and extremer outliers.

4.2.1. Examining Significantly Low Performing Nodes with HPCG

Before we begin to reproduce the results and to investigate outliers, we look into the two discrete values of the temporal variations. Among other things, we aim to determine whether there are certain aspects and conditions to consider on some nodes in order to prevent temporal variation, or whether several repetitions are necessary in the following measurements in order to compensate for these variations.

We find that the nodes with about 2% temporal variation are all located within the `600` series rack of the cluster (see Figure B.6 in Appendix B). This can indicate, for example, differences in the cooling of the individual racks, which is why we use Grafana to examine the status data of these nodes. We do not find any differences in temperature. In addition, it is noticeable that among the 30 executed measurements on these nodes only one single measurement with about 11% less performance occurs. We moreover notice that by simultaneously submitting our jobs on the weekend, when the cluster is typically not heavily used, several jobs start at once. Thus, all 395 jobs that experienced temporal fluctuations of 2% were executed at the same time. Further analysis reveals that the power draw, CPU load, and memory bandwidth are all simultaneously dropping on all each nodes for about ten minutes. Since this is a non-representative, non-reproducible effect, we exclude this single run from the evaluation as a measurement error. For the nodes that run at said incident, only 29 executions are thus analyzed. This reduces the temporal variation on average to 0.43% (before: 1.05%) while the average performance increases to 37.37 $GFlop/s$ (before: 37.31 $GFlop/s$). This small temporal variation allows us to perform only one warm-up run and one measurement per node. If a performance heat map is now created (Figure B.7), again the position of the nodes in the cluster does not seem to correlate with the performance.

To validate the previous results, we remeasure the fastest three nodes and the slowest eight nodes as identified in the initial HPCG runs in Section 3.2.2. The script for monitoring the temperature and frequency of the individual cores results in approximately 5% overhead. Therefore, average values of around 35.64 $GFlop/s$ are expected. Since we find significant differences in the first run of these remeasurements compared to the initial runs, we decide to observe these eleven nodes over a longer period of time. In the course of this, we try to measure the day-to-day variation by running HPCG at different times of the day and on different days of the week. The results are shown in Figure 4.2. On the abscissa, there is a time line

and the performance in $GFlop/s$ is illustrated on the ordinate. In addition to the measured performance, the ⟳-symbol on the abscissa indicates a reboot of nodes in the corresponding color.

Of the former eight nodes identified as slow outliers, only three nodes remain slow. Some of the outliers achieve twice as much $Flop/s$ as before and perform as good as an average node now. The variations of the performance from measurement to measurement all lie within the temporal variations of the respective nodes. During our observations, the node `ncm0583` was randomly restarted on July 28th, presumably due to a node failure. Afterward, we detect an increase of 22.50% in performance, improving it to the same level as the other reference nodes. The node maintains the same improved performance during subsequent measurements until the end of our study. Therefore, we investigate the effects of planned restarts and whether this result is reproducible. On August 7th, we hence intentionally reboot the remaining slow nodes `ncm0085` and `ncm0610`. In the first measurement after the reboot, both nodes achieve respectively 19.36% and 21.97% more $GFlop/s$ than before. In addition to HPCG, Linpack is also used to examine whether changes in performance can be detected after the reboot. However, only differences in the per thousand range are found.

While the HPCG performance remains measurably improved on the nodes `ncm0583` and `ncm0610` over a longer period of at least three weeks, `ncm0085` drops back to 29.90 $GFlop/s$ after less than 29 hours. Between these two measurements, only three further jobs of other users have been executed on this node. We also notice that the first run after the restart consumes an average power of 358 W, is 70 $°C$ warm, and clocks with a frequency of 2.504 GHz. In the second measurement with less performance, a power consumption of 345 W, an temperature of 70 $°C$, and a frequency of 2.532 GHz was measured on average. Especially when the frequency is compared at core-level, significantly increased values of 2.6 GHz, compared to

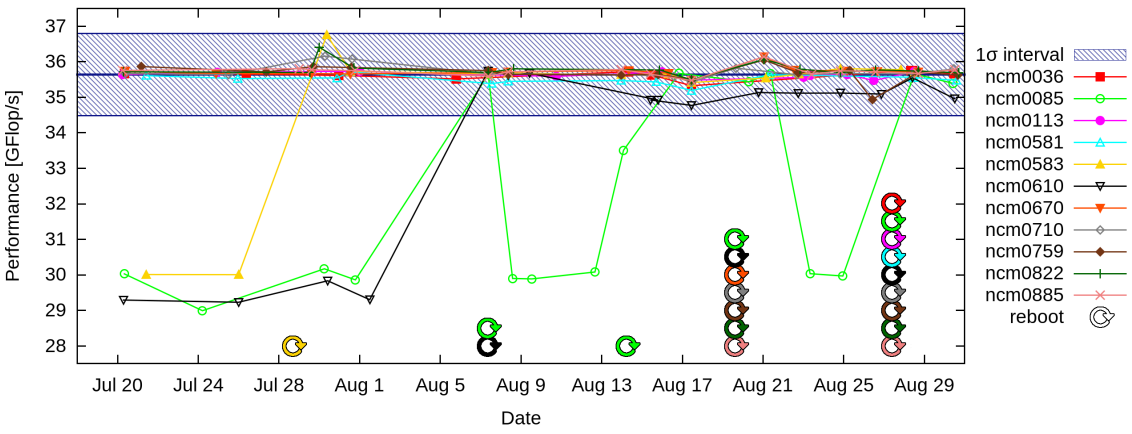

Figure 4.2.: Monitoring the performance of selected CLAIX2018 nodes over a longer period of time with HPCG. Reboots of nodes are indicated by the ⟳-symbol in the corresponding color.

the normally expected 2.5 GHz, are noticeable. Based on the observation, that the power consumption of modern CPUs is roughly proportional to the third power of the clock frequency [19], an increasing frequency results in a larger power draw and thus in an increasing performance. However, since we measure an increase in frequency and a decrease in the performance and power consumption at the same time, something else has to happen. The execution of HPCG on all cores under Turbo Boost should clock with a frequency of 2.5 GHz. However, since we find some cores with frequencies of 2.6 GHz, this is not the case. In addition, the lower energy consumption indicates that not all AVX2 units are used, so we assume that the CPU is used by other processes and is still partially occupied.

To test this hypothesis, we divide the following job into two parts. The first step is to idle for 25 minutes and to output the *load average* at equidistant intervals of five minutes, which is followed by an ordinary HPCG run. In addition, the job is monitored with our own scripts and retrospectively with Grafana. As a reference node to `ncm0085` we choose `ncm0670` and submit the same job scripts.

The load average is a measure for CPU usage [44]. Only the actual CPU load is measured and processes or threads that wait for I/O, network, or anything else that does not require the CPU are not included. It focuses only on processes that actively take CPU time and indicates whether the physical CPU was over- or underutilized in the last one, five, and fifteen minutes. The output is not normalized and must therefore be divided by the number of cores for a correct interpretation. For example, when the command `uptime` returns the output

```
20:00:48 up 50 days, 3:58, 0 users, load average:  0.18, 21.44, 48.51
```

on a compute node with 48 cores, we see that the node is in operation for 50 days and nearly 4 hours since the last restart and that no user are connected via ssh sessions. During the last minute, the node had a workload of 0.18/48 = 0.38% on average, and was utilized 21.44/48 = 44.67% of the time during the last five minutes. However, during the last 15 minutes, it had a workload of 48.51/48 = 101.06% and was therefore overloaded by 1.06%. After at least 15 minutes of idle time, we therefore expect a load average close to 0 and thus a normalized workload of 0% in all three values of the load triplet.

The results of these experiments are collected together with the previously obtained findings in Table 4.1. This includes the status, such as frequency, temperature, and power draw, of the node `ncm0085` before and directly after a restart, as well as the normalized CPU load average depending on the time in idle mode and the HPCG performance achieved afterwards. In addition to the differences in status already described, the different idle load is directly noticeable. The processors of `ncm0085`, for example, are still running at 47.94% capacity on average during the last 15 minutes, while they are idling for 15 minutes. Even after 25 minutes without starting a new process, the utilization rate is still at 17.13%. If this is compared with the same node after a restart, we notice significantly lower load averages of 0.10% and 0.13%. At the same time, the performance increases by 18.58% from

	`ncm0085` before reboot			`ncm0085` after reboot			`ncm0670`		
Frequency	2.532 GHz			2.504 GHz			2.520 GHz		
Temperature	70 $°C$			70 $°C$			70 $°C$		
Power draw	345 W			358 W			361 W		
Load average after idling for	1 min-average	5 min-average	15 min-average	1 min-average	5 min-average	15 min-average	1 min-average	5 min-average	15 min-average
0 min	22.75%	74.67%	91.48%	0.50%	0.12%	0.12%	2.25%	6.44%	7.75%
5 min	0.35%	27.35%	66.23%	0.27%	0.19%	0.12%	0.33%	0.56%	5.69%
10 min	0.00%	10.02%	47.94%	0.10%	0.13%	0.10%	0.02%	0.01%	4.15%
15 min	0.04%	3.73%	34.90%	0.02%	0.13%	0.10%	0.35%	0.54%	3.06%
20 min	0.35%	1.60%	23.60%	0.06%	0.10%	0.10%	0.06%	0.31%	2.27%
25 min	0.33%	0.69%	17.13%	0.06%	0.17%	0.13%	0.04%	0.17%	1.69%
Performance	30.08 $GFlop/s$			35.67 $GFlop/s$			35.84 $GFlop/s$		

Table 4.1.: Performance differences and measured status data of selected CLAIX2018 nodes. The frequency, temperature, power draw, normalized average CPU load after a certain amount of time in idle state, and the achieved HPCG performance is shown.

30.08 $GFlop/s$ before the restart to 35.76 $GFlop/s$ after rebooting. For reference, we recorded the same data on the node `ncm0670`, which has a low idle load average of 1.69% after 25 minutes idle and reaches 35.84 $GFlop/s$. This indicates that background processes or unfinished processes of other users could still be running on some nodes, which therefore deliver significantly less performance in HPCG.

To conclude, our long-term investigation with HPCG on CLAIX2018 shows that the performance losses are due to presumably zombie or orphaned processes of other users or OS daemons running in the background. They partly occupy the processors, which decreases the performance on affected nodes. Analogous to the results on CLAIX2016, the performance does not seem to correlate with the position of the nodes within the cluster. In addition, a cluster-wide effect caused performance losses during the initial measurement in Section 3.2.2.

4.2.2. Observing Outliers with Linpack

With Linpack we start our investigation by reviewing the results of the initial measurement. For this purpose, single-node measurements are repeated on the fastest three and the slowest five nodes. We notice performance increases from 4.69% to 13.72% on the nodes that were initially classified as “slow”. The other nodes bring 99.20% of the original computing power despite the additional monitoring script. Due to longer jobs of other users, reservations, and SLURM errors, our experiments have less time points than the HPCG measurement running at the same time. Figure 4.3 documents the temporal course of the performance. In the figure, the 1σ interval of the expected performance is highlighted and the ⟳-symbol indicates restarts. During the first four weeks, we do not notice any significant differences in the per-

formance of the nodes that do not lie within the temporal variations. Given the success of rebooting slow nodes and mitigating non-terminated processes of other users, we request reboots for all observed slow HPL nodes and measure the idle load average.

When observing the idle load, processes that continue to consume resources can neither be detected on nodes with high performance, nor on those with less performance. Likewise, we do not detect any significant changes after planed restarts. Thus, the low computing power in Linpack results is apparently not caused by zombie or orphaned processes.

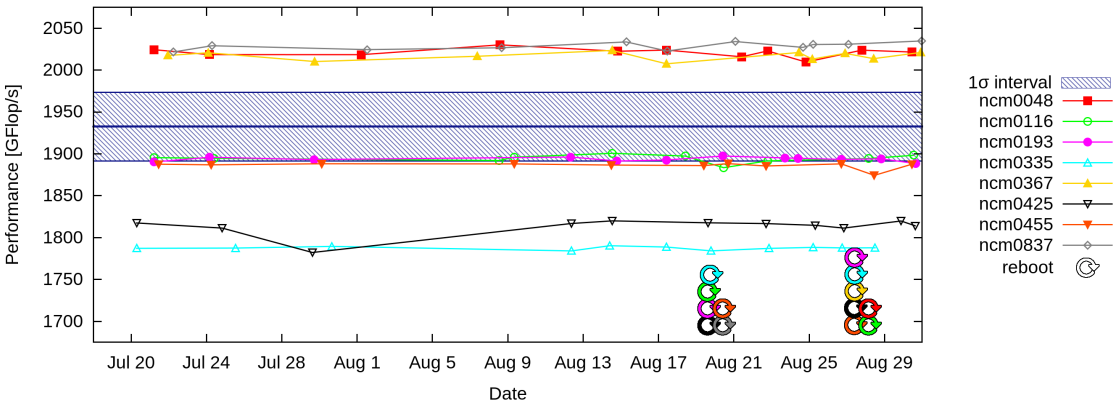

Figure 4.3.: Monitoring the performance of selected CLAIX2018 nodes over a longer period of time with HPL. The blue highlighted area is the expected performance within the 1σ interval. Reboots of nodes are indicated by the ⟳-symbol in the corresponding color.

Because we do not notice any additional load on the slow nodes and because we exclude several issues that can be caused by the software, such as the OS, usage of different drivers, and background processes, the hardware is again considered. Again, we compile a performance heat map, which can be found in Appendix B as Figure B.5. Linpack has a larger spatial variation, which is almost normally distributed. This can be perceived in the colorful mapping in many different shades. The placement of performance in the second cluster also appears to be randomly distributed for HPL within the racks. Slow and fast nodes can be found in every rack and at every position. Thus, the cluster appears to be cooled homogeneously and there seems to be no correlation between the performance and the placement within the cluster.

In summary, the performance variation in the measurements with Linpack is not influenced by processes running in the background. Additionally, the position within the server rack has presumably no influence on the performance. Likewise, we do not detect any day-to-day fluctuations and restarts also have no impact on the peak performance of a node. Further ideas are discussed in Section 5.

5. Discussion

The results in Section 3.2 indicate that statistically significant and practically significant variations in performance occur between single-node runs on the different compute nodes of a compute cluster. With the usage of the well-known and widely accepted benchmarks Linpack, HPCG, STREAM, and FWQ, which test many relevant characteristics of a modern parallel computer, we demonstrate run-to-run fluctuations on two independent cluster systems. We are able to characterize the temporal and spatial variations of both systems, as well as to identify some sources that caused significant variability. Moreover, we can exclude other factors and lay the foundation for future work.

As soon as a job with several nodes is executed, the network of the cluster is used for communication. This almost always causes additional variation with competing access and possible multiple hops over routing and must therefore be taken into account in every multi-node run. For example, the results of simulations by Jain *et al.* [24], who investigate performance comparisons and variations for different topologies and different workloads, can be used together with our new determined node-to-node variations to further determine which topology is best suited for clusters with this particular workload and produces the least fluctuations. However, this is out of the scope of this thesis. Furthermore, the network variation is already part of several other studies [4, 24, 40, 46] and has been well investigated. We focus on the single-node variations that are always given as background noises on the different nodes and within a compute cluster and thus take a different approach.

This thesis' findings in Section 3.2.1 with the measurements using the FWQ benchmark reproduce results of Petrini *et al.* [39] and Jones *et al.* [25], showing that the operating system and its interrupts cause variation in program execution. Furthermore, the observed patterns of increasing system activity in equidistant intervals are consistent with the work of León *et al.* [29], which assigns regular interruptions and recurring patterns to as many as 735 background processes and sources on their system.

In addition to confirming related work, we furthermore show that the occurrence and length of these interrupts is the same on all nodes. The variations are statistically and practically not significant (from $\pm 0.04\%$ to $\pm 0.07\%$ within 1σ) and a user will not notice them in the execution of their programs. Only in exceptional cases (2 out of 1,624) significantly more system noise was found for a limited period of time. This suggests that the operating system does not have to be considered, when investigating fluctuations in performance at the node-level when the same ker-

nel version and patch level is installed on the whole system. Hence, to the best of our knowledge, the operating system can be ruled out as a source for node-to-node fluctuations.

For this reason, the FWQ benchmark is particularly well suited to detecting faulty configurations. Problems with the OS on individual nodes lead to an increased amount of interrupts. So if there were any problems with the OS on individual nodes, they would immediately be recognized as they would stand out from the otherwise variation-free cluster.

The measurements in Section 3.2.2 indicate that the heat leak from adjacent nodes has only a negligible effect on the performance. The results reveal that only minor performance differences occur, when neighboring nodes emit strong heat through computational-intensive applications compared to a measurement with idling neighbors. The fluctuations in performance caused by this waste heat are only within 0.25σ and therefore statistically not significant. Thus, it is not necessary to consider the workload of other nodes because their emitted heat does not affect our single-node measurements significantly in practice.

The temporal variations on a single node ranges, with a confidence interval of 1σ, from $\pm 0.07\%$ (see Section 3.2.2) to 1.05% (e.g. Section 3.2.3), regardless of which cluster or node is used. These results signify that performance tuning can be correctly evaluated on these clusters as long as the expected performance gains do not fall below 1.05%. In practice, developers often try to achieve performance improvements in the range of 5% to 10% [4]. Thus, this thesis confirms that the tuning of the runtime of applications in practice is advisable, if the programmer evaluates the changes to the program code on the same compute node and if the aimed gains exceed the maximum measured temporal variations of 1.05%

In addition, individual single-node measurements on our systems have a maximum accuracy of 1.05%. If higher accuracy is required, several repetitions to determine a mean value are inevitable.

It is further noticeable that the variation is greater for memory-bound applications (like HPCG and STREAM) than for compute-bound benchmarks (HPL). A programmer should thus be aware of how his application behaves and whether its kernel's performance is limited by the memory bandwidth or the processor's compute capability in order to avoid possible errors. Hence, some memory-bound applications, as they often occur in scientific computing, might have greater temporal fluctuations than computation-intensive codes.

If the spatial variation of the peak performance is taken into account, we find a normally distributed computing power on both systems, which varies by 1.52% and 2.01%, respectively. On CLAIX2016, three single nodes stand out with significantly low performance, which were successfully attributed to a configuration error leading to an average performance loss of 7.82% and whose correction contributes to the stability of the cluster in all three benchmarks (see Section 4.1). Individual compute

nodes are often reserved for separate measurements and are deliberately configured differently from their default settings. If these systems are excluded from the normal batch operation, no problems arise. However, if these changes are neither noticed nor reset before they are released to the normal user, this can lead to significant performance losses. Therefore, it would be beneficial to have an automated system to detect and resolve any configurations that might deviate from the default settings. Alternatively, a hard reset to restore the default values could be routinely carried out, after an experiment has been completed.

With the exception of this one error, the peak performance on CLAIX2016 is furthermore bimodal and has two peaks. This suggests that there are two populations of nodes, which perform differently and therefore are expected to differ in some aspect. Hence, this thesis examines whether connections between the performance of the nodes and their placement within the machine can be determined. For this purpose, various position heat maps can be found in Appendix B, which illustrate the performance of each node within the server racks in every evaluated measurement. Since the two-color heat map in Figure B.2 shows that the nodes of the two peaks are seemingly distributed randomly in the cluster, other factors must cause the differences in performance. As already mentioned, the heat maps of the other benchmarks do not show any patterns or correlations as well. Neither some racks, nor certain heights or sides favor particularly good or bad results. Conversely, this means the cooling system of both machines ensures a homogeneous cooling without individual nodes being affected by strong waste heat. Another possible assumption is again based on the hardware. For example, the CPUs could be produced in two different batches, which can be checked by means of the serial number. The verification of this hypothesis would be associated with removing all processors from several hundred nodes and therefore lies outside the scope of this thesis. The spatial Linpack variation on CLAIX2018 is also probably due to hardware variation. In Section 4.2, no day-to-day variation is measured, so that temporal influences can be excluded to the best of our knowledge. For example, even tiny differences in the application of the heat conducting paste or in the manufacturing of an arbitrary hardware part can suffice for a node to systematically achieve less performance than other nodes.

To keep performance fluctuations in compute-bound kernels that are executed on multiple nodes on CLAIX2016 low, the cluster could be divided into two partitions. For example, Bhatele *et al.* [4] show that job fragmentation has only an insignificant impact on the performance. The performance is only decreased, when the job of interest uses the network a lot and is surrounded by other strong communication. The nodes of the lower peak on CLAIX2016 can form the first partition and the remaining one the second. The scheduler can internally assign that multi-node jobs only run on nodes of the same partition. This allows the spatial variation of the measurements to be reduced and decreases the amount of load imbalances due to uneven performance between individual nodes. It is therefore advisable to implement specialized compute-bound scheduling on CLAIX2016 in order to reduce variation in certain cases as a workaround and evaluate the effects in future work.

The spatial variation of memory-bound applications are mainly caused by single infrequent outliers, which achieve in the worst case only 41.31% of the average performance and result in strongly left-skewed distributions (Section 3.2.2). This causes serious problems especially with multi-node jobs and with static load balancing because one slow node is sufficient to slow down the entire run. Further experiments reveal a correlation between unfinished processes running in the background on some CLAIX2018 nodes and their HPCG performance. Their source and frequency has not yet been clarified and is thus to be further researched.

A possible source are jobs from other users. All nodes are available in the normal batch operation and we measure significant performance losses after the execution of three jobs by other end users. Individual processes can become orphaned or continue to run in the background as zombie processes. Hence, these three application executions can be considered whether they can cause this phenomenon on the corresponding node.

There is also the possibility that individual OS daemons may consume more resources on some nodes, or that problems may occur with the SLURM workload manager. These additionally possible causes should also be investigated in order to effectively prevent the occurrences of outliers in the future.

If the outliers identified on CLAIX2018 are disregarded, the performance with HPCG and STREAM is approximately normally distributed and slightly left-skewed with a spatial variation of 0.18% and 0.26%, respectively. Likewise, the spatial variation of these benchmarks on CLAIX2016 is at a low level (0.11% and 0.33%) and shows no outliers. This is an indicator that some memory-bound applications and some important scalable applications, but not all, might run with virtually no significant node-to-node variations on a large part of the nodes. Thus, efforts should be made to handle the few outliers. This will significantly improve the system's stability. Considering the spatial variation, a programmer should analyze whether their program code is compute-bound or memory-bound, in order to be able to better estimate the expected variation.

In addition, we find uptimes of well over 50 days on about 90% of all nodes on both systems and only a few nodes were restarted in between. On the one hand, this means that the individual nodes run very reliably and on a stable operating system. They last a long time without crashing or having to be restarted for maintenance and administration purposes.

On the other hand, the long times between suggestive reboots can lead to problems because critical updates can require reboots. Additionally, the longer a computer is in uninterrupted operation, the higher is the probability of unwanted terminated processes that continue to occur in the process table and consume a small amount of resources or of processes whose parent process has already been terminated. Thus, it should be investigated how often restarts should be performed. The dialog systems for log-in, for example, are rebooted every Monday at 6am. It must therefore

be considered whether it benefits to restart all nodes every x days, which consumes 5,328 core-hours with an average downtime of 5 minutes, or whether a restart should only be performed before larger measurements with multiple nodes.

Various studies have shown that variation is constantly increasing [36] and will be a growing problem in the future [1, 26, 27]. Therefore, it would be interesting to run a VARIATION500 list that ranks the variation in performance of the TOP500 supercomputers. This would allow the HPC community to point out the problems of performance variation and create more awareness, which an individual customer cannot do. Moreover, already known issues, such as the increasing hardware variation, could be better addressed. The list could additionally contribute sustainably to a better understanding of the sources of performance variation, as it would collect more data and would encourage more research.

At the time of these measurements, CLAIX2018 is a relatively new cluster that has recently completed its pilot phase and might further improve its variability characteristics and thus its performance after troubleshooting. With regard to the TOP500 list, computers can be recognized that have significantly improved their first reported performanceafter installation in later runs with identical hardware and same theoretical peak performance. For example, Sequoia at the Lawrence Livermore National laboratory improved by 5.20% from 16,423.8 $TFlop/s$ to 17,173.2 $TFlop/s$ while the theoretical peak performance remains at 20,132.7 $TFlop/s$ and while the total cores count remains the same in both cases [12]. The Cray XC40 system at the United Kingdom Meteorological Office and Cascade at the Pacific Northwest National Laboratory are further examples, who enhanced by 4.05% and by 8.24%, respectively. Since the hardware is seemingly unchanged, the gained performance could be due to less variation, or other improvement to the software.

The installation of a new cluster is usually followed by several acceptance-, stress-, and burn-in tests [37]. The operator of a cluster places the requirement on the manufacturer that all nodes must achieve a minimum performance [34]. Otherwise, the node is regarded as faulty, will be further examined and possible replaced. Practical experience and our results conform that this regulation is considered reasonable. Further, it is feasible to expand the requirements in the future to reduce the spatial variability within a cluster. With a normally distributed performance, less variation is expected on average compared to a strongly skewed distribution or with several peaks. Thus, besides to define a minimum performance threshold for each node, a spatial variation restriction can be established. This can specify that the achievable performance should be distributed normally across the cluster and should moreover vary spatially below a fixed threshold.

Furthermore, we recommend that these tests are not only performed immediately after installation, but also repeat them as a regular maintenance work. On the one hand, otherwise unnoticed configuration errors can be detected, which improves the stability of the system. In addition to our results, other studies also report on the importance of long observation periods in the investigation of the causes of variations [4, 38, 41]. On the other hand, the performance of individual nodes can

change over time. Individual circumstances can alter, such as that some hardware components' characteristics change with increasing age [17], and problems often occur more frequent after an installation. Therefore, frequent health checks can help to guarantee a stable performance. A first approach is laid with this thesis, which documented the performance and its variation and investigated significantly low performing nodes over a longer period of time. A collection of scripts with the presented benchmarks is composed into a toolkit, which enables automated measurements and monthly reports. These evaluation also flag nodes with significantly downwards deviating performance and enables to separately monitor these outliers. More information and details can be found in Section 6, where we go in detail and explain its usage.

6. Toolkit for Automatic Measurements and Monthly Reports

In the course of this thesis, a toolkit is developed, which automates the submission and evaluation of benchmarks. Job scripts of the presented benchmarks are fully automatically generated, submitted, and evaluated after a successful termination. The gained measurement data are stored in a table with a clear structure and graphs are created. This allows automatic monthly measurements and reports to be integrated into the daily operation of CLAIX2016 and CLAIX2018, which generates data to facilitate investigation and a better understanding of performance variation over time and system age. This data can be used for future work in the HPC group. This tool is strongly adapted to the clusters of the RWTH Aachen University, their hardware, and the used software. Therefore, transfers to other systems may require further adjustments.

In the following, the kernels integrated in this version and the mechanics of usage of the toolkit are explained.

6.1. Kernels

Since the toolkit is to be used beyond this thesis and in order to enable a smooth integration into the day-to-day operations of the systems, not all benchmarks presented in Section 3.1.1 are included, although we cannot rely on a single benchmark either. For example, our results show that nodes that deliver a significantly low performance in one benchmark do not necessarily have the same low performance in other benchmarks. Likewise, a single benchmark cannot judge the overall performance of a system or consider all aspects. In addition, each measurement should consume as few computing resources as possible. Thus, benchmarks with short runtimes are selected or configured in a way that conserves resources. It is recommended to continue the measurements with the three benchmarks STREAM, Linpack, and HPCG, which are described in detail in Section 3.1.1. We exclude FWQ because we have found no significant fluctuations and each measurement generates roughly 70 GB of data that must be evaluated and stored.

To measure the memory bandwidth, we use a multi-threaded STREAM benchmark. With its very short runtime, even 100 runs can be measured again. Depending on the cluster, 24 and 48 OMP threads are used to obtain the maximum bandwidth.

The variation in compute power is determined using the compute-bound Linpack benchmark, as well as the memory-bound benchmark HPCG. However, the benchmarks are configured to consume as few resources as possible, but still to reflect on the variation at the node-level. The runtimes are set to approximately five minutes and three repetitions in this toolkit. This means that the HPCG results cannot be used for official TOP500 measurements, which require a running time of 30 minutes, but we still achieve a similar performance. Linpack's runtime and measured performance depends on the problem size N. This means that the maximum peak performance is only achieved with the largest N possible. However, we configure Linpack to have a runtime of about five minutes to conserve resources. Thus, we measure a sub-optimal performance, but can still reflect on the variation between runs and clusters. In addition, warm-up phases of the hardware are also considered at this point.

The toolkit has a modular structure and can easily be extended by further benchmarks. All previously presented benchmarks are already included and can be deployed with little effort. Further benchmarks can be added easily, and the evaluation can be adapted to meet future requirements easily.

6.2. Mechanics of Usage

This section describes the main functions and use of the toolkit, i.e. the preparation and installation, as well as the evaluation are explained.

6.2.1. Mechanics of Building

The toolkit is located in the GitLab repository `2019_BA_Hanneken_Performance-Variation`[1] in branch `develop`. To customize the project ID under which the jobs are submitted, the user can specify it in line 8 of the `make.sh` script. If `none` is specified, the system works under the user account and therefore without a project ID. To build, one simply has to navigate into the top-level directory and execute `sh make.sh`, which generates the jobscripts for STREAM, HPCG, and HPL for CLAIX2016 and CLAIX2018. To keep the data comparable, the binaries are not recompiled, but Linpack and HPCG of the MKL and an already compiled STREAM benchmark are included in the `sourcecodes/` directory. Building also generates folders for storing the results (`data/`) and the `run.sh` script that is used for running and evaluating series of measurements.

[1] `https://git.rwth-aachen.de/hpc-thesis-artefacts/2019_ba_hanneken_performance-variation`

6.2.2. Mechanics of Running and Evaluating

In lines seven through twelve of the `run.sh` script, the user can specify which of the pre-installed benchmarks shall run in the current measurement. Nevertheless, all measurements carried out previously are evaluated. With the command `sh run.sh`, the main script is executed, which starts to evaluate the previously executed measurements by first canceling not yet started but submitted jobs. At the same time, new folders are created in which the evaluation is saved. The folders are given the current date as name. For each benchmark and cluster, a sub-directory is generated, which gives us the structure `data/<date>/<benchmark_ma- chine>/`. With the help of Gnuplot (version 4.7, also tested with version 5.3), the measurements are evaluated statistically. The evaluation generates graphs for each benchmark and saves all data for future use. In addition, a table is created under `data/statistics.dat`, which is supplemented by the new results after each evaluation. We record the average performance, its standard deviation, the coefficient of variation, and the quotient of min/mean.

The generated output files are moved to a sub-folder, which can also be found under `data/<date>/<benchmark_machine>/outputfiles`. A maximum of 200 MB of data is generated per benchmark.

6.2.3. Remarks and Recommendations

In Section 4.2, we continuously measure individually selected nodes at intervals of one to seven days over a period of one and a half months. Since the temporal variation on normal, not further significantly fluctuating nodes did not exceed a CoV of 1.05%, it is sufficient to measure all nodes once a month. Previous work by the HPC Group has shown that the average energy consumption at the weekends on CLAIX2016 is significantly lower than the average energy consumption on weekdays, which leads to the conclusion that the cluster is used less at the weekend [5]. Therefore, it is advisable to start large measurements on all nodes in these empty spaces at the weekend with a low priority. This will reduce the load on the normal batch operation as much as possible. The measurements on all nodes form the basis for further investigations that focus only on some slow nodes. Outliers of this measurement (deviation of more than 2σ) should be considered more often and with additional tests. Since fast nodes have no negative influence on the performance of the system or on multi-node jobs, we also recommend that only significantly slow nodes are observed. With this toolkit, only outliers that deviate downwards are recognized and are marked in the file `data/outliers.dat`. However, it may help to continue using a few normal nodes as a reference if necessary. As an additional test the idle load can be measured with the command `sh run.sh idle` of all discovered outliers on the list. The repeated measurement of these outliers can also be performed and evaluated with `sh run.sh outliers` with the selected benchmarks (STREAM, Linpack, and HPCG). This is recommended on a weekly basis in order to examine the fluctuations and evaluate them.

7. Conclusion and Future Work

The goal of this thesis was to investigate the node-level performance variability in parallel applications running on HPC platforms. In experiments on two different parallel computers, we measured the order of magnitude of the temporal variations of the individual compute nodes of a parallel computer in single-node runs Moreover, we determined how strongly the spatial variation at cluster-level influences the performance on these machines, and were able to identify sources of possible variation, and excluded some other factors. Furthermore, a toolkit was created as part of this thesis, which enables the automated measurement and evaluation of benchmarks. This allows the performance, and thus its fluctuation, to be monitored over a long period of time during the normal operation of a cluster and additionally, it allows to identify outlying nodes. The obtained data can form the basis for future work, might contribute to a better understanding of the variations with increasing age of the clusters, and might help in the evaluation of new variation-aware programming techniques.

We investigated the influences of neighboring jobs that emit strong heat by performing compute-intensive applications, of the operating system in the form of interrupts, and of a node's position within the server rack and conclude that these effects only have an insignificant correlation with performance variation. When adjacent nodes radiate waste heat, small differences in performance are measurable. These performance decrements are not further considered with a maximum deviation of 0.25σ. Similarly, our experiments and measurements showed that the OS is a potential cause for variation, as it interrupted the execution of user applications, but the occurrence and length of interrupts did not differ significantly from node to node and therefore did not cause any differences on the node-level. The placement of nodes within a cluster, such as height, relative direction, and distinct racks, show no correlation either. Thus, these factors can be excluded as possible causes for fluctuations in performance on the node-level.

The temporal variation is at a similar level on all nodes on both clusters. However, it must be distinguished whether an application is limited by the computing power or the memory access. Memory-bound applications vary in time on our systems by up to $\pm 1.05\%$, while compute-intensive applications have negligible low fluctuations of $\pm 0.25\%$. Thus, a user must be aware of how their applications' performance is limited and that performance tuning can only be evaluated, if the expected performance gain is greater than 1.05%. Likewise, individual single-node measurements have a maximum accuracy of 1.05% and several repetitions are necessary, when a higher precision is needed.

Excluding outliers, the spatial performance is approximately normally distributed. Moreover, the restriction of an applications' performance is important again. If a kernel is compute-bound, fluctuations of up to $\pm 1.98\%$ occur, while memory-bound code only varies up to $\pm 0.33\%$. However, the spatial variation becomes significantly greater, if outliers are also considered. For example, we measured performance losses of up to 58.69% (16.59σ below the mean value) on individual CLAIX2018 nodes with HPCG. Further testing and analysis has shown that presumably zombie and orphaned processes or OS daemons run in the background and consume resources, even when the node is otherwise idling. One possible solution to this phenomenon is to restart the affected nodes. However, this is not optimal, so that further research is needed to identify the source of the occupancy of the CPUs and to determine its frequency in occurrence. The restart cycle of all nodes can also be examined to determine whether the period between successive restarts should be adjusted or whether one reboot should be performed before large multi-node jobs. Additionally, slow nodes on CLAIX2016 in different benchmarks revealed a configuration error on some nodes. This again illustrates how important it is to recognize and prevent the infrequent outliers. In the future, data centers should therefore pay particular attention to individual outliers because they significantly worsen a system's stability, while otherwise a stable performance was found.

Moreover, it is interesting to investigate the magnitude of performance variation in jobs with multiple nodes. The single-node data collected within this thesis for each node can serve as a basis for this future work. Neglecting the variation by the network, Monte Carlo methods can be used to perform simulations to determine the average variation of a multi-node job. These network effects and the utilized network architecture, as well as I/O influences, can be investigated further. Since individual nodes are connected by more hops than others due to the network topology, or compete over the shared resource, variations on the node-level are expected here. With further data, such as the average node request for jobs, and the scheduling procedure with SLURM, an expected value for the variations of multi-node measurements can be determined more elaborated.

In summary, significant variations in performance on the node-level of modern parallel computers are mainly induced by infrequent slow outliers that impair the stability of the whole system. Otherwise, the performance is normally distributed and varies from $\pm 0.07\%$ to $\pm 2.01\%$ within 1σ. Temporal variations are particularly small in compute-bound kernels, while memory-bound code tends to experience smaller spatial variations.

Acknowledgments

This work would have never been successful without the incredible support and encouragement of my family, friends, and supervisor.

I want to thank Daniel Schürhoff for his supervision and constructive criticism during this work. He supported me continuously, contributed to the success of this thesis with his help and guidance, and provided valuable feedback in many meetings.

My appreciation is also expressed to the Chair for High Performance Computing and various staff members that never ceased to help and suggested new ideas. Furthermore, simulations were performed with computing resources granted by RWTH Aachen University under project `thes0552`.

Last but not least, I express my deep gratitude to my family and friends, who supported and believed in me through the last three years of my studies and this work.

A. Multi-Threaded STREAM with Different Number of OMP Threads

STREAM is also further investigated on both systems with bandwidth heat maps created from the initial measurements. They can be found in Appendix B as Figure B.4 and Figure B.8. Both mappings are free of patterns and shows no conspicuity, so additional experiments are conducted to examine the outliers. For this, we consider additional multi-threaded STREAM measurements with one to 24 or 48 OMP threads each, depending on the system The threads are each bound to a physical core with the `spread` binding policy.

CLAIX2016 The memory bandwidth from one to 24 OMP threads with the standard derivation and the occurring variation is shown in Figure A.1. As the number of threads increases, the reached bandwidth also increases overall. Especially from one to twelve threads, it increases strongly and already reaches 82.59% of the maximum bandwidth at twelve threads. In addition, the variation is particularly large for one to six threads. Hence, the best bandwidth per variation rate is reached with 24 threads.

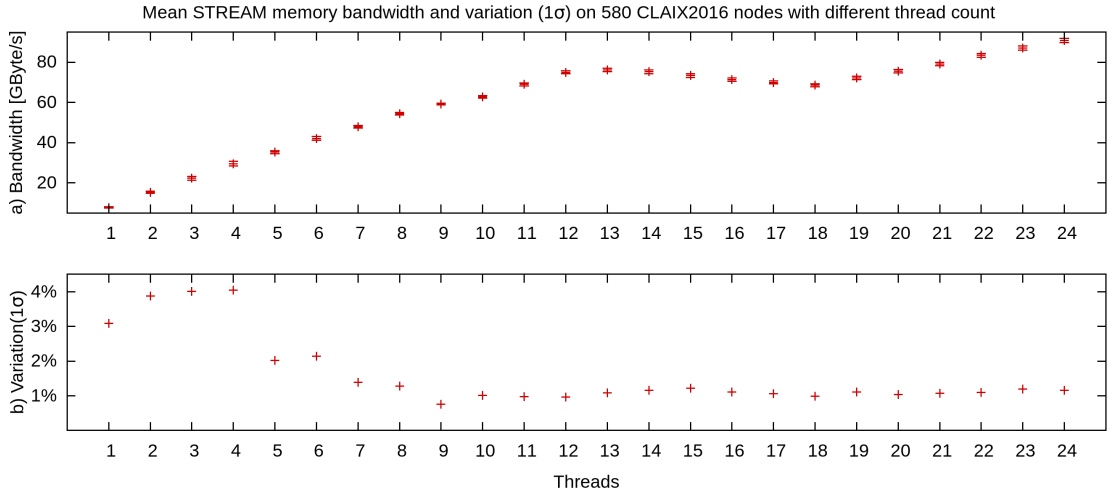

Figure A.1.: Memory bandwidth with different numbers of OMP threads on CLAIX2016.

CLAIX2018 As expected, the bandwidth increases as the number of OMP threads increases. The maximum bandwidth is measured with 48 threads (see Figure A.2). Peaks at ten, sixteen, 26 and 29 threads can be observed. In addition, the bandwidth of one to sixteen threads increases much more than in the other range, reaching already 77.99% of the maximum bandwidth with sixteen threads. Furthermore, the variation is particularly high when using one to seven, ten, 47, and 48 threads. Hence, the best bandwidth per variation rate is reached with 22 threads, which reaches 67.79% of the optimal bandwidth and reduces variation at the same time.

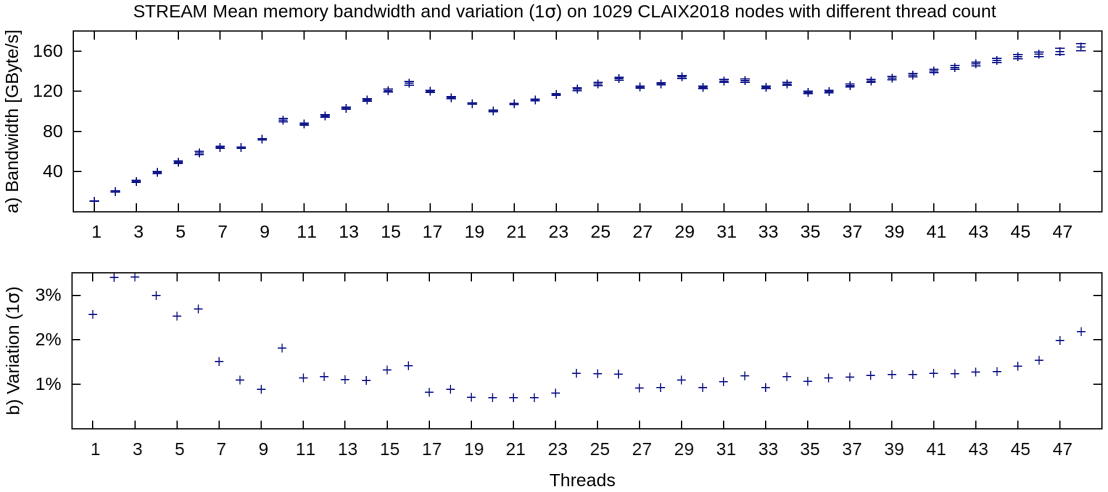

Figure A.2.: Memory bandwidth with different numbers of OMP threads on CLAIX2018.

B. Performance-Placement Heat Maps

This section provides the figures of the heat maps from the Performance Placement Analysis from Section 4. This assignment is colored in a green-yellow-red color scale that indicates where each value lies within this range. Particularly high values are highlighted in green, performances in midfield are given a yellow color, and red indicates slow nodes. If a node was not available, it is grayed out. A more detailed description can be found in the corresponding section.

CLAIX2016

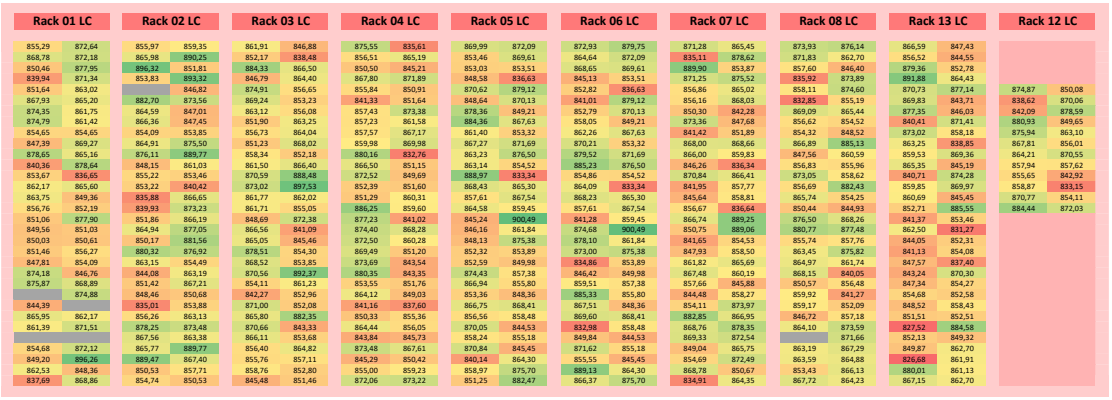

Figure B.1.: Performance placement heat map for Linpack on CLAIX2016.

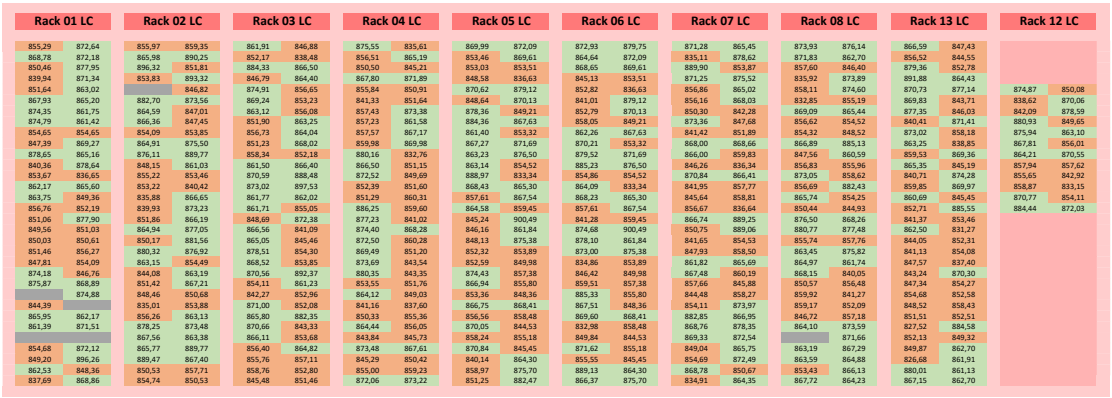

Figure B.2.: Identical to Figure 4.1. Two-colored heat map for Linpack on CLAIX2016. Nodes of the first peak are colored in orange and nodes of the second peak in green.

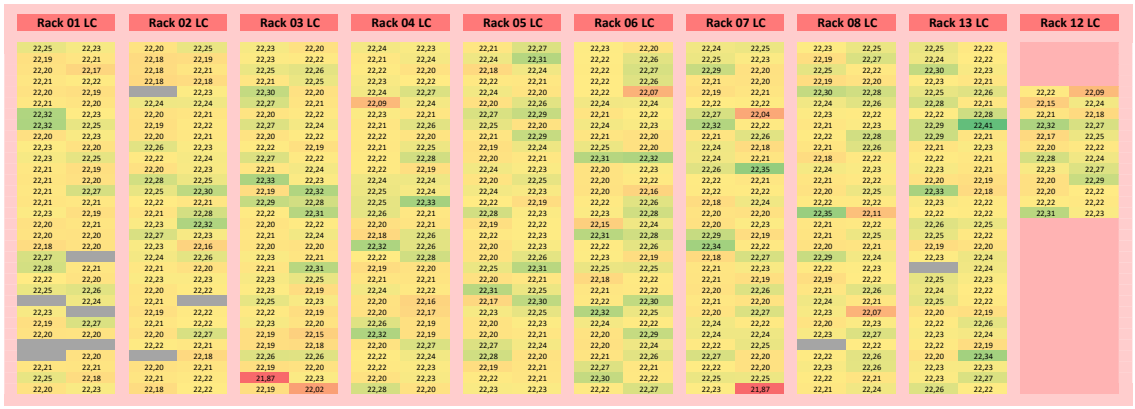

Figure B.3.: Performance placement heat map for HPCG on CLAIX2016.

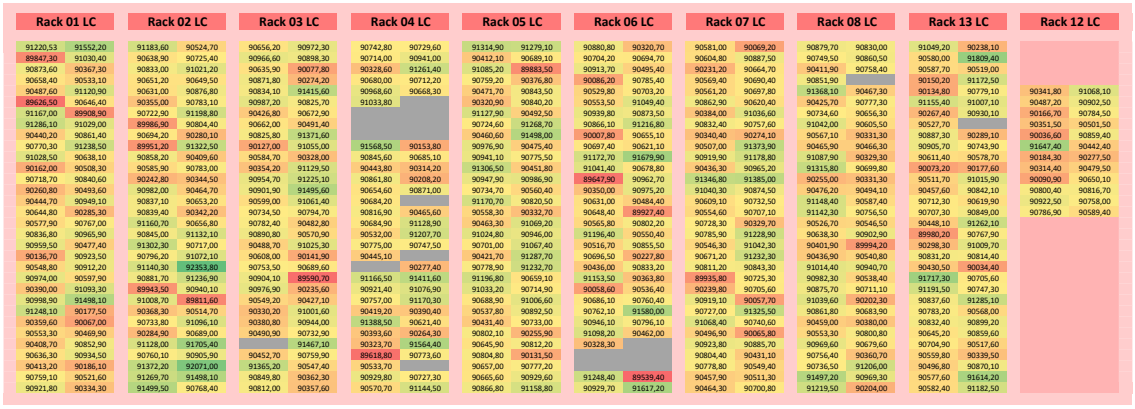

Figure B.4.: Performance placement heat map for multi-threaded STREAM (24 OMP threads) on CLAIX2016.

CLAIX2018

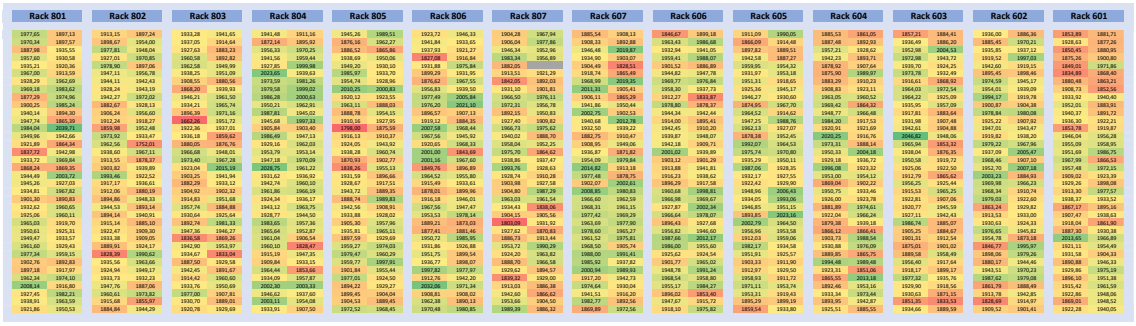

Figure B.5.: Performance placement heat map for Linpack on CLAIX2018

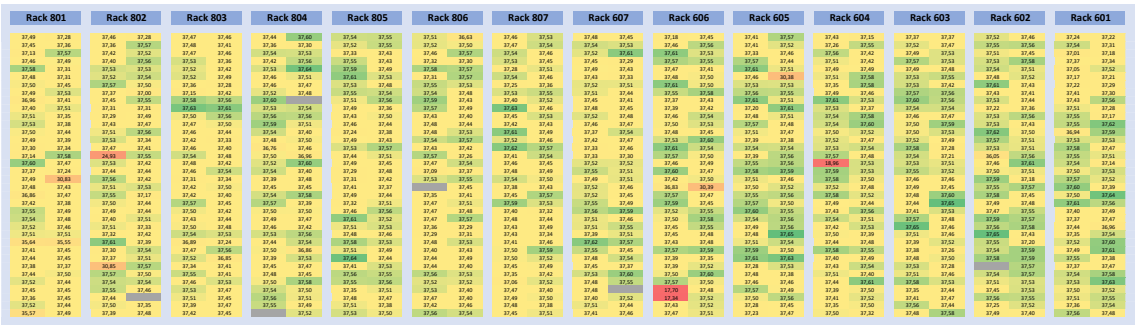

Figure B.6.: Performance placement heat map for HPCG on CLAIX2018

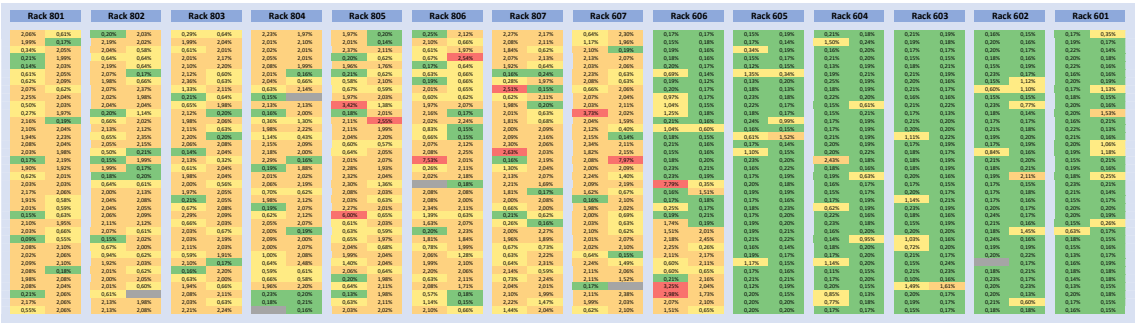

Figure B.7.: Temporal variation heat map for HPCG on CLAIX2018

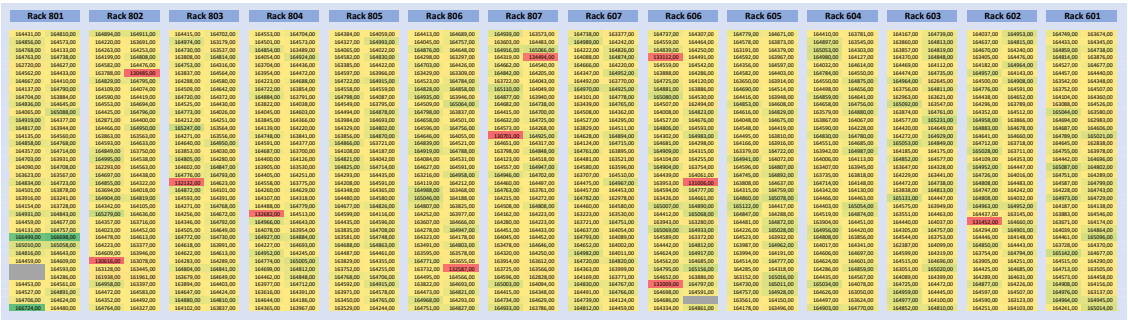

Figure B.8.: Performance placement heat map for multi-threaded STREAM (48 OMP threads) on CLAIX2018

Bibliography

[1] B. Acun, P. Miller, and L. V. Kale. Variation Among Processors Under Turbo Boost in HPC Systems. In *Proceedings of the 2016 International Conference on Supercomputing*, page 6. ACM, 2016.

[2] M. Adams, P. O. Schwartz, H. Johansen, P. Colella, T. J. Ligocki, D. Martin, N. Keen, D. Graves, D. Modiano, B. Van Straalen, et al. Chombo Software Package for AMR Applications-Design Document. Technical report, Lawrence Berkeley National Laboratory, 2015.

[3] A. Apon, S. Ahalt, V. Dantuluri, C. Gurdgiev, M. Limayem, L. Ngo, and M. Stealey. High Performance Computing Instrumentation and Research Productivity in US Universities. *Journal of Information Technology Impact*, 10(2):87–98, 2010.

[4] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs. There Goes the Neighborhood: Performance Degradation Due to Nearby Jobs. In *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 41:1–41:12, New York, NY, USA, 2013. ACM.

[5] H. Brädle and S. Wienke. HPC Centers and Electricity Service Providers - Demand and Budgeting. Unpublished seminar thesis, RWTH Aachen University, Jan 2019.

[6] J.-W. Buurlage, T. Bannink, and R. H. Bisseling. Bulk: A Modern C++ Interface for Bulk-Synchronous Parallel Programs. In *European Conference on Parallel Processing*, pages 519–532. Springer.

[7] S. Chunduri, K. Harms, S. Parker, V. Morozov, S. Oshin, N. Cherukuri, and K. Kumaran. Run-to-run Variability on Xeon Phi based Cray XC Systems. In *SC '17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 52:1–52:13, New York, NY, USA, 2017. ACM.

[8] L. DeRose, B. Homer, and D. Johnson. Detecting Application Load Imbalance on High End Massively Parallel Systems. In A.-M. Kermarrec, L. Bougé, and T. Priol, editors, *Euro-Par 2007 Parallel Processing*, pages 150–159, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[9] S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. K. De, and S. Borkar. Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor. *IEEE Journal of Solid-State Circuits*, 46(1):184–193, Jan 2011.

[10] J. J. Dongarra, M. A. Heroux, and P. Luszczek. High-Performance Conjugate-Gradient Benchmark: A New Metric for Ranking High-Performance Computing Systems. *The International Journal of High Performance Computing Applications*, 30(1):3–10, 2016.

[11] J. J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK Benchmark: Past, Present and Future. *Concurrency and Computation: practice and experience*, 15(9):803–820, 2003.

[12] J. J. Dongarra, H. W. Meuer, E. Strohmaier, et al. Website: TOP500 Supercomputer Sites, 1993-2019. URL `https://www.top500.org/`, accessed April 16, 2019.

[13] K. Du Bois, S. Eyerman, J. B. Sartor, and L. Eeckhout. Criticality Stacks: Identifying Critical Threads in Parallel Programs Using Synchronization Behavior. *ACM SIGARCH Computer Architecture News*, 41(3):511–522, 2013.

[14] M. Feldman. Website: China Will Deploy Exascale Prototype This Year, Jan 2017. URL `https://www.top500.org/news/china-will-deploy-exascale-prototype-this-year/`, accessed July 13, 2019.

[15] S. Futral. Website: ASC Sequoia Benchmark Codes, 2013. URL `https://asc.llnl.gov/sequoia/benchmarks/`, accessed Mai 07, 2019.

[16] F. E. Grubbs. Procedures for Detecting Outlying Observations in Samples. *Technometrics*, 11(1):1–21, 1969.

[17] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing Flash Memory: Anomalies, Observations, and Applications. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 24–33. IEEE, 2009.

[18] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, et al. Underdesigned and Opportunistic Computing in Presence of Hardware Variability. *IEEE Transactions on Computer-Aided Design of integrated circuits and systems*, 32(1):8–23, 2012.

[19] G. Hager and G. Wellein. *Introduction to High-Performance Computing for Scientists and Engineers*. CRC Press, 2010.

[20] M. A. Heroux, J. Dongarra, and P. Luszczek. HPCG Technical Specification. *Sandia report SAND2013-8752*, pages 42:1–42:21, 2013.

[21] Intel Corporation. Website: Products formerly Broadwell. URL `https://ark.intel.com/content/www/us/en/ark/products/codename/38530/broadwell.html`, accessed Juli 16, 2019.

[22] Intel Corporation. Intel® Math Kernel Library Benchmarks. In *Intel® Math Kernel Library for Linux* Developer Guide*, pages 95–107. Intel Corporation, 2019 Update 3.

[23] IT Center, RWTH Aachen University. Website: Hardware of the RWTH Compute Cluster. URL `https://doc.itc.rwth-aachen.de/display/CC/Hardware+of+the+RWTH+Compute+Cluster`, accessed August 30, 2019.

[24] N. Jain, A. Bhatele, S. White, T. Gamblin, and L. V. Kale. Evaluating HPC Networks via Simulation of Parallel Workloads. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 154–165, Nov 2016.

[25] T. Jones, S. Dawson, R. Neely, W. Tuel, L. Brenner, J. Fier, R. Blackmore, P. Caffrey, B. Maskell, P. Tomlinson, and M. Roberts. Improving the Scalability of Parallel Jobs by adding Parallel Awareness to the Operating System. In *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, pages 10:1–10:20, Nov 2003.

[26] B. Kocoloski and J. Lange. Varbench: an Experimental Framework to Measure and Characterize Performance Variability. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 18:1–18:10. ACM, 2018.

[27] B. Kocoloski, L. Piga, W. Huang, I. Paul, and J. Lange. A Case for Criticality Models in Exascale Systems. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 213–216. IEEE, 2016.

[28] W. T. Kramer and C. Ryan. Performance Variability of Highly Parallel Architectures. In *International Conference on Computational Science*, pages 560–569. Springer, 2003.

[29] E. A. León, I. Karlin, and A. T. Moody. System Noise Revisited: Enabling Application Scalability and Reproducibility with SMT. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 596–607, May 2016.

[30] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf. Managing Variability in the IO Performance of Petascale Storage Systems. In *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 37:1–37:12, Nov 2010.

[31] V. Marjanović, J. Gracia, and C. W. Glass. Performance Modeling of the HPCG Benchmark. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pages 172–192. Springer International Publishing, 2014.

[32] J. D. McCalpin. Performance Variability due to Snoop Filter Conflicts on the Xeon Gold and Platinum Processors with 18, 20, 22, 24, 26 and 28 cores - and Xeon Phi x200 (KNL). SC18 presentation, 2018. URL `https://sites.utexas.edu/jdm4372/2019/01/07/sc18-paper-hpl-and-dgemm-performance-variability-on-intel-xeon-platinum-8160-processors/`, accessed July 31, 2019.

[33] J. D. McCalpin. Website: STREAM Benchmark Reference Information. URL `http://www.cs.virginia.edu/stream/ref.html`, accessed July 14, 2019.

[34] J. D. McCalpin. HPL and DGEMM Performance Variability on the Xeon Platinum 8160 Processor. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 225–237. IEEE, 2018.

[35] J. D. McCalpin et al. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE computer society technical committee on computer architecture (TCCA) newsletter*, 2(19–25), 1995.

[36] S. Mittal. A Survey Of Architectural Techniques for Managing Process Variation. *ACM Computing Surveys*, 48, 02 2016.

[37] M. S. Müller, G. Juckeland, M. Jurenz, and M. Kluge. Quality Assurance for Clusters: Acceptance-, Stress-, and Burn-In Tests for General Purpose Clusters. In *High Performance Computing and Communications*, pages 44–52, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[38] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. De Supinski. Practical Resource Management in Power-Constrained, High Performance Computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 121–132. ACM, 2015.

[39] F. Petrini, D. J. Kerbyson, and S. Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, pages 55:1–55:17, Nov 2003.

[40] B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, and T. Hoefler. Efficient Task Placement and Routing of Nearest Neighbor Exchanges in Dragonfly Networks. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 129–140. ACM, 2014.

[41] D. Skinner and W. Kramer. Understanding the Causes of Performance Variability in HPC Workloads. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.*, pages 137–149. IEEE, 2005.

[42] H. Toutenburg and C. Heumann. *Deskriptive Statistik - Eine Einführung in Methoden und Anwendungen mit R und SPSS*. Springer-Lehrbuch, 6th edition, 2008.

[43] L. G. Valiant. A Bridging Model for Parallel Computation. *J Communications of the ACM*, 33(8):103–111, 1990.

[44] R. Walker. Examining Load Average - Understanding workload averages as opposed to CPU usage. *Linux Journal*, 152:80–82, Dec 2006.

[45] H. Weisbach, B. Gerofi, B. Kocoloski, H. Härtig, and Y. Ishikawa. Hardware Performance Variation: A Comparative Study using Lightweight Kernels. In *International Conference on High Performance Computing*, pages 246–265. Springer, 2018.

[46] N. J. Wright, S. Smallen, C. M. Olschanowsky, J. Hayes, and A. Snavely. Measuring and Understanding Variation in Benchmark Performance. In *2009 DoD High Performance Computing Modernization Program Users Group Conference*, pages 438–443. IEEE, 2009.