H2020-FETHPC-01-2016

DEEP-EST

DEEP Extreme Scale Technologies

Grant Agreement Number: 754304

D2.2

Initial Application Analysis and Models

Final

Version:	1.0
Author(s):	J. Corbalan (BSC)
Contributor(s):	R. Fischer (BSC), J. Gimenez (BSC), C. Navarrete (BAdW-LRZ), G. Llort (BSC), L. Mercadal (BSC)
Date:	30.07.2019

Project and Deliverable Information Sheet

DEEP-EST Project	**Project ref. No.:**	754304
	Project Title:	DEEP Extreme Scale Technologies
	Project Web Site:	http://www.deep-projects.eu/
	Deliverable ID:	D2.2
	Deliverable Nature:	Report
	Deliverable Level: PU *	**Contractual Date of Delivery:** 30/June/2019
		Actual Date of Delivery: 30/July/2019
	EC Project Officer:	Juan Pelegrin

*– The dissemination level are indicated as follows: **PU** - Public, **PP** - Restricted to other participants (including the Comissions Services), **RE** - Restricted to a group specified by the consortium (including the Comission Services), **CO** - Confidential, only for members of the consortium (including the Comission Services).

Document Control Sheet

Document	**Title:** Initial Application Analysis and Models	
	ID: D2.2	
	Version: 1.0	**Status:** Final
	Available at: http://www.deep-projects.eu/	
	Software Tool: LaTex	
	File(s): DEEP-EST_Initial Application Analysis and Models_1.0.pdf	
Authorship	**Written by:**	J. Corbalan (BSC)
	Contributors:	R. Fischer (BSC), J. Gimenez (BSC), C. Navarrete (BAdW-LRZ),G. Llort (BSC), L. Mercadal (BSC)
	Reviewed by:	H. E. Plesser (NMBU) M. Bamberg(JUELICH)
	Approved by:	BoP/PMT

Document Status Sheet

Version	Date	Status	Comments
1.0	30.07.2019	Final	EC submission

Document Keywords

Keywords:	DEEP-EST, HPC, Exascale, Performance models, Energy models, Workload models

Table of Contents

List of Figures

List of Tables

Executive Summary

WP2 covers different aspects related to the evaluation and modeling of the DEEP-EST project. At the end of the project WP2 will provide:

- A benchmark suite as well as the automatic execution and evaluation in the JUBE tool.
- A set of workloads to be used for WP5 policies evaluation.
- Performance scalability models.
- Energy models.

Performance models and Energy models can be used in two contexts. Firstly, for application analysis and prediction, secondly, as part of the input for WP5.

This deliverable covers: workload models, performance models, and energy models. As the main contribution of the deliverable, we present initial results in application analysis and performance models. We will show initial results for the workload model generation. And finally, we also present the status and first results for the energy models.

1 Introduction

The performance of a systems comes from several sources. Individual applications can be efficiently designed to be scalable and to exploit the benefits of a modular architecture, but the specific mix of applications, or workload, will have a significant impact on the global performance. Application characteristics such as the flexibility to be executed on different modules rather than on a fixed one, the load of the different modules, the percentage of jobs asking for more than one module at the same time, will also influence the system utilization and average slowdown. For instance, if one job can be executed in two different modules, and one module is less loaded than the other, the scheduler could decide to start the job in the less loaded module reducing the wait time and alleviating the load of the module where it was initially allocated.

We have studied the state of the art workloads and workload models to select the best suited and already available workload model as starting point for us. We also analysed if some of the available workload traces in heterogeneous systems included information that can be utilized in our work showing different loads depending on the sub-cluster or different application sizes. Based on this previous study, we concluded there are no models or workload traces that can reflect the workload characteristics in a modular system. We have decided to extend one of the most used workload models to consider multiple modules and the possibility to execute a given job in more than one module. This initial work only covers one of the workload requirements: applications being able to run in more than one module. The second requirement, applications requesting more than one module to run is currently being analysed in cooperation with WP1 and will be included in our model in the next iteration of the process. To be able to provide some results about how workloads will provide different system metrics depending on the amount of job flexibility existing in workload, we have used our approach to generate workload traces where some jobs were allowed to run on (potentially) different partitions. Multiple partitions support is a SLURM feature with some similarities to the current Tk5.5 work in terms of scheduling. In this deliverable we are focussing on the potential of this approach and not on a quantitative evaluation of the results.

In this deliverable we focus on the analysis on the GROMACS application with respect to the performance model. GROMACS is a well known parallel code for molecular dynamics that does not follow Single Program Multiple Data (SPMD) paradigm making it attractive for an execution in a heterogeneous platform. The analysis demonstrates the high coupling of the processes on the current version as well as that load balance is the current main bottleneck. Strong scaling makes it difficult to extrapolate to large scale as the input size determines the boundaries in the process range for the execution. Using an input case that targets between 1 and 64 nodes (with 24 processes per node) and collecting traces of 1,2,4,8, and 16 nodes we extrapolated the results and validated them with non-instrumented runs up to 256 nodes. The deliverable shows the potential of the BSC performance tools and models with a focus on different study types that can be faced with the Dimemas simulator.

Finally, we are also working on a hot topic: energy management, with a focus on energy modelling. Currently, systems are not accounting users for energy utilization, but it is just a matter of time before system administrators and policy centers will define energy utilization, as well as a maximum number of CPU hours, and a maximum number of Joules to be consumed by users. In the end, energy is a much more generic metric, including CPU hours, and has a direct

impact on electricity bill. CPU hours can be envisioned as a metric that reflects the static cost of the infrastructure, but energy reflects the dynamic utilization. Given these considerations, using energy in the most efficient way will be key for HPC systems in the near future. WP2 contribution in this deliverable is limited to a first version of an energy model for Skylake architectures. Current ongoing work includes specific synthetic kernels and metrics for GPUs to extend the model with HW characteristics existing in the DAM and the ESB modules. Additionally, we describe the benchmark used to collect metrics that are the input for the modelling process. This benchmark executes a list of synthetic codes, experiments, with different characteristics and under conditions such as compiler flags, frequencies, vector sizes etc. All the experiments are executed in single node mode, generating a set of metrics per experiment as a result. The computation of the model is based on a Principal Components Analysis (PCA) [17], filtering not relevant values and resulting in a model made by 8 input parameters. The main conclusion is that the energy, which highly depends on the processor type and its features, can be easily predicted and its computation consists of a simple matrix multiplication operation.

Using a model calculated for a certain processor that does not correspond to the processor on which the application will run can lead to relative errors much greater than desired and hence, there is a need to apply a specific model for the current architecture. As it will be shown in the section 4, the maximum relative error between the foreseen energy use and the measured one is always lower than 5%.

2 Modelling and Validation of Scheduling Policies

This Chapter covers the analyses of workload models for MSA scheduling evaluation, including the classification of different job types that need to be considered in the workload.

1. Single-module job - with strict hardware (HW) requirements (e.g. requires GPU acceleration or a considerable amount of RAM memory to process). These jobs can only run in one specific module and they correspond to the traditional HPC applications.
2. Single-module jobs with flexible HW requirements - run on a single module but could run on another one (e.g. general purpose, do not require any specific resource). These jobs are traditional HPC jobs where different binaries can be provided or same binary can dynamically detect the available HW resources and select different kernels or libraries. This scenario is not new related to applications, but in terms of scheduling, since multiple modules are envisioned as a single system.
3. Modular jobs are jobs running on more than one module at the same time (e.g. partial run on nodes with GPU acceleration, partly run on nodes of DAM with substantial RAM memory requirements).
4. Modular jobs with dependencies between components - jobs running on more than one module where one component has to be executed before the others since it provides some preprocessing. Dependency can be for the whole or partly execution time.

Modular jobs can also be named heterogeneous jobs and we can refer to modular jobs with dependencies as workflows.

Single-module jobs with flexible HW requirements can positively impact the system utilization by improving the global average slowdown. This deliverable shows the methodology to create workloads for modular architectures and in particular targets these types of jobs. We will also show our initial results from our experiments with multiple modules. Section 2.1 checks existent models for workloads that can represent the characteristics that we discussed above. Therefore, we analysed the existent models that generate synthetic traces based on real production systems. We were unable to find a model that represents the desired characteristics. Consequently, we have decided to produce a variation of an existing model that fits our requirements and allows customization of the produced traces.

2.1 Workload Models

One workload is a sequence of enumerated jobs where detailed information for each job can be found to be either simulated or reproduced. Workloads are used for job scheduling evaluation and/or system analysis. Workloads used in job scheduling come from two sources: workload traces coming from real production systems or workload traces generated using workload models. Since the DEEP-EST project is proposing a modular architecture, there are no existing traces to be used as reference. Existing workload models are created by analysing workload

traces and extracting main characteristics. The benefit of workload models against workload traces is that they allow creating new traces according to the system size, experiment duration etc.

Existent workload models aim to realistically represent production systems and therefore, their main goal is to provide the closest possible characterization for a submission/arrival pattern to the system, allowing the evaluation of the scheduling polices. The models allow customization of details such as when jobs are submitted to run on the system, at which frequency, on which modules, and number of nodes. Producing a realistic model derived from existent traces from production systems was a main component for the proper evaluation of scheduling polices for parallel computers.

A trace repository exists that is widely used for scheduling evaluation, including workload logs [19] and models [20]. For this deliverable, we have used traces and models available there as reference. For the DEEP-EST project we faced the challenge of developing a workload model that focusses on the modular architecture, which is a challenge for the particular novelties on such system. There are no previous references and thus, it is challenging to determine which traces are representative. This demands an envisioning characterisation of what would be the main traits of a modular system rather than just a more simple and practical analyses and replication of production traces. Therefore, we needed to identify the characteristics and particularities of the system by envisioning how the jobs will be placed on the system, instead of overhauling existing traces to confirm the accuracy of the model for further calibration. Consequently, we decided to create our traces based on existing models that would take into account traditional aspects such as: arrival patterns, job size, and synthetically generated variations of these traces to explore the effect of adding jobs with the characteristics presented before with different percentages. So traces generated with the model could be seen as workload templates extended with modular characteristics.

Lublin [12] is a very detailed workload model for rigid jobs that includes an arrival pattern with a daily cycle, runtimes that are correlated with the number of nodes, and a distinction between interactive and batch jobs. This model has been extensively used for scheduling evaluation [21]. The outputs of this model are based on the Standard Workload Format and are effortlessly translatable into traces to be used by the SLURM Simulator [3].

We also evaluate the possibility to customize this model according to our needs and we conclude that modifying its source code to generate traces suitable for each of the different modules was a feasible task to be performed. Also, we could easily generate Modular Workload Format traces derived from the Standard Workload Format (SWF) traces obtained by using the Lublin model.

Due to the modular characteristic of the project, we needed to consider the creation of independent traces for each module, respecting limitation of system size and the particular specifications of the nodes. We created a tool that combines several independent outputs (each one is a SWF trace for an independent module) from the Lublin model to generate heterogeneous traces composed by several jobs that can run on different modules. This approach produced a preliminary workload version with duplicated IDs, as each trace was generated for a single module. Since the DEEP-EST architecture offers a unified system, a post-processing of the workload was still needed . As the final step, we unified the IDs by combining all generated traces from different modules into one single output and then we modified the IDs properly

with a final modification that generates new IDs considering the arrival time on the system. Therefore, our methodology can be summarised as:

1. We generate several traces, each trace for a different module.
2. All generated traces are combined into a single file.
3. A post-processing tool replaces all the jobs IDs by proper IDs that are obtained considering the arrival time on the system.
4. The produced output is ready to be run by the SLURM Simulator.

2.2 Efficient Job scheduling for modular architectures

There are two new challenging scenarios the MSA scheduler needs to manage to provide support for single-module jobs with flexible requirements and modular jobs:

1. The application can be executed on different modules and the scheduler should choose the module that gives the lowest slowdown, and
2. the application is executed simultaneously on multiple modules.

The next sections will describe how to enable efficient scheduling considering the first scenario. The work done on WP5 (Tk5.5) can benefit from our early results in order to design and prototype a more efficient job scheduler that fits the modular architecture of DEEP-EST. Concerning second scenario, we have collected information from WP1 to characterise how many applications will be able to run on more than one module at a time and which are the most typical combinations of modules we could expect. In a next step we will include this information in our traces to consider this type of jobs. We have worked on developing the required functionality on the simulator to support this scenario, but the description of this work is out of the scope of the deliverable.

2.2.1 Preliminary analysis of workloads with module flexibility

The evaluation of the impact of module flexibility in system metrics cannot be done at this moment since the scheduler extension is still under development. One of the main technical challenges when dealing with a single job submission is the execution on different modules and job requirements are different in terms of number of cpus, execution time, etc. HW requirements are needed inputs that must be either provided by the user or estimated by the system automatically. Asking N resource requirements (1 per module) to users when submitting jobs rather than one would increase the complexity of the system utilization and could have a negative impact on the success of the project. Therefore, we decided in cooperation with WP5, resource requirements for additional modules would be automatically estimated based on the requirements provided by users for the preferred module. Given that decision, our traces only have to include, apart from standard fields, the list of ordered modules where each job could be executed.

With this simplification, the idea behind this contribution that will support jobs running in different modules is conceptually similar to what is already considered in SLURM partitions. Because of that, we have used the already existing SLURM partition mechanism to simulate our traces and to provide a preliminary evaluation of the impact when simulating the modular architecture of the project (taking into account the effect itself of having different architectures can not be evaluated yet).

The concept of partitions is used in SLURM to separate resources in multiple subsets, in order to specify different limitations in each of them in terms of maximum wall clock time, maximum number of nodes requested, etc.

These resources are supposed to be homogeneous among partitions, since users are submitting same sets of requirements and wallclock time to all the partitions specified. It is possible that a user specifies the list of partitions on which the job can be executed [6].

As long as the requested resources and time are within the per-partition limits, SLURM can choose any of the partitions in the user specified list from which to allocate resources for user's job. SLURM creates one register job_queue_rec per partition, and once a job is allocated resources, other records for that job in the queue will be ignored.

The traces created for this preliminary evaluation consider jobs that are allowed to run on multiple partitions. It is only needed to substitute the partition field by the module list field once Tk5.5 work is ready to be evaluated.

Additionally, we have considered the information provided by WP1 regarding the benchmark applications and the possibility for each application to be run on more than one module. Currently we have 6 out of 14 applications that can potentially benefit from the possibility to be run on another module[1].

Table 1 shows the list of partners and applications and the modules, on which each application could potentially run. Some of the scenarios were available at the beginning of the project and some of them are ongoing work. It is not reflected in the table, but some of these applications cooperate in the fork of a workflow, such as the Imager and Correlator from ASTROM, and some of them, as the case for GROMACS, are modular applications that could use more than one module at the same time to take benefit of the specific HW characteristics.

We estimate that no more than 40% of the applications that run on the CM can also run on the ESB or applications that run on the DAM can also run on the ESB. Therefore, we conducted the following experiments on our SLURM's simulator. We wanted to evaluate the benefit of having jobs that allow to be run on a second module. With this in mind, we use the mechanism of partitions on SLURM's simulator to create the modular architecture and therefore, simulate different modules.

We have created a synthetic scenario representing three modules with different HW characteristics in terms of number of nodes and node size. The current Tk5.5 includes an execution time model that will be incorporated in the simulator to include modules with heterogeneous characteristics. MOD-S is a small one, MOD-M medium and MOD-XL is the big one.

- Partition MOD-S with 25 nodes, Sockets=2 CoresPerSocket=20 ThreadsPerCore=1.
- Partition MOD-M with 50 nodes, Sockets=2 CoresPerSocket=12 ThreadsPerCore=1.

[1] partitions in these experiments

Partner	Application	CM	ESB	DAM
NMBU	NEST	X		
	Arbor/HybridLFP		X	
	Elephant			X
NCSA	GROMACS	X	X	
ASTRON	Imager			X
	Correlator		X	X
KU Leuven	xPic-Fields	X		
	xPic-Particles		X	
	DLMOS			X
UoI	NextDBSCAN	X	X	X
	PiSVM	X	X	X
	Deep-Learning		X	
CERN	Reconstruction		X	X
	Classification	X	X	X

Table 1: WP1 applications and modules on which they could potentially run.

- Partition MOD-XL with 150 nodes, Sockets=2 CoresPerSocket=28 ThreadsPerCore=1.

To evaluate the impact of having more or fewer jobs being able to run on more than one module, we have generated several workload trace files where the percentage of jobs allowed to run on more than one module varies from 0 (baseline) up to 40%. In this deliverable and based on WP1 applications, we assumed jobs submitted to MOD-XL are the ones that can be also executed in some cases either in MOD-M or MOD-S.

Table 2 summarises the input values used in Lublin model to create our workload trace as well as some information that reflects the characteristics of traces such as the maximum duration. For the evaluation presented in this deliverable, traces with 3.000 jobs have been used. We will create large traces with 10.000 jobs for further evaluation. Characteristics described in Table 2 are common to all the traces and what is changed between traces is the percentage of jobs submitted to more than one module.

Unified trace size	3,000		
Modules (MOD)	S	M	XL
Trace size (per module)	1,000		
Minimum #nodes	1		
Maximum #nodes	25	50	150
Minimum job duration (s)	1		
Maximum job duration (s)	8,055		
Minimum arrival time (s)	3		
Maximum arrival time (s)	1,990	1,986	1,985

Table 2: Workload trace characteristics.

Since the computational resources of the partitions are equal, we are not considering the differences on runtime that will exist on the production system (further details in Tk5.5 of WP5).

Then, we sweep from a range of 0% (baseline), up to 40% of jobs that when submitted, the related value (in %) of jobs requesting either to be run on MOD-S or the MOD-M that are also allowed to run on the MOD-XL. Such a sweep made it possible to verify the benefit of having a second partition option for smaller partitions that saturate quicker under similar loads.

$$WaitTime(J_i) = StartTime(J_i) - SubmitTime(J_i) \tag{2.1}$$

$$Slowdown(J_i) = \frac{WaitTime(J_i) + ExecutionTime(J_i)}{ExecutionTime(J_i)} \tag{2.2}$$

Figures 1 and 2 show the impact on wait time (per partition and global), see equation 2.1. Figures 3 and 4 show the the impact on slowdown (per partition and global), see equation 2.2 when varying the percentage of modular flexibility in job requirements. It is important to remark the percentage shown on x-axis is the percentage of jobs requesting MOD-S or MOD-M that will additionally include MOD-XL in their requirements, so the total percentage with respect the whole workload is slightly lower than these values. For instance, for the 10% scenario the total percentage is the 6% and for the 40% scenario percentage over the total is 26% . We present both metrics since they provide different kind of information. A wait time of 1 hour is acceptable for one job requesting 10 hours, however, it is not for a job requesting 10 min. This additional perception of the trade-off between execution time and wait time is provided by the slowdown.

The results present the impact on system metrics when the percentage of jobs that can be executed on either the MOD-S or the MOD-M can also be run on the MOD-XL increases. The base case is the workload where each job can be executed on one strictly specified partition (either only MOD-S or only MOD-M). Finally, we present the sweep across the x-axis, which is the percentage of jobs requesting either MOD-S or MOD-M that also request MOD-XL. We also included the results for jobs that can run only on the MOD-XL so we can verify how such jobs are affected.

Figures 1 and 3 present the same sweep across the x-axis, which is also the percentage of jobs requesting either MOD-S or MOD-M also request MOD-XL. As more jobs are allowed to run on the MOD-XL module, less jobs are executed in MOD-S or MOD-M, translating into the reduction of the average wait time per module as seen in Figure 1. Figures 2 and 4 present the results for the global system metrics, which shows the impact of the global wait time and the global slowdown time, respectively, for the modular architecture as well as for the system as a whole.

Finally, Figure 5 shows how the number of jobs run on the much bigger and more available Module MOD-XL increases as the sweep on values from 0 to 40% is performed. Similarly, it also shows how the number of jobs run on the smaller modules MOD-M and MOD-S decreases. We conclude that there is a clear benefit for allowing more jobs to run on different modules, and that if such flexibility exists, it should be allowed so the modules can benefit from decreases on important system metrics for availability and usability of the cluster. Even though results only show the potential of the proposal, it is relevant that no significant percentage of flexible jobs is needed to report benefits. This conclusion is crucial since in these results the impact of the heterogeneity is not yet considered, resulting in the same execution time in the three modules.

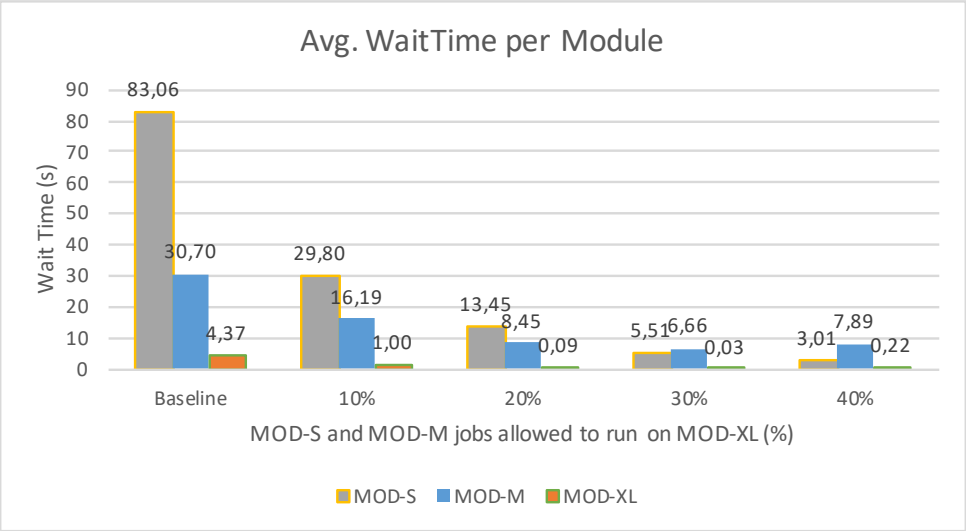


Figure 1: Impact on the average wait time when requesting MOD-S or MOD-M module's resources as a first choice and MOD-XL module's resources as a second choice.

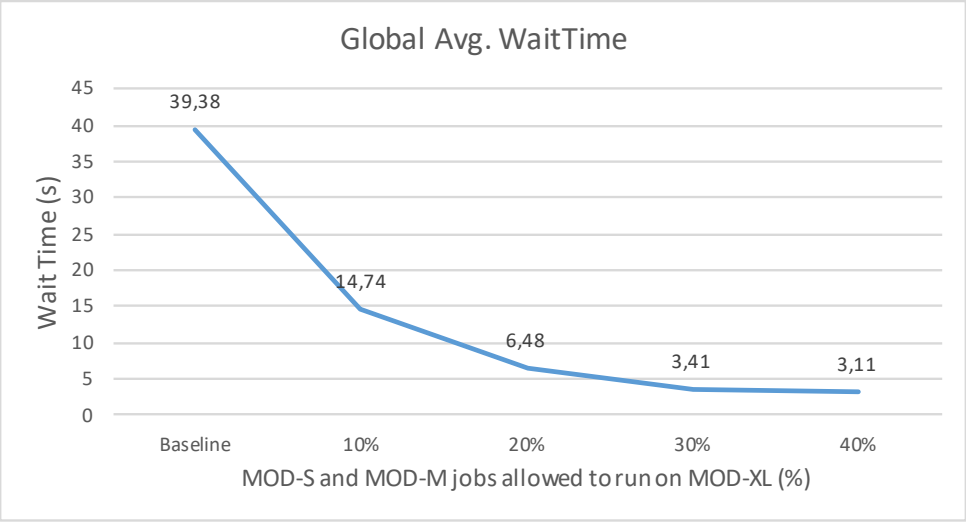


Figure 2: Impact on the global wait time when requesting MOD-S or MOD-M module's resources as a first choice and MOD-XL module's resources as a second choice.

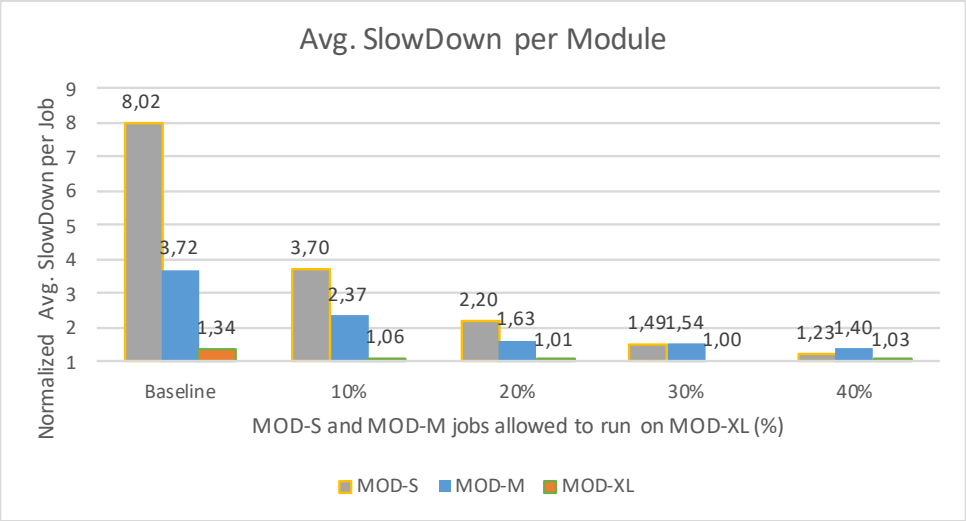


Figure 3: Impact on the average slow down when requesting MOD-S or MOD-M module's resources as a first choice and MOD-XL module's resources as a second choice.

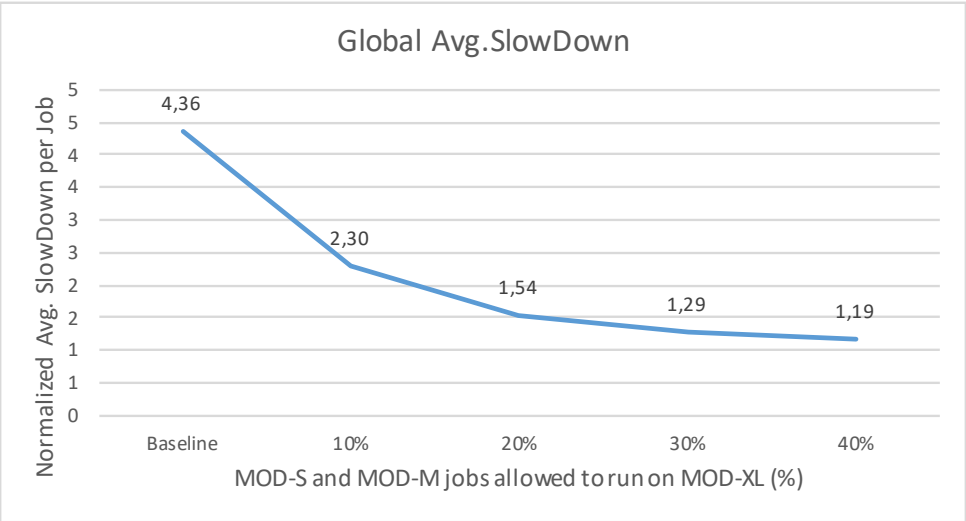


Figure 4: Impact on the global slow down when requesting MOD-S or MOD-M module's resources as a first choice and MOD-XL module's resources as a second choice.

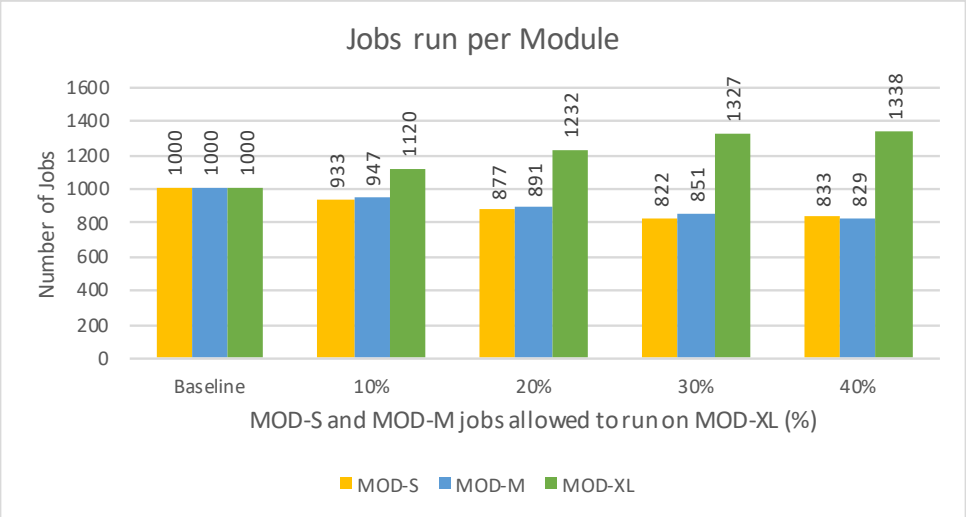


Figure 5: Number of jobs run on each module when requesting MOD-S or MOD-M module's resources as a first choice and MOD-XL module's resources as a second choice.

Two interesting characteristics can be also observed from the experiments done: First adding module flexibility have not impacted negatively in performance metrics, and second, when looking at maximum values, we have observed maximum values for wait time and slowdown have been significantly reduced, improving the worst case results for the system. For instance, the maximum wait time computed for the baseline scenario is 488 seconds whereas in the 10% scenario it is 272 seconds.

2.3 Conclusions

The work done in T2.2 covers the first requirement for the modular system, supporting jobs to run on different modules. We started with this scenario, since it will be the first new feature ready to be evaluated. Based on already existing workload models we have created a methodology to create workload traces for modular systems where a given percentage of jobs are flexible in terms of module requirements. These workloads have been configured guided by WP1 applications. Preliminar results evaluating the impact of this characteristic on system performance shows a high potential for improvement. With less than a 10% of jobs with modular flexibility, the average wait time was reduced from 39 seconds to 14 seconds. Even though the execution time impact has not been considered in these experiments, we expect that there will still be some benefit when taking it into account. We have observed the Lublin model generates workload traces with periods of low activity based on daily periods, however, this characteristic is something to be reconsidered in actual systems and could result in a variation in the model to be adapted to generate higher loads. Next steps will include the generation of workload traces considering modular jobs and modular jobs with dependencies as well as including an execution time model in the SLURM simulator. We will coordinate with WP1 for workload characteristics and with WP5 to select the most beneficial characteristic to be implemented first based on their plans.

3 Performance Modelling And Extrapolation: Initial application analysis and performance models

The goal of the performance modelling and extrapolation is twofold. Firstly, to analyse the performance and study the scalability of the applications; secondly, to model with Dimemas the DEEP-EST architecture to evaluate efficiency, predict executions at larger scale, and to study the impact of different mappings. The analysis is performed using the tools developed at BSC that were previously used in DEEP and DEEP-ER projects.

3.1 BSC modelling tools

The performance modelling approach used in DEEP-EST is based on the BSC efficiency model. This is a multiplicative model that characterizes the performance of the applications using fundamental factors and that can be applied to understand and predict application scalability. The factors are measured as values between 0 and 1, the higher the better. An efficiency of 0.8 indicates that the use of the resources was 80%, so 20% of the corresponding resources allocated to the execution are unused. This value can be considered a boundary between good and bad performance.

The global efficiency of a parallel application can be decomposed into two main factors:

- Parallel efficiency represents the percentage of time spent on the computation (useful work) with respect to the total execution time.
- Computation scalability reports the scaling of the computation itself. For instance, an application that suffers code replication when scaling will report degradation in the computation scalability.

Similarly, the parallel efficiency can be decomposed into three main factors:

- Load balance (LB) measures the efficiency losses due to differences on the computing time between processes. If some processes take more time in the computation, the other processes would have to wait for them in the synchronization of the MPI calls for instance.
- Transfer (Trf) measures the reductions on the efficiency due to the need to transfer data between processes. If the application is dominated by communications, transfer efficiency would be low.
- Serialisation (μLB) measures the efficiency losses due to dependencies during the execution. This factor also reflects load imbalances that can be compensated along time. If on the even iterations half of the processes do more work than the other half but on the odd iterations the behaviour is just the opposite, load balance would report a good efficiency while serialisation would reflect the loss of efficiency.

The load balance can be directly measured from an instrumented execution. In order to compute the transfer and serialisation factors, it is necessary to isolate the application execution

from the network characteristics. This can be done using the Dimemas simulator to predict the behaviour running on an instantaneous network.

The BSC efficiency model characterizes the execution of a given application on a given platform (the platform component not only corresponds to the hardware, but also to the software stack, mapping of the processes, etc.). Additionally, if the application is very sensitive to the input data, different models may be generated from various inputs. As the model is based on a trace of timestamped calls to the parallel runtime and the ratio between the computing time and the communication/synchronization time per process, relevant modifications in the code structure or in the MPI calls used, have a high impact on the resulting model and may render a model generated before these modifications useless.

Once the model is computed, it can be used to extrapolate the behaviour on a larger scale as it was demonstrated on previous DEEP projects. The input for the extrapolation is a set of traces for at least four or five executions increasing in scale. The model set-up is adjusted to analyse the collected traces by looking for behavioural trends, while increasing the scale. By default, the extrapolations are done based on Amdahl's law. The Amdahl model would reflect the contention/serialisation on a given resource.

$$Amdahl_{fit} = \frac{metric_0}{f_{metric} + (1 - f_{metric}) * P} \tag{3.1}$$

Where metric is the efficiency that we are modelling and P the number of processors for a given run. If the collected traces allow identifying an underlying physical phenomenon that follows a more specific law for a given efficiency factor, the parameter P in the previous formula can be substituted by the corresponding/approximate function based on the number of processes.

If the goal is to predict the behaviour at several orders of magnitude bigger than the instrumented executions, it is important to validate the model with non-instrumented runs with increasing scale. Validating the scale half way from the largest run used for modelling and the target prediction seems a safe and reasonable approach.

Even though the model is developed for a given platform, the BSC Dimemas simulator can be used to estimate the execution on a different platform. Before extrapolating the simulated results, it may be recomendable to check whether we have to readjust the model looking at the traces generated from the simulations. With this approach, we can predict the execution of a given application on architectures that are not available, or we can study the sensitivity of a given code to the key factors that characterize the Dimemas network: bandwidth, communication latency, the number of messages coming in and out for a given node (input/output links), and the number of messages that can use concurrently the network (number of buses). The main target for Dimemas is modelling the network, but can also be used to simulate the porting to a processor twice as fast, or the impact of improving the computations from specific parts of the code. It is also possible to adapt the mapping of the processes to the available resources; but there is neither modelling of the memory hierarchy, nor multi-core sharing effects.

Using Dimemas simulations to feed the BSC efficiency model, we can build new models of an application for desired architectures and configurations. The results of the models generated are providing us with very interesting insights about constrains, requirements, and potential of a given application.

3.2 Modelling the DEEP-EST applications

The application selected as the first target for the performance modelling study is GROMACS. GROMACS is a well known parallel code for molecular dynamics that is implemented with two types of processes (Particle-Mesh Ewald (PME) and Particle-Particle (PP)), instead of the typical Single Program Multiple Data (SPMD) approach, which characterizes most of the HPC codes. PMEs and PPs have a very different granularity and behaviour, but are very tightly coupled and synchronised during execution.

The non-SPMDiness of GROMACS makes it very interesting to exploit the DEEP-EST MSA, despite the fact that the code has been developed and optimized to run on homogeneous HPC systems, where both types of processes are typically mapped sharing the node resources.

To simplify the extrapolation of the model in the DEEP-EST architecture, the only restriction we enforce for the study of the applications is to maintain the same ratio of processes across the modules when scaling the code. For instance, the ratio between processes that would run on the ESB module vs. Cluster module is maintained in all the executions.

As the DEEP-EST architecture is not yet available, the second requirement to start this study was to select a platform to run the application. The goal was to do a first analysis and modelling on a system that would enable us to validate the results. The selected platform is the JURECA Cluster Module at Jülich [22]. This system consists of 1,872 compute nodes, each equipped with two 2x12-core Intel Xeon E5-2650 v3 Haswell processors clocked at 2.5 GHz with a Mellanox EDR Infiniband interconnect; where we can run executions with a maximum of 256 nodes (6,144 cores). Considering this boundary, we set the limit for the instrumentation to 16 nodes to allow a distance of a little bit more than one order of magnitude between the largest instrumented run used to build the model and the largest non-instrumented run used for the validation.

To be able to model the DEEP-EST architecture with Dimemas we need to have estimators that characterize the network at different levels, as well as the target core performance compared to the performance of the core in the platform used to collect the traces. With Dimemas we will be able to investigate the potential behaviour running on the DEEP-EST architecture, before it can be used and additionally at larger scale than with the planned prototype. Nevertheless, the goal is to validate the predictions once the platform becomes available to run the applications.

Defining the network before the prototype is available may introduce errors that may lead to wrong conclusions. To avoid this risk, we decided to drive the study based on the results of the application efficiency analysis, building models with Dimemas in order to explore the potential benefits of some scenarios selected. The study will provide input on how to improve the scaling and what it is the application sensitivity to the network characteristics.

This approach increases the number of models that can be generated for any given application, but we decided it is more interesting to fully explore the potential with different alternatives for few key applications than to generate one model for all the codes involved in the project.

3.3 GROMACS Analysis

The first step is to analyse the behaviour of the application in the target platform. WP1 collected the traces for the study selecting an appropriate set-up and configuration to run from 1 to 16 nodes.

Figure 6 displays a timeline of the MPI activity for the execution with 16 nodes (384 cores). We can identify 3 internal subiterations with some differences between them. This is a very small interval inside one of the main iterations and there is a high variability between subiterations. We selected one region with less perturbations to illustrate the application behaviour.

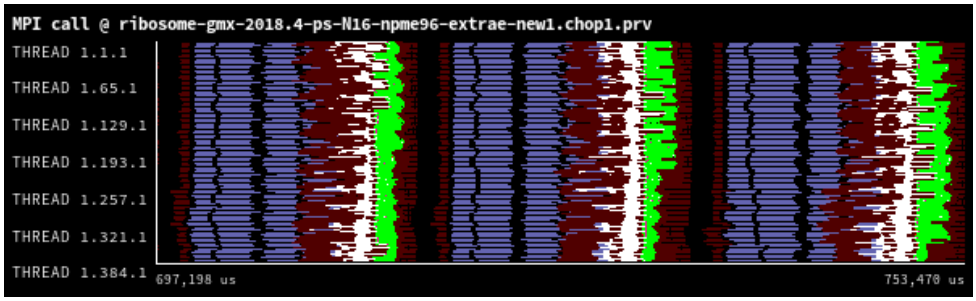


Figure 6: MPI call timeline for the execution with 16 nodes and 384 processes.

Zooming into a small subset of the processes, we can see more details of the subiteration structure. The MPI_Alltoall (purple) is executed to communicate between the PMEs while the PPs are computing. As on each subiteration PME and PP processes are synchronised (through point to point calls) when the PPs end their main computation they wait for their PME in the MPI_Recv (white). Then the subiteration ends the communication phase and a new subiteration is started.

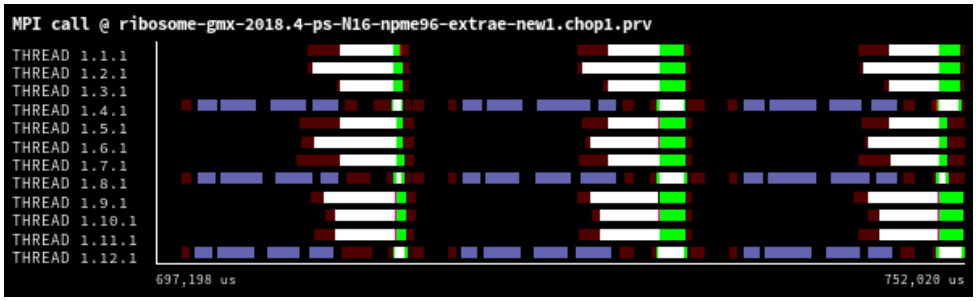


Figure 7: Zoom in the previous timeline to focus on 3 subsets of processes, each of them with 4 processes (3 PPs and 1 PME).

Table 3 report the values measured from the traces using one of the last iterations close to the end of a 2,000 iterations run as input. The same iteration was used on all the executions, except when significant noise was identified.

ParaStation MPI	24 cores	48 cores	96 cores	192 cores	384 cores
Global efficicency	90.79%	86.71%	80.51%	83.02%	61.81%
Parallel efficiency	90.79%	88.03%	84.45%	81.64%	61.83%
Load Balance	92.62%	91.33%	87.57%	86.95%	68.01%
Communication efficiency	98.02%	96.39%	96.43%	93.89%	90.93%
Serialisation efficicency	99.37%	98.79%	98.25%	97.61%	99.33%
Transfer efficiency	98.64%	97.57%	98.15%	96.19%	91.55%
Computation scalability	100.00%	98.50%	95.34%	101.68%	99.96%
IPC scalability	100.00%	102.54%	106.06%	108.62%	111.54%
Instructions scalability	100.00%	96.57%	89.66%	93.39%	89.97%

Table 3: GROMACS efficiency metrics expressed as percentage.

3.4 GROMACS Extrapolation

As a previous step for the extrapolation, we investigated in detail the scaling behaviour in the traces with ParaStation MPI. The objective was to confirm the trends identified on the efficiencies analysis by using traces. The clearest example is the transfer factor. By analysing the traces we can find, for instance, if the number of MPI calls increases linearly or logarithmically with respect to the number of processes. The objective is to select the model (Amdahl, linear, constant...) to fit each of the components of the efficiency model.

Both computation scaling and serialization efficiency have a quite uniform behaviour on the different runs, so the modelling as a constant can be considered a good first estimation. Nevertheless, applying Amdahl to very similar values should also approximate as a constant value, so we can use the default Amdahl fit.

The increase of the MPI calls and volume of data transferred through MPI is directly driven by the domain decomposition. Consequently, we can assume it will follow Amdahl's law linearly with the number of processes.

The modelling for the load balance was the most difficult aspect, as it depends not only on the domain decomposition, but correlates to the balance between PMEs and PPs and has many fluctuations (Table 3). For this component, we considered the Amdahl approximation.

As we have assumed that computations would not degrade with the scale, we can simplify the model and base it on the parallel efficiency. Figure 8 plots the parallel efficiency and its components as measured in the traces, as well as the prediction based on the adjustment described in the previous paragraphs. We plot the values up to 6,144 cores as it would be the target for the largest execution used in the validation.

The extrapolation predicts a very poor parallel efficiency with 6,144 cores (6.3%) caused mainly by load balance (15%), and also reports a problem with transfer (44%).

We can get some insight on the prediction for the different factors that limit the scaling. The parallel efficiency is highly correlated with the load balance, which is the most relevant factor on the analysed scale. After discussions between the project partners, we identified that the poor load balance is due to the restriction of a uniform scaling. In order to efficiently run GROMACS

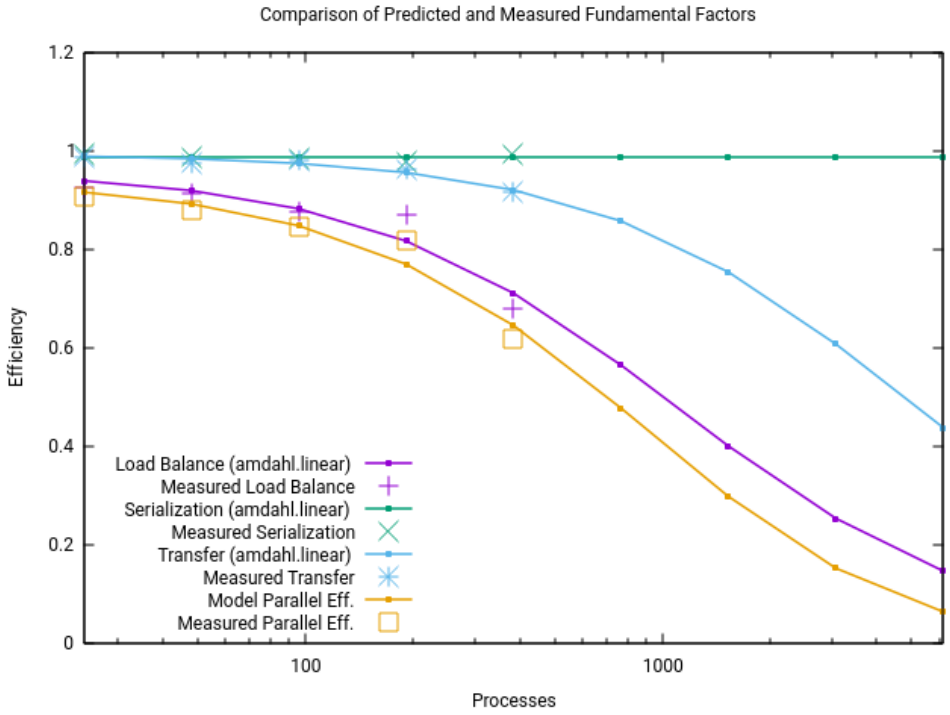


Figure 8: Efficiencies measured from the instrumented run up to 384 cores and prediction up to 6,144 cores.

for each configuration the balance between PMEs and PPs needs to be adjusted, instead of forcing to maintain the initial ratio of 3 PP for each PME.

Transfer efficiency has good values in the range of processes we analysed in the traces (lower value is 91.55%). Nevertheless, the prediction forecasts poor transfer efficiency already with 64 nodes (1,536 cores). The loss of transfer efficiency from the measured 16 nodes to the predicted 256 nodes is similar to the reduction suffered in the load balance (48% vs. 53%).

To understand the prediction of the transfer efficiency we have to consider that using strong scaling the ratio between computations and communications is reduced with the scale. We received confirmation from the GROMACS developers that due to the size of the domain the input used would typically run in a range of 1 to 64 nodes.

We can also verify that the model predicted a constant behaviour for the serialization, even the fit was done using Amdahl. Serialization is a factor that describes the internal unbalances and dependencies of the code. Having a constant behaviour is a good property of GROMACS.

We used non-instrumented runs from 1 node to 256 nodes to validate the predictions of the model. Figures 9 and 10 show the execution time and the efficiency comparing the predicted values with the timing provided by GROMACS on the non-instrumented runs. The efficiency is relative to the execution with the smallest run (one node).

When comparing the prediction of the execution time to the time measured, the results seem accurate up to 64 nodes. With 128 nodes, the non-instrumented run abruptly increases the elapsed time close to the 16 nodes run. The model predicts that the application will start

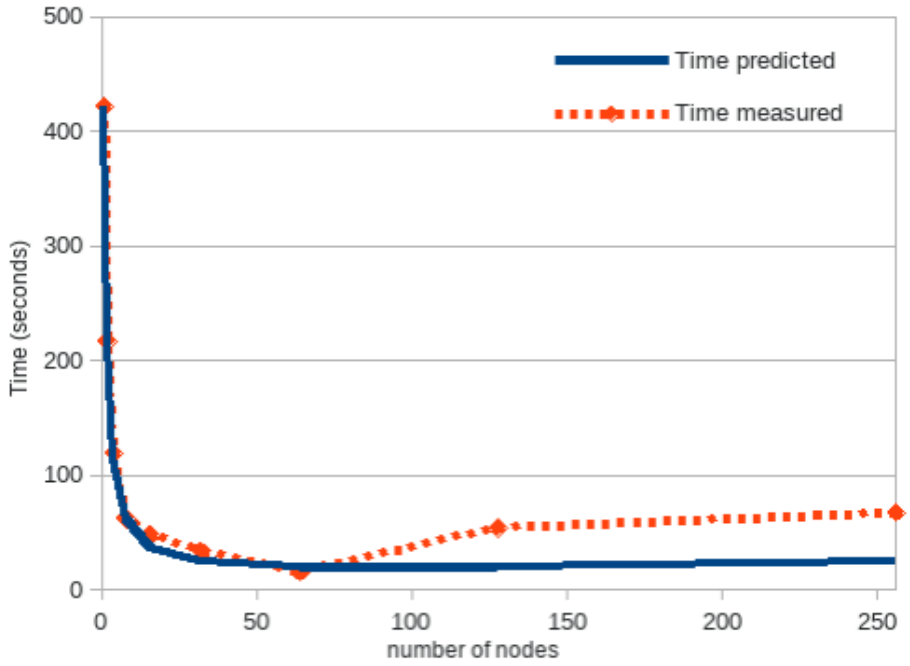


Figure 9: Validation with respect to the execution time comparing the predicted values from modelling 1 iteration with the timing provided by GROMACS on the non-instrumented runs of 2,000 iterations.

degrading at 256 nodes, and this degradation to be softer. To measure the differences, it is better to use as reference the efficiency reported in Figure 10.

Our model cannot predict the noisy behaviour we see in the efficiency scaling, but rather the model will provide a softer estimate for the efficiencies. From the efficiency plot in Figure 10, we can identify that the predicted efficiency up to 256 nodes is a reasonable adjustment for the measures without instrumentation as it fits around the measured values. The biggest difference is 0.17 with an average of 0.055 in absolute numbers.

Both Figures (9 and 10) indicate that beyond 128 nodes the model is too optimistic. In fact, we can see that already with 16 nodes the prediction shows an efficiency significantly higher than the measurement. The reason for this divergence may be related to the fact that we identified a noisy behaviour in the traces and we selected one iteration without large system noise. This has a higher impact in the predictions with 128 and 256 nodes, because the system noise becomes more important with the scale.

In conclusion, as the GROMACS developer told us that a reasonable scaling for the input used would be between 1 and 64 nodes, we consider the model successfully passed the validation, and the traces can be used to simulate new scenarios with Dimemas.

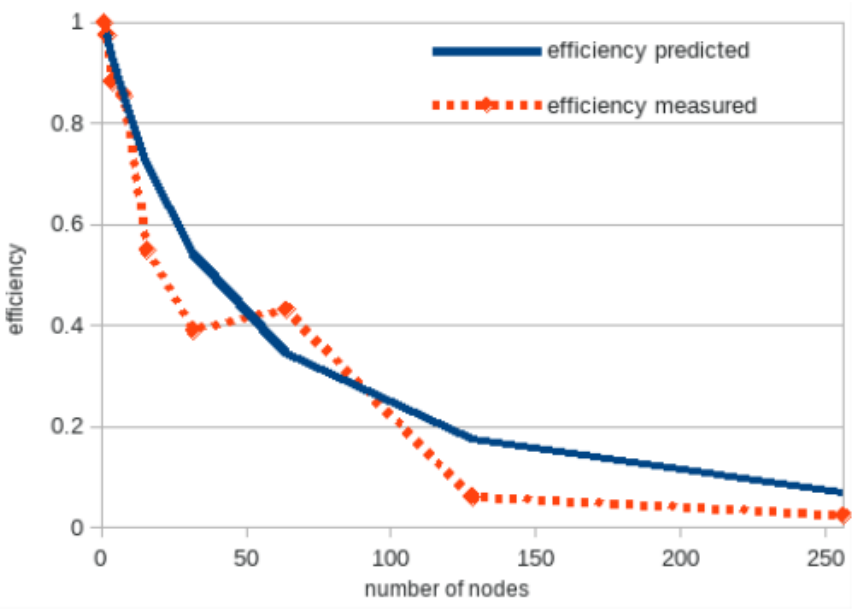


Figure 10: Validation with respect to the execution efficiency comparing the predicted values from modelling 1 iteration with the timing provided by GROMACS on the non-instrumented runs of 2,000 iterations.

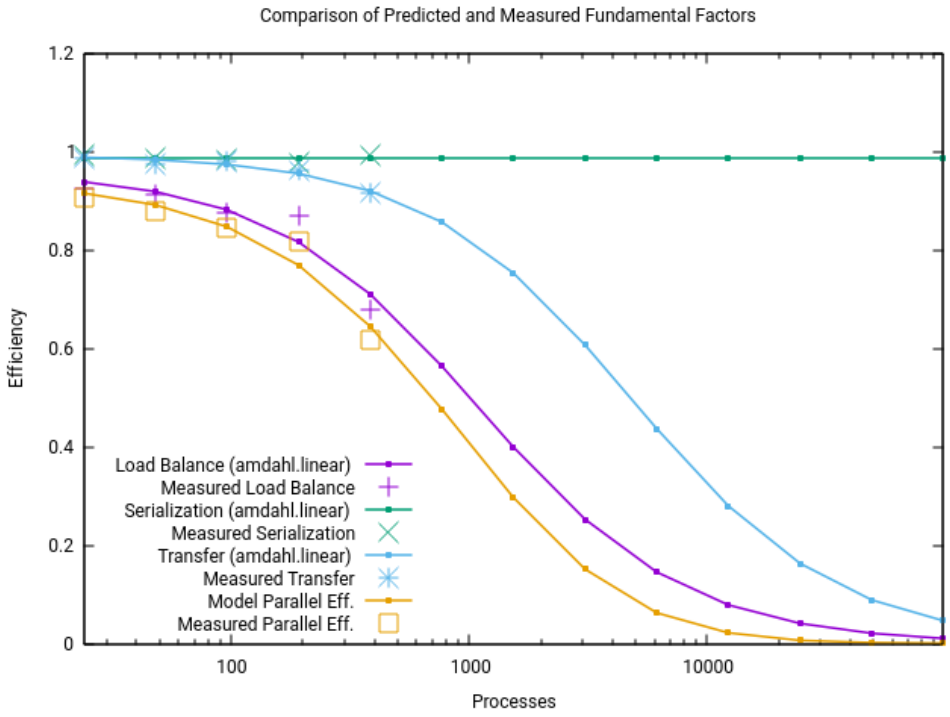


Figure 11: *Reference* model up to 4,096 nodes (98,304 cores).

3.5 Modelling GROMACS with Dimemas

The objective of the studies carried out with Dimemas is to analyse different selected scenarios for the GROMACS application comparing the predicted efficiency and scaling. Before describing the work done we should mention a limitation of Dimemas simulating the MPI_Waitall call used by GROMACS. The current implementation of the simulator does not allow changing the sequence of receptions captured during the execution. This limitation may create some small delays that would not happen on the real execution.

In this section we selected as upper limit for the extrapolation the value of 98,304 cores. This is 256 times the largest instrumented run used as input (384 cores) and 16 times the largest non-instrumented run used for the validation (6,144 cores). As reference, Figure 11 plots the efficiencies of the original run on the selected range of cores.

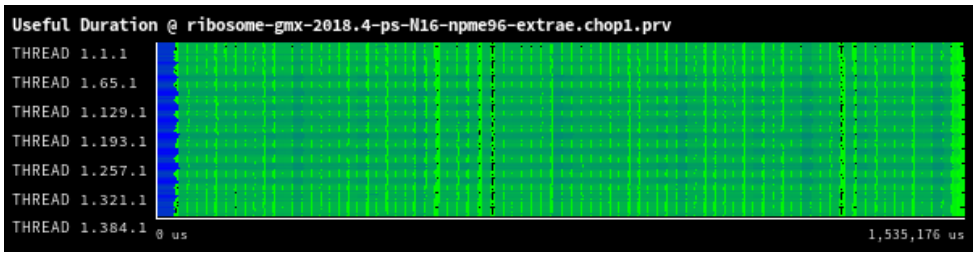
We study three alternatives targeting to improve the measured efficiency. The first one focuses on the communications modelling a faster network, while the latter two focus on the computations, one considering speeding up all computations, and the other concentrated on accelerating only the main computing kernels. Henceforth, we refer to the configurations studied as *FasterNetwork*, *FasterCPU* and *KernelsAccelerated*, respectively.

3.5.1 Evaluating the impact of network settings in GROMACS scalability

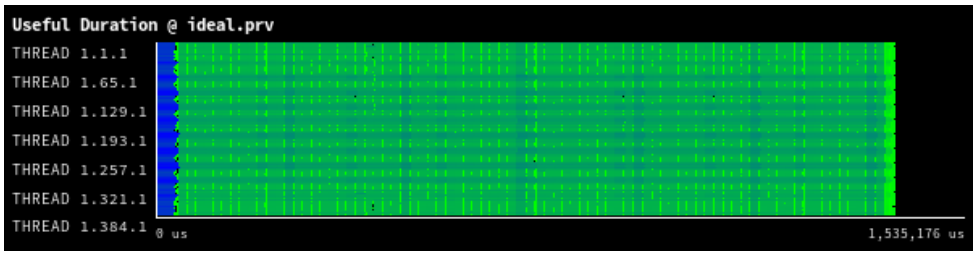
The first selected scenario tries to overcome the limited transfer scaling identified by the efficiencies analysis. The Dimemas instantaneous network simulation identified that the execution would benefit from a faster network, so we started studying the network characteristics required to get a performance closer to the instantaneous communication. We used as reference the trace with 16 nodes (384 cores) that reports the lowest transfer efficiency.

Figure 12 shows how the full execution timeline for the real run (top), shortens when simulating a network with instantaneous communications (middle), and also when simulating a network setting the bandwidth to 10GB/s and the latency to 3μs (bottom). Since the latter configuration is already quite close to the performance achieved for the simulation of an instantaneous network, we selected this configuration for the extrapolation analysis. With these parameters, the transfer efficiency was increased from 91.55% for the real run to 97%, and the results of the extrapolation are displayed in Figure 13. The extrapolation predicts a good scaling of the transfer efficiency up to 512 nodes (12,288 cores) with efficiency values above 80%. However, load balance is still the main factor limiting parallel efficiency.

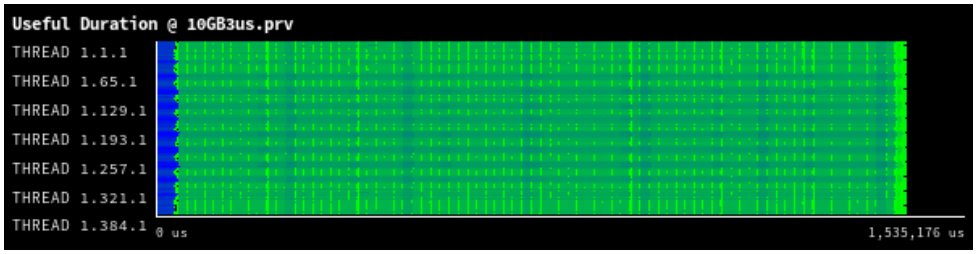
Figure 14 compares the transfer efficiency between the *Reference* and *FasterNetwork* models. Solid lines show efficiencies computed with reference values, and dashed lines show projections with 10GB/s bandwidth. Comparing between solid and dashed lines with the same color, we can observe the impact in efficiency when modeling a faster network. Focusing on the blue lines, up to 4 nodes (96 cores), both configurations report very similar transfer efficiency. This implies that at small scales, the effective bandwidth achieved in the real executions was already as high as 10GB/s, thus the simulation does not report any improvements. While maintaining the same number of processes per node, the runs with 8 and 16 nodes (192 and 384 cores) see their performance improved in the simulation. This fact indicates that the real effective band-



(a) Real execution



(b) Ideal simulation (Bandwidth=∞, Latency=0s)



(c) Simulation with Bandwidth=10GB/s, Latency=3μs

Figure 12: Selection of a faster network configuration

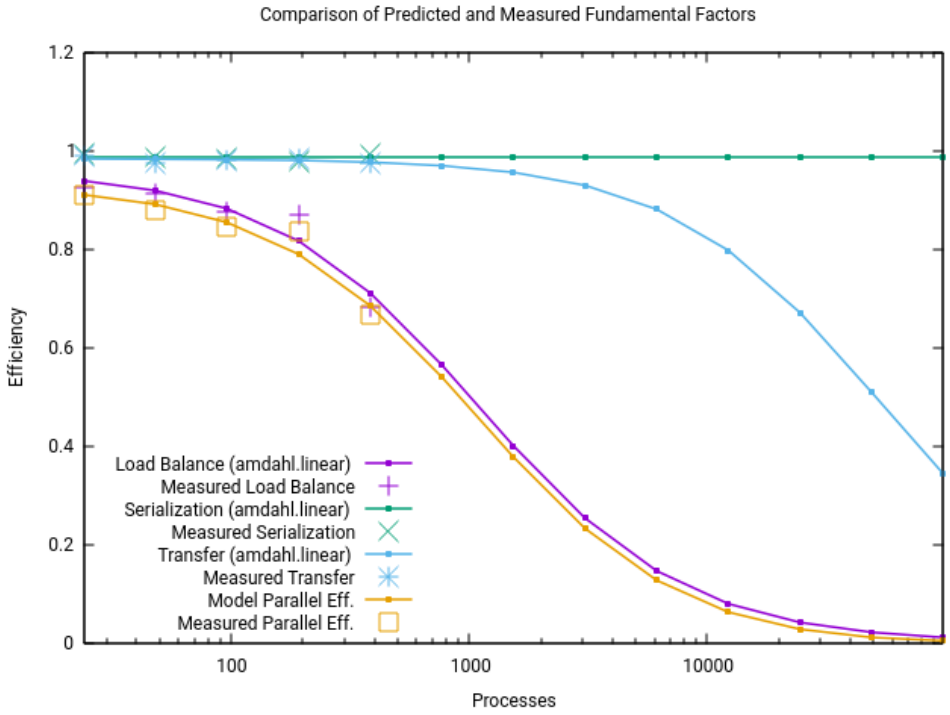


Figure 13: Simulated model (*FasterNetwork*) using a network of 10GB/s and a latency of 3μs.

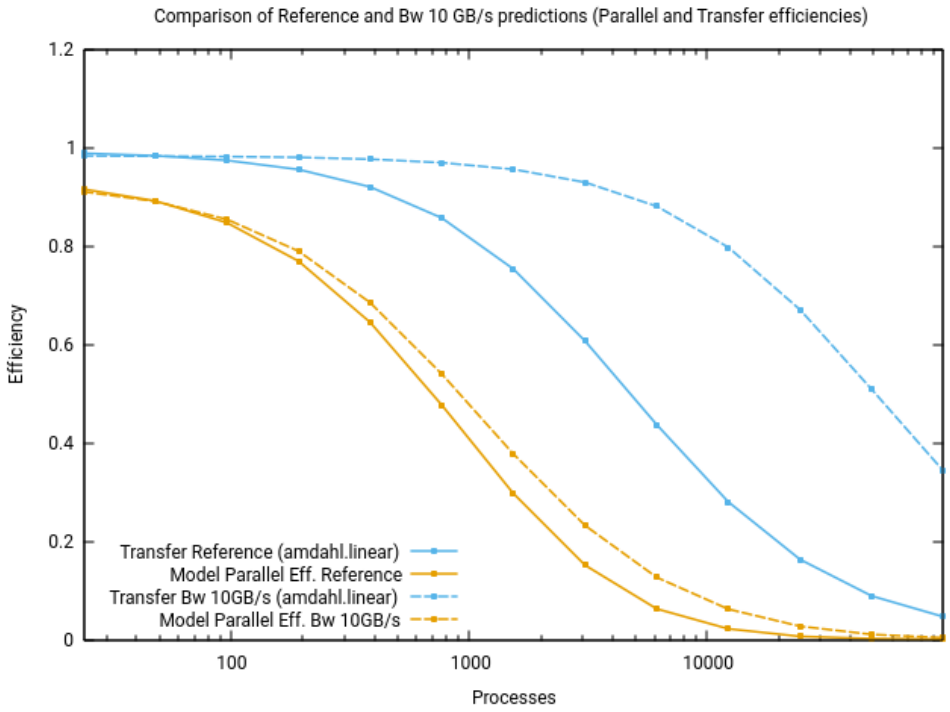


Figure 14: Comparison of transfer and parallel efficiency between *Reference* and *FasterNetwork* models using a network of 10GB/s and a latency of 3μs.

width achieved was lower, which suggests that there may be some global network contention in JURECA that limits the effective bandwidth when higher core counts demand higher network traffic. Focusing on the yellow lines, we can see that the impact of this improvement on the parallel efficiency results in a smaller profit margin due to this is still limited by load balance, as shown previously in Figures 11 and 13.

As the simulation with 10 GB/s network bandwidth fits the behavior of the small scale real executions, we fix this setting as the baseline network configuration for further simulations. Table 4 summarizes the Dimemas network parameters used to simulate the selected settings.

	Communication parameters	
	Intra-node	Inter-node
Latency	3μs	3μs
Bandwidth	10GB/s	10GB/s
Number of Buses	1	0 (unlimited)
Input Links	1	1
Output Links	0	1

Table 4: Description of Dimemas network parameters

3.5.2 Evaluating the impact of the computational load balance between PP and PME processes in GROMACS scalability

The second selected scenario targets to improve the imbalance problem of the executions. This is a quite ambitious target due to the non-SPMDiness of the code as we have two different unbalances: between types of processes and within each type of processes. Our goal is to improve the balance between the two types of processes.

Table 5 characterises the two-level unbalances on the different traces. We report the percentage of time computing per type of process with the minimum, maximum, average and load balance, as well as the ratio between the average columns.

	PME				PP				Ratio
Nodes	min	max	avg	LB	min	max	avg	LB	avg
1	71%	78%	75.56%	96.87%	94%	98%	95.86%	97.82%	1.27%
2	61%	78%	70.25%	90.06%	93%	96%	93.96%	97.87%	1.34%
4	49%	64%	57.77%	90.27%	90%	96%	93.34%	97.23%	1.62%
8	48%	59%	55.29%	93.71%	88%	94%	90.43%	96.20%	1.64%
16	35%	41%	38.32%	93.46%	62%	90%	69.67%	77.41%	1.82%
avg			59.44%				88.65%		1.49%

Table 5: GROMACS load balance characterization.

We can see that there is imbalance within each type of process being worse for the PME processes (except with 16 nodes). Nevertheless, as our target is the balance between the PMEs and the PPs we should focus on the last column of this table. The values indicate that the ratio between the two types of processes becomes worse with the scale.

The objective is to speedup the PP processes so their percentage of time computing is closer to the PME processes. That should reduce the time the PMEs are waiting inside MPI for the PPs. If we select the ratio with 1 or 2 nodes, the impact on the 16 nodes will be very limited. On the other side, if we select the ratio with 16 nodes we will increase the unbalance for the lower core counts, for that reason we selected the ratio from the average of the different traces (1.49).

We approached this target with two different alternatives. The first one corresponds to the scenario of executing the PP processes on faster CPUs. This means that all the computations executed by the PP processes are recomputed applying the selected ratio. Figure 15 plots the efficiency reported by the simulated model *FasterCPU*.

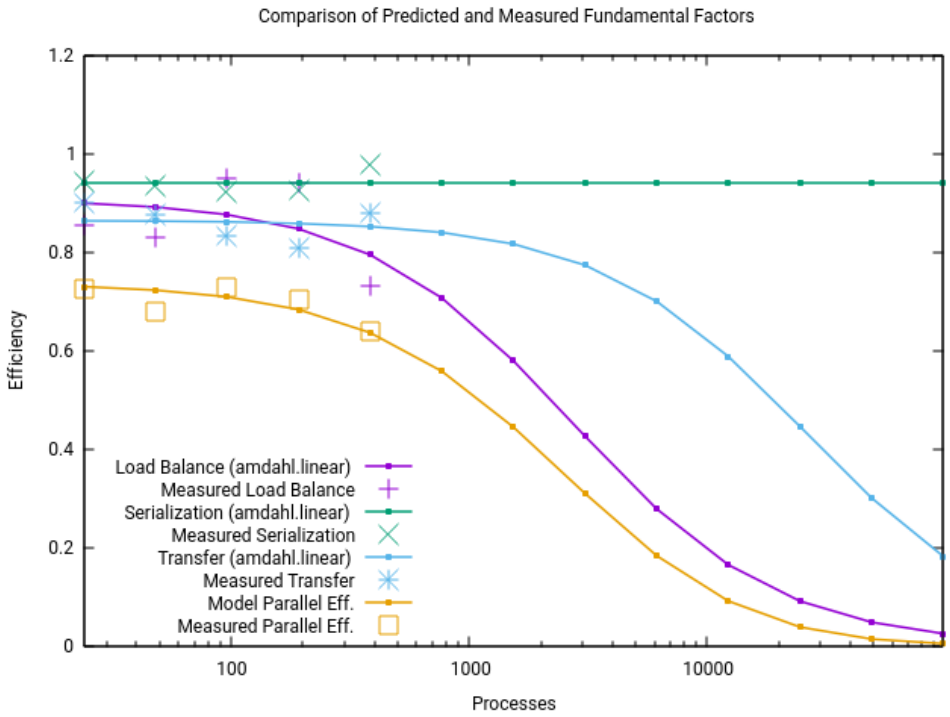


Figure 15: Simulated model (*FasterCPU*) using a network of 10GB/s and a latency of 3μs, and a CPU 1.49 times faster for the PP processes.

From this plot we can extract two insights:

- We have been able to improve the balance, achieving better results with 4 and 8 nodes (the ones closer to the selected ratio). In turn, the load balance efficiency is reduced for 1 and 2 nodes as expected, because the used ratio creates unbalance as waiting time on the PP processes. Nevertheless, this approach significantly improves the predictions, but load balance is still the factor that dominates the parallel efficiency.
- Improving the balance seems to increase the network requirements likely due to a higher injection rate, as the transfer efficiency is lower than reported in Figure 13. Even with just one node, the transfer efficiency goes down from 98.8% to 90%. Now the transfer efficiency reports good values up to 64 nodes instead of 512.

This reduction of the network efficiency drives us to consider a different approach where the improvement is focused on the PP main computation, and the rest of the computations keep their original duration. This scenario corresponds to a porting with either OpenMP or accelerators where only the main kernels get a time improvement. We used the same ratio (1.49), but it is only applied to the main computation of the PP. The results of the simulated model *KernelsAccelerated* are reported in Figure 16.

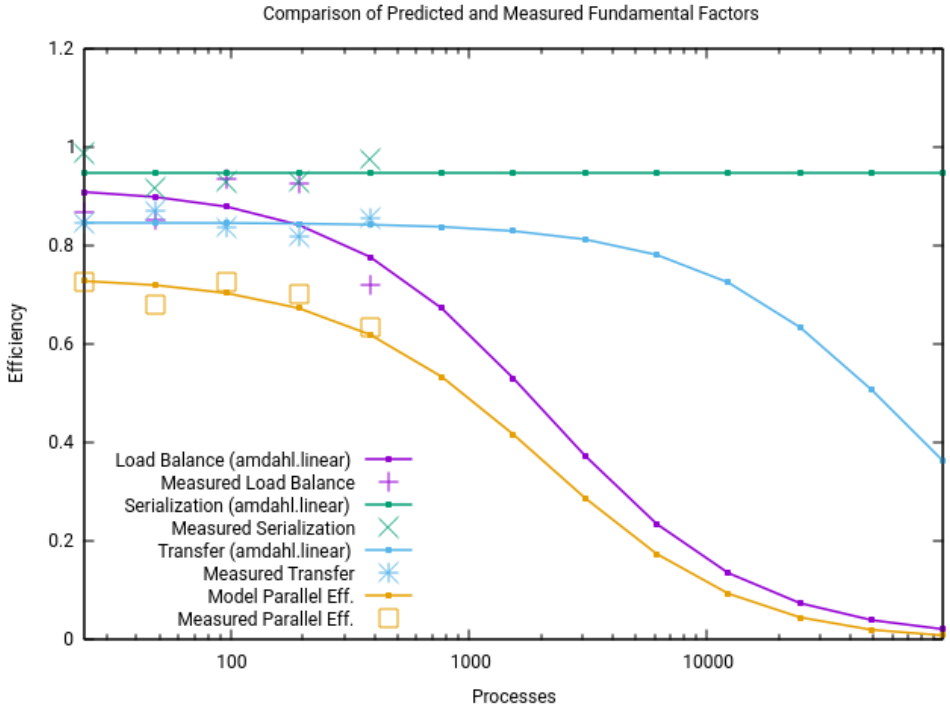


Figure 16: Simulated model (*KernelsAccelerated*) using a network of 10GB/s and a latency of 3μs, and a CPU 1.49 times faster for the PP processes main computation.

If we compare Figure 15 with Figure 16 we can see both report very similar values for the parallel efficiency, but in this last scenario, the load balance is a little bit worse and the transfer efficiency is a little bit better. Now the scaling of the transfer efficiency with the same network characteristics is good up to 128 / 256 nodes.

To better compare the three scenarios described so far, Figure 17 plots the improvement achieved in the parallel efficiency with respect to the *Reference* model from the instrumented runs. This improvement is computed as the ratio dividing the efficiency for a given configuration and core count by the efficiency of the *Reference* model for the same core count. We refer to the different configurations studied as *FasterNetwork* (10GB/s bandwidth, 3μs latency); *FasterCPU* (CPU speed increased by a factor of 1.49); and *KernelsAccelerated* (only main computations accelerated by a factor of 1.49). It is important to recall that both *FasterCPU* and *KernelsAccelerated* are incremental improvements on top of *FasterNetwork*, so all these simulations share the same improved network characteristics.

Focusing on the executions from 1 to 16 nodes (24 to 384 cores), all the simulations provide a very uniform ratio of improvement. As we move to the right of the plot, the higher the scale the

higher becomes the achieved improvement. Most of the benefit comes from the improvement on the network due to the very tight integration and synchronisation.

When comparing the results between *FasterCPU* and *KernelsAccelerated*, both show similar results for small core counts up to 16 nodes (384 cores). Up to 256 nodes (6,144 cores), the application would have a slightly higher benefit with a faster CPU that would speed up all computations (*FasterCPU*), but beyond 512 nodes (12,288 cores), it would benefit the most from accelerating just its main computing kernels (*KernelsAccelerated*). Without further improvements in the network speed, it is better just to accelerate the key kernels, and not the complete application, which would overload the network.

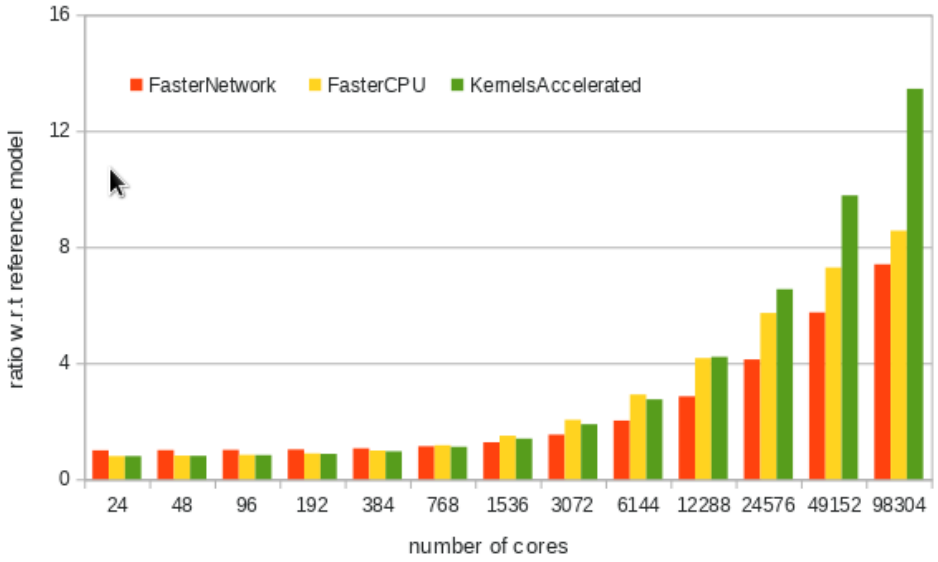


Figure 17: Comparison of the efficiency achieved by the different models obtained with Dimemas. The plotted values are the ratio of improvement with respect to the *Reference* model obtained from the traces.

3.6 Conclusions

The study of GROMACS allowed us to evaluate its efficiency and scaling as well as to explore different potential scenarios to improve the limitations identified by the analysis. These scenarios target different components of the architecture as the network bandwidth, the processor speed, and the use of accelerators.

The non-SPMD structure of GROMACS may seem the most attractive for a heterogeneous platform but maybe it is also one of the most ambitious targets as the application is tightly coupled and optimised to run on homogeneous platforms.

As the most frequent usage of GROMACS is strong scaling, we evaluated the behaviour in that mode. Strong scaling limits the range of scaling and instead over dimensioning a test case to run it with few nodes to be able to scale it to thousands, we decided to evaluate a real input to fit in the scale of one to sixty four nodes. We think this is a good approach to target the

application analysis, as the use of a very large input case will produce higher efficiencies with lower core counts due to a very good computation-communication ratio. Furthermore, running a huge problem on only a few cores would require excessive time and resources, and that limits the applicability.

The goal of the requirement we made to maintain a sequence of scaling was to minimise the variability between executions. However in the case of GROMACS, a fixed ratio between PME and PP processes generates more unbalance and with more variability than the configurations typically tuned by the users. The tools do not have any constrains that impose this limitation, so for future studies emphasis will be placed on minimising variability instead of maintaining a sequence of scaling.

The noisy behaviour of GROMACS load balance that is also reflected in the global efficiency makes more difficult to generate accurate estimators in the extrapolation. This fact as well as the noisy behaviour identified in the traces can explain the deviations we saw on the validation of the model. Potential network contention in JURECA would also explain the divergence with 128 and 256 nodes that can be validated collecting traces for these core counts.

4 Energy Modelling

The following section will describe the advances done in the task regarding the calculation of the energy, power, and time models developed during the project.

The main goal of the energy model is to minimize the amount of energy that the HPC system consumes when executing an application. The power consumed in a processor is proportional to the square of the voltage and the frequency of the processor. Since the energy is the product of power and time, lowering the frequency of the processor leads to a minimization of the average power consumption. However, in some cases lowering the frequency will result in a longer runtime of the application, which in turn results in an increase of the energy consumed. Consequently, it is necessary to find the right frequencies that optimize the energy consumption of an application (see Figure 18).

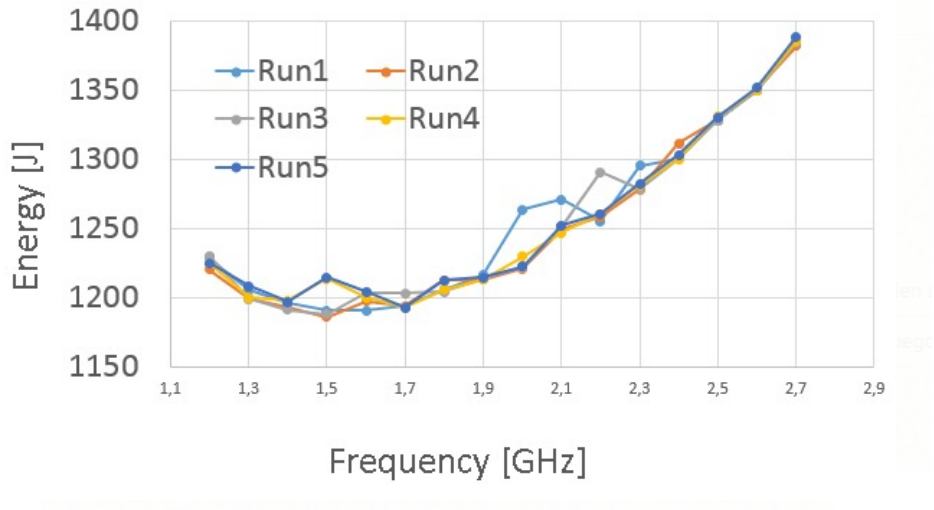


Figure 18: Typical energy curves of an application depending on the clock frequency.

4.1 Data acquisition for the model

For the calculation of energy projections it is necessary to obtain characteristic values of applications and to calculate the formula that best adjusts the energy used by the applications according to these characteristic values. For this purpose, a micro-benchmark has been implemented that extracts the value of certain parameters; afterwards we study the relationship between the energy used and the value of the parameters.

4.1.1 Micro-benchmark

For measurements, we used a modified version of the APEX-MAP benchmark that generates artificial calculations and memory accesses. The initial idea of the Apex project [8, 9, 10] is the assumption that the performance behaviour of any scientific application can be modelled by a set of specific performance factors. Thus, combining these factors, applications that avoid hardware specific models can be designed to simulate typical application performance. Taking into account that the two most dominant performance factors are memory accesses and computational intensity, this benchmark simulates typical memory access patterns of scientific applications. The final goal of this benchmark is to simulate compute and memory bound applications.

The APEX-MAP benchmark implements two different memory access modes. A "strided" mode, where every n^{th} memory cell is accessed, and a "random" mode, where memory locations to be accessed are precomputed with a randomizing function. As a result of precomputation of the location, the randomizing does not interfere with performance measurements, though it is unavoidably included in runtime and power measurements. The randomizing function takes an additional parameter ($0 < K \leq 1$) as the reciprocal exponent to a random number that shapes the spatial distribution of the memory accesses within the virtual address space. With small K ($K \approx 0$), memory locations with small virtual addresses get accessed more often; with large K ($K \approx 1$), the distribution is more uniform. For $K = 1$ every memory block has the same probability of being accessed. Concerning the regularity of the memory access, the original APEX-MAP benchmark focused on random access patterns inside an allocated memory block. The implementation also considers strided access patterns, which are common in many scientific applications. The benchmark written in the style of APEX-MAP has the following 7 parameters:

- M: total size of the allocated memory block data in which data accesses are simulated.
- L: vector length of data access, (sub-blocks of length. $L < M$ starting at ind[i] are accessed in succession), describes the Spatial Locality.
- K: shape parameter of power distribution function ($0 \leq K \leq 1$) determines the random starting addresses.
- ind: array that describes the temporal locality.
- S: stride length.
- C: a parameter used to increase the computational intensity.
- I: length of the index buffer ind.

In the case of strided access only the parameters M, S, and C are relevant.

The modified version of the APEX-MAP includes different test cases whose behaviour simulates different memory accesses and computational intensity patterns depending on the input parameters of the application. It also includes a "sleep" algorithm, which is used as validation, since during the sleep period neither computation operations nor memory accesses are performed. This test case serves as reference values for the computation of energy and time models.

The different test cases are encapsulated between PAPI library [13] function calls therefore, only events during the runtime of the test case are measured. The PAPI events set consists of:

1. Computational intensity related events:
    - Cycles: number of cycles elapsed during the execution of the test case.
    - Instructions: number of instructions executed by the test case.
    - Double precision scalar flops: number of flops related to 64 bit registers measured during the run of the test case.
    - Double precision 128-bit flops: number of flops related to 128 bit registers measured during the run of the test case.
    - Double precision 256-bit flops: number of flops related to 256 bit registers measured during the run of the test case.
    - Double precision 512-bit flops: number of flops related to 512 bit registers measured during the run of the test case.
    - Double precision flops: aggregation of flops measured during the execution of the test case.
2. Memory usage related events:
    - L1: L1 cache misses count.
    - L2: L2 cache misses count.
    - L3: L3 cache misses count.

Additionally, other events out of the PAPI [13, 14] context, related to the uncore subsystem are measured:

- Bandwidth: memory bandwidth consumed by the test case.
- Uncore frequency: frequency achieved by the memory during the execution of the test case. This is related to the vector usage intensity of the test case. There are two uncore frequencies, each of them related to each socket.

In order to compare the energy and time projected by the model, it is also necessary to measure the time and energy (or power) consumed by each test case during its execution:

- Package power/energy: power/energy consumed/used by each of the processor sockets during the execution of the test case.
- DRAM power/energy: power/energy consumed/used by each of the DDRAM modules during the execution of the test case.
- Runtime: Time elapsed between the beginning and end of the test case.

In total 18 events were measured per test case.

All of the event values can be measured by unprivileged users; in order to be able to measure the uncore events and the energy or power values, the application must be launched as superuser (*root* user in GNU/Linux). Thus, an important machine configuration task has to be performed before being able to run the micro-benchmark.

4.1.2 Compilation

For measuring the energy consumed by the APEX-MAP application it must be linked to the PAPI library and also to the implemented library that is needed to access and read the uncore registers.

Since the micro-benchmark is not optimized for any particular architecture, in order to obtain the value of the specific counters, the application must be compiled with architecture-specific variables. Additionally, the behaviour with respect to other architectures must remain, so that the energy model can be applied even when an application is not optimized for a superior architecture.

In this particular case, the benchmark has been compiled with five different sets of compilation flags resulting in five different binary files from the same source code:

1. -O0: no compilation optimization applied.
2. -O3 -msse4.2: loop optimizations without vectorization. No AVX optimizations applied.
3. -O3 -xAVX -nolib-inline: vectorization optimizations applied. Only 128-bit AVX instructions (*AVX*) will appear in the assembler code.
4. -O3 -xCORE-AVX2 -nolib-inline: vectorization optimizations applied. Only 256-bit AVX instructions (*AVX2*) will be taken into account.
5. -O3 -xCOMMON-AVX512 -nolib-inline: vectorization optimization applied. Only 512-bit AVX instructions (*AVX512*) will be counted.

4.1.3 Execution and output

The tests were performed on an Intel Skylake compute node (48 cores, 2 threads per core) at the Leibniz Supercomputing Centre of the Bavarian Academy of Science and Humanities. This node is very similar to the ones included in the DEEP-EST cluster, but at the time when the model was computed, there was no access to the DEEP-EST cluster. Moving the code from one machine to another with similar characteristics implies a new computation of the model coefficients, but there is no need of recompiling the code or creating new compiler sets.

The measurements were performed in a reserved compute node so that no other users could perform actions which could interfere with the event measurements. The micro-benchmark, due to the reasons explained before, was executed as a super user. Furthermore, no other benchmarks or computations were running on the machine while performing the measurements.

Each micro-benchmark binary runs independently with a fixed frequency and for all of the available frequency in the system that means the sixteen frequencies from 1.2GHz to 2.7GHz with intervals of 100MHz. The application is configured to fill the compute node, i.e. use 96 MPI tasks (48 hyperthreaded cores) and it allocates all the available memory of the node. The output of the micro-benchmark consists of a set of metrics per experiment.

Thus, the complete input set for the modelling procedure corresponds to:

- Five different binaries according to the five different sets of compilation flags.
- Sixteen different fixed frequencies used to run each application.

- Two different test cases per application varying the input L and K parameters, resulting in forty experiments per test case.
- An additional test case for reference values resulting in one experiment.
- Eighteen hardware counters and events per record.

The complete input set for the modelling procedure consists of 6480 records each of 25 parameters (compilation flags set, configured fixed frequency, elapsed runtime, five input parameters, seven computational intensity events, three memory usage events, three uncore events, and four energy/power hardware counters) or components.

4.2 Model calculation

Auweter et al. [16] described a model for energy aware scheduling optimizations for Intel SandyBridge architecture based on a base frequency selected for an entire application. They reported a good accuracy with less than 3% deviation from real values in 5 out of 6 applications which were analysed [16].

The approximation of the power consumption was done through the following equation:

$$PWR(f_n) = PWR(f_0) * \Big[A_n * GIPS(f_0) + B_n * \frac{1}{GIPS(f_0)} + C_n * CPI(f_0) + D_n * \frac{1}{CPI(f_0)} + E_n * GL3PS(f_0) + F_n * \frac{GL3PS(f_0)}{GIPS(f_0)} + G_n * GL2PS(f_0) + H_n * \frac{GL2PS(f_0)}{GIPS(f_0)}\Big] \quad (4.1)$$

The definitions of each term are shown in Table 6.

$PWR(f_n)$	Predicted power at frequency n.
A_n, B_n, C_n, D_n, E_n, F_n, G_n, H_n	Model constants characterizing a given platform at frequency n
$GIPS(f_0)$	Giga instructions per second at the nominal frequency f_0
$CPI(f_0)$	Clocks per instruction at the nominal frequency f_0
$GL2PS(f_0)$	Giga L2 cache misses per second at the nominal frequency f_0
$GL3PS(f_0)$	Giga L3 cache misses per second at the nominal frequency f_0

Table 6: Model definitions.

The constants A_n, B_n, C_n, D_n, E_n, F_n, G_n, H_n are calculated for the model only once and stored in the plugin. The other terms $GIPS(f_0)$, $CPI(f_0)$, $GL2PS(f_0)$, $GL3PS(f_0)$ are measured for each application at the nominal frequency f_0, which is the frequency used to define the model constants.

The equation 4.1 is applied for a fixed frequency n, so to predict the power consumption for any frequency from the available ones, the expression can be rewritten in a matrix multiplication notation as follows:

$$\frac{PWR(f_n)}{PWR(f_0)} = M_{F,C} * P_{C,1} \tag{4.2}$$

$$M = \begin{pmatrix} A_1 & B_1 & C_1 & D_1 & E_1 & F_1 & G_1 & H_1 \\ A_2 & B_2 & C_2 & D_2 & E_2 & F_2 & G_2 & H_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ A_i & B_i & C_i & D_i & E_i & F_i & G_i & H_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ A_n & B_n & C_n & D_n & E_n & F_n & G_n & H_n \end{pmatrix}_{F \times C} \tag{4.3}$$

$$P = \begin{pmatrix} GIPS(f_0) \\ \frac{1}{GIPS(f_0)} \\ CPI(f_0) \\ \frac{1}{CPI(f_0)} \\ GL3PS(f_0) \\ \frac{GL3PS(f_0)}{GIPS(f_0)} \\ GL2PS(f_0) \\ \frac{GL2PS(f_0)}{GIPS(f_0)} \end{pmatrix}_{C \times 1} \tag{4.4}$$

where,

- f_0 is the nominal/default frequency,
- f_n is the frequency on study,
- $M_{F,C}$ is the coefficients matrix result of a linear regression procedure,
- F is the number of frequencies (=16 in out current study),
- C is the number of components (=8 from A_i until H_i), and
- P is the vector of the values of components measured at nominal/default frequency.

The idea behind this model is to run the application at nominal frequency measuring the value of the hardware counters which will be represented under the P vector. This vector P remains constant for the application. The matrix M is machine dependent and defines the contribution of each component to the total energy use.

The equation 4.2 was described for SandyBridge processors but it cannot be applied to the new processors, due to their new features; there were no processors with AVX512 instruction set and the memory bandwidth was estimated using the cache misses at all cache levels. If we applied this model to foresee the energy use of an application running on a current processor, such as Skylake or Knight Landing, the accuracy of the projection would be very low. Thus, we need models in which the new features (AVX2, AVX512) of the new processor families are taken into account.

The idea proposed to compute the energy/power model follows the expression described in 4.2 but new components should be added to both matrices; new coefficients must be calculated for the new expression components and the behaviour of the application regarding these new components must be known. Furthermore, the model should be extended for the new available frequencies of planned processors.

4.2.1 Components reduction

Each of the measured components can be considered to represent a dimension/axis in space. The union of all these dimensions results in the energy curve used by the application as a function of each of the components. The measured energy of every register is a point in this multidimensional space.

Since there is the possibility that several components are related to each other (e.g. cycles, instructions and frequency, or memory bandwidth, and last cache misses), we must look for the set of linearly independent dimensions that define the same space, while losing as little information as possible. This is exactly the motivation of the Principal Component Analysis (PCA) methodology [17].

PCA is a dimension-reduction tool that can be used to reduce a large set of variables to a small set that still contains most of the information of the large sets. It is a mathematical procedure that transforms a number of (possibly) correlated variables into a possibly smaller number of uncorrelated variables called principal components. The methodology is based on the computation of the eigenvalues and eigenvectors of the matrix consisting of the measured components; if an eigenvalue is zero, it means that two axis of the space are linearly dependent, so one of them does not contribute any new information to the equation and thus, this component can be eliminated from the expression.

The space which represents the energy use of the system by the micro-benchmark was initially defined by 25 components, but since the energy curve should not rely on the application input parameters, the energy space can already be reduced to 20 parameters. After applying the PCA reduction method, the space was reduced from 20 parameters to maximally 12 and minimally 5 parameters. The difference between 12 and 5 parameters relies on different computed models (see section 4.3).

4.3 Results

Several R scripts [18] to perform the PCA reduction and compute the linear regression were implemented with the R-studio IDE. R-studio is a free and open-source integrated development environment (IDE) for R, a programming language for statistical computing and graphics. Furthermore, R-studio provides a package to work with PCA algorithms.

As mentioned before, the input data for the model consists of 6480 records. Bearing in mind that these records must be used both for model computation and for feeding the model with real data and calculate the fitting, a random partition of the records into 2 groups of the same

size was performed. A first group was used to compute the matrix of system coefficients, while the second group was used to check the accuracy of the model.

One of the first steps to get an idea about the modelling procedure is to calculate the principal components of the energy consumption space and depict their appearance. Instructions, cycles, and memory bandwidth components must be part of the model expression, since these three components mostly explain the energy values distribution. This can be seen in Figure 19. Additionally, a high range of energy values where observed while using the different compilation flag sets defined. Therefore, the presence of the flops counters in the final model expression will be mandatory. In any case, one of the big differences between SandyBridge and Skylake processors is exactly the AVX512 instruction set, so it makes sense to study the energy use behaviour concerning the instruction set of processors.

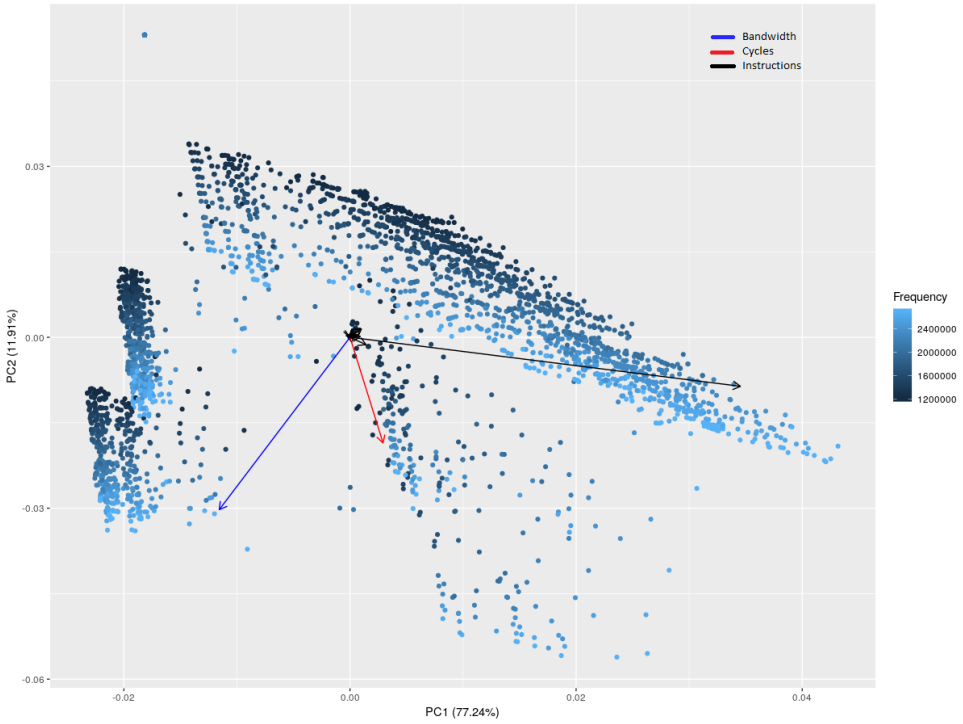


Figure 19: Visualization of the principal components of the space defining the energy use of the micro-benchmark. Three main components can be seen: Instructions, Cycles, and Memory Bandwidth. Other components are also present but the contribution is lower.

Taking all these restrictions into account (the presence of instructions, cycles, memory bandwidth, flops events, and hardware counters in the energy model), it is just a matter of experimenting with different combinations of these mandatory components and computing the accuracy of the model with respect to the energy fitting.

4.3.1 Study of tentative energy models

Seven models with different component combinations were tested and studied. However, only the three most relevant ones will be introduced here. For all of them, the root mean squared error ($RMSE$) between the measured and the predicted energy was calculated. This value helped to identify if a certain combination of components better fits the energy curve or not (see Table 7). For each of the possible models, three different accuracy tests were performed:

- Fitting of the experiments, which already were executed at nominal frequency. In this case, the fitting should be essentially perfect, since an energy projection for the nominal frequency is calculated using the same nominal frequency as input frequency.
- Fitting of the first 1000 records, which implies the experiments with the lower energy consumption.
- Fitting of the complete set of records.

Model Id	$RMSE_{2.3GHz}$	$RMSE_{1000}$	$RMSE_{global}$	Counters
1	301	906	933	Instr, CYC, CPI, 1/CPI, L1, L2, L3, Band, DP_esc, DP_128, DP_256, DP_512
2	253	3425	3578	Instr, CPI, 1/CPI, Band, DP_esc, DP_128, DP_256, DP_512/Instr
7	243	289	283	Instr, CPI, Band, DP_esc, (DP_128 + DP_256 + DP_512)/CYC

Table 7: Possible energy models study depending on the set of components and their contribution. $RMSE$ stands for root mean squared error, $Instr$ stands for instructions, CYC for cycles, CPI for clocks per instruction, $L1$ for level 1 cache misses, $L2$ for level 2 cache misses, $L3$ for level 3 cache misses, $Band$ for memory bandwidth, DP_esc for 64 bits register flops, DP_128 for 128 bits register flops, DP_256 for 256 bits register flops and DP_512 stands for 512 bits register flops.

Model 1

The first developed model, identified by *Model 1*, contains the complete set of parameters of the SandyBridge model plus the AVX hardware counters. The component $\frac{1}{CPI}$ corresponds to the derivative response causing the energy use to decrease if the CPI variable increases rapidly.

The value of the relative mean error as a consequence of the fit between the measured and projected energy curves at a frequency equal to the nominal frequency is relative high (301J from about 4200J means 7% aproximatelly), while a RMSE value between both curves taking into account just only the first 1000 experiments of 906J (nearly 22%) is observed. Using the complete set of records implies a RMSE value of approximately 24%. This high increment of the RMSE value corresponds to the noise introduced by the components included in the energy model which should be eliminated from the equation, since they do not contribute with relevant information for the fitting. Filtering these components will result in a better adjust.

The corresponding fittings are depicted in Figures 20, 21, and 22.

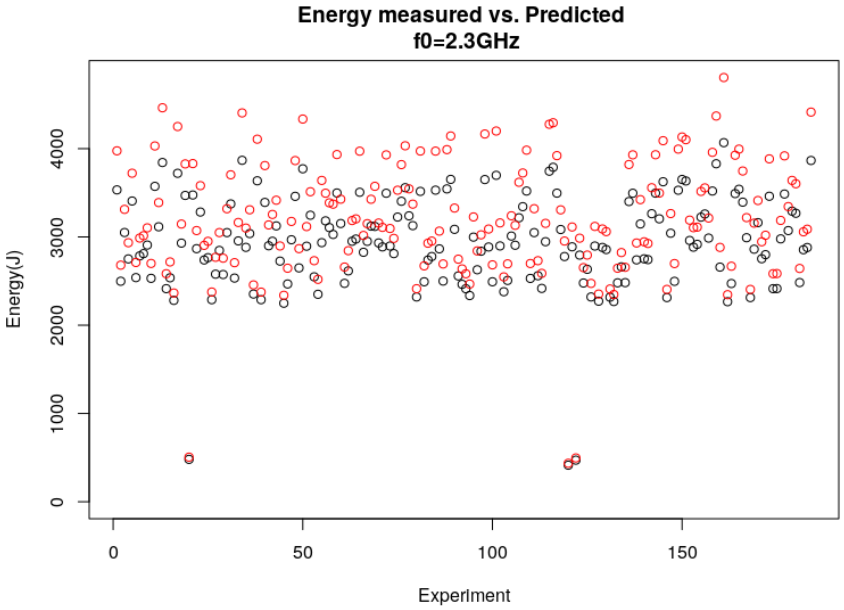


Figure 20: Scattered plot of measured (black) and predicted energy (red) values using the *Model 1* for those applications, which were already running at nominal frequency. Since the frequency used for the prediction is the same as the nominal one f_0, the predicted energy used should be essentially the same.

This first model can be described as a model with an excessive number of parameters.

Model 2

The *Model 2* contains a set of components similar to the one of *model one*, but in this case, the cycles component is missing. Since the cycles component is one of the three most important components (see Figure 19 for details), filtering this component, causes high RMSE values; the RMSE values for the 1000 first experiments are nearly 82% and close to 85% when using the complete set of registers.

The corresponding fittings are depicted in Figures 23, 24, and 25.

The Model 2 can be defined, opposite to Model 1, as a model with too few parameters.

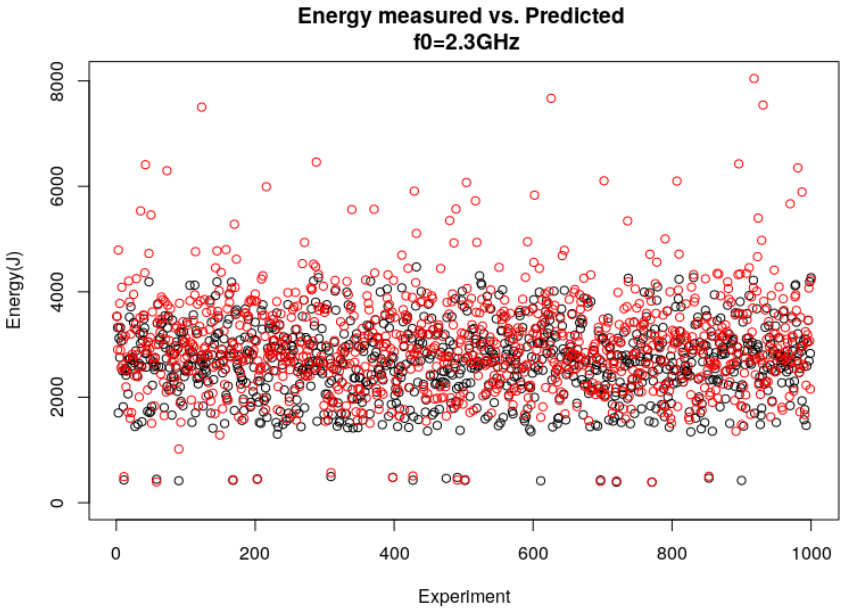


Figure 21: Scattered plot of measured (black) and predicted energy (red) using the *Model 1* for only the first 1000 registers. Lower energy values are better predicted that high energy values in which we can see registers with more than 80% of absolute error.

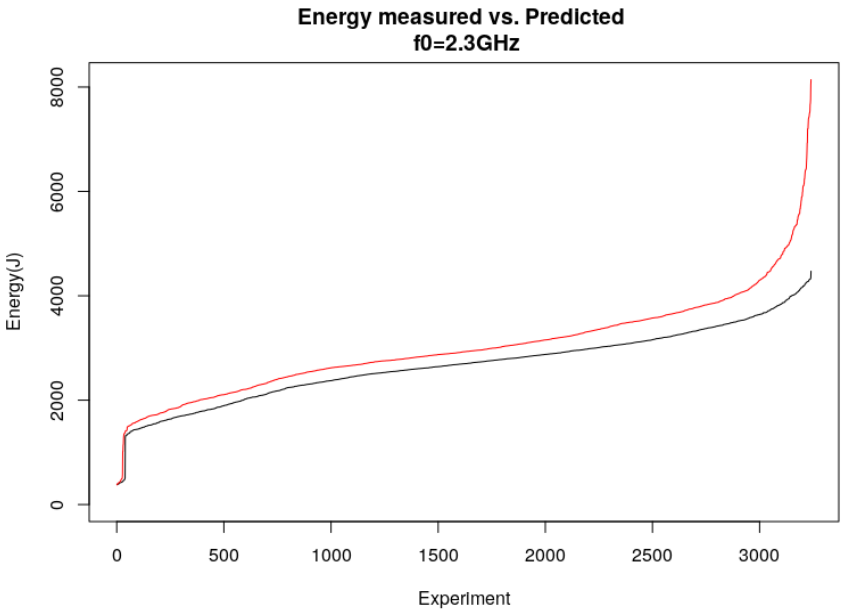


Figure 22: Comparison between the energy measured (black) and energy predicted (red) values using as input data the complete registers collection regarding the use of the *Model 1*. Experiments have been sorted by energy consumption non independently, i.e. the energy measured and predicted data points for the same experiment number belong to the same experiment.

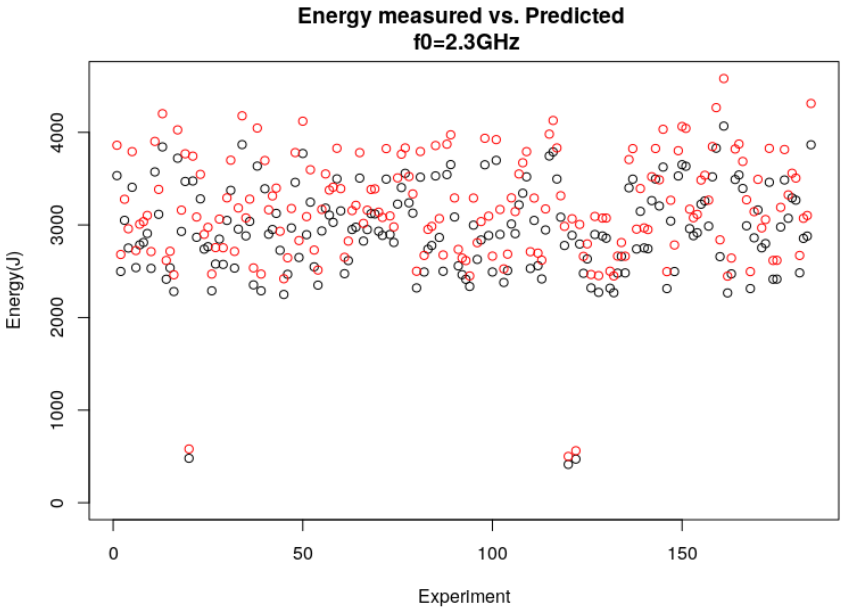


Figure 23: Scattered plot of measured (black) and predicted energy (red) values using the *Model 2* for those applications that were already executed at nominal frequency. Since the frequency used for the prediction is the same as the nominal one f_0, the predicted energy is essentially the same.

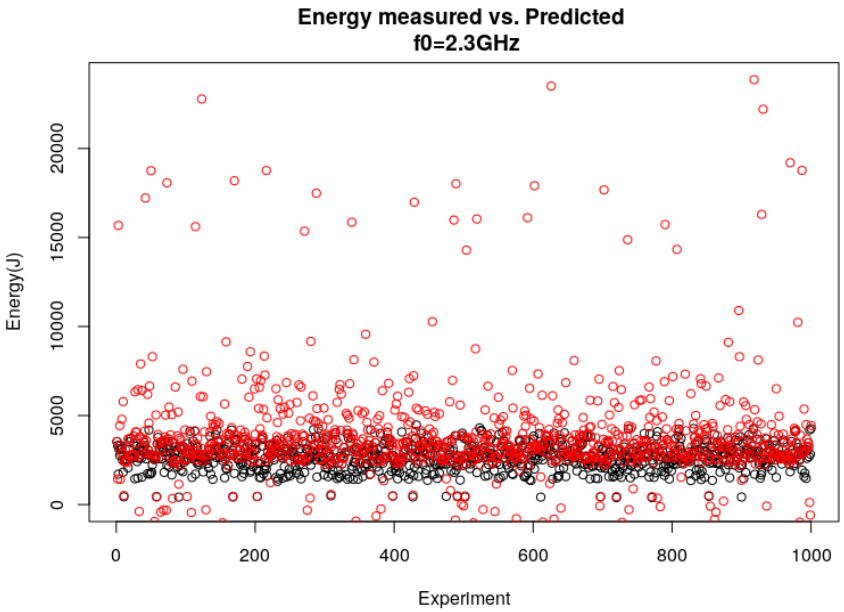


Figure 24: Scattered plot of measured (black) and predicted energy (red) values using the *Model 2* for only the first 1000 data points. In this case, since one of the principal components is missing from the model expression, the absolute errors are much bigger than using the *Model1* (Figure 21).

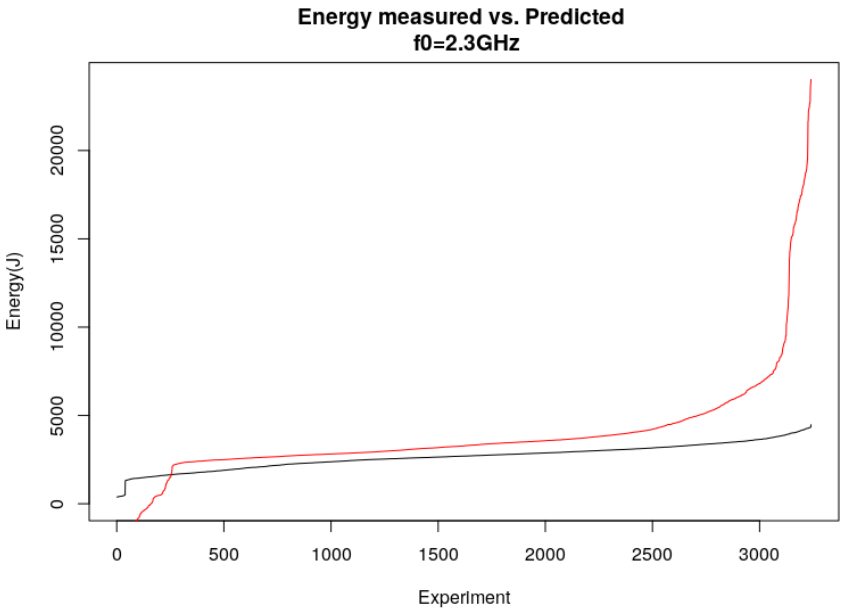


Figure 25: Comparison between the energy measured (black) and energy predicted (red) values using as input data the complete set of data points regarding the use of the *Model 2*. Since a principal component, cycles, were filtered from the equation, the accuracy of the fitting is very low.

Model 7

At the moment, the model called *Model 7* is the most accurate one. In this model a new parameter was introduced that balances the number of AVX instructions (whether 128, 256, or 512 bits) with respect to the number of cycles. This new parameter gives us a measure on the vectorization of the code.

The RMSE value for this model varies from 5.8% when predicting the same energy as measured, to 6.8% and 6.9% using the first 1000 registers or the complete input data correspondingly.

The corresponding fittings are depicted in Figures 26, 27, and 28.

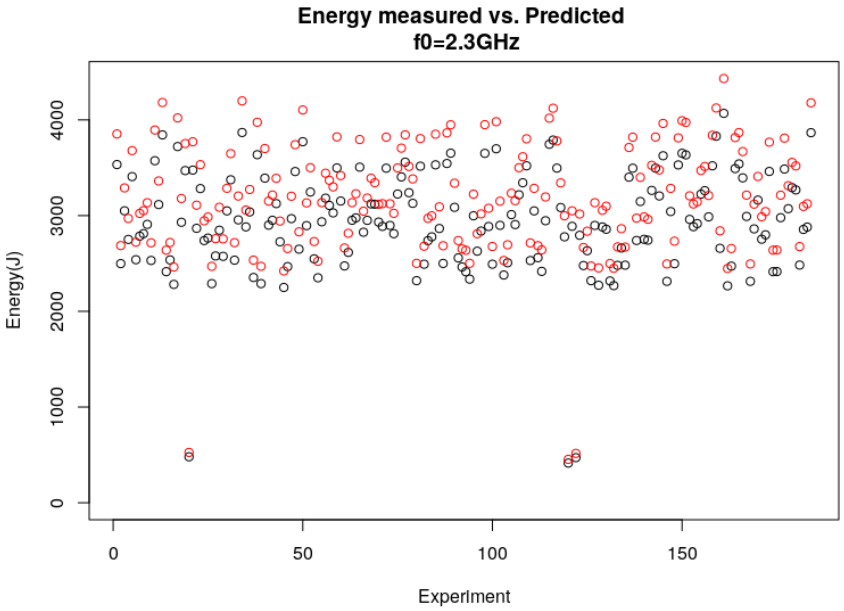


Figure 26: Scatter plot of measured (black) and predicted energy (red) values using the *Model 7* for those applications that were already running at nominal frequency. Since the frequency used for the prediction is the same as the nominal one f_0, the predicted energy used should be essentially the same.

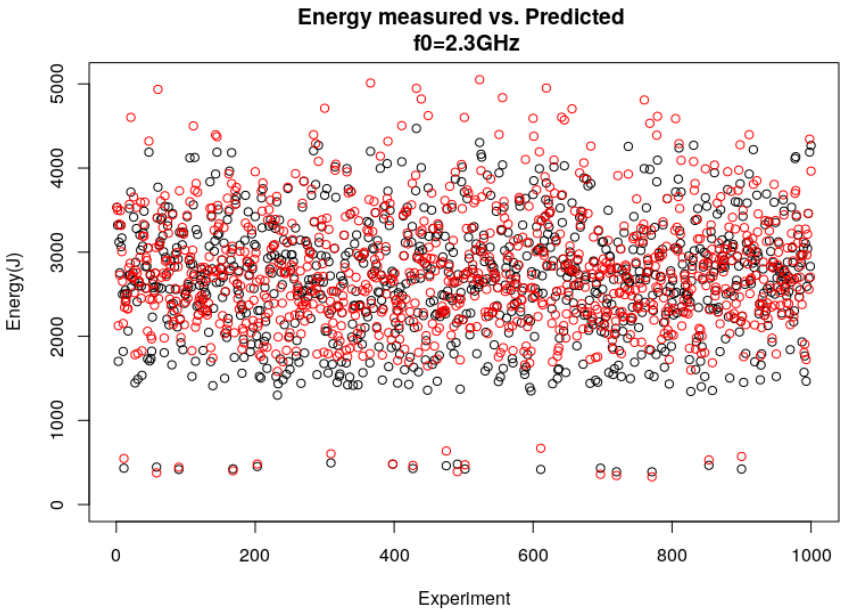


Figure 27: Scattered plot of measured (black) and predicted energy (red) values using the *Model 7* for only the first 1000 registers. In this case, a better fitting in comparison with the previous models can be seen (Figure 21 and Figure 24). Maximal absolute error is around 20%.

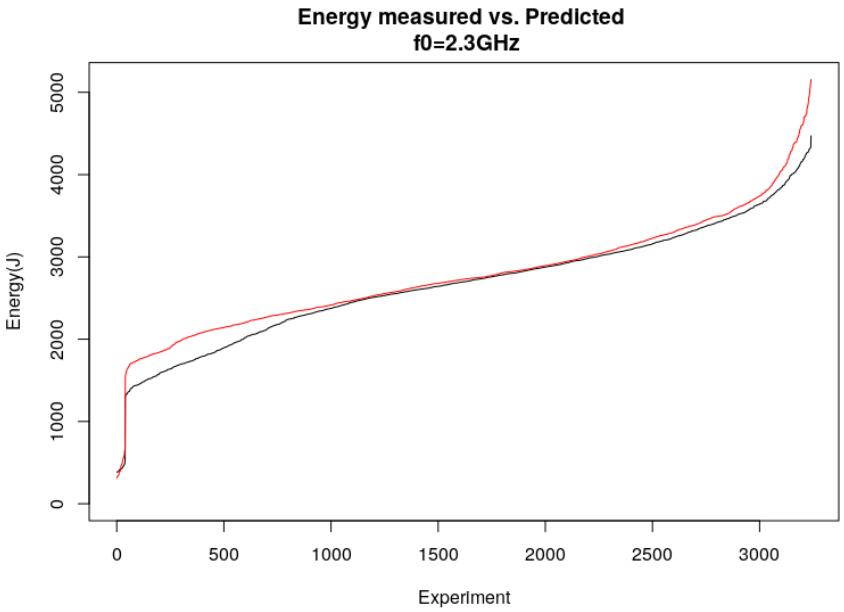


Figure 28: Comparison between the energy measured (black) and energy predicted (red) values using as input data the complete registers collection regarding the use of the *Model 7* energy model.

4.3.2 Comparison of tentative energy models

In order to easily compare the accuracy of the three possible models studied, a comparison plot of the evolution of the relative error between the energy predicted by the model and the real energy used by the experiment can be found in Figure 29.

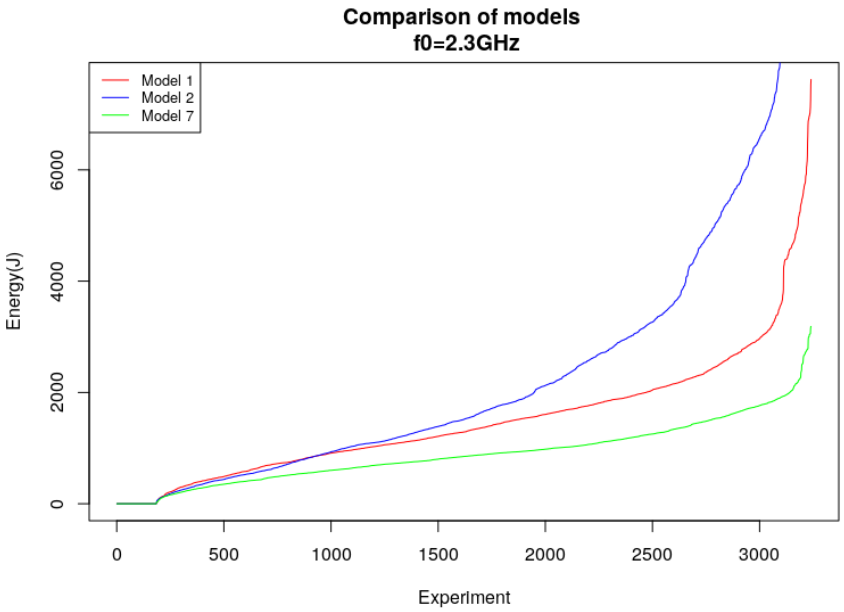


Figure 29: Comparison of the evolution of the relative error between the energy predicted by each of the models and the real energy used by the experiment. The values are ordered by energy used in ascending order.

4.4 Conclusions

In this document a working methodology for the calculation of energy models was presented. This methodology was applied to an architecture similar to that of one of the modules of the project cluster, obtaining a model with an average error of less than 7%. The next step is to apply this methodology to calculate the energy consumption models of applications for the other modules.

5 Summary

This deliverable presents the first proposals and analysis in three of the four topics for WP2: Application performance models, workload models, and energy models.

Regarding performance models, we have demonstrated how we can use the BSC performance tools to build a model based on the key factors that determine the efficiency with which an application is executed. By measuring these factors we can identify the current bottlenecks of the execution as well as to compute estimators of the behaviour at a larger scale. The first study has been done with GROMACS. Using the Dimemas tool we have been able to evaluate the potential of different optimization alternatives that were selected based on the results of the efficiency analysis. The study shows that the benefit of these optimizations would help to improve the scalability without a significant penalty at low corecounts (which are not our target). This is a very important insight, because GROMACS typically runs in strong scaling mode.

Workload traces or models are used for job scheduling evaluation. Because there are no existing modular systems, we decided to create a workload model that will allow us to create a diversity of traces with different characteristics and to evaluate the performance of WP5 proposals under various conditions. A modular system is not as simple as a heterogeneous cluster or N homogeneous clusters joined. There are new alternatives that must be taken into account and reflected when generating a workload. To generate workload as representative as possible (given it is not enough experience to define what ”representative” means in this context), we have used as reference existing workload models together with the characteristics of the different applications, in terms of module flexibility, and their plans in the near future in the context of the project.

Since at this stage of the project we cannot provide evaluation results of WP5 contributions, we have evaluated the impact of the different workloads generated by applying our ”module list” in the ”partition list” field because they present some similarities from the point of view of the impact of wait time. Results, even though they can not be interpreted from a numerical point of view, show a high potential in terms of wait time reduction with not strong requirements with respect the amount of jobs with module flexibility required to improve system performance. The next steps will include the creation of medium and big workload traces for WP5 evaluation where only single-module jobs are considered and the extension of our workload model to include restrictions such as ”jobs for module A cannot be executed in module B”. We will also include multi-module jobs in our traces, generating a second set of traces with this additional characteristic.

Energy management can be addressed from different perspectives: from hardware facilities to high level analysis such as the one presented here. WP2 contribution in this specific topic is providing energy models to be used in application analysis, optimization, scheduling decisions, etc. Energy modelling is a process that must be repeated for each new set of applications and/or architectures. This deliverable presents the methodology to create the model, the benchmark used (ported to the new architecture as part of the work done in the project), the energy model, and evaluation. This model is suitable for the Cluster Module (CM) of DEEP-EST. The ongoing work includes the extension of the APEX-MAP benchmark for GPUS existing in the DAM and ESB modules.

List of Acronyms and Abbreviations

C

CM Cluster Module: with its Cluster Nodes (CN) containing high-end general-purpose processors and a relatively large amount of memory per core

D

DAM Data Analytics Module: with nodes (DN) based on general-purpose processors, a huge amount of (non-volatile) memory per core, and support for the specific requirements of data-intensive applications

DEEP-EST DEEP - Extreme Scale Technologies

E

ESB Extreme Scale Booster: with highly energy-efficient many-core processors as Booster Nodes (BN), but a reduced amount of memory per core at high bandwidth

H

HPC High Performance Computing

P

PCA Principal Component Analysis

S

SLURM Job scheduler that will be used and extended in the DEEP-EST prototype

SPMD Single Program Multiple Data

Bibliography

[1] Morris A. Jette, Andy B. Yoo and Mark Grondona: *SLURM: Simple Linux Utility for Resource Management*, in Proceedings of the 9th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP), Springer, Lecture Notes in Computer Science (LNCS), volume 2862, pages 44–60, http://dx.doi.org/10.1007/10968987_3

[2] D. Krause and P. Thörnig: *JURECA: General-purpose supercomputer at Jülich Supercomputing Centre* in Journal of Large-scale Research Facilities (JLSRF), volume 2, article 62, http://dx.doi.org/10.17815/jlsrf-2-121

[3] Stephen Trofinoff and Massimo Benini: *Using and Modifying the BSC Slurm Workload Simulator* in Slurm User Group Meeting, 2015, https://slurm.schedmd.com/SLUG15/BSC_Slurm_Workload_Simulator_Enhancements.pdf

[4] *Heterogeneous Job Support in Slurm* [Online], Available: https://slurm.schedmd.com/heterogeneous_jobs.html

[5] *Consumable Resources in Slurm* [Online], Available: https://slurm.schedmd.com/cons_res.html

[6] *Jobs Submitted to Multiple Partitions, p6* [Online], Available: https://slurm.schedmd.com/SUG14/sched_tutorial.pdf

[7] *The RICC log* [Online], Available: http://www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html

[8] Weinberg V, Brehm M, Christadler I. 2010. *OMI4papps: Optimisation, Modelling and Implementation for Highly Parallel Applications*.

[9] Strohmaier E, Shan H. 2004. *Architecture independent performance characterization and benchmarking for scientific applications*. International symposium on modeling, analysis and simulation of computer telecommunications systems.

[10] Strohmaier E, Shan H. 2005. *Apex-Map: A global data access benchmark to analyze HPC systems and parallel programming paradigms*. Proceedings of SC2005.

[11] *Extrae instrumentation tool*[Online], Available: https://tools.bsc.es/extrae

[12] *The workload on parallel supercomputers: modeling the characteristics of rigid*. Journal of Parallel and Distributed Computing Year 2003, Volume 63, Number 11, Pages 1105-1122.

[13] Browne S, Deane C, Ho G, Mucci P. 1999. *PAPI: A Portable Interface to Hardware Performance Counters*. Proceedings of Department of Defense HPCMP Users Group Conference

[14] David H, Gorbato E, Hanebutte U, Khanna R, Le C. 2010. *RAPL: memory power estimation and capping*. Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design. ACM.

[15] Intel Corp. 2012. *Intel 64 and IA-32 Architectures Software Developer Manual*.

[16] Auweter A., Bode A, Brehm M, Brochard L, Hammer N, Huber H, Panda R, Thomas F, Wilde T. 2014. *A Case Study of Energy Aware Scheduling on SuperMUC*. International Supercomputing Conference (ISC).

[17] del Moral M.J., Valderrama M.J. 1997 *A principal component approach to dynamic regression models*. Journal of Forecasting (13), 237244.

[18] *R Studio tool*[Online], Available: http://www.rstudio.com

[19] *Workload logs*[ONLINE] Available: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html

[20] *Workload models*[ONLINE] Available: http://www.cs.huji.ac.il/labs/parallel/workload/models.html

[21] *List of publications for Lublin model*[ONLINE] Available: http://www.cs.huji.ac.il/labs/parallel/workload/models.html#lublin99

[22] *JURECA system overview*[Online] Available: https://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html