Decision Criteria and Benchmark Description

SuperMUC-NG

(21. August 2017)

**Aug 2017 LRZ-Bericht 2017-3**

Content:

[1 Decision Methodology and Terms 1](#_Toc490731229)

[1.1 Benchmark Sources 1](#_Toc490731230)

[1.2 Aggregation of Performance 1](#_Toc490731231)

[1.3 Aggregation for Different Node Types 1](#_Toc490731232)

[1.4 Quantitative Evaluation by Means of the Aggregate Compute Performance 2](#_Toc490731233)

[1.5 Qualitative evaluation and final ranking 2](#_Toc490731234)

[1.6 Award of contract 3](#_Toc490731235)

[1.7 Weights of the Benchmarks 4](#_Toc490731236)

[1.8 Criteria for the qualitative evaluation 6](#_Toc490731237)

[1.9 Evaluation Scheme 9](#_Toc490731238)

[1.10 Rules for the Benchmarks and their Verification 10](#_Toc490731239)

[1.10.1 Confidentiality 10](#_Toc490731240)

[1.10.2 Code Modifications 10](#_Toc490731241)

[1.10.3 Additional results for non-compliant benchmark versions 11](#_Toc490731242)

[1.10.4 Limitations of Code Optimisation 11](#_Toc490731243)

[1.10.5 Precautions Regarding Cache in the Intel MPI Benchmarks 11](#_Toc490731244)

[1.10.6 Conversion of Timing to Performance Values 11](#_Toc490731245)

[1.10.7 Frequency and Power Envelope Settings 12](#_Toc490731246)

[1.10.8 Rules for Systems which are not available for Benchmarking 12](#_Toc490731247)

[1.10.9 Reference Output 12](#_Toc490731248)

[1.10.10 Benchmark Report 12](#_Toc490731249)

[1.11 Commitments by the Tenderer and Submission of Results to LRZ 12](#_Toc490731250)

[1.12 Delivery of updated Commitments (compared to an Intermediate Offer) 12](#_Toc490731251)

[1.13 Preparation and Running the Benchmarks 14](#_Toc490731252)

[1.14 Verification of the Benchmark Commitments 14](#_Toc490731253)

[1.14.1 Verification for Phase 1 14](#_Toc490731254)

[1.14.2 Verification of the Improvement Ratio for Phase 2 15](#_Toc490731255)

[2 Benchmarks 16](#_Toc490731256)

[2.1 High Performance Interconnect-related benchmarks 16](#_Toc490731257)

[2.1.1 MPI Aggregate messaging bandwidth of a node 16](#_Toc490731258)

[2.1.2 MPI Messaging rate of a node 17](#_Toc490731259)

[2.1.3 MPI startup time 18](#_Toc490731260)

[2.1.4 MPI Link bandwidth and latency of a node, one task per node, intra-island 19](#_Toc490731261)

[2.1.5 MPI Bisection bandwidth and latency, intra-island and inter-island, many tasks per node 20](#_Toc490731262)

[2.1.6 MPI collective communication 23](#_Toc490731263)

[2.1.7 MPI Shared Memory Test 24](#_Toc490731264)

[2.2 Low Level and Kernel Benchmarks 26](#_Toc490731265)

[2.2.1 FFTW 26](#_Toc490731266)

[2.2.2 GRAPH500 27](#_Toc490731267)

[2.2.3 HPCG 28](#_Toc490731268)

[2.2.4 HIGH PERFORMACE LINPACK (HPL) 29](#_Toc490731269)

[2.2.5 LRZSORT 30](#_Toc490731270)

[2.2.6 PMATMUL 32](#_Toc490731271)

[2.2.7 RBench (Bioconductor, sequence alignment / pattern matching) 33](#_Toc490731272)

[2.2.8 RINF 34](#_Toc490731273)

[2.2.9 SIP 37](#_Toc490731274)

[2.2.10 SPECrate® CPU2006 of a node 39](#_Toc490731275)

[2.3 Application Benchmarks 41](#_Toc490731276)

[2.3.1 BQCD 41](#_Toc490731277)

[2.3.2 Gadget 43](#_Toc490731278)

[2.3.3 GROMACS 45](#_Toc490731279)

[2.3.4 Iphigenie/CPMD (QMMM Simulation of a Biomolecule) 47](#_Toc490731280)

[2.3.5 SeisSol 49](#_Toc490731281)

[2.3.6 SUSPENSE 53](#_Toc490731282)

[2.4 Storage Subsystem Benchmarks 55](#_Toc490731283)

[2.4.1 IOR benchmark 56](#_Toc490731284)

[2.4.2 FIO benchmark 59](#_Toc490731285)

[2.4.3 IOZONE benchmark 60](#_Toc490731286)

[2.4.4 Bonnie++ MPI metadata benchmark 61](#_Toc490731287)

[2.5 Energy efficiency 62](#_Toc490731288)

[2.6 Estimated Mean Power Draw and Energy Cost 63](#_Toc490731289)

# Decision Methodology and Terms

## Benchmark Sources

Programs, input files, and detailed instructions on the benchmarks are deposited on an FTP server of the LRZ and are subject to access control.

The Tenderer has received an FTP password and login that grants read access on the FTP server. After login via an FTP client the Tenderer will be automatically directed to the directory containing all necessary files.

ftp ftp.lrz.de  
user: sng  
password: SNGrunde2 dir

**Benchmark sources and inputs have not been changed since the previous release** for the first round of the competative dialogue.

## Aggregation of Performance

The term *aggregate compute performance* refers to LRZ’s method calculating the performance of a **fully loaded system** for any given benchmark. In simplified terms, the method can be described as follows:

The system has to be **loaded with as many as possible identical copies** of a given benchmark. The Tenderer ascertains the **minimum** **performance** for each instance of the simultaneously running benchmarks, each running on a given **fraction** of the system. LRZ then **multiplies** this performance with the **inverse** of this fraction to yield the **aggregate compute** performance of the system.

The rationale behind this is that under certain circumstances (e.g., because of bottlenecks of memory access or of the high performance interconnect) the aggregate performance for simultaneously running **n** identical copies of a program is not always simply **n** times the performance of a single program run.

It may happen that a number of nodes of the system cannot be used by the benchmark configuration. In order to account for these left-out cores in the aggregate performance, the Tenderer must

* run a version of the program with a smaller problem size on the remaining processors as a dummy program to produce some artificial workload, ***and/or***
* run appropriate copies of the benchmark HIGH PERFORMANCE LINPACK (HPL) as a dummy program to produce some artificial compute and network load workload.

The dummy programs are only used to model memory and/or interconnect bottlenecks and do not themselves contribute to the performance calculations.

Nodes or cores purely dedicated for service purposes and cores in the login cloud cannot contribute to the aggregate benchmark performance. The same applies for cores which are not used for computational work, while their attached physical memory is used by other cores running a benchmark program.

For simplicity, the command lines provided for the individual benchmarks indicate the requirement that the system

must be filled with identical copies of a benchmark by using the flags "-C ncopies".

## Aggregation for Different Node Types

If a benchmark is run on different node types (e.g., on general purpose nodes and on many-core or accelerated nodes) then the aggregation of performance is the sum of aggregate performance of each node type e.g.,

Where Pi,A is the aggregate performance for benchmark i and nodes type A. The aggregate performance for node type A is calculated by using “Total number of nodes of this type in system”. For the purpose of performance aggregation, a specific node can only belong to exactly one node type.

For benchmarks which report latencies (Ti,part) and which cannot be executed on the complete system (i.e. for MPI Link latency of a node, MPI Bisection latency, MPI RMA latency, MPI Collectives Log latency), the aggregation will be performed by summing up across system parts with weights proportional to the relative HPL performance of that system part (PHPL,part).

All inter-island latency measurements must be performed on the complete system, therefore no aggregation is needed and a single latency number will be directly evaluated.

A separate benchmark evaluation for the **large shared memory compute nodes** (fat nodes, see: *Description of Goods and Services SuperMUC-NG, Section 3.2.3*) is not required. The performance of these nodes is considered equal to that of the compute nodes with less memory. The number of nodes taken for aggregation is the sum of fat and thin nodes.

## Quantitative Evaluation by Means of the Aggregate Compute Performance

Aside from adjustments to enable the **evaluation of qualitative factors**, the decision will be based on the **benchmark performance offered for a fixed total budget** (which includes investment as well as energy, air conditioning, and maintenance costs).

The evaluation procedure aims at ranking the offered systems according to their aggregate compute performance and evaluating their relative compute performance when compared with each other. The following evaluation scheme ensures this.

To normalize the different characteristics of the individual benchmarks, the value Vi for the ***relative*** **aggregate compute performance** for each individual benchmark is defined as the ratio between to the aggregate compute performance *Pi*and the best performance Pi,best among all the offers.

If performance data (like GFlop/s, GByte/s, 1/s) are compared (“higher is better”), then

If latencies or timings etc. are compared (“lower is better”), then

In the unlikely case that no Tenderer provides results for a benchmark case, i.e. 0/0, then Vi is set to 0 for all Tenderers. Thus, for each offered system, numerical values between 0 and 1 are obtained for each individual benchmark.

These ratios will be multiplied by the weight factors gi stipulated for the benchmarks in section 1.7 and subsequently summed up:

R is denoted as the **ranking value** and is considered a measure for the relative strength of an offered system with respect to all offered systems. The weights are normalized to yield: 0 ≤ R ≤ 1.

The performance of Phase 2 is included in these considerations by prescribing a fixed improvement ratio (see document “*Goods and Services for SuperMUC-NG”* section 3.4.3 as well as section 1.14.2 of this document here).

## Qualitative evaluation and final ranking

To obtain a qualitative overall impression of the system, all benchmarks as well as other aspects of the offer will be examined for conspicuous characteristics that may have an impact on the achievable compute performance or the usability and manageability of the system.

For characteristics that conspicuously deviate from the average, a corresponding evaluation weight will be assigned. This will usually happen if a system differs substantially from the other systems or a clear failure to fulfil the requirements stated in this document is observed. Qualitative corrections of this kind are performed carefully by LRZ by judging against the state-of-the-art, and will be justified and explained.

The committed compute performance values will not be re-evaluated, since this information is already contained in the benchmark results themselves. However, the scaling behaviour or non-compliant versions of the code will be taken into account.

The criteria for applying corrections are described in Section 1.8.

The ranking number R determined according to the description given in the previous section will be multiplied with a factor 1+η in order to obtain the final qualified ranking number Q.

Q = R (1+η)

η is the capped sum of the qualitative corrections for the particular feature groups (see Section 1.8)

The final qualitative correction by LRZ is restricted to a final range of

-0.25 ≤ η ≤ 0.25

even if the summation over all feature groups would lead to lower or higher values.

## Award of contract

The contract will be awarded to the Tenderer with the highest qualified ranking number Q.

## Weights of the Benchmarks

The weights for the particular benchmarks are given in **Table 1**.

Benchmarks with weight 0% will be only used to verify the requirements defined in the document “*Goods and Services for SuperMUC-NG”* and for qualitative assessment of the offer.

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Benchmark | Weight gi  (%) | Reference  Section |
|  | **High performance interconnect-related benchmarks** | **24%** | **2.1** |
| 1 | MPI Messaging bandwidth of a node | 1.5% | 2.1.1 |
| 2 | MPI Messaging rate of a node | 1.5% | 2.1.2 |
| 3 | MPI Startup Time quarter of system | 1.0% | 2.1.3 |
| 4 | MPI Startup Time half of system | 1.0% | 2.1.3 |
| 5 | MPI Startup Time entire system | 1.0% | 2.1.3 |
| 6 | MPI Link bandwidth of a node, intra-island, pingping | 1.0% | 2.1.4 |
| 7 | MPI Link latency of a node, intra-island, pingping | 1.0% | 2.1.4 |
| 8 | MPI Bisection bandwidth, intra island, exchange/sendrecv | 2.0% | 2.1.5 |
| 9 | MPI Bisection bandwidth, inter island, exchange/sendrecv | 1.5% | 2.1.5 |
| 10 | MPI Bisection bandwidth, intra island, biput/biget | 1.0% | 2.1.5 |
| 11 | MPI Bisection bandwidth, inter island, biput/biget | 1.0% | 2.1.5 |
| 12 | MPI Bisection latency, intra island, exchange/sendrecv | 2.0% | 2.1.5 |
| 13 | MPI Bisection latency, inter island, exchange/sendrecv | 1.5% | 2.1.5 |
| 14 | MPI Bisection latency, intra island, biput/biget | 1.0% | 2.1.5 |
| 15 | MPI Bisection latency, inter island, biput/biget | 1.0% | 2.1.5 |
| 16 | MPI Barrier Log Latency | 1.0% | 2.1.6 |
| 16 | MPI Allreduce Log Latency | 2.0% | 2.1.6 |
| 17 | MPI IAllreduce Log Latency | 1.0% | 2.1.6 |
| 18 | MPI SharedMem | 1.0% | 2.1.7 |
|  | **Low Level and Kernel Benchmarks** | **38%** | **2.2** |
| 19 | FFTW | 4.0% | 2.2.1 |
| 20 | GRAPH500 | 3.0% | 2.2.2 |
| 21 | HPCG intra island | 4.0% | 2.2.3 |
| 22 | HPCG entire system | 3.0% | 2.2.3 |
| 23 | HPL intra island | 3.0% | 2.2.4 |
| 24 | HPL entire system | 3.0% | 2.2.4 |
| 25 | LRZSORT | 3.0% | 2.2.5 |
| 26 | PMATMUL | 3.0% | 2.2.6 |
| 27 | RBench | 3.0% | 2.2.7 |
| 28 | RINF MPI-only | 3.0% | 2.2.8 |
| 29 | RINF OpenMP-only | 3.0% | 2.2.8 |
| 30 | SIPBENCH | 3.0% | 2.2.9 |
| 31 | SPECRate FP (base) | 0.0% | 2.2.10 |
| 32 | SPECRate INT (base) | 0.0% | 2.2.10 |
|  | **Application Benchmarks** | **38%** | **2.3** |
| 33 | BQCD | 6.0% | 2.3.1 |
| 34 | Gadget | 6.0% | 2.3.2 |
| 35 | GROMACS | 6.0% | 2.3.3 |
| 36 | Iphigenie/CPMD | 6.0% | 2.3.4 |
| 37 | SeisSol TPV27 64 | 2.0% | 2.3.5 |
| 38 | SeisSol Drplast | 4.0% | 2.3.5 |
| 39 | SUSPENSE SCALE=8 | 2.0% | 2.3.6 |
| 40 | SUSPENSE SCALE=16 | 2.0% | 2.3.6 |
| 41 | SUSPENSE SCALE=24 | 2.0% | 2.3.6 |
| 42 | SUSPENSE SCALE=32 | 2.0% | 2.3.6 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Storage Subsystem Benchmarks (only for qualitative assessment and minimum performance) | 0 % | 2.4 |
| 43 | IOR | 0% | 2.4.1 |
| 44 | FIO | 0% | 2.4.2 |
| 45 | IOZONE | 0% | 2.4.3 |
| 46 | Bonnie | 0% | 2.4.4 |
| 47 | **Energy Efficiency (only for qualitative assessment)** | **0%** | **2.5** |
| 48 | **Estimated mean power draw (only for qualitative assessment)** | **0%** | **2.6** |

**Table 1:** Weights of benchmarks

## Criteria for the qualitative evaluation

Each requirement in the document *Description of Goods and Services SuperMUC-NG* is assigned to one of the following feature groups. Each feature group is separately evaluated with particular regard to the source of the requirement (see also *Description of Goods and Services SuperMUC-NG, Section 1.9*). An overall rating for each feature group is established. The ratings are summed up and optionally capped to their final value.

|  |  |  |
| --- | --- | --- |
| Feature group (main sources of requirements[[1]](#footnote-2)) | Lower  boundary for η | Upper  boundary for η |
| General Objectives (ALL) | -0.25 | +0.25 |
| Date of system readiness (POL) | -0.05 | +0.05 |
| Likelihood of operational success (PROJ, TECH, REL) | -0.05 | +0.05 |
| Features and characteristics of the system architecture, compute nodes and interconnect (TECH, PERF, ENGY, INFRA) | -0.20 | +0.20 |
| Features and characteristics of the storage subsystems (TECH, REL, PERF) | -0.20 | +0.20 |
| Reliability, resiliency, redundancy, usability, flexibility (excluding I/O subsystem) (REL, TECH) | -0.20 | +0.20 |
| Operating system, resource management system, system management software (ADMIN, TECH, REL, STD) | -0.15 | +0.15 |
| Software, programming environment and user aspects (USER, PERF, TECH, STD) | -0.25 | +0.25 |
| Additional characteristics of the benchmarks (PERF) | -0.10 | +0.10 |
| Ease of software porting (PERF, STD, USER) | -0.25 | +0.25 |
| Infrastructure and energy efficiency (INFRA, ENGY) | -0.20 | +0.20 |
| Support and user transition (SUPP, ADMIN, USER) | -0.10 | +0.10 |
| Collaboration and reference installations (POL) | -0.05 | +0.05 |
| **Capped final sum** | **-0.25** | **+0.25** |

**Table 2**: Feature groups and their contribution to the qualitative assessment.

LRZ will assess how well the proposal satisfies the objectives and requirements setup in the document “*Goods and Services for SuperMUC-NG”* and the performance proposed for the benchmarks.

Evaluation criteria shall include, but are not be limited to, the following feature groups and items:

**General objectives**

* How well the proposed solution meets the overall objectives and the system concept
* Completeness of the Tenderer’s response to the requirements
* Overall quality and credibility of the proposal
* Versatility of the potential usage of the system
* General purpose characteristics of the system and its software
* Diversity of key technologies and system architectures (hardware and software)
* Concept for Phase 2

**Date of system readiness**

* Revaluation or devaluation of Tenderer’s offers between +5% for October 2018 and -5% for December 2019

**Likelihood of operational success**

* Quality and stringency of project/site preparation and installation plan
* Technological and commercial risks of the proposed solution
* Fall back and mitigation scenarios
* Complexity of the proposed system and the required infrastructure
* Number of novel or critical features and components as well as their specific risks
* Dependencies from suppliers and third party components
* Timely availability of a test system

**Features and characteristics of the system architecture, compute nodes and interconnect**

* Size, volume, and characteristics of hardware resources (cores, aggregate memory, caches, interconnect, etc.)
* Type of node memory, size of the usable memory of a single compute node
* Topology of the internal network
* Relative and absolute bandwidths and latencies of the memory hierarchy and the interconnect
* Capability of the system to support the scaling of applications to high core counts and high parallel performance
* Monitoring capabilities (component errors, power draw, thermals, network traffic, etc.)
* Capabilities to optimise and control the usage of system resources

**Features and characteristics of the storage subsystems**

* Size, capacity, capability, and performance characteristics of file systems
* Performance characteristics of the I/O benchmarks which are not directly covered by the benchmark rating
* Ease of integration of DSS and HOME into the high performance interconnect
* Homogeneity of the file system solutions and support
* Data integrity solution
* IO specific reliability, resiliency, redundancy, usability, flexibility
* Characteristics of the gateways to the tape archive and login cloud

**Reliability, resiliency, redundancy, usability, flexibility**

* Hardware and software fail-safety features
* Expected reliability and resiliency of hard- and software
* Quality and scalability of diagnostics, monitoring and administration tools
* Features for system monitoring and control of system usage
* Expected downtimes caused by software maintenances and upgrades of the operating system
* Seamless integration and usage of Phase 2
* Handling of replacement parts
* Re-routing and dead-link detection features within the interconnect
* Requirements on the infrastructure

**Operating system, resource management system, system management software**

* Possibilities provided by the system software to ensure high usability, a well-balanced load distribution, and operational stability as well as a high energy efficiency for the expected application profile
* Well-defined processes for fixing bugs in the software stack (e.g. OS, compilers and tools)
* Capabilities and efficiency of the resource management system
* Capabilities for managing the interconnect fabric

**Software, programming environment and user aspects**

* Suitability for the current user community and expected acceptance of the system by the user community
* Ease of use of the programming model
* Capabilities, quality, and availability of compilers, debuggers, test aids and tools
* Scope and availability of optimized scientific libraries, tools, and applications (including third party software)
* Interoperability with the LRZ environment, particularly for visualisation, archiving and backup
* Capabilities, quality, and availability of the software virtualisation solution

**Additional characteristics of the benchmarks**

* Quality and expressiveness of the benchmark report
* Performance characteristics which are not directly covered by the benchmark rating, e.g. scaling behavior
* Positive or negative performance outliers
* Commensurate and balanced performance for all benchmarks or parts of a benchmark
* Identification and resolution of bottlenecks in benchmarks
* Delivery of optional results
* Performance of non-compliant versions

**Ease of software porting**

* Porting efforts and need for code modifications (in particular for the benchmark codes; see also **Table 3**)
  + Changes which can be performed within 2 weeks will be positively evaluated (“good”)
  + Changes requiring up to 1 month are evaluated “neutral”
  + Changes requiring up to 2 months are evaluated as “marginal”
  + Changes requiring more than 2 months will be evaluated as “poor”
  + Extensive non-portable code modifications (e.g. CUDA, intrinsic instructions, etc.) will be evaluated as “poor”
* Migration path from the current application base
* Significant performance gains by optimization that are easy to perform
* Automatically achievable compute performance (compiler switches, auto-parallelisation/auto-vectorisation, etc.)

**Infrastructure and energy efficiency**

* Energy and power measurement capabilities
* Infrastructure requirements and needed modifications to the existing infrastructure and the corresponding costs that are not covered by the procurement
* Cooling concept
* Extent of the free cooling capability available at LRZ (e.g., maximum tolerable system inlet temperature, ΔT, ...)
* Possibilities for system waste heat reuse

**Support and user transition**

* Extent and quality of support
* Conditions for support contract
* Number of support personnel
* Support and assistance of user transition by the Tenderer

**Collaboration and reference installations**

* Reference installations, number and size of comparable installations
* Market penetration
* Possibilities for collaboration and joint projects.

## Evaluation Scheme

**Compared to the first round of the competitive dialogue, the focus of the evaluation is on the direct comparison of the offers of the two remaining Tenderers.**

The Tenderer’s response is evaluated according to the following scheme:

|  |  |  |  |
| --- | --- | --- | --- |
| General Rating | Degree of performance | Comparison with other Tenderer’s offer | Value for η |
| Good | Requirements are fulfilled or exceeded.  High quality in many aspects of the proposal. | Above the other Tenderer’s offer | Up to the upper bound for the particular feature group |
| Neutral | The proposal fulfills most elements of the requirements. | Only marginal differences between the offers | Zero |
| Poor | The proposal misses some of the requirements.  Low quality in some aspects of the proposal. | Below the other Tenderer’s offer | Down to the lower bound for the particular feature group |
| Unacceptable | One or more mandatory criteria are not fulfilled. |  | **Exclusion from the procurement process** |

**Table 3**: Evaluation Scoring

If there are conflicting evaluations for different columns in the above table, the worst case is retained.

## Rules for the Benchmarks and their Verification

### Confidentiality

Under no circumstances shall any information related to the design, algorithms, or source code of the bench­marks be disclosed to a third party without written consent of LRZ and the authors of the particular bench­mark.

1. A Non-Disclosure Agreement between the Tenderer and LRZ must be signed before getting access to the benchmark sources.[[2]](#footnote-3)

LRZ may disclose the optimisations for the application bench­marks to the respective authors of the particular benchmark and will discuss the modifications with them.

### Code Modifications

Only modifications that **do not change** the intention, functionality and complexity order of the underlying algorithms and do not lower the precision of the calculations are permitted. The changes must be at least as robust as the baseline algorithm with respect to numerical stability.

Any change of the source code must be fully disclosed.

A short rationale for all code changes must be provided.

Allowable modifications include:

* Compiler flags that are generally available for most compilers or compilation or linkage flags that are supported and documented for a specific compiler are permitted.
* Compiler directives and pragmas, which are supported and documented for the compiler, may be used. Language extensions, e.g. CILK, instead of directives are also permitted.
* Loop unrolling or fusion as well as loop blocking, including the variation of block sizes is permitted. Inner and outer loops can be interchanged.
* Changes to the data layout may be done locally in some routines or globally for the whole code. Note that this may involve a large number of changed code lines and, correspondingly, a large porting effort.
* Data alignment is permitted, e.g. by compiler directives or compiler flags.
* Code substitution by calls to well-established performance library subroutines is permissible. Here, pattern-matching techniques for replacing original code with calls to libraries are permissible. Linking to optimized and supported versions of Tenderer’s performance libraries is permitted and encouraged. The usage of all libraries must be disclosed together with the submission of the results.
* Implicit or explicit function inlining is allowed.
* The use of threads is allowed, either implicit by TBB, OpenMP, OpenACC etc. or explicit by POSIX threads.
* The porting of code to CUDA or OpenCL is permissible. With respect to ease of use, this might be negatively evaluated (i.e., the use of portable directives is preferred).
* The use of architecture-specific intrinsics is permitted. With respect to ease of use, this will be negatively evaluated (i.e., a portable, architecture-independent approach is preferred).

Starting point for optimization and porting of the application benchmarks is the respective source code supplied with the benchmark or any explicitly allowed version of it. If newer versions of a specific source code are available, their optimizations may only be backported. A complete replacement of the source code with a newer version is not permissible.

Special cases for permitted optimization:

* For HPL, HPCG, and Graph500 only reference implementations are provided. Here the Tenderer is free to implement any solution which is conforming with the general rules of the particular benchmark.
* For FFTW a vendor specific library can and should be used.
* For Seissol and CPMD additional rules are provided in the benchmark description.

Number of MPI ranks in the kernel and application benchmarks

* The specification of MPI ranks for the kernel and application benchmarks can be considered as a definition of the problem size for an instance of the benchmark. Any number of MPI ranks can be used as long as that problem size and the underlying algorithms of the benchmark are retained.

OMP\_NUM\_THREADS in the kernel and application benchmarks

* The specification of OMP\_NUM\_THREADS for the kernel and application benchmarks can be replaced by other resource metrics on accelerated nodes (e.g. the product of gangs and threads in OpenACC), as long as the problem size and the underlying algorithms of the benchmark are retained.

### Additional results for non-compliant benchmark versions

Going beyond the required commitments, **additional** results for versions of the benchmarks that are not compliant with the rules in Section 1.10.2 may be supplied. These additional commitments will be evaluated in the qualitative assessment only. Examples of non-compliant versions include but are not limited to:

* newer and better optimized versions of a code (e.g. GPU optimized version of BQCD or CPMD)
* intrusive changes in the data layout
* deeper algorithmic changes
* usage of special optimized libraries (e.g. libxsmm)

In any case, the commitments for the compliant versions must be supplied.

If additional versions are provided, the porting effort and the differences to the compliant version must be documented in the benchmark report.

### Limitations of Code Optimisation

We take it for granted that the Tenderer understands the intention of the kernels (especially those of the low-level benchmarks) and undertakes no action to circumvent the intended measurements.

The following optimisations are **not** permitted:

* **Code to circumvent the actual computation:** Any modification of the code to circumvent the actual computation is not permitted, e.g., it is not permitted to use Strassen's algorithm for DGEMM matrix multiplication. This prohibition does of course not cover standard optimisation techniques, whether by the compiler or carried out manually, which pull redundant code out of the loops and precompute it.
* **Eliminate code or bypass code** which is not covered in the actual benchmark case but may be covered in other benchmark cases.
* **Preimposing the knowledge of the solution:** Any modification of the code or input data sets which makes use of known properties of the solution is not permitted.
* **Input files** of application benchmarks must not be modified to change the simulation model or algorithm. Exceptions for permissible parameter changes are explicitly stated in the description of the respective benchmark.

### Precautions Regarding Cache in the Intel MPI Benchmarks

For evaluation of the interconnect properties version 4.1 of the Intel MPI benchmark (https://software.intel.com/en-us/articles/intel-mpi-benchmarks) is used, which provides a highly configurable set of MPI kernels. Apart from minor changes to the build system, the only change to the benchmark source is the increase of MAXMSGLOG from 22 to 24 in the header file “**IMB\_settings.h”**. In the multi-process group version of the bench­mark, disjoint groups of 2, 4, 8, etc. processes will be formed, which will all simultaneously run the bench­mark routines.

Please consult the User’s Guide provided with the benchmark sources, which is also located at  
*$BENCH/mpi/imb/doc/IMB\_Users\_Guide.pdf* in the LRZ benchmark source tree.

**To ensure that the cache is invalidated** for each measurement iteration of the program, it is required to use the   
-off\_cache switch of the executable. The environment variable CCONF referenced in the command lines for executing the benchmark programs must contain the following two numbers, separated by a comma:

* The size of the last level cache in MBytes
* The cache line size for that cache in Bytes
* Example: CCONF=32,64

### Conversion of Timing to Performance Values

For some programs timing values are converted to floating point operations per second by dividing a predefined operation count by the execution time (wall clock time). However, in some cases only parts of the program are measured e.g., omitting the initial phase or the final phase.

It is not allowed to change the conversion factors, even if they may appear wrong or inappropriate.

For any given bench­mark program we consider the converted values as just another measurement unit of execution time, which serves to rank the results against the best performance obtained among Tenderers.

### Frequency and Power Envelope Settings

Benchmarks for the performance commitments may be run at any frequency (including turbo steps) and power envelope setting available on the proposed system. However, the peak electrical power draw of the total system specified by the Tenderer in the document *Description of Goods and Services SuperMUC-NG* must not be exceeded.

### Rules for Systems which are not available for Benchmarking

The description of each bench­mark program specifies which results must be delivered. In case that a bench­mark program cannot be run on the proposed system, either because there is no system of the offered size available or because the proposed system hardware or software cannot yet be bench­marked at all, predictions and commitments have to be made by the Tenderer.

These predictions are considered as committed minimal performance results. They should be based on measurements done on an available system that is similar in architecture to the one offered. Results obtained on smaller systems need to be carefully scaled to the predicted values for the proposed system size or to the required bench­mark size. The reasoning behind all estimates and predictions should be explained. A rationale for the assumptions underlying the predictions should be given.

### Reference Output

Many benchmark directories contain reference outputs. The Tenderer should not take the performance values provided in these outputs as a basis for performance estimations and predictions on current or future systems. The performance values have been obtained under various conditions e.g., they may have been obtained with lower frequency settings or with non-optimal compiler flags.

### Benchmark Report

A benchmark report should be provided together with the benchmark results. The report should be comprehensible and informative with respect to the proposed code modifications, projection methods, compiler flags, and runtime settings.

## Commitments by the Tenderer and Submission of Results to LRZ

Commitments in red fields must be filled in.  
Committed values must be demonstrated by the Tenderer during the acceptance procedure for the system. Failure to reproduce the committed performance requires that the Tenderer will take appropriate counter measures.

Fields in green are for consistency checking during the aggregation calculation, which is performed by LRZ. LRZ will compare the value provided by the tenderer with its own calculations and in case of mismatch try to clarify the discrepancies.

Fields in white are optional, but should be filled for qualitative evaluation.

All required results and the source code of the programs must be supplied to LRZ in electronic form. The tenderer should keep an exact copy and checksum of it.

## Delivery of updated Commitments (compared to an Intermediate Offer)

Updated commitments are possible after the first round of the Competitive Dialogue. However, only adjustments for better overall performance are permitted. This includes, but is not limited to:

* delivering better benchmark results
* delivering more hardware (which will increase the aggregate performance)
* further qualitative improvements

If downward adjustments are performed for a specific benchmark, an irrefutable reason must be provided, and compensation through improvements in other benchmarks is required. In this case, the intermediate offer is considered as a competing one. The two offers are compared against each other according to the rules given in Section 1.3. The benchmark ranking number of the final offer must be greater or equal to the ranking number of the intermediate offer (i.e. the committed benchmark performance must not be decreased).

Rfinal ≥ Rintermediate

Improvements of the benchmark performance values of the final offer must not be obtained by significant code modifications that would incur a down evaluation compared to the intermediate offer. Delivery of output files and documentation may contain results unchanged from the intermediate proposal. Files and documentation that have changed against the intermediate offer should be clearly marked.

## Preparation and Running the Benchmarks

Many of the benchmarks refer to a common include file for “make”, to common settings and libraries. For using this mechanism, define the variable BENCH and source the prepared setup.sh:

BENCH=…  
cd $BENCH  
. setup.sh

The tenderer may need to modify setup.sh and the related include files or write its own ones..

A sample script $BENCH/bin/run is provided which can be used to start up the benchmark programs. In the benchmark description, the script is denoted as $RUN:

RUN=$BENCH/bin/run

In addition to the internal time measurement within the bench­mark programs this script performs an external time measurement.

The script takes six input parameters beyond the executable and execution flags to be run:

-n: the (total) number of MPI processes

-N: the number of nodes

-I: the number of islands

-T the number of tasks per node

-t: the number of threads

-C: the number of copies

**Not all parameters need to be specified for the execution of the benchmarks**

**Modifications:** Modify the script according to your needs e.g., insert calls to *mpiexec* to start the executables. Using other calls than the *time* command that provide additional information about the performance of the system (e.g., Flop-counters, hardware performance monitors/counters) is appreciated, but these commands must at least report the elapsed (real) time, user time, and system time.

This script can also be modified to fill the system with identical copies of the program, as stipulated in Section 1.3.

## Verification of the Benchmark Commitments

### Verification for Phase 1

During the **acceptance test of Phase 1,** the performance of the individual benchmarks for the interconnect-related benchmarks, for the low-level and kernel benchmarks, and for the application benchmarks has to be demonstrated. For each benchmark (i) the relative deviation from the committed values is computed:

The performance of individual benchmarks may fail to reach the committed values by a margin of 15%. Such deviations must be compensated by achieving better-than-committed values in other benchmarks. The weighted sum of all deviations of the individual benchmarks is computed as follows (see Section 1.7 for the values of gi) and must be greater than or equal to zero

1. The relative deviation for each individual benchmark (Di) must be greater or equal than **-**15 % (-0.15).

The weighted sum of all relative deviations (D) must be greater or equal than zero.

To confirm your accordance with this, check here **[ ]**

### Verification of the Improvement Ratio for Phase 2[[3]](#footnote-4)

The benchmark set for the verification of the improvement ratio (see *Description of Goods and Services SuperMUC-NG* in section 3.4.3) consists of the following benchmarks:

1: HPL for Phase 2

2: Rinf Benchmark

3: SPECRate FP

4: SPECRate INT

5: MPI Messaging rate of a node

6: Aggregate Bisectional MPI Bandwidth for Phase 2

7-12:Six application benchmarks

The individual improvement ratio IRi for each of the 12 benchmarks is calculated as the ratio of the aggregate performance of Phase 2 to the aggregate performance of Phase 1 (as measured during the acceptance test of Phase 1

The overall improvement ratio IR is calculated as the average of the individual improvement ratios.

1. For Phase 2, the average of the individual improvement ratios must be equal or greater than the improvement ratio stipulated in *Description of Goods and Services SuperMUC-NG, Section 3.4.2*.

To confirm your accordance with this, check here **[ ]**

# Benchmarks

Aggregation for different node types (e.g., general purpose, GPU, or many-core nodes) are only performed for those types for which commitments are delivered (see also Section 1.2 and 1.3 of this document)

If needed, provide separate commitments for each node type by making appropriate copies of result tables.

## High Performance Interconnect-related benchmarks

### MPI Aggregate messaging bandwidth of a node

Purpose: Measure the single node aggregate bandwidth and report the total aggregate interconnect bandwidth, by aggregating the bandwidth of all available links on the node. The benchmark utilizes point-to-point communications. The Benchmark is a modification of the Argonne Leadership Computing Facility MPI Benchmark (see: <https://www.alcf.anl.gov/software>).

Source: $BENCH/src/mpi/ALCF\_MPI/aggregate

Executable: aggregate

Compile make

License: Open Source License, see $BENCH/src/mpi/ALCF\_MPI/LICENSE

Procedure: Run the executable on multiples of 10 nodes i.e., the benchmark is run on several groups consisting of 10 nodes each. This scale-out is accomplished to assess the variation of performance over the nodes of the system. SCALE should be set in a manner that at least 10% of all nodes take part in a single copy of the benchmark. An appropriate number of copies of the benchmark should be simultaneously be started to fill the complete system (e.g., 10). It is also possible to run a single copy of the benchmark. All nodes of the system should be covered by the benchmark.

The minimum performance measured by these copies is taken as the performance indicator. It is not allowed to change the size of the messages (LENGTH).

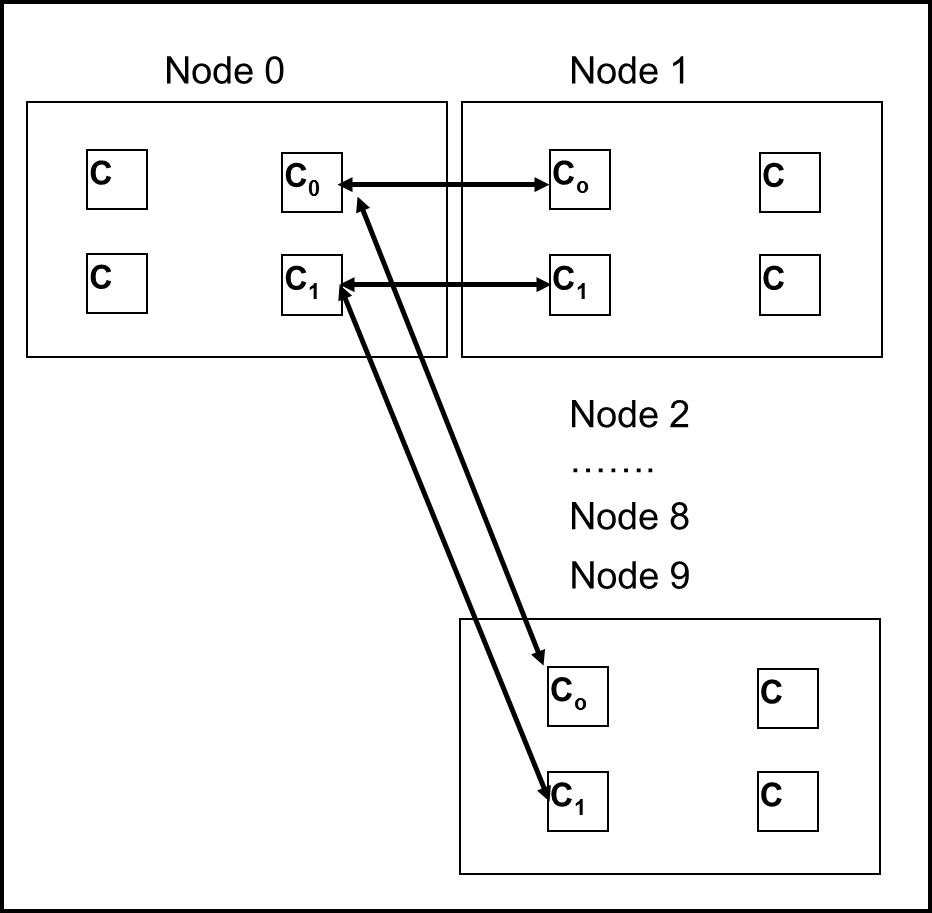
Command line: SCALE=1000 (TBD, a large number)  
TASKS\_PER\_NODE=22 (TBD, optimal number)  
NODES=$(($SCALE\*10))  
TASKS=$(($NODES\*$ TASKS\_PER\_NODE)  
$RUN –C 10 -N $NODES –n $TASKS -T $TASKS\_PER\_NODE -t 1 -I 1 ./aggregate

Hints: It is necessary to determine an optimal number of tasks per node.

The Tenderer may choose an optimal placement of the neighboring nodes. This can be performed externally (by hostfile, etc ) or by modification of getranks.c. Nodes may be arranged to perform nearest-neighbor communications. The executable is instrumented to output information about the placement of tasks. These lines may be skipped.

For this Benchmark: 1 GB = 109 Byte

Example: Communication pattern for 10 nodes i.e., 9 neighbors, each having 4 cores and TASKS\_PER\_NODE=2.



Performance: The performance of a single copy is contained in the output in the line starting with  
“RESULT”.   
To account for variations across the entire system and to account for contention etc., the minimum performance of the simultaneous running copies must be taken.

Aggregation: The performance indicator is multiplied by the number of nodes of the system.

Commitment:

|  |  |
| --- | --- |
| **Messaging Bandwidth** |  |
| (Minimum) aggregate Bandwidth of one node (GB/s): |  |
| For formal evaluation:  Aggregate Performance of the system (GB/s) =  (MinAggBW) \* (Total number of nodes of this type in system) |  |

### MPI Messaging rate of a node

Purpose: Measure the messaging rate of a node. The Benchmark is a modification of the Argonne Leadership Computing Facility MPI Benchmark (see: https://www.alcf.anl.gov/software).

Source: $BENCH/mpi/ALCF\_MPI/mmps

Executable: mmps

Compile make

License: Open Source License, see $BENCH/mpi/ALCF\_MPI/LICENSE

Procedure: Run the executable on multiples of 10 nodes i.e., the benchmark is run on several groups consisting of 10 nodes each. This scale-out is accomplished to assess the variation of performance over the nodes of the system. SCALE should be set in a manner that at least 10% of all nodes take part in a single copy of the benchmark. An appropriate number of copies of the benchmark should be simultaneously be started to fill the complete system (e.g., 10). It is also possible to run a single copy of the benchmark. All nodes of the system should be covered by the benchmark.

Hints: It is necessary to determine an optimal number of tasks per node.

The Tenderer may choose an optimal placement of the neighboring nodes. This can be performed externally (by hostfile etc ) or by modification of getranks.c. Nodes may be arranged to perform nearest-neighbor communications. The executable is instrumented to output information about the placement of tasks. These lines may be skipped.

If still necessary to maximize the rate, the Tenderer may choose appropriate numbers for the window size - the parameter, which determines the number of messaging sent simultaneously to each neighbor.

Command line: SCALE=1000 (TBD, a large number)  
TASKS\_PER\_NODE=22 (TBD, optimal)  
NODES=$(($SCALE\*10))  
TASKS=$(($NODES\*$ TASKS\_PER\_NODE)  
$RUN –C 10 -N $NODES –n $TASKS -T $TASKS\_PER\_NODE -t 1 -I 1 ./mmps

Example: For a Figure see: 2.1.1

Performance: The performance of a single copy is contained in the output in the line starting with  
“RESULTS: Nodes”.   
To account for variations across the entire system and to account for contention etc., the minimum performance of the simultaneous running copies must be taken.

The minimum of "Max Mrate" of the 10 simultaneously running copies.

Aggregation: This (Minimum) messaging rate of one node is multiplied by the number of nodes of this type in the system.

Commitment:

|  |  |
| --- | --- |
| **Messaging Rate** |  |
| (Minimum) Messaging Rate of one node: |  |
| For formal evaluation: Performance Aggregation: (MinMsgRate) \* (Total number of nodes of this type in system) |  |

### MPI startup time

Purpose: Measure the to start and finish an MPI application

Source: $BENCH/mpi/ALCF\_MPI/init

Executable: init

Compile make

License: free, LRZ development

Procedure: Run the executable

1. on a quarter of the system with the number of MPI tasks equal to the number of compute cores of this partition
2. on half of the system with the number of MPI tasks equal to the number of compute cores of this partition
3. on the entire system with the number of MPI tasks equal to the number of compute cores of the entire system

Command line: $RUN –C 1 -N $NODES –n $TASKS -T $TASKS\_PER\_NODE -t 1 –L LOGFILE ./init

Performance: Inverse of the WALLTIME reported in LOGFILE.

Aggregation: No aggregation of performance.

Commitments:

|  |  |  |
| --- | --- | --- |
| **Benchmark** | **Startup time (sec)** | **Performance,  Inverse of Startup  (1/sec)** |
| Case a) Startup Time for quarter of the system |  |  |
| Case b) Startup Time for half of system |  |  |
| Case c) Startup Time for entire system |  |  |

### MPI Link bandwidth and latency of a node, one task per node, intra-island

Purpose: Measure the (one-directional) bandwidth available to applications that use only a single MPI task per compute node.

Source: $BENCH/mpi/imb/src

License: Common Public License Version 1.0, see directory license

Original source: https://software.intel.com/en-us/articles/intel-mpi-benchmarks

Executable: IMB-MPI1

Compile make MPI1

License:

Procedure: Run the executable IMB-MPI1 between the two nodes of one arbitrarily (i.e., worst case) chosen pair of compute nodes within an island. Run **exactly one** MPI task on each node. Leave the other cores of the node idle.  
The message lengths is fixed to 8 Bytes and 1 MiB (1048576 Byte), resp.

Command line:

TASKS=2  
$RUN –C 1 -I 1 -N $TASKS –n $TASKS –T 1 -t 1 \  
 ./IMB-MPI1 -msglen LENGTHS –off\_cache $CCONF pingping

Example: Two nodes, each having 4 cores. Only one core of each node takes part in the communication:



Results: Report the bandwidth and the latency for 8 bytes and the bandwidth for the message size of 1 MiByte (=1048576 Bytes).

Take the latency and bandwidth value from the output, where "1 group of 2 processes" are formed (i.e., MPI Communication Group 0 consists of the processes 0 and 1 etc.).

Aggregation (will be performed by LRZ): The bandwidth will be multiplied by the number of nodes in the system.

Commitments:

|  |  |
| --- | --- |
| **Link Bandwidth and Latency**  No aggregation for the latency will be performed |  |
| Pingping latency, 8 byte, t[µsec]  (choose right column!) |  |
|  | |
| Pingping link bandwidth, for 1 MiByte [in Mbytes/sec] (choose right column!) |  |
| For formal evaluation:  Performance Aggregation: = (PingPingLinkBW) \* (Total number of nodes of this type in system) |  |

### MPI Bisection bandwidth and latency, intra-island and inter-island, many tasks per node

Purpose: Measure the bisection bandwidth and latency.

* For the bandwidth, the transfer rate for 1 MiB is taken.
* For the latency, the time needed to transfer 8 Byte.

The bisection bandwidth is the minimum bandwidth over all possible bisection bandwidths. Hence, the worst-case bisectional configuration within the involved nodes must be measured.

Both MPI-1 and one-sided MPI-2 calls are considered.

Source: $BENCH/mpi/imb/src

Executable: IMB-MPI1, IMB-RMA

Compile: make MPI1 RMA

License: Common Public License Version 1.0, see directory license

Original source: https://software.intel.com/en-us/articles/intel-mpi-benchmarks

Procedure: The benchmark must be executed on the following *node groupings* of every installation phase:

**Case 1, intra-island: ALL** nodes within an island of an installation phase. If the system has no island concept, an appropriate group of nodes consisting of at least 16384 compute cores can be taken or the same performance value as for Case 2 can be reported.

**Case 2, inter-island:** **ALL** nodes of an installation phase

Divide all nodes of a *node grouping* into two equally sized sets, called “left” and “right” in the following. Use as many cores on a node as you need to reach the maximum aggregate bandwidth of the internal network, but **at least one core of each node** must take part in the communication.

Run the bench­mark with pairs of cores, where the two tasks of a pair are on different sides of the configuration. For the inter-island tests half of the islands is on the left side and half is on the right side. If the number of islands is uneven, then the nodes of one island are divided half to the left and half to the right.

**The NON-AGGREGATE measurements for the one-sided benchmarks (put/get) have to be taken.**

Hints: Probably the best results can be obtained when all cores of a node take part.  
-multi 0: only the lowest performance group will be output.

Command lines:

TASK\_PER\_NODE=<TBD, optimal>

# Intra Island  
$RUN –C $COPIES -I 1 –N $NODES -n $TASKS –T $TASKS\_PER\_NODE –t 1 \  
 ./IMB-MPI1 –msglen LENGTHS –multi 0 \  
 –off\_cache $CCONF –map $(($TASKS/2))x2 sendrecv exchange

# Inter Island  
$RUN –C 1 -I $ISLANDS –N $NODES -n $TASKS –T $TASKS\_PER\_NODE –t 1 \  
 ./IMB-MPI1 –msglen LENGTHS –multi 0 \  
 –off\_cache $CCONF –map $(($TASKS/2))x2 sendrecv exchange

# Intra Island   
$RUN –C $COPIES -I 1 –N $NODES -n $TASKS –T $TASKS\_PER\_NODE –t 1 \  
 ./IMB-RMA –msglen LENGTHS –multi 0 \  
 –off\_cache $CCONF –map $(($TASKS/2))x2 bidir\_put bidir\_get

# Inter Island   
$RUN –C 1 -I $ISLANDS –N $NODES -n $TASKS –T $TASKS\_PER\_NODE –t 1 \  
 ./IMB-RAM –msglen LENGTHS –multi 0 \  
 –off\_cache $CCONF –map $(($TASKS/2))x2 bidir\_put bidir\_get

Example: Four nodes with 6 MPI task on a node, 4 nodes and 2 islands take processes take part in this example:



Results: Take the bandwidth value for the first part of the output, where "TASKS/2 groups of 2 processors" are formed (i.e., MPI Communication Group 0 consists of the processes 0 and NTASKS/2 etc) and for the message size of 1 MiByte.

Take the timings for the first part of the output where "NTASKS/2 groups of 2 processors" are formed, for a message size of 8 Byte. This is used as definition of the MPI latency.

Commitments:

|  |  |  |
| --- | --- | --- |
| **Bisection Latency 8 Byte take the column for: t\_avg** | **Latency MPI1** | |
| **Intra-Island t\_avg [µs]** | **Inter-Island t\_avg [µs]** |
| exchange |  |  |
| sendrecv |  |  |
| **Take minimum of the above values** | | |
| Latency  No aggregation for the latency will be performed |  |  |

|  |  |  |
| --- | --- | --- |
| **Bisection Bandwidth 1 MiByte take the column for: Mbytes/s** | **Bandwidth MPI1** | |
| **Intra-Island [MByte/s]** | **Inter-Island [MByte/s]** |
| exchange |  |  |
| sendrecv |  |  |
| **Take maximum of the above values** | | |
| Bisection bandwidth (per task pair) |  |  |
| Tasks per node |  |  |
| Bisection bandwidth per Node (=BisectBW\*TaskPerNode) |  |  |
| For formal evaluation:  Performance Aggregation: (BisectionBW per Node) \* (Total number of nodes of this type in system) |  |  |

|  |  |  |
| --- | --- | --- |
| **Bisection Latency 8 Byte take the column for: t\_avg MODE: NON-AGGREGATE** | **Latency RMA** | |
| **Intra-Island [µs]** | **Inter-Island [µs]** |
| Bidir\_get |  |  |
| Bidir\_put |  |  |
| **Take minimum of the above values** | | |
| **Latency**  No aggregation for the latency will be performed |  |  |

|  |  |  |
| --- | --- | --- |
| **Bisection Bandwidth 1 MiByte take the column for: Mbytes/s MODE: NON-AGGREGATE** | **Bandwidth RMA** | |
| **Intra-Island [MByte/s]** | **Inter-Island [MByte/s]** |
| Bidir\_get |  |  |
| Bidir\_put |  |  |
| **Take maximum of the above values** | | |
| Bisection bandwidth (per core pair) |  |  |
| Tasks per node |  |  |
| Bisection bandwidth per Node (=BisectBW\*TaskPerNode) |  |  |
| For formal evaluation: Performance Aggregation: (BisectionBW per Node) \* (Total number of nodes of this type in system) |  |  |

### MPI collective communication

Purpose: Measure MPI collective execution time as a function of task count. The following collective routines are measured: MPI\_Barrier, MPI\_Allreduce,

Source etc.: $BENCH/mpi/imb/src

Executable: IMB-MPI1, IMB-NBC

Compile: make MPI1 NBC

Procedure: The benchmark must be executed with 4096, 8192, 16384, 32768, 65536, and 131072 tasks on an installation phase of the offered system. A subset of nodes should be chosen which contains at least as many (potentially hyperthreaded) cores as MPI tasks are needed for the benchmark. For the reduction operation, a size of 8 Bytes is chosen. The task layout can be optimized for the target system and must be disclosed to LRZ.

License: Common Public License Version 1.0, see directory license

Original source: https://software.intel.com/en-us/articles/intel-mpi-benchmarks

Command lines:

$RUN –C 1 –I <islands> -N <nodes> -n 131072 –T $TASKS\_PER\_NODE –t 1\  
 ./IMB-MPI1 –msglen LENGTHS\_COLLECTIVE \  
 -sync yes –off\_cache $CCONF –multi 0 barrier allreduce

$RUN –C 1 –I <islands> -N <nodes> -n 131072 –T $TASKS\_PER\_NODE –t 1\  
 ./IMB-NBC –msglen LENGTHS\_COLLECTIVE \  
 -sync yes –off\_cache $CCONF –multi 0 Iallreduce

Results: **barrier and allreduce**: Take the average timings **t\_max** from the output, where "1 groups of 131072 processors", 2 groups of 65536 processors”, etc.

**Iallreduce**: Take the average timings t\_ovrl from the output, where "1 groups of 131072 processors", 2 groups of 65536 processors”, etc.



Commitments:

|  |  |  |  |
| --- | --- | --- | --- |
| **Barrier**  **n, groups of … processes** | **t\_max [µs]** | **/ log2(n)** | **L(n)** |
| 4096 |  | /12 |  |
| 8192 |  | /13 |  |
| 16384 |  | /14 |  |
| 32768 |  | /15 |  |
| 65536 |  | /16 |  |
| 131072 |  | /17 |  |
|  | | (these values are for reference and qualitative evaluation, take the average of the above values as commitment) | |
| **Average of L(n) for barrier** No further aggregation will be performed | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **(Multi-)Allreduce**  **Tasks, n** | **t\_max [µs]** | **/ log2(n)** | **L(n)** |
| 4096 |  | /12 |  |
| 8192 |  | /13 |  |
| 16384 |  | /14 |  |
| 32768 |  | /15 |  |
| 65536 |  | /16 |  |
| 131072 |  | /17 |  |
|  | | (these values are for reference and qualitative evaluation, take the average of the above values as commitment) | |
| **Average of L(n) for collective operations** No further aggregation will be performed | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Iallreduce n, groups of … processes** | **t\_ovrl [µs]** | **/ log2(n)** | **L(n)** |
| 4096 |  | /12 |  |
| 8192 |  | /13 |  |
| 16384 |  | /14 |  |
| 32768 |  | /15 |  |
| 65536 |  | /16 |  |
| 131072 |  | /17 |  |
|  | | (these values are for reference and qualitative evaluation, take the average of the above values as commitment) | |
| **Average of L(n) for collective operations** No further aggregation weill be performed | |  | |

### MPI Shared Memory Test

Commitment:

|  |  |
| --- | --- |
| **Sharedmemtest** |  |
| Windows/sec per node |  |
| For formal evaluation: Performance Aggregation: (Windows/sec) \* (Total number of nodes of this type in system) |  |

## Low Level and Kernel Benchmarks

### FFTW

Purpose: Test the functionality and the performance of the FFTW Fast Fourier Transformation.

ftw-bench times in-place double-precision complex 3-dimensional FFT and in-place and out-of-place double-precision real 3-dimensional FFT. It is written in Fortran and parallelized with OpenMP.

Source etc.: $BENCH/kernel/fftw

FFTW library: The FFT library is not included as source files in the benchmark. FFTW can be downloaded from:  
<http://www.fftw.org/>.

Version FFT 3.3.3 or higher or any compliant version can be used, including vendor or architecture-specific versions.

Executable: fftw-bench

Compile: make

License: GPLv3, see file COPYING

Original source: http://loto.sourceforge.net/feram/src/fft\_check.html

Procedure: The benchmark must be executed for a 8192x8192x8192 problem. The number of iterations is set to 10. However, 5 dummy loops are executed in the beginning.

Minimum Performance Requirements:

For any single copy of the benchmark a minimum performance of

**10 TFlop/s = 10\*1012 Flop/s**

must be achieved with an arbitrary number of nodes, tasks, cores and threads, which may be chosen by the Tenderer. Since the performance of all copies will be aggregated, the Tenderer will typically only use the minimum number of nodes which allow to reach this requirement.

Hints: Aggregation of Performance Results is based on nodes. Therefore, always try to use all cores on a node.

If the -+numbers of dimensions of an array are powers of two, "memory or cache conflicts" may occur. Padding may be introduced but makes the code complicated.

For creating the MPI plan for in-place forward transformation (note dimension reversal) the parameters FFTW\_MEASURE, FFTW\_PATIENT or FFTW\_EXHAUSTIVE may be used.

Multithreading: Multithreading (OpenMP) may be used.

Command lines:

NTIMES=10

NODES=<tbd>  
TASKS\_PER\_NODE=<tbd>  
OMP\_NUM\_THREADS=<tbd>  
COPIES=<tbd>, to fill the system.

$RUN –C $COPIES –I <islands> -N $NODES -n $(($NODES\*$TASKS\_PER\_NODE)) \  
 –T $TASKS\_PER\_NODE –t $OMP\_NUM\_THREADS \  
 fftw-bench $NTIMES 8192 8192 8192

Results: The benchmark outputs the performance as

==== PERFORMANCE [Flop/s]

(Also the performance per task and per thread are displayed; do not mix them up)

The minimum of the performance of all COPIES must be taken.

Commitment:

|  |  |
| --- | --- |
| **FFTW** |  |
| PERFORMANCE [Flop/s] (with the provision that the performance of 10 TFlop/s must be exceeded by a single copy) |  |
| NODES, Number of nodes for a single instance |  |
| For formal evaluation:  Performance Aggregation: (PERFORMANCE) / NODES \* (Total number of nodes of this type in system) |  |

### GRAPH500

Purpose: The Graph500 is a rating of supercomputer systems, focused on [data intensive loads](https://en.wikipedia.org/wiki/Data_Intensive_Computing).   
An undirected graph is constructed. Then a breadth-first search in a large undirected graph is performed. While graphs can have any number of vertices and edges, the Graph 500 benchmark has particular specifications for them. It sets the scale (*SCALE)* of a graph and then requires that the number of vertices be a power of 2, that is, |*V* | = 2*SCALE*, and that there are 16 edges per vertex. Thus, the number of edges is |*E*| = 16 |*V* |. The benchmark uses the harmonic mean of the traversed edges per second (TEPS) as the performance metric.

Rules: The benchmark must be performed according to the rule of Graph500.org.

Description and Reference Implementations:

For a full description and rules for the benchmark see: http://www.graph500.org or https://github.com/graph500

Reference implementations can be found there.

However; any other implementation including vendor or architecture-specific versions can be used.

Source: A copy of a **reference implementation** is included in $BENCH/kernel/graph500/mpi.

Hint: The reference implementation will provide suboptimal results! The Tenderer is allowed to use its own implementation

License: For the reference implementation: see file COPYING

Compile: make

Procedure: The benchmark is defined by two parameter:

SCALE: The logarithm base two of the number of vertices.  
 EDGEFACTOR: The ratio of the graph's edge count to its vertex count. It has to be set to 16.

The benchmark must be run for the problem size

**SCALE ≥ 35.**

Choose the problem and measure the performance (TEPS) according to the rules of the benchmark.  
The result will be aggregated to the whole system, if not all nodes are used.

SCALE=<tbd>

NODES=<tbd>  
TASKS=<tbd>  
OMP\_NUM\_THREADS=<tbd>  
$RUN -C <number of copies> -I <islands> –N $NODES \  
 –n TASKS –t $OMP\_NUM\_THREADS ./graph500.exe $SCALE 16

Commitment:

|  |  |
| --- | --- |
| **GRAPH500 (SCALE ≥ 35):** |  |
| PERFORMANCE [harmonic\_mean\_TEPS] (provide full number not GTEPS or TTEPS, e.g. 1.234e+12) |  |
| NODES, Number of nodes for a single instance |  |
| For formal evaluation: Performance Aggregation: (TEPS) / NODES \* (Total number of nodes of this type in system) |  |

### HPCG

Purpose: The High Performance Conjugate Gradients (HPCG) Benchmark project is an effort to create a new metric for ranking HPC systems. HPCG is designed to exercise computational and data access patterns that more closely match a different and broad set of important applications, and to give incentive to computer system designers to invest in capabilities that will have impact on the collective performance of these applications.

HPCG is a complete, stand-alone code that measures the performance of basic operations in a unified code:

* + - Sparse matrix-vector multiplication.
    - Sparse triangular solve.
    - Vector updates.
    - Global dot products.
    - Local symmetric Gauss-Seidel smoother.
    - Driven by multigrid preconditioned conjugate gradient algorithm that exercises the key kernels on a nested set of coarse grids.

Description and Reference Implementation:

The description, rules and reference implementations can be found under:   
http://hpcg-benchmark.org/

Hints: Probably HPCG can benefit from large memory footprints.

Procedure: Choose an appropriate problem size and measure the performance. According to the rules of the benchmark the runtime must be at least 1800 seconds.

The required performance values are contained in the yaml-file in the line:

HPCG result is VALID with a GFLOP/S rating of: XXXX

**Intra Island**: Assess throughput (with the number of cores equal to the preferred size of an island, see *Description of Goods and Services SuperMUC-NG, Section 1.2*)

NODES=<tbd>  
TASKS=<tbd>  
OMP\_NUM\_THREADS=<tbd>

SCALE=<tbd, local Problem Dimensions>  
$RUN -C <number of copies> -I 1 –N $NODES \  
 –n TASKS –t $OMP\_NUM\_THREADS \

./hpcg.exe –nx SCALE –ny $SCALE –nz $SCALE –rt 1800

The performance will be aggregated to the entire system.

**Entire System:** Measure performance for the entire system or all nodes of this type.

NODES=<number of node of whole system>  
TASKS\_PER\_NODE=<tbd>  
NUMBER\_OF\_TASKS=$(($ NODES\*$TASKS\_PER\_NODE))  
OMP\_NUM\_THREADS=<tbd>  
$RUN -C 1 -I <islands> –N $NODES -n $ TASKS \

./hpcg.exe –nx SCALE –ny $SCALE –nz $SCALE –rt 1800

Commitment:

|  |  |  |
| --- | --- | --- |
| HPCG | intra island | entire system |
| SCALE, local problem dimension |  |  |
| Total execution time [s] |  |  |
| PERFORMANCE (GFLOP/S rating) |  |  |
| NODES, Number of nodes for a single instance |  |  |
| For formal evaluation:  Performance Aggregation: (PERFORMANCE) / NODES \* (Total number of nodes of this type in system) |  |  |

### HIGH PERFORMACE LINPACK (HPL)

Purpose: The standard parallel HPL benchmark is used to obtain an estimate of the peak performance of the system; this benchmark also gives an impression of the quality of the Tenderer’s BLAS implementation. For a given problem size N, αN2 bytes of memory storage are required; ideally, α should not be much larger than 8 if eight-byte floating-point words are used. If this storage is distributed across P tasks, the amount of memory per task required will be

.

For execution on a large parallel system, a compromise may need to be made between the long run time needed versus the high fraction of peak performance achieved for large problems.

The performance L(P, N) is determined by

,

where Tmeasured is the execution time for problem size N and task count P.

Note that the implementation is required to preserve the operation count specified above, i.e. use of a Winograd-Strassen or related algorithm for performing matrix multiplications is prohibited. Otherwise, Tenderer-specific implementations with respect to coding, communication mechanism and (possibly multi-threaded) BLAS library implementation are encouraged.

Description and Reference Implementation:

A description and reference implementation for HPL is available at <http://www.netlib.org/benchmark/hpl/>

Procedure: Choose an appropriate problem size and measure the total performance Rmax.

**Intra Island**: Assess throughput (with the number of cores equal to the preferred size of an island, see *Description of Goods and Services SuperMUC-NG,* *Section 1.2*)

NODES=<tbd>  
TASKS=<tbd>  
OMP\_NUM\_THREADS=<tbd>  
$RUN -C <number of copies> -I 1 –N $NODES \  
 –n TASKS –t $OMP\_NUM\_THREADS ./linpack.exe

The performance will be aggregated to the entire system

**Entire System:** measure performance for the entire system.

NODES=<number of node of entire system>  
TASKS\_PER\_NODE=<tbd>  
TASKS=$(($ NODES\*$TASKS\_PER\_NODE))  
OMP\_NUM\_THREADS=<tbd>  
$RUN -C 1 -I <islands> –N $NODES -n $ TASKS –t $OMP\_NUM\_THREADS ./linpack.exe

Assessment of performance of general purpose compute nodes:

The aggregate compute performance for Case SCALE=8 might also be taken for assessing the performance of the general purpose compute nodes.

Commitment:

|  |  |  |  |
| --- | --- | --- | --- |
| **HPL** | **intra island** | **entire system** | **may be also used for  For assessment of the  general purpose compute  performancenodes,**  **Only general purpose nodes** |
| Problem Size N |  |  |  |
| Total execution time [s] |  |  |  |
| Rmax [PFlop/s] |  |  |  |
| NODES, Number of nodes for a single  instance |  |  |  |
| For formal evaluation:  Performance Aggregation: (Rmax) / NODES \*(Total number of nodes of this type in system) |  |  |  |

### LRZSORT

Purpose: lrzsort is a parallel sort program, which reads a file and writes another one with the sorted results. It originates from ([http://sortbenchmark.org](http://sortbenchmark.org/)).

The input data contain records of 100 Bytes, where the first 10 Bytes are the key and the next 90 Bytes are the values.

Algorithm: The input data are distributed over all processes. Then there are several local sorting phases with subsequent exchange of data with other processes. Finally, the data are written to the output files.

Python: The benchmark is written in Python and also tests the MPI4py implementation. It uses MPI4py, particularly:

* MPI-IO
* MPI-Collectives
* MPI-alltoallv

Source: $BENCH/kernel/lrzsort

The sources for the data generation and for the validation of the results are from   
<http://www.ordinal.com/gensort.html>

Restrictions: There are several restrictions. The number of processes must be a perfect square (of 64, 256, 1024, 4096, 16384, … processes), and BUFFSIZE (the size per process) must be dividable by the number of tasks i.e., BUFSIZE / TASKS = whole number.

Multithreading: The generate and validate phases are multithreaded.

Use –t<number> for generate and validate.

The sort phase is not yet multithreaded. However, multithreading is allowed, as long as this is supported by the Python versions from [www.python.org](http://www.python.org). For instance, it is not allowed to use a specially compiled multi-threaded version of “.sort”, if this version is not generally available.

Hints: Use “,buf” after the file for the generate and validate phase. Also use “,trans”.

See: “generate –h” or “validate –h” for more information.

Topology: The nodes need not be fully filled with processes to aggregate more IO bandwidth for the benchmark.

Procedure: There are 3 phases for the benchmark

* generate
* sort
* validate

Only the sorting phase is evaluated.

To help around the above restrictions, LRZ provides a small script “lrzsort.sh”

Usage: lrzsort DIRECTORY PHASE SCALE 4096

**DIRECTORY** is the directory in which the files are stored

**PHASE** is the phase as described above

**SCALE** is the scaling of the benchmark. SCALE must be greater or equal 10 (i.e. files are approx. 8 TiB)

**4096** is a fixed number for the large scale benchmarks

It may be necessary to modify the line with “mpiexec” in the script to include the actual number of processes.

#generate and validate executables

make

DIR=<directory for the files>  
NNN=4096 # can be 64, 256, 1024 for tests (perfect square)  
SCALE=<tbd by Tenderer> # scaling for the size of the Benchmark,

# must be greater or equal 10.

#Run on one node, need only be done once, file can be reused.

./lrzsort.sh $DIR generate $SCALE $NNN  
  
#Run with exactly 16384 Workers (e.g., MPI tasks), modify the line with mpiexec for this. Must be perfect square  
./lrzsort.sh $DIR sort $SCALE $NNN

#Run on one node,

./lrzsort.sh $DIR generate $SCALE $NNN

Different node types:

If there are different node types in the system, only the result for the best performing type may be committed.

No aggregation will be performed.

Results: The benchmark outputs the performance as

Results: xxxx.xxxxx MB/s (Take this number !)  
--------------------------------  
====> TIME

The line marked with “Results:” is based on internal timings of the benchmark, where the line with   
“===> TIME” is based on external timings, which include the startup time for the benchmark.

Take the performance result from the internal one.

Commitment:

|  |  |
| --- | --- |
| **LRZSORT, NNN=4096, 16384 Workers** |  |
| MPI Tasks |  |
| Results [MB/s] (no further aggregation will be performed) |  |
| TIME (no further aggregation will be performed) |  |
| Number of nodes used for the benchmark: |  |

### PMATMUL

Purpose: This benchmark tests the [divide and conquer algorithm](https://en.wikipedia.org/wiki/Divide_and_conquer_algorithm) with block partitioning for the parallel multiplication of square matrices.

Source: $BENCH/low\_level/rin1

Compile: make

Code Owner: LRZ

Hints: The benchmark can use two different communication algorithms and different block sizes for communication, which is controlled by the input file. Format of the input file

M, N, K, BLOCKSIZE, METHOD

M=N=K: Size of the matrices  
BLOCKSIZE: Size of the submatrices  
METHOD: 1 = Broadcast of the submatrices, 2 = Ringcast

Probably, the Ringcast algorithm is fastest.

The input can contain multiple lines for which the benchmark is to be performed.

Estimations of the performance during the runs are written to stderr.

An optimized version for DGEMM should be used. However, Strassen’s algorithm is not allowed.

Procedure: Run the benchmark for a matrix size of **600000.**

The number of nodes, tasks, tasks per node, and threads can be chosen by the Tenderer. The result will be aggregated to the whole system, if not all nodes are used. Typically, the Tenderer will use the smallest number of nodes for which the benchmark can be run.

COPIES=<tbd, to fill the whole system>

TASKS\_PER\_NODE=<tbd>  
NODES=<tbd>   
TASKS=$((TASKS\_PER\_NODE\* NODES))  
OMP\_NUM\_THREADS=<tbd>

copy input.bench.dat input  
# optionally modify the input deck to contain more experiments and   
# optimum parameter for BLOCKSIZE and METHODE

$RUN –C $COPIES –n $TASKS –N $NODES –T $TASKS\_PER\_NODE \  
 –t $OMP\_NUM\_THREADS ./pmatmul

Results: The performance for a specific run is contained in the column “TotalGFlops”. The best performing combination of BLOCKSIZE and METHOD should be taken for the commitment. If not all copies show the same performance, the minimum performance of all copies must be taken.

Commitment:

|  |  |
| --- | --- |
| **PMATMUL, Matrix Size M=N=K=600000** |  |
| NODES, Number of nodes for a single instance |  |
| TotalGFLOPS (of one copy) |  |
| For formal evaluation:  Performance Aggregation: TotalGflops / NODES \* (Total number of nodes of this type in system) |  |

### RBench (Bioconductor, sequence alignment / pattern matching)

Purpose: The application R is widely used in statistical and Big Data analytics. In this case it will be used for a fuzzy pattern matching algorithm on genetic data. For this purpose, the algorithm 'countPattern' from the Bioconductor package 'Biostrings' will be used. Since a large number of these pattern matchings have to be performed in practice, the benchmark combines this algorithm with a job farming approach performed by several worker processes running R. These workers connect to a **redis** database in order to fetch their tasks. The benchmark allows for flexible choice of workers per node. The number of total tasks 'N', however, is fixed for both the test run (N=200) as well as for the actual benchmark run (N=**500000**).

Building: A detailed description for compiling all necessary components can be found in the text file   
compiledescription.sh’ in the top-level directory of the repository.

Hints: Please ensure that there are no running redis-servers blocking port 6378 or 6379 when the benchmark is started.

The Benchmark itself is embarrassingly parallel.

Code Owner: Bernau/LRZ

Source: $BENCH/kernel/R-Benchmark

All necessary source files can be found in the repository in the subdirectory ‘DEPS’.

Executable: R, Rscript, and redis-server

Versions: The versions of redis, R and the R-packages required for the benchmark are the following:  
Executables: redis 3.0.7, R 3.2.4  
R-Packages: rredis\_1.7.0, Biobase\_2.30.0, BiocGenerics\_0.16.1, IRanges\_2.4.8, IRanges\_2.4.8, S4Vectors\_0.8.11, XVector\_0.10.0, zlibbioc\_1.16.0, Biostrings\_2.38.4.  
The sources for this are contained in the repository.

Testing: A test case with 200 pattern matchings can be run by setting the environmental variable ‘CASE’  
in the file ‘bashrun.sh’ to ‘SMALLCASE’.

Input: Several parameters have to be adjusted in the section ‘environment variables to be set by user’. These are specifications of the paths to the necessary binaries as well as specifications of the number of nodes and processes and workers to be used (further details regarding these parameters are explained directly in bashrun.sh). The parameter PATTERNS has to be set to AUTO for the actual benchmark  which causes this parameter to be set to 200 for the small test case whereas it will be set to 500000 in the large case. For test purposes, one may also set this parameter to an arbitrary nonnegative value.

Procedure: Once the parameters mentioned in section ‘Input’ have been set correctly the benchmark can be run by copying the complete content of the file ‘bashrun.sh’ into a batch scheduler script or by running ‘./bashrun.sh’ inside the benchmark folder (i.e., the top-level folder of the repository).

Logging: Both the small and the large run, produce the following log files in the subdirectory 'logs':

* db1.txt and db2.txt with information on the two redis-databases
* startup.log and check.log reporting problems with the startup of the databases
* submit.log writing information on task submission
* workers/worker.log collecting information on processing of pattern matchings
* collect.log logging progress of collection of the results
* postproc.log reporting information on results, time measurements and validation of results

Output: For both the reference and the benchmark the tasks are divided into 10 blocks which are processed one after the other. For each block, one output file is created in the subfolder 'results'. These output files include time measurements for the overall benchmark and the individual tasks. Moreover, they include the number of matches for each sequence alignment performed during the run which can be used as a reference for validating the correctness of the results. Furthermore, the .pdf file 'results.pdf' and a data file 'summary\_results.RData' are created in the end of the benchmark. The pdf shows a distribution of the runtimes for the individual pattern matching tasks as well as a bar plot of the pattern hits. In the case of the small reference run, it also includes a comparison of the pattern hits for both the current run and a previously performed reference run. If there are any mismatches between these two tables, the results can be considered invalid. Further details on the differences to the reference, can be found in the logfile 'logs/postproc.txt'.

Performance: The walltime of the benchmark is written at the end of the logfile ‘postproc.log’ (‘BENCHMARK WALLTIME’).

The performance per node is computed as:

Performance\_per\_node = (1./WALLTIME) / NODES

Procedure: Start the modified script: bashrun.sh. Execute the benchmark with a resulting WALLTIME of not more than 700 seconds.

**WALLTIME ≤ 700 secs**

Typically the tenderer will take the minimum number of nodes or cores falling just below the demanded WALLTIME.

Commitment:

|  |  |
| --- | --- |
| **PATTERNS=500000** |  |
| WALLTIME (sec) (with the provision that WALLTIME < 700 secs) |  |
| NODES |  |
| Performance\_per\_node (1/sec) |  |
| For formal evaluation: Performance Aggregation = Performance\_per node \* (Total number of nodes of this type in system) |  |

### RINF

Purpose: This benchmark tests the floating point and integer performance of various one-dimensional loop kernels. Depending on the access pattern, various aspects of the processor architecture and the memory hierarchy are tested.

The case relevant for the quantitative evaluation is:

**Test Case 2:** Double Precision Vector Triad

A(I) = B(I) \* C(I) + D(I)

evaluated as 3 loads, 1 store and 2 floating-point operations (A potential load for “read for ownership” is not accounted).

The aggregate performance of a node is assessed. Scaling the benchmark for more than one node has the only purpose of assessing performance variations across cores or nodes.

Two subcases are considered:

* **rinf1.small.conf:** probably best suited for filling the nodes only with (many) MPI tasks, without multithreading
* **rinf1.large.conf:** probably best suited for filling the nodes with just one or a few MPI tasks with multithreading.

Source: $BENCH/low\_level/rinf

Code Owner: LRZ

Hints: The benchmark code is written in standard Fortran, with some infrastructure as well as kernel code in standard C. Support for C interoperability and dynamic memory management from the Fortran 2003 standard is required. The MPI variant of the code presumes the availability of a standard-conforming MPI implementation including the Fortran module **mpi**; but only functionality from MPI-1 is needed.

Reference output: see subdirectory REFERENCE-OUTPUT

Removing the dummy routine:

Line 105 of the benchmark code contains a dummy routine, which prevents code removal by the compiler. This dummy routine can be deleted, if it is assured that all iterations of the nested loops are actually executed.

102 t1 = MPI\_WTIME()

103 !$OMP parallel private(i,it)

104 do it=1,inp%nit

**105 call dummy()**

106 !no$OMP do schedule(static,omp\_chunk)

107 !$OMP do simd

…

117 do i=1,llen STRIDED

118 #endif

119 a(i) = d(i) + b(i) \* c(i)

120 end do

Procedure: Compile two versions of the code

make rinf1.small (SIZE is set to 2000000)

make rinf1.large (SIZE is set to 400000000)

The input for the particular benchmark must be copied to “rinf1.conf”. It contains

* The number of vector distinct lengths for which the benchmark is performed. Based on this, the actual vector lengths are compute in subroutine select\_lengths.
* The approximate time for which each vector length is tested (10 secs, which is rather long and it may be lowered if still valid results can be obtained)
* The ID of the test case (=2, do not change)
* The stride of access (=1, do not change)
* How data are allocated (=.true., do not change)
* How the average internally computed in module rinf1\_wp. For ‘l’ the logarithmic average starting with the smallest vector length is computed, for ‘L’ the logarithmic average starts above vector length > 10000. Do not change this setting.

MPI is only used for taking the potential variation of performance across the nodes into account. Ideally, the performance should be predictable from a single node case.

The benchmark should be run using at least 10 percent of all nodes and then with 10 copies, but also 100% and only 1 copy can be used. The time used for MPI communication is negligible.

The minimum performance for the copies must be reported, bearing in mind variations across copies, nodes, or cores.

COPIES=<approx. 10, to fill the system>

TASKS\_PER\_NODE=<TBD, optimal>  
NODES=<at least 10% of the number of nodes of whole system>  
TASKS=$((TASKS\_PER\_NODE\* NODES))  
OMP\_NUM\_THREADS=1

**cp rinf1.small.conf rinf1.conf**  
$RUN –C $COPIES –n $TASKS –N $NODES –T $TASKS\_PER\_NODE \  
 –t $OMP\_NUM\_THREADS rinf1.small.exe  
#The minimum performance for the 10 copies should be reported.

COPIES=<approx. 10, to fill the system>

TASKS\_PER\_NODE=1  
NODES=<at least10% of the number of node of whole system>  
TASKS=$((TASKS\_PER\_NODE\* NODES))  
OMP\_NUM\_THREADS=<TBD, optimal>

**cp rinf1.large.conf rinf1.conf**  
$RUN –C $COPIES –n $TASKS –N $NODES –T $TASKS\_PER\_NODE \  
 –t $OMP\_NUM\_THREADS rinf1.large.exe

Results: The performance is displayed in the last lines of the output as

>>> Case: 2 Stride 1 Weighted Performance value PER JOB (Mflop/s)

Commitments:

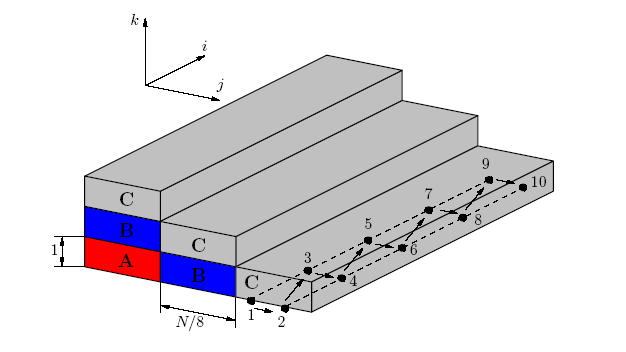
|  |  |  |
| --- | --- | --- |
| **RINF** | **rinf1.small.conf** | **rinf1.large.conf** |
| OMP\_NUM\_THREADS |  |  |
| TASKS\_PER\_NODE |  |  |
| NODES |  |  |
| Logarithmic weighted performance PER JOB [MFlop/s]: (scientific format) |  |  |
| Performance of a node [Mflop/s] =Log. weighted\_performance PER JOB / NODES |  |  |
| For formal evaluation:  Performance Aggregation [Mflop/s] (Performance of a node) \* (Total number of nodes of this type in system) |  |  |

### SIP

Purpose: This is a multi-threaded strongly-implicit procedure (SIP) solver, suitable for solving systems of linear equations resulting from a discretisation of partial differential equations. It is widely used in fluid mechanics and therefore of great practical importance. Regarding the implementation, different approaches are possible. The original paper by Deserno et al. from RRZE is available at  
<http://www.rrze.uni-erlangen.de/dienste/arbeiten-rechnen/hpc/Projekte/OptGuide.pdf>

The following versions of the solver can be used:

**Case 101 (**SipThreeDSolver): a straightforward way is to iterate over all nodes (i, j, k) in 3 do-loops.  
**Case 102** (SipThreeDSolver\_regopt): same as case 101, but one loop is split in multiple parts to improve register usage.  
**Case 103** (SipThreeDSolver\_ppp, pipeline parallel processing): Data dependencies may prohibit automatic parallelisation of the 3D-Version of the SIP-Solver. However, the Fortran90 compiler on the Hitachi SR8000 was able to resolve the dependencies with its specific pipeline parallel processing technique. This technique is re-programmed here with explicit OpenMP statements. The system is divided up into chunks of a certain size (see the following figure). Parallelisation is applied to the loop along the j-direction (the middle loop). Calculation of any chunk is delayed by a barrier until the chunk left of it has been processed. Consequently, blocks with equal colour in the figure are calculated concurrently. This leads to load imbalance in the “wind-up” and ”wind-down” phases of this pipeline since some CPUs have to wait for the first few chunks to be calculated; this effect is negligible for a sufficiently large lattice.



**Case 104** (SipHyperplaneSolver): A hyperplane is defined as L = i + j + k = const.  
**Case 105 (SipHyperplaneSolver\_sr8k):** Pipeline-parallel variant of case 104 optimized for   
Hitachi SR8000 automatic wavefront parallelisation.  
**Case 106 (SipHyperLineSolver):** Similar to hyperplanes one can define hyperlines for which   
L = j + k = const.

Code Owner: RRZE Erlangen

Hints: Probably Case 103 will be the best-performing one.

For getting good performance, it may be necessary to ensure that the data layout in the initialisation phase is the same as during the computation phase. For Shared Memory systems the allocation policy (e.g., First Touch) determines the appropriate initialisation strategy.

**Current versions of the Intel MKL libraries may interfere with the threading library of the compilers** (libiomp5.so). Avoid this by ensuring that any MKL specific entries in LD\_LIBRARY\_PATH appear after those for the compilers.

Source: $BENCH/low\_level/sip

Compile: make

Procedure: Execute the benchmark using one node.

**Modify the input SipInput.bench.dat and SipInput.dat for the problem size (default: 800).**

The problem size must be greater or equal **800**.

Run the benchmark with an appropriate number of OpenMP threads e.g.,

OMP\_NUM\_THREADS=<tbd>

$RUN –C 1 -I 1 –N 1 –n 1 -t $OMP\_NUM\_THREADS ./SipBench.exe

Reference output: see directory REFERENCE-OUTPUT.

Results: Take the maximum performance over the six code versions. In the box below, please enter the performance numbers for a single instance (potentially running on a completely filled node).

The measured performance for each case is given in the output line:

"Performance per core Instance (thread, core, cpu, etc) (MFlop/s):"

Commitments:

|  |  |
| --- | --- |
| **SipBench** | Performance per Thread [MFlop/s] |
| 101: Sip3DSolver (optional, not considered as a commitment) |  |
| 102: Sip3DSolver\_regopt (optional, not considered as a commitment) |  |
| 103:Sip3DSolver\_ppp (probably the best perfoming variant)  (optional, not considered as a commitment) |  |
| 104: SipHyperPlSolver (optional, not considered as a commitment) |  |
| 105: SipHyperPlSolver\_sr8k (optional, not considered as a commitment) |  |
| 106: SipHyperLineSolver (optional, not considered as a commitment) |  |
| Performance per THREAD for the **best performing variant** from above [MFlop/s] (considered as a commitment) |  |
| OMP\_NUM\_THREADS |  |
| SIZE that is chosen for the benchmark (≥800) |  |
| Performance of one node [Mflop/s] **=**OMP\_NUM\_THREADS\* (Performance of best perf. Variant) (should be equal to the performance per TASK) |  |
| For formal evaluation:  Performance Aggregation: (Performance of one node) \* (Total number of nodes of this type in system) |  |

### SPECrate® CPU2006 of a node

Purpose: The SPEC CPU™ 2006 benchmark(http://www.spec.org/) is an industry-standardized, CPU-intensive benchmark suite, stressing a system's processor, memory subsystem and compiler.

The SPECrate® metrics measure the throughput or rate of a machine carrying out a number of tasks.

For the measurement of thebase **metrics**[[4]](#footnote-5) the same flags must be used in the same order for all benchmarks of a given language.

Source: Sources are not included. Tenderers are encouraged to use their own license. LRZ has a valid license to run the benchmarks during the verification process.

Rules: The same rules as for the original SPEC benchmark apply.

Compilation: Consistent compiler options across all programs of a given language must be used.

Results: The benchmark has zero weight and will only be evaluated qualitatively.

However, the base metrics for the SPECint\_rate20006 and SPECfp\_rate2006 benchmark must be provided at least for the general purpose part of the system.

Additionally, the Tender can supply results for the accelerated nodes (e.g. many-core nodes).

Aggregation to the full system size will be performed.

Hints: This benchmark might be limited by the amount of memory of a node.

Reference: see results in Throughput [SPECint\_rate2006](https://www.spec.org/cpu2006/results/rint2006.html) and [SPECfp\_rate2006](https://www.spec.org/cpu2006/results/rfp2006.html) in  
<https://www.spec.org/cpu2006/results/>.

Commitment:

|  |  |  |
| --- | --- | --- |
| **SPECrate CPU 2006, Base optimization** | **CINT2006 Rate at least for general purpose part** | **CFP2006 Rate at least for general purpose part** |
| PERFORMANCE=Max. throughput rate for a single node |  |  |
| For formal evaluation: Performance Aggregation  PERFORMANCE \* (Total number of nodes of this type in system) |  |  |

|  |  |  |
| --- | --- | --- |
| **SPECrate CPU 2006, Base optimization** | **CINT2006 Rate optional: accelerated part** | **CFP2006 Rate optional: accelerated part** |
| PERFORMANCE=Max. throughput rate for a single node |  |  |
| For formal evaluation: Performance Aggregation  PERFORMANCE \* (Total number of nodes of this type in system) |  |  |

## Application Benchmarks

### BQCD

Purpose: BQCD (Berlin QCD Program) is a hybrid Monte-Carlo program that simulates Quantum Chromodynamics with dynamical standard Wilson fermions. The computations take place on a four-dimensional regular grid with periodic boundary conditions. The updates are local (i.e., only nearest neighbours are needed). The kernel of the program is a standard conjugate gradient solver with even/odd preconditioning. As a consequence, all arrays are stored in an even/odd ordered fashion and the four indices are collapsed into a single one. The access to neighbours is handled by lists.

The code is written in Fortran and parallelised with MPI and OpenMP, for larger runs almost 20 % of the walltime is spent on the MPI communication calls (point-to-point & collectives), and ~80 % on the program-kernels.

The parallelisation is done through a regular grid decomposition in the highest 3 dimensions. The values from the boundaries of the neighbouring processors are stored in the same array as the local values. The local values have indices 1, ..., volume/2. The boundary values have indices greater than volume/2.

The total domain size in the example input files is 48 x 48 x 48 x 96. The memory for the arrays is dynamically allocated during initialisation.

Apart from rounding errors the program gives identical results for any grid decomposition.

Source: $BENCH/src/apps/BQCD

Building: see: README

make prep-<platform>  
make

Executable: ./bqcd

For more information about the compiled executable type: mpiexec –n 1 ./bqcd -V

Modifications: libd21.a (set by libd=21 in Makefile.var) may be replaced by a different version of the library. The sources for the different libraries are contained in the subfolder: src/d.  
(21, 3, and 2 have already been tested, 21 shows the best performance on SuperMUC).

In order to build a GPU version of BQCD, a configuration is available on "src/platform/Makefile-quda.var". This file includes the QUDA library in order to perform calculations on graphics processing units (GPUs) see: <https://lattice.github.io/quda/>

Which library is used must be disclosed.

Porting to and optimisation for your hardware may be required. Choice of compilers and flags may vary from system to system.

In addition to the usual standard optimisation techniques mentioned in 1.10.2 you are allowed to perform the following modifications for the routines **d** and **d\_dag**:

* The sequence of floating point operations may be changed according to the rules of algebra.
* The sequence of communication operations may be changed as long as remote data is available at the right point in time.
* The data layout may be changed.
* The subroutine structure within d() and d\_dag() may be modified.
* Fortran, C,or C++ may be used as programming languages.
* Any communication library (MPI, SHMEM, OpenMP or combination of MPI and OpenMP) may be used.
* If you change the data layout, you have to make sure that outside cg() the original data structures can be used i.e., at the beginning of cg(), data would have to be copied from the original data structures to the new ones and at the end of cg() data would have to be copied back correspondingly. At a few locations outside of cg() the subroutines d() and d\_dag() are called. Here necessary copy operations are also permitted.

Multithreading: Multithreading is permitted and may be useful, particularly for large core counts. By running several benchmark configurations, we compared the effect of two communication modes, pure MPI and hybrid (OpenMP+MPI). We observed some accelerations by hybrid parallelisation and using the advantage of AVX2 (compared to the MPI only version performance on the SuperMUC Phase 2 machine).

Please report the number of nodes, processes and threads used.

Procedure: The input for the benchmark run is stored in the file TEST/input.BENCH. In input.BENCH the domain size is defined in the line:

lattice LX LY LZ LT

The actually used number of MPI tasks per x-, y-, z- and t-direction has to be defined in the line:

processes NPEX NPEY NPEZ NPET

The total number of MPI processes is

NPEX \* NPEY \* NPEZ \* NPET

bqcd binary will check automatically the variable $OMP\_NUM\_THREADS to calculate the total number of cores using:

NPEX \* NPEY \* NPEZ \* NPET \* $OMP\_NUM\_THREADS = SIZE

NPEX NPEY NPEZ and NPET must be dividers of LX, LY, LZ and LT, respectively. They may be changed as long as they fit the number of cores required for benchmarking. Examples for input files are also available as TEST/input.BENCH-\*.

Command Line: Prepare the six input data sets with the desired settings for the layout (48 x 48 x 48 x 96) lattice size, number of the MPI-processes and threads

SIZE=<set to corresponding value>  
OMP\_NUM\_THREADS=<tbd>

TASKS=<number of MPI tasks>  
TASK\_PER\_NODE=<tbd>S=<td>

$BENCH/bin/run –C … -I … –N $NODES –n $TASKS –t $OMP\_NUM\_THREADS ./bqcd input.BENCH > out.BENCH  
etc.

Testing**:** Smaller test cases may be generated by modifying LX, LY, LZ, LT in   
input.BENCH to fit to the number of processors and memory.

Hints: On SuperMUC Phase 2 processors with 28 physical cores there is no natural mapping for QCD projects (the prime 7 is first time used in lattice dimensions). One can "rotate" the decomposition (by setting up an appropriate machine-file) in such a way that the fourth dimension of the lattice is the fastest running dimension in the machine-file.

BQCD is parallelized by domain decomposition, and  decomposing the lattice for a large number of processes has two effects. First, at some stage a domain might completely fit into the data cache. Second, the data from the relatively large surface of the small domains has to be communicated to eight nearest neighbor processes. The communication becomes dominant and requires an excellent network.  For these reasons, in two lines of the input file the lattice size and the number of MPI-processes per direction must be specified:

lattice 48 48 48 96  
processes 2 8 8 16

If a machine file can be set up for a parallel benchmark that specifies the MPI rank by the order in which the execution hosts appear in it, it is permitted to specify a host order that permutes the MPI tasks. This might be done such that all tasks corresponding to increments of 1 in the last column of the “processes” line above are assigned neighboring MPI task indices.

Examples: Reference input and output can be found in the directory output-reference.

Results: Please deliver all result files from stdout

The performance per core of the **CG-solver** can be obtained from the output.

The result file contains, 7 colons, where the minimum value (Gflops), max (Gflops),  mean (Gflops) average and Total performance with (Tflop/s) are given, see:

region #calls time mean min max Total

s Gflop/s Gflop/s Gflop/s Tflop/s

CG 43  22.27 0.56  0.56  0.56 10.01

For the commitment the value of the last colon (Total) with Tflop/s is required, for the example above the value will be:  “10.01”. Also report the column time, here “22.27”

Run the program for lattice size (48 x 48 x 48 x 96) with an increasing number of processes and/or threads (starting from the lowest possible number) up to the required number to show **the scaling behaviour.**

**A minimum performance for CG for the smallest configuration must be 10 Tflop/s or more..**

The results for this are taken for the quantitative evaluation.

Double the SIZE of the configuration for the further steps. The results for this are taken for the quantitative evaluation.

Commitment:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SIZE= MPI-TASKS \* OMP\_NUM\_THREADS** | **MPI\_ TASKS** | **OMP\_ NUM\_ THREADS** | **TASKS\_ PER\_ NODE** | **NODES** | **time[s]** | **TotalCG  Performance [TFlop/s]** | **Performance aggregation = (TotalCG) / (NODES) \*  (Total number of nodes of this type in system)** |
| smallest SIZE with CG Performance ≥ 10 Tflop/s |  |  |  |  |  |  |  |
| 2 \* smallest SIZE |  |  |  |  |  |  |  |
| 4\* smallest SIZE |  |  |  |  |  |  |  |
| 8\* smallest SIZE |  |  |  |  |  |  |  |
| 16\* smallest SIZE |  |  |  |  |  |  |  |

### Gadget

Purpose: Gadget is a code for cosmological N-body simulations. It is written in C and was designed primarily for computers with distributed memory using an explicit communication model implemented via MPI, but in the current benchmark version (P-Gadget3) most core routines are optimized for mixed shared/distributed memory architectures, making use of OpenMP. The code can be used from personal workstations, clusters and massive parallel computers with several thousands of cores. Gadget computes gravitational forces with a Tree-PM algorithm and represents fluids by means of smoothed particle hydrodynamics (SPH). The code can be used to address a wide array of problems in astrophysics and with the inclusion of additional physical processes such as radiating cooling and heating, it can also be used to study the dynamics of the gaseous intergalactic medium, or to address star formation and its regulation by feedback processes.

Code Owner: Volker Springel (HITS). Main developer of this benchmark version: Dolag (USM).

Source: $BENCH/applications/Gadget

Input: The input files for Gadget are contained in two separate files in the download area:

gadget-input.tar.bz2 (approx. 540 GB)  
gadget-intermediate-size.tar.bz2 (approx. 25 GB)

Example: $BENCH/applications/Gadget/sample-gadget-logs

This directory contains a subset of logfiles generated by a typical Gadget run. We removed only the files which were referred to machine-specific debugging or to astrophysical information, and had little use for a benchmark. The required standard logfile of the run corresponds to the file standard-output.out . The other files are provided "as-is" and without description, for convenience of the benchmarkers.

Requirements: GSL (gnu scientific library) version 1.16.   
Newer versions might also work: see <http://www.gnu.org/software/gsl/>

FFTW version 3.3: <http://www.fftw.org/>

HDF5 version 1.8: <http://www.hdfgroup.org/HDF5/>

Modifications: Porting to and optimisation for your hardware may be required.

Hints: Concepts for further optimization to specific hardware may be found in:

Performance Optimisation of Smoothed Particle Hydrodynamics Algorithms for Multi/Many-Core Architectures. Authors: [Fabio Baruffa](https://arxiv.org/find/cs/1/au:+Baruffa_F/0/1/0/all/0/1), [Luigi Iapichino](https://arxiv.org/find/cs/1/au:+Iapichino_L/0/1/0/all/0/1), [Nicolay J. Hammer](https://arxiv.org/find/cs/1/au:+Hammer_N/0/1/0/all/0/1), [Vasileios Karakasis](https://arxiv.org/find/cs/1/au:+Karakasis_V/0/1/0/all/0/1)

See: <https://arxiv.org/abs/1612.06090>

Compile: In the source code directory there are two Config files, named Hr5Config.sh and Hr2Config.sh (set $CONFIG\_FILE to one of those). They will be used for compiling separately the single-node test and the large-scale test, respectively. Define a name for the binary executable to be build (i.e. replace Gadget\_EXEC\_BIN by your choice). Define an appropriate SYSTYPE for your system (compare "SuperMucFFTW3" in the Makefile and Makefile.systype), based on your installation (e.g. GSL, HDF5, FFTW, MPI, etc.).

In either case, to build the code,

cd $BENCH/applications/Gadget  
make clean  
make CONFIG=$CONFIG\_FILE EXEC=Gadget\_EXEC\_BIN

Memory tuning: The memory requirement of the code can be lowered to some extent by adjusting the parameters PartAllocFactor, MaxMemSize and BufferSize (in the \*benchmark\_test.par files).

The PartAllocFactor is acting as an overhead factor for all data allocation to deal with memory imbalances at run time. Suggested value: 1.5; maximum value: 3.0.

The MaxMemSize is the limit of total memory (in MiB) per MPI task. Suggested and maximum value are architecture and setup dependent.

The BufferSize is controlling the size (in MiB) of the Gadget internal MPI send/recv buffers. Suggested value: 150; maximum value 500.

As a general advice, low values of these parameters lead to lower memory footprints. However, this may result in degraded performance or allocation errors. In this case, adjusting those values might solve the problems.

Procedure: Goal of the benchmark is to fill the whole machine with multiple replicas of the larger benchmark system, where all replicas run under a single (MPI parallel) program instance.

*Single-node test:* Copy the executable Gadget\_EXEC\_BIN generated in $BENCH/applications/Gadget for the one-node test in the directory 1-node/P-Gadget3.

The parameter file is 1-node/box5\_hr\_benchmark\_test.par , the ICs are in the directory 1-node/snapdir\_058, and the other files contain input needed by the several physics modules during the run.

To execute Gadget:

$RUN –n $TASKS ./P-Gadget3/Gadget\_EXEC\_BIN box5\_hr\_benchmark\_test.par 2

where $TASKS is the number of MPI tasks to be run (syntax may change). This test has been designed to run on one node and has been successfully tested with 2, 4, 8 and 16 MPI tasks. The number of OpenMP threads per task is a free parameter. On some systems, it might be necessary to delete the output \*.txt files before running again the benchmark.

For testing a larger/smaller run time than the given one, please set TimeMax (in box5\_hr\_benchmark\_test.par, l. 42) to a larger/smaller value (within the range 0.50 - 1.0).

The test is meant for checking the minimal functionality of the code with small computing requirements, and for checking the correctness of results (see *Reference* below). Its results are not required for the performance evaluation of the benchmark.

*Intermediate-scale test*: An additional testcase is provided as a separate file in the download area as:

Gadget-intermediate-size.tar.bz2

This file contains a README with instructions.

The test is meant for further testing the functionality of the code. Its results are not required for the performance evaluation of the benchmark.

*Large-scale test*: Copy the executable Gadget\_EXEC\_BIN generated in $BENCH/applications/Gadget for the large-scale test in the directory 1-island/P-Gadget3.

The parameter file is 1-island/box2\_hr\_benchmark\_test.par , the ICs are in the directory 1-island/snapdir\_060, and the other files contain input needed by the several physics modules during the run.

To execute Gadget:

$RUN –C $COPIES –n $TASKS ./P-Gadget3/Gadget\_EXEC\_BIN \

box2\_hr\_benchmark\_test.par 2

where $TASKS is the number of MPI tasks to be run (syntax may change). On some systems, it might be necessary to delete the output \*.txt files before running again the benchmark.

This test has been prepared to run on 512 nodes and has been successfully tested with 1, 2 and 4 MPI tasks per node. Please be aware, when changing the number of MPI tasks, P-Gagdet3 has to re-calculate the domain decomposition during the initialization phase. This will result in a "memory allocation peak" which will set the size of the minimal required memory. The run may stop due to an allocation error, resulting in an MP\_Abort error.

You may be able to use the steps described under *Memory Tuning* to optimize the memory usage of the setup. However, depending of the available memory of a system, the required number of nodes will be determined by the problem size (i.e. size of the initial conditions).

Relevant for the reference results are the first 300 timesteps executed. After their completion (depending on the system, order of 10 minutes wallclock time) the run can be manually stopped.

Reference: To check the correctness of your result compare the last line of 1-node/reference/energy.txt with the last line of energy.txt produced during a single-node test:

|  |
| --- |
| 1-node/reference/energy.txt , last line: |
| 0.56 5.38669e+06 -4.79705e+08 1.40894e+09 5.38669e+06 -8.68839e+07 \ 2.30622e+08 0 -3.90147e+08 1.17743e+09 0 0 0 0 0 0 0 -2.63922e+06 \ 883333 0 -33954.2 5145.49 7366.75 36646.7 0 0 14.3552 0.137915 |

The second value has to be equal to *5.38669e+06* with at least 3 digits of accuracy. The other numbers must not be *NaN* or *Inf*.

Results: The performance and the performance per node are computed as:

where *Walltime* is computed by means of the script 1-island/get\_timing.bash , which computes the wallclock time of the required 300 timesteps. The first timestep is thus excluded by the results, because of its initialization overhead.

**A minimum performance of 5.0e-03 (1/sec) should be achieved   
(which corresponds to a walltime of 200 sec).**

If this goal cannot be reached due to load imbalances, the following holds

**Alternatively, the benchmark must be executed on not less one eighth of the nodes of the complete system.**

Commitments:

|  |  |
| --- | --- |
| **Gadget, large scale test** |  |
| WALLTIME (sec) (≤ 200 s, or alternatively on not less than one eighth of the nodes of the entire system) |  |
| NODES |  |
| Performance (1/sec) |  |
| Performance\_per\_node (1/sec) |  |
| For formal evaluation: Performance Aggregation = (Perf\_per\_node) \* (Total number of nodes of this type in system) |  |

### GROMACS

Purpose: GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions, but since GROMACS is extremely fast at calculating the nobonded interactions (that usually dominate simulations) many groups are also using it for research on non-biological systems, e.g. polymers. It is one of the fastest and most popular software packages available and can run on CPU as well as GPUs.

Source: GROMACS is a freely available program under GNU License, written in C/C++, and version 5.1.1 of the code is available in $BENCH//applications/GROMACS (given as a reference installation),

This benchmark can be downloaded from the Gromacs webpage, e.g. via ftp

<ftp://ftp.gromacs.org/pub/gromacs/gromacs-2016.2.tar.gz>

or via http from

<http://ftp.gromacs.org/pub/gromacs/gromacs-2016.2.tar.gz> .

**The following version can be used: 5.1.1, 2016.1, 2016.2 or 2016.3. Newer versions are not permitted.**

License: GROMACS is Free Software, available under the GNU Lesser General Public License (LGPL), version 2.1. You can redistribute it and/or modify it under the terms of the LGPL as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

Compiling: A detailed configuration for compiling GROMACS can be found in the script “$BENCH/applications/GROMACS/install.sh”, please type the command

./install.sh

The command above will extract the source “tar.gz” file, create a build directory where GROMACS will be compiled and build “gmx\_mpi” binary.

Please note that this benchmark must be executed with single floating point precision.

GPU: GROMACS provides support for GPU-accelerated systems, for more details, e.g: about compiling, please read the documentation at: <http://www.gromacs.org/GPU_acceleration>

Executable: The executable gmx\_mpi (parallel) will be found at “./build/bin” folder.

Prerequisites: FFTW libraries (also contain in MKL) are needed.

Executable: gmx\_mpi

Testing:Once your executable is created you can test the binary with a small single node test case in the $BENCH/applications/GROMACS/runs/1\_node/ directory.

Result: A reference-output done on SuperMUC Phase1 can be found on the reference-output folder.

Procedure: In the runs folder, there are 2 sub-folders:

* bench: a large input file created for the benchmark run is stored in the file “soup10k\_nstlist40\_NPT2.tpr”
* test: contains a small test case, only for testing the compiled executable.

Execute your runs with varying numbers of MPI tasks and OpenMP threads, compute the performance and deliver the results of the following runs:

EXE=../build/bin/gmx\_mpi   
OMP\_NUM\_THREADS=<tbd>

NODES=<tbd>

$RUN -n NCPUs –t $OMP\_NUM\_THREADS –N $NODES -T… -C… -I… $EXE mdrun \  
 -s soup10k\_nstlist40\_NPT2.tpr \  
 -noconfout -resethway > gromacs\_sp\_ph1\_soup\_cores\_NCPUs.out

Please deliver all result files.

Results: At the end of the output file, performance and the timing results are printed. Please report these two lines:

Time: ….

**Performance:** ……[ns/day]

For each run the performance per node is computed as:

Gromacs benchmark does not specify or require any number of nodes or MPI processes that should be used, instead, **a minimum performance of 40 ns/day** using the input soup10k\_nstlist40\_NPT2.tpr will be required. Select the node number accordingly in order to meet this requirement. You are free to assign ranks per nodes or OMP\_NUM\_THREADS per node, which may be beneficial for the target performance . You are allowed to use the number of PME nodes using the flag “-npme” on the mdrun command, and specify the number of PME threads using "-ntomp\_pme” or environment variable GMX\_PME\_NUM\_THREADS but only on the 2 allowed versions, if they are not supported you will be limited to use the provided commands on the description.

Additionally, scaling strong scaling results for a minimum performance of 5 ns/day, 10 ns/day, 20 ns/day should be provided using the (smallest) appropriate number of nodes to achieve this. The scaling results are used for the qualitative evaluation.

Commitment:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **GROMACS** | | | | | | |  |
| Min. application performance [ns/day] | MPI Tasks | OMP\_NUM\_ THREADS | NUMBER\_ OF\_CORES | NODES\_ | Wall\_t [sec] | Performance  (ns/day) | Performance Aggregation = Performance / NODES \* (Total number of nodes of this type in system) |
| 5 |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |

### Iphigenie/CPMD (QMMM Simulation of a Biomolecule)

Purpose: The Iphigenie/CPMD program couples classical molecular dynamics with polarizable force fields (Iphigenie) with density functional theory expanded in plane waves (CPMD), where the latter requires more than 90% of the total computational effort. The codes are written in C (Iphigenie) and Fortran90 (CPMD) and are MPI (Iphigenie) and MPI+OpenMP (CPMD) parallelized.

Web links: Iphigenie: <https://sourceforge.net/projects/iphigenie/>  
CPMD: <http://www.cpmd.org/)>

Building: A detailed description for compiling all necessary components can be found in the text file $BENCH/applications/Iphigenie-CPMD/README.md.   
In particular, item 1) refers to CPMD and item 2) refers to Iphigenie.

In the section “Benchmark instructions” item 3) references data are given with their tolerable relative errors.   
The executable that is used for the benchmark results must reproduce these results within the given error margins.

Extra Download: None.

Hints: Hints on compilation and optimization are given in the file:   
$BENCH/applications/Iphigenie-CPMD/README.md.

Code Owner: G. Mathias/LRZ (Iphigenie); The CPMD consortium (CPMD).

License: Running this benchmark requires a non-free license for CPMD, which can be requested on the CPMD website http://cpmd.org/ . The Iphigenie program is GPL code.

Source: $BENCH/applications/Iphigenie-CPMD/source

All necessary source files are located in the repository in the subdirectories ‘cpmd’ and ‘iphigenie’.

Porting and commitments for the quantitative evaluation must be based on the delivered source code.

Versions: **Additional results can be delivered that are based on other versions of CPMD than the supplied one. These will be considered for qualitative evaluation only.**

Executable: ‘iffi.<suffix>’

Testing: The benchmark system is a 220 atom peptide model (Trpzip) immersed in a (76 Å)3 box of polarizable water molecules. The peptide is treated by DFT, the water environment by a force field. The DFT electron density of the peptide is expanded on a 3003 spatial grid for a test and optimization setup and on a 5123 spatial grid for the benchmark setup.

*See* $BENCH/applications/Iphigenie-CPMD/README.md for details.

Goal of the benchmark is to fill the entire machine with multiple replicas, where all replicas run under a single MPI instance.

Input: Input files and sample batch scripts (based on LoadLeveler) are found in the directory $BENCH/applications/Iphigenie-CPMD/input.

*See* $BENCH/applications/Iphigenie-CPMD/README.md for details.

Preparation: Prior to the benchmarking a RESTART file containing the initial electron density has to be generated via the input file ‘rho.xml’ and the corresponding ‘rho.ll’ batch script. The results of this preparatory run also serve to check the numerical validity of the results (c.f. “Building” above).

The script ‘setup.sh’ then generates the input structure for the subsequent test and benchmark runs.

*See* ‘$BENCH/applications/Iphigenie-CPMD/README.md’ for details.

Logging: The log files of the runs are written to the directories/files ‘log\_<run name/replica index>/log\_<start\_time>-[replica index]-<mpi rank>.{out,err}’, where run name is either ‘rho’ or ‘single’ and the replica index is given only if multiple replicas are used. The ‘.out’ files of rank 000 contain the regular run output relevant the the benchmarks, whereas the ‘.err’ files contain warnings and run errors. Runs using the test input additionally contain ‘small’ in the file and path names. Relevant for the benchmark evaluation is the log file ‘log\_000/log\_<start time>-r000-n000.out’.

Output: Output directories start with ‘out\_’ and follow an equivalent naming scheme as the log directories. Reference output and logfiles are given in the directory ‘$BENCH/applications/Iphigenie-CPMD/reference\_output’.

Performance: The performance of the benchmark is measured in integration steps per minute of walltime. At the end of the log file the **last column** of the line starting with ‘

total................walltime

yields the **mean time per integration step.**

**LRZ requires a minimum performance per replica of: 0.0166 step/sec.**

The number of MPI process and/or OpenMP threads per replica has to be adjusted accordingly. The number of nodes used for a single replica has to be chosen by the Tenderer to satisfy this requirement. Typically, he will choose the smallest number of nodes which meets this criterion. Remaining nodes, that cannot filled with a replica need to be treated as described in Section 1.3. See also the README file in the benchmark directory.

The simulation speed (i.e. steps/sec) for all replicas is equal!

The performance per node is computed as:

Procedure: Fill the entire machine with multiple replicas of benchmark system (*5123 spatial grid*) where all replicas run under a single MPI instance with the provision, that the single replica exceed the performance requirement set above.

The batch script ‘mult.ll’ contains an example setup, where the number of replicas has to be set by the variable NUMREP.

Commitment:

|  |  |  |
| --- | --- | --- |
| **Iphpigenie/CPMD** | Quantitative evaluation  **CPMD 4.1** | Qualitative evaluation (optional)  **Other CPMD  Version** |
| Mean Time per Integration Step [sec] |  |  |
| Number of nodes per replica |  |  |
| Number of replicas to fill the system or all nodes of this type |  |  |
| Performance per replica [1/sec]= 1./(Mean Time per Integration Step) = **(Must exceed: 0.0166 step/sec)** |  |  |
| Performance per node [1/sec] = (Performance per replica) / (Nodes per replica) |  |  |
| Performance Aggregation = (Performance\_per\_node) \* (Total number of nodes of this type in system) |  |  |

### SeisSol

SeisSol is a solver for the seismic wave equations for complex geometries and materials with advanced rheology (including viscoelasticity, rupture dynamics and off-fault plastic deformation). The solver is written in Fortran, C/C++ and made parallel using a hybrid approach based on MPI and OpenMP. Associated tools are mostly written in Python. The code provides various I/O options that require the availability of POSIX-I/O, MPI-I/O, MPI-based HDF5, and NetCDF.

The SCons based build process includes generation of compute kernels based on the LIBXSMM[[5]](#footnote-6) library and code generator for small-scale matrix-matrix-multiplications.

Hints: Numerical background, applications characteristics, and hints for optimization of the small matrix kernels are given in “High performance seismic simulations”[[6]](#footnote-7).

Purpose: The code is currently mainly used in seismological projects for the simulations of realistic earthquake scenarios using a model of a heterogeneous subsurface with (visco)-elastic bulk material, pre-defined frictional interfaces (introduces nonlinear behaviour) and off-fault plastic deformation. The accurate numerical simulation of such advanced earthquake scenarios helps to understand complicated wave phenomena related to earthquake source propagation and earthquake triggering, strong ground motion, or Tsunami generation (among others). Discretization is based on unstructured tetrahedral meshes to account for complex geometries. The internal algorithms (ADER-DG) use arbitrarily high-order approximations in time and space as well as the option for an explicit local time stepping algorithm, such that each element with different sizes may run with different time steps which in general may reduce the overall computation time.

Code Owner: SeisSol Group (see authors for details)

Authors: $BENCH/applications/SeisSol/code/AUTHORS.txt

License: BSD-like license:

Copyright (c) 2012-2016, SeisSol Group, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Source: $BENCH/applications/SeisSol/code.tgz

Input data: Input data are contained in the separate file drplast.tgz in the download area.

Prerequisits: C/C++/Fortran compilers, MPI, OpenMP, HDF5, NetCDF, LIBXSMM[[7]](#footnote-8), Python, SCons

Readme: All details on compiling and running the code are provided within the README-files:

$BENCH/applications/SeisSol/README.txt or $BENCH/applications/SeisSol/README.docx

Build: Use case specific compilation steps are described within the README files.

Based on editable build configuration files (input for SCons-based build process) the generated executable binaries are named SeisSol\_<config-options> and can be found within

$BENCH/applications/SeisSol/code/build/

directory after successful compilation.

LIBXSMM: LIBXSMM is a library for small dense and small sparse matrix-matrix multiplications as well as for deep learning primitives. Details can be found in: https://github.com/hfp/libxsmm.

Note that the LIBXSMM-kernels that have to be used within this benchmark must be generated using the build-option ‘**dnoarch’** (as given also by build-configuration template config\_template.py located in the top directory of the benchmark). This option lets LIBXSMM generate architectural independent C-source code. The generated code is contained in the directory:

$BENCH/applications/SeisSol/code/build/build\_<programName>/generated\_code.

The actual programName is defined in config\_template.py.

All code modifications as described in 1.10 may be applied to the complete benchmark, particularly the files in “generated\_code”. This may include different compiler flags, directives and **alignment** than implied by the original **dnoarch**-flag of the build-process as defined in site\_scons/arch.py.

Also, during the code-generation process it might be advantageous or even necessary to change the **alignment specifications** accordingly. This can be done within :

generated\_code/gemmgen/Arch.py, e.g.

def getAlignment(architecture):

alignments = {

**'noarch': 16**,

'wsm': 16,

'snb': 32,

'hsw': 32,

'knc': 64,

'knl': 64

} return alignments[ getCpu(architecture) ]

The performance commitments must be based on the C-code version of LIBXSMM routines, using the rules described in 1.10.

**Optimized version:**

**Additional performance numbers based on any optimized version of LIBXSMM or of the source files of the “generated\_code” (including assembler/intrisic based and architectural specific code optimizations) may be provided. Kernels for this may be generated using the LIBXSMM framework or any suitable replacement.**

The reported performance will be assessed in the qualitative evaluation only (case: “drplast fully optimized LIBXSMM”).

Any of the provided partitionings might be used.

Benchmarks: Two strong scaling cases are provided.

* **TPV27\_1km**: a small-scale code-correctness benchmark which is located within the repository in $BENCH/applications/SeisSol/benchmarks/TPV27\_1km suitable for 1, 8, 16, 32, 64 MPI-tasks.
* **drplast:** a large case for performance evaluation which has to be downloaded as a tar-file from the FTP server. Domain partitionings for running the code on 128, 256, 512, 1024, 2048 or 4096 MPI-tasks are provided. The tar-file needs to be unpacked into the $BENCH/applications/SeisSol/benchmarks/drplast directory as described inside the README-files.

Workflow: Both cases must be run using the same executable binary.

Each case follows the following generic workflow:

* configuration step.
* preprocessing step,
* running SeisSol,
* postprocessing step.

All four steps are executed within the provided example batch-file in a sequence. Adoption of the provided example batch-file batch.sh (scheduler dependency) and the config.sh file as well as potentially the MPI startup within run.sh are typically required.

Code validation: For the code validation the case TPV27\_1km can be used.

N time-series of physical fields at specified locations must be compared with the reference-solutions (generated on SuperMUC). The provided Python-script should be used for the comparison ($BENCH/applications/SeisSol/benchmarks/tools/compare.py). Results get accepted if the reference (ref[:,i] and the test series (test[:,i]) are elementwise equal within a given tolerance value (atol) for each of the N normalized time-series (devider mx) with test and reference time-series stored within 2-D Numpy-arrays as scetched in the following Python pseudo-code:

atol=1.e-7

mx=abs(ref[:,i]).max()

if np.isclose(test[:,i]/mx,ref[:,i]/mx,atol=atol\*mx).all():

print ‘accepted’

The validation is automated within the post-processing part of the workflow (post.sh) which invoces compare.py, or can be invoked explicitly by the user. Use python compare.py –h to get help on its usage.

Performance: The main metric for the performance of each use case is the time spent inside the main loop of the application, the **looptime** **Tl**. which may be extracted from the logfile. Also the applicationwalltime Taw must be extracted.

Both time intervals looptime Tl and application walltime Taw should be recorded for the drplast benchmark. The difference Ti = Taw –Tl, the initialization-time, reflects the time spent for model initializations. (More details and an example are provided inside the README.txt within the repository).

Commitment: The input files for the benchmarks comprise (among others) several pre-partitioned computational meshes.   
The benchmarks should be run on different partitionings which makes it possible to provide information on strong-scaling behavior.

Note that in a real scientific simulation the time-loop will be iterated upon many more iterations (factor of about 5000 to 50000) which should render the initialization-time Ti typically negligible. Note that the performance based on the time-loop is most relevant, but the application walltime Taw should also be provided for potential bottleneck identification.

Commitments:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SeisSol use case Strong scaling** | **TPV27\_1km** | **TPV27\_1km** | **TPV27\_1km** | **TPV27\_1km** |
| Number of MPI tasks | 8 | 16 | 32 | 64 |
| Application walltime Taw [s] |  |  |  |  |
| OMP\_NUM\_THREADS |  |  |  |  |
| Validation passed [True or False] |  |  |  |  |
| LOOPTIME Tl [s] |  |  |  |  |
| NUMBER\_OF\_NODES |  |  |  |  |
| Loop-Performance [1/s] =  1/Tl |  |  |  |  |
| Loop-Performance per Node= Loop-Performance / NUMBER\_OF\_NODES |  |  |  |  |

Commitment:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SeisSol use case Partitioning Strong scaling** | **Drplast 128** | **Drplast 256** | **Drplast 512** | **Drplast 1024** | **Drplast 2048** | **Drplast 4096** | **Drplast best value (Loop performance per node) from  columns from the left** |
| Application walltime Taw [s] (not considered as a commitment) |  |  |  |  |  |  |  |
| Number of MPI tasks |  |  |  |  |  |  |  |
| OMP\_NUM\_THREADS |  |  |  |  |  |  |  |
| NUMBER\_OF\_NODES |  |  |  |  |  |  |  |
| LOOPTIME Tl [s] |  |  |  |  |  |  |  |
| Loop-Performance [1/s] = 1/Tl |  |  |  |  |  |  |  |
| Loop-Performance per Node= Loop-Performance / NUMBER\_OF\_NODES |  |  |  |  |  |  |  |
| Loop-Performace Aggregation= Loop-Performance \* (Number of Nodes of system) |  |  |  |  |  |  |  |

Optional information:

|  |  |
| --- | --- |
| **SeisSol use case Partitioning  fully optimized version LIBXSMM (optional)** | **Drplast** \_\_\_\_\_\_ |
| Application walltime Taw [s] (not considered as a commitment) |  |
| Number of MPI tasks |  |
| OMP\_NUM\_THREADS |  |
| NUMBER\_OF\_NODES |  |
| LOOPTIME Tl [s] |  |
| Loop-Performance [1/s] = 1/Tl |  |
| Loop-Performance per Node= Loop-Performance / NUMBER\_OF\_NODES |  |
| Loop-Performace Aggregation= Loop-Performance \* (Number of Nodes of system) |  |

### SUSPENSE

Purpose: The program solves the incompressible Navier-Stokes equations in primitive variable formulation, using a fractional-step (pressure-correction) algorithm. Temporal discretization is semi-implicit (Crank-Nicolson and Runge-Kutta-3), while the spatial operators are discretized with the aid of second-order, staggered finite-differences on a uniform isotropic mesh. The presence of solid rigid particles is described by an immersed boundary method which resolves the phase-interfaces. The algorithm is parallelized with the aid of three-dimensional Cartesian domain decomposition, and a specific protocol is employed for the handling of the particle motion crossing (processor) sub-domain boundaries. Finally, energy input into the system (turbulence forcing) is done through random Uhlenbeck-Ornstein processes in Fourier space. The aim of our simulations is to understand the interaction between particles with a size larger than the smallest flow scales and the turbulent energy cascade.

The benchmark is setup for weak scaling.

Code Owner: Uhlmann/KIT

Source: $BENCH/applications/SUSPENSE

Hints: LRZSCALE determines the number of MPI tasks.

Each task writes an output file of approximately 70 MB into one directory. This may result in contention or blocking for very large number of tasks. It might be advantageous to modify subroutine SAVE\_4FIELDS in this case (e.g. by using multiple directories, writing output into one file, or by using optimized IO schemes/libraries). However, the complete output must be produced by all simultaneous copies.

Compile: cd $BENCH/applications/SUSPENSE/SRC  
make veryclean  
**#modify LRZSCALE in dimensionp.dat\_in (this will change dimension.dat during make)**  
make

Procedure: Run the compile and run the benchmark for:

* + - CASE 1: LRZSCALE=8 (small)
    - CASE 2: LRZSCALE=16 (medium)
    - CASE 3: LRZSCALE=24 (large)
    - CASE 4: LRZSCALE=32 (very large)

#Preparation

rm –rf $SCRATCH/SUSPENSE  
mkdir $SCRATCH/SUSPENSE  
mkdir $SCRATCH/SUSPENSE/output  
cp $BENCH/applications/SUSPENSE/run\_LRZ/\* $SCRATCH/SUSPENSE

#Execution

COPIES=1

TASKS=$(($SCALE\*$SCALE\*$SCALE))

TASKS\_PER\_NODE=<tbd>  
NODES=$(( ($TASKS+$TASKS\_PER\_NODE-1) / $ TASKS\_PER\_NODE ))   
OMP\_NUM\_THREADS=<tbd>

cd $SCRATCH/SUSPENSE  
$RUN –C $COPIES –n $TASKS –N $NODES –T $TASKS\_PER\_NODE \  
 –t $OMP\_NUM\_THREADS $BENCH/applications/SUSPENSE/SRC/MAINP

Results: Timing results are given in the last lines of the benchmark. As the metric the duration of the time iteration loops is taken, which is contained in the line

RESULTS – Loops (secs):

The inverse of the duration is taken as the metric for the performance. An aggregation to the whole system is performed

Assessment of performance of general purpose compute nodes:

The aggregate compute performance for Case SCALE=8 might also be taken for assessing the performance of the general purpose compute nodes.

Commitments:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SUSPENSE**  **Weak scaling** | **LRZSCALE=8** | **LRZSCALE=16** | **LRZSCALE=24** | **LRZSCALE=32** | **LRZSCALE=8 may be also used for the assessment of the general  purpose compute  performance.**  **Only general purpose nodes** |
| RESULTS – Loops (secs) |  |  |  |  |  |
| NODES |  |  |  |  |  |
| Performance (1/sec) =  1 / (RESULT-Loops) |  |  |  |  |  |
| Performance per node =  Performance/NODES |  |  |  |  |  |
| For formal evaluation:  Performance Aggregation = (Performance per node) \* (Total number of nodes of this type in system) |  |  |  |  | Should be ≥ 0.33 |

## Storage Subsystem Benchmarks

This section refers to the file systems described in Section 3.5 of the *Description of Goods and Services SuperMUC-NG*. The benchmarks are included to verify the commitments for I/O performance.

The I/O benchmarks for performance evaluation must be set up so as to exhaust the memory buffers to ensure that the performance of the storage subsystem is actually being measured. To do this, the Tenderer must select the size of file(s) so that caching effect on IO performance are avoided. The following table lists the system parameters that must be defined and used for running the I/O benchmarks.

|  |  |
| --- | --- |
| I/O Parameters | Variables |
| Path to File System | FS |
| File system Block Size | FS\_BLOCK\_SIZE |
| Memory per Compute Node | MEM\_PER\_NODE |
| Cores per Compute Node | CORES\_PER\_NODE |
| Number of Compute Nodes | NODES |
| Number of MPI tasks per Compute Node | TASKS\_PER\_NODE |
| Path to HOME file system. Data Science Storage. | HOME |
| Path to PROJECT file system. Data Science Storage. Expected bandwidth 50 GiB/sec. | PROJECT |
| Path to SCRATCH or WORK file space. High Performance Parallel File System. Expected bandwidth 500 GiB/sec. | SCRATCH WORK |
| SSD storage layer | SSD\_STORAGE |
| Aggregate file size | AggFileSize |
|  | Flags |
| api: I/O Library for the test. It must be set to one of POSIX, MPIIO or HDF5 depending on test. | -a api |
| Reorder Tasks: changes task ordering to n+1 ordering for readback | -C |
| Intra test barriers: use barriers between open, write/read, and close. | -g |
| perform fsync upon POSIX write | -e |
| writes file(s), first deleting any existing file. | -w |
| reads existing file(s) (from current or previous run) | -r |
| accesses a single (separate) file for each task. | -F |
| Repetitions - number of times to run each test | -i |
| size of transfer (in bytes) that is the size of a single data buffer to be transferred in a single I/O call.  Note: transferSize – may be expressed as multiple of the FS\_BLOCK\_SIZE to achieve the maximum data transfer rate. For small files testing this can be less than FS\_BLOCK\_SIZE. | -t TransferSize |
| size (in bytes) of a contiguous chunk of data accessed by a single process; it is comprised of one or more transfers. | -b BlockSize |
| number of segments in file/s. A segment is a contiguous chunk of data accessed by multiple processes each writing/reading their own contiguous data; comprised of blocks accessed by multiple clients. With HDF5 this repeats the pattern of an entire shared dataset. | -s SegmentCount |
| name of the output file/s. | -o testFile |
| use unique directory name for each file-per-process. This should be included for more than 1000 files. | -u uniqueDir |
| Deadline for Stonewalling. Seconds before stopping write or read phase. Incompatible with data checking. This option should be used with care. It must not circumvent the intension of the benchmarks. | -D seconds |

### IOR benchmark

Purpose: IOR benchmark measures the I/O performance of parallel file systems by using different I/O libraries such as HDF5, MPIIO and POSIX-IO. IOR uses MPI for tasks synchronization.

Source: $BENCH/IO-bench/IOR-2.10.3

Original source: http://sourceforge.net/projects/ior-sio

License: GNU General Public License version 2.0 (GPLv2). See $BENCH/IO-bench/IOR-2.10.3/COPYRIGHT.

Executable: IOR (in IOR-2.10.3/src/C)

Compile: For MPIIO and POSIX:

gmake mpiio CC=mpicc

For HDF5:

module load hdf5/mpi ## Load parallel hdf5

gmake hdf5 CC=h5pcc

Modifications: For Tests 1 to 4 the transfer size (parameter –t) can be adjusted by the Tenderer to yield optimal throughput. The data per node must be greater or equal than the amount of memory of a compute node.

Using POSIX-IO the data per node must be at least MEM\_PER\_NODE; and for MPI-IO and HDF5 interface it must be at least 2\*MEM\_PER\_NODE.

FS must be adjusted for the different file systems.

Use of the -D option is optional and the procedure to use it must follow the recommendation of the section “HOW DO I USE STONEWALLING?” of the USER\_GUIDE of IOR version 2.10.3.

Results: The data transfer rate or bandwidth is calculated as the amount of data transferred divided by the elapsed time where timing is from the first file open to last file closed.

IOR generates an aggregate filesize equal to where corresponds to NODES or NODES\*TASKS\_PER\_NODE depending on the number of the Test (1 to 6).

IOR bandwidth is expressed in MiB where   
1 MiB = 220 bytes = 1024 kibibytes = 1048576 bytes.

Output example:

IOR-2.10.3: MPI Coordinated Test of Parallel I/O

Run began: Tue Feb 9 11:50:49 2016

Command line used: /home/hpc/pr28fa/di98het/IOR/src/C/IOR-ibmpi\_1.4-Ph1 -a MPIIO -Cge -vv -wWr -i 5 -b 256m -t 8m

Machine: Linux i19r02a26 3.0.101-0.47.67-default #1 SMP Wed Aug 19 14:12:37 UTC 2015 (265cc32) x86\_64

Using synchronized MPI timer

Start time skew across all tasks: 0.01 sec

Path: /gss/scratch/pr28fa/di98het/IOR-MPIIO-SMUCNG-RS

FS: 5247.4 TiB Used FS: 76.8% Inodes: 512.0 Mi Used Inodes: 41.8%

Participating tasks: 256

Summary:

api = MPIIO (version=3, subversion=0)

test filename = testFile

access = single-shared-file, independent

pattern = segmented (1 segment)

ordering in a file = sequential offsets

ordering inter file=constant task offsets = 1

clients = 256 (16 per node)

repetitions = 5

xfersize = 8 MiB

blocksize = 256 MiB

aggregate filesize = 64 GiB

Operation Max(MiB) Min(MiB) Mean(MiB) StdDev Max(OPs) Min(OPs) Mean(OPs) StdDev Mean(s)

write 36189.11 33235.68 34576.44 1262.14 4523.64 4154.46 4322.06 157.77 1.89791

read 40014.46 35984.78 38639.80 1487.93 5001.81 4498.10 4829.97 185.99 1.69869

Max Write: 36189.11 MiB/sec (37947.03 MB/sec)

Max Read: 40014.46 MiB/sec (41958.20 MB/sec)

Run finished: Tue Feb 9 11:51:16 2016

Procedure: The following table presents the minimum values for aggregate file size considering the IOR Blocksize (parameter b), the number of MPI tasks and segments (parameter s). Values are selected to produce an I/O burst of 60 seconds that is repeated 10 times and must provide at least the expected minimum data transfer rate presented in Table 1.

|  |  |
| --- | --- |
| Filesystem | IOR parameters |
| HOME  (SSD and  HDD Tiers) | **Test Type 1: POSIX One file per node**  NODES ≥ 32 BlockSize ≥ MEM\_PER\_NODE  TransferSize: may be multiples of FS\_BLOCK\_SIZE AggFileSize ≥ 1TiByte $RUN –n NODES –N NODES –T 1 \  ./IOR –a POSIX –Cge –wWr –F –i 10   –t TransferSize –b BlockSize –o FS/testFile |
| PROJECT | **Test Type 1: POSIX One file per node**  NODES ≥ 32 BlockSize ≥ MEM\_PER\_NODE and  BlockSize ≥ 10 TiByte/NODES TransferSize: may be multiples of FS\_BLOCK\_SIZE AggFileSize ≥ 10TiByte $RUN –n NODES –N NODES –T 1 \  ./IOR –a POSIX –Cge –wWr –F –i 10   –t TransferSize –b BlockSize –o FS/testFile |
| SCRATCH WORK  Test must be run with SSD\_STORAGE disabled | **Test Type 2: POSIX One file per MPI task, l*arge*** *files*  NODES ≥ 64 BlockSize ≥ MEM\_PER\_NODE/MPI\_TASKS\_PER\_NODE and  BlockSize ≥ 100 TiByte/(NODES\*MPI\_TASKS\_PER\_NODE) TransferSize: may be multiples of FS\_BLOCK\_SIZE AggFileSize ≥ 100TiByte $RUN –n NODES –N NODES\*MPI\_TASKS\_PER\_NODE \  –T MPI\_TASKS\_PER\_NODE\  ./IOR –a POSIX –Cge –wWr –F –i 10 \  –t TransferSize –b BlockSize –o FS/testFile  **Test Type 3: MPIIO Shared File, one writer per MPI task, strided pattern**  NODES ≥ 64 SegmentCount=16 BlockSize ≥ 2\*MEM\_PER\_NODE/(MPI\_TASKS\_PER\_NODE\*SegmentCount) and BlockSize ≥ 100 TiByte/(NODES\*MPI\_TASKS\_PER\_NODE\*SegmentCount) TransferSize: may be multiples of FS\_BLOCK\_SIZE AggFileSize ≥ 100TiByte  $RUN –n NODES –N NODES\*MPI\_TASKS\_PER\_NODE \  –T MPI\_TASKS\_PER\_NODE \  ./IOR –a MPIIO –Cg –wWr –i 10 \  –t TransferSize –b BlockSize –s SegmentCount \  –o FS/testFile  **Test Type 4: HDF5 One file per node**  NODES ≥ 64 BlockSize ≥ 2\*MEM\_PER\_NODE and BlockSize ≥ 100 TiByte/NODES TransferSize: may be multiples of FS\_BLOCK\_SIZE AggFileSize ≥ 100TiByte  $RUN –n NODES –N NODES –T 1 \  ./IOR –a HDF5 –Cg –wWr –F –i 10 \  –t TransferSize –b BlockSize –o FS/testFile |
| SSD\_STORAGE on SCRATCH  Testing the mechanism to accelerate latency-bound IO patterns.  Each test must be run with SSD\_STORAGE enabled. | **Test Type 2: POSIX One file per MPI task, large files**  NODES ≥ 64 BlockSize ≥ MEM\_PER\_NODE/MPI\_TASKS\_PER\_NODE and  BlockSize ≥ 50 TiByte/(NODES\*MPI\_TASKS\_PER\_NODE) TransferSize: may be multiples of FS\_BLOCK\_SIZE AggFileSize ≥ 50TiByte $RUN –n NODES –N NODES\*MPI\_TASKS\_PER\_NODE \  –T MPI\_TASKS\_PER\_NODE\  ./IOR –a POSIX –Ce –wWr –F –i 10 \  –t TransferSize –b BlockSize –o FS/testFile  **Test Type 5: MPIIO A single shared file**  NODES ≥ 64 BlockSize ≥ MEM\_PER\_NODE/MPI\_TASKS\_PER\_NODE and  BlockSize ≥ 50 TiByte/(NODES\*MPI\_TASKS\_PER\_NODE) TransferSize: may be multiples of FS\_BLOCK\_SIZE AggFileSize ≥ 50TiByte  $RUN –n NODES –N NODES –T 1\  ./IOR –a MPIIO –Cg –wr –i 5  –t TransferSize –b BlockSize –o FS/testFile |

Commitments:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **File  system** | **Test Type** | **Read Mean (MiB/s)** | **Write Mean (MiB/s)** | **Requirement** |
| HOME SSD tier (primary file system) | 1 |  |  | No Requirement |
| HOME HDD tier (primary file system) | 1 |  |  | No Requirement |
| PROJECT | 1 |  |  | 50 GiB GB/s ≤ MIN(READ Testcase, WRITE Testcase) |
| SCRATCH  WORK | 2 |  |  | 500 GiB GB/s ≤ MIN(READ Testcase, WRITE Testcase) |
| 3 |  |  | No Requirement |
| 4 |  |  | No Requirement |
| SSD\_STORAGE on SCRATCH | 2 |  |  | No Requirement |
| 5 |  |  | No Requirement |

Mean(MiB)in the output is the mean aggregate data rate of i iterations that is expressed in MiB/sec.

### FIO benchmark

Purpose: FIO is an I/O tool meant to be used both, for benchmark and stress/hardware verification. FIO is used to generate the random I/O operations and it can be used to report the I/O operations per second (IOPS).

Source: $BENCH/IO-bench/fio-2.12

Original source: http://git.kernel.dk/cgit/fio/

License: GNU General Public License version 2.0 (GPLv2). See $BENCH/IO-bench/fio-2.12/COPYING.

Executable: fio

Compile: ./configure --cc=$CC --prefix=$INSTALL-PATH  
make  
make install

Procedure: FIO benchmark must be executed on both the SSD and HDD tiers of primary HOME. The FIO’s output is used to report the IOPS.

Command line:

fio --ioengine=libaio --rw=… --bs=… --size=TOTAL\_IO \

--refill\_buffers --direct=1

--rw= string must be set to randread and randwrite

--bs=*integer* Block size (BLOCK\_SIZE) for I/O units that must be **4K.**

--size=*integer* Total size of I/O for the job (much larger than the caches before the SSDs)

Modifications: iodepth (Default 1) and numjobs (Default 1) parameters should be select by the Tenderers to obtain the maximum reachable IOPS. If the Client/Server mode is used, The Tenderer should select the number of clients to obtain the maximum IOPS provide by the file system to evaluate.

Reference: The file(s) should be much larger than the caches before the SSDs. Every SSD must be filled at least once to see how the SSD behaves under real world conditions (internal garbage collection running).

Results: IOPS for read and write operations using BLOCK\_SIZE: 4K and 8K.

Output example: If the FIO’s output is used to commitment, then it must report the field iops of the output. Below, an example is shown for such case.

For the command line:

fio --ioengine=libaio --iodepth=1 --direct=1 --refill\_buffers \

--size=256m --directory=FS/FIO-TEST --bs=4k --rw=randwrite

The output reads like:

**write**: io=262144KB, bw=1051.4KB/s, iops=262, runt=249337msec

Commitments:

|  |  |  |
| --- | --- | --- |
| **Filesystem** | **HOME (SSD Tier)** | **HOME (HDD Tier)** |
| 4K Read IOPS |  |  |
| 4K Write IOPS |  |  |

### IOZONE benchmark

Purpose: IOzone is a file system benchmark tool that generates and measures a variety of file operations. IOzone is used to measure the bandwidth in an exported view with NFS or CIFS of a file system.

Source: $BENCH/IO-bench/iozone3\_430

Original source: <http://www.iozone.org/src/current>

License: <http://www.iozone.org/docs/Iozone_License.txt>

Executable: iozone

Compile: cd $BENCH/IO-bench/iozone3\_430/src/current

make $TARGET-SYSTEM

Where $TARGET-SYSTEM depends on the architecture of the machine. For example: $TARGET-SYSTEM=linux-AMD64 for architectures x86\_64.

Procedure: IOzone benchmark must be run for NFS and CIFS exports of primary HOME and PROJECT via CES.

Command line: $BENCH/IO-bench/iozone3\_430/src/current/iozone –i 0 –i 1 \

–r FS\_BLOCK\_SIZE –s 1t

Where:

-i Used to specify which tests to run (0=write/rewrite, 1=read/re-read)

-r Used to specify the record size, in Kbytes, to test. One may also specify -r #k (size in Kbytes) or -r #m (size in Mbytes) or -r #g (size in Gbytes).

-s Used to specify the size, in Kbytes, of the file to test. One may also specify -s #k (size in Kbytes) or -s #m (size in Mbytes) or -s #g (size in Gbytes).

Modifications: The number of parallel processes and nodes are subject to the Tenderer.

Results: The Output is in kBytes/sec. The Tenderer must report values in GiB/sec for the read and write operations.

Commitments:

|  |  |  |  |
| --- | --- | --- | --- |
| **Filesystem** | **Read (GiB/s)** | **Write (GiB/s)** | **Requirement** |
| HOME or PROJECT via CES-NFS |  |  | 20 GiB/s ≤  MIN(READ Testcase, WRITE Testcase) |
| HOME or PROJECT via CES-CIFS |  |  | 4 Gib/s ≤  MIN(READ Testcase, WRITE Testcase) |

### Bonnie++ MPI metadata benchmark

Purpose: This benchmark tests the metadata performance for the different file systems. In this benchmark (based on Bonnie++), each MPI tasks writes files into a separate directory.

Source: $BENCH/IO-bench/mpibonnie

License: Bonnie++ License see $BENCH/IO-bench/mpibonnie/copyright.txt

Executable: mpibonnie

Build: ./configure --prefix=$INSTALL/io\_bench/mpibonnie/build

make

make install

Procedure: The benchmark creates a directory for every MPI task and then creates 0-Byte files inside the directories. You have to provide a path where the directories will be created.

For the acceptance test the benchmark will be run for two kind of tests:

* Test 1: One MPI task per compute using at least 64 NODES
* Test 2: CORES\_PER\_NODE tasks per compute node using at least 64 NODES.

The benchmark must be executed with the thin storage layer enabled and disabled, resp.

The whole benchmark (creating files, using stat() and deleting files) must be completed successfully but only the file creation performance is evaluated.

Running the benchmark:

mpirun –np TASKS mpibonnie -n FILES/1024 -d PATH\_FILESYSTEM -x 1 -s 0 -f > log.out

Where:

-n number of file to create for each MPI task. This is measured in multiples of 1024 files.

-d path to filesystem.

-x number of tests.

-s size of file in MiB.

-f in this case without argument to skip tests per-char.

Post-process the results:

perl bonniempi\_postprocess.pl log.out

Modifications: Distributing directories to different parts of a namespace is allowed. The numbers of files per MPI task can be selected by the Tenderer to provide the maximum metadata rate.

Results: The result is the number shown in the column “Sequential create”/”Create/sec” in the output of the post.process-script (first column).

Commitments:

|  |  |  |
| --- | --- | --- |
| **File System** | **File Creations/sec** | |
| **Test 1** | **Test 2** |
| SCRATCH/WORK SSD\_STORAGE **disabled** |  |  |
| SCRATCH/WORK SSD\_STORAGE **enabled** |  |  |

## Energy efficiency

Because of the high energy prices in Europe and particularly in Germany, only highly energy efficient systems are economically viable in the future. Also considering environmental and ethical aspects, energy efficiency is considered a value itself. Therefore, energy efficiency is separately evaluated.

For the HPL benchmark, the AC power draw of the system has to be measured (excluding the disks and not taking into account the AC power for cooling and air conditioning but including the high performance interconnect). The benchmark can also repeatedly run for a timespan which is long enough to yield reliable energy and power measurements.

**For the AC power measurements the power measurement devices delivered with SuperMUC-NG Phase 1 will be used (see Description of Goods and Services SuperMUC-NG, chapter 3.6).**

The energy efficiency is calculated as

For Rmax the configuration used for the HPL benchmark must be used (section 2.2.4, case “Entire System”).

PowerHPL-CN is the mean power draw of involved compute nodes during this HPL run.

Commitment:

|  |  |
| --- | --- |
| **Energy Efficiency** |  |
| Rmax HPL performance (from Section 2.2.4) [PFlop/s] |  |
| PowerHPL-CN, Mean Power draw of involved compute nodes during the HPL benchmark [MW] |  |
| eff (energy efficiency of the system [PFlop/s per MW]) |  |

## Estimated Mean Power Draw and Energy Cost

The estimated cost for energy of the system for an operational period of 6 years must be calculated as defined in the documents “*Anschreiben SuperMUC-NG* (page 2 and 3)” and “*Bewerbungs- und Vertragsbedingungen SuperMUC-NG*” with regard to the percentage of heat which is dissipated into the cold water loop, the warm water loop and into air. The percentage of heat dissipated into cold water loop, warm water loop, and into air is denoted as fcold, fwarm, and fair, respectively.

To determine the estimated mean AC power draw of the system for an operational period of 6 years, the following formula must be used:

EnergyCostestimated = (fcold \*1.40 + fwarm \*1.06 + fair \*1.60) \* \* 5,000,000 €/MW

with (fcold + fwarm + fair) = 1

Where is the sum of the following system power draw values:

* AC power draw of the SuperMUC-NG Phase 1 system executing the HPL benchmark for the TOP500 list
* AC power draw of all cloud nodes running single node instances of the HPL benchmark on all computational components
* AC power draw of all SuperMUC-NG Phase 1 login and management nodes
* AC power draw of WORK and SCRATCH storage and all Ethernet network components of SuperMUC-NG Phase 1

Note: PowerHPL-CN are different.

By deploying adsorption cooling to convert process heat to a cold-water equivalent, the fractions fcold and fair will be reduced in favour of a higher fwarm. Since the COP for adsorption cooling is considered to also be 1.06, no change to the formula for energy cost estimation is needed.

The power draw of the system during normal user operation must be calculated as follows:

Powerestimated = 0.65 \*PowerHPL

|  |  |  |
| --- | --- | --- |
| **Estimated mean power draw and energy cost** | Without adsorption  cooling | If applicable, with  adsorption cooling |
| PowerHPL [MW] |  | |
| Powerestimated, Estimated mean power of the entire system [MW] |  | |
| fcold, percentage of dissipated heat into cold water |  |  |
| fwarm,percentage of dissipated heat into warm water |  |  |
| fair, percentage of dissipated heat into air |  |  |
| EnergyCostestimated, estimated energy cost [€] to be subtracted from the total available budget. |  |  |

1. The estimated cost for energy must be subtracted from the total available budget to yield the budget for investment and maintenance (see “*Anschreiben SuperMUC-NG*”).

To confirm your accordance with this, check here **[ ]**

END OF DOCUMENT

1. The sources of the requirements are detailed in the *Description of Goods and Services SuperMUC-NG*, Section 1.9. [↑](#footnote-ref-2)
2. The NDAs of all Tenderers are already accepted by LRZ and need not be submitted again. [↑](#footnote-ref-3)
3. For commitments relating to Phase 2 see “*Anschreiben SuperMUC-NG*” [↑](#footnote-ref-4)
4. https://www.spec.org/cpu2006/Docs/readme1st.html#Q14 [↑](#footnote-ref-5)
5. https://github.com/hfp/libxsmm [↑](#footnote-ref-6)
6. http://dx.doi.org/10.1016/B978-0-12-809194-4.00021-1 [↑](#footnote-ref-7)
7. https://github.com/hfp/libxsmm [↑](#footnote-ref-8)