THE UNIVERSITY *of* EDINBURGH

Optimisation of lattice simulations energy efficiency

for the NVIDIA A100 and ATOS BullSequana XH2000 platforms

Antonin Portelli (School of Physics & Astronomy)

October 14, 2022

Copyright

Supplemental Data

Data and code associated with this study is available at https://doi.org/10.5281/zenodo.7057645.

Acknowledgements

AP warmly thanks the HPC System Team at the Advanced Computing Facility in Edinburgh for the technical support they dedicated to this study. This work was funded by STFC DiRAC grant ST/S003762/1. AP also received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreements No 757646 & 813942. Grid (https://github.com/paboyle/Grid) is free software under the GNU General Public License v2.

Contact

James Clerk Maxwell Building,
Peter Guthrie Tait Road,
Edinburgh EH9 3FD, UK
Email: antonin.portelli@ed.ac.uk

Changelog

v0.1	02/09/2022	first draft submitted to DiRAC for feedback
v1.0	08/09/2022	first complete draft for release
v1.1	14/10/2022	first release after comments from DiRAC

Contents

1 Motivations

It is crucial for the scientific community to be concerned by the environmental impact of their computing practices, especially in communities relying critically on large-scale simulations that can only be ran on supercomputers. Additionally, in the current context of energy supply issues and price surges, the impact of energy costs on operational budgets can rapidly lead to damages to scientific programmes, in the eventuality price volatility goes well beyond what was projected at the budgeting stage. It is therefore crucial to be aware of how efficiently a given computation task can be performed, from the point of view of achieved computational work in units of energy spent. Additionally, it is important to notice that different workflows and software will have different energy profiles, and that finding energy-optimal hardware settings will be domain-specific in the large majority of cases.

This study aims at studying and optimising the energy efficiency of lattice simulations, specifically simulations based on the Grid library[1], and targetted at the NVIDIA A100 GPU, integrated into the ATOS BullSequana XH2000 system. This work was specifically targetted a the STFC DiRAC supercomputer "Tursa", however large systems with the same configuration are present in Europe, specifically JUWELS Booster at Forschungszentrum Jülich[2], and the future pre-exascale system Leonardo at Cineca[3]. We expect the conclusions of this study applies to these systems as well.

2 Protocol

All the tests described in this study were performed on the STFC DiRAC supercomputer "Tursa" at the University of Edinburgh. This system features 112 nodes with an NVIDIA RedStone board with 4 A100-40 GPUs, 2 AMD EPYC 7302 CPUs, 4 HDR200 network interfaces, and 1 TiB of host memory. Nodes are integrated in ATOS BullSequana XH2000 racks. GPU nodes are spread across 5 racks, rack 0 contains 16 nodes, and racks 1 to 4 each contain 24 nodes.

2.1 Grid DWF benchmark

To simulate a typical lattice QCD calculation, we use the single-precision domain-wall fermion (DWF) benchmark `Benchmark_dwf_fp32` included in the Grid library. This benchmark contains two successive phases where a random vector is multiplied a large number of times by a sparse matrix, and the performance in MFlop/s over the whole sequence is measured. In the first phase, the sparse matrix used is the hopping term of the DWF operator, and in the second phase the even-odd part of the same operator is used. For the non-specialist, inversions of the DWF operator generally dominates the compute time used in lattice QCD calculations using domain-wall fermions. These inversions are generally performed using iterative Krylov solvers, and therefore the inversion cost comes from a large number of multiplications by the operator, which depends on the chosen algorithm and the condition number of the operator. Other choices for fermion operators are used in the lattice community, and the DWF ones have the particularity that they act on a five-dimensional space, increasing the intensity of the operation compared to four-dimensional formulations such as Wilson fermions. The even-odd version of the operator is used in red-black preconditionned solvers.

[1] https://github.com/paboyle/Grid
[2] https://www.fz-juelich.de/en/ias/jsc/systems/supercomputers/juwels
[3] https://leonardo-supercomputer.cineca.eu/

Finally, it is worth noting that a large majority of the Tursa system is currently used for lattice QCD calculations using the DWF formulation, implemented using Grid.

We used the library at commit 188d2c7a[4] and the benchmark was slightly modified to increase by a factor 100 the number of matrix multiplications, in order to mitigate uncertainties due to the coarse sampling of power monitoring tools. The modification of the benchmark is given by the change below to be apply to the file `benchmarks/Benchmark_dwf_fp32.cpp` at line 194.

```
-  int ncall =300;
+  int ncall =30000;
```

When using this benchmark, the global four-dimensional lattice volume can be specified using the `--grid` command-line option. We chose two local lattice volumes for this study. The first volume named “C0” is $24 \times 24 \times 24 \times 12$, it is named after a dataset used in production on Tursa, and constitutes a problem size representative of most of the current calculations on this system. The second local volume named “loc32” is $32 \times 32 \times 32 \times 32$, and is a good representation of larger calculations to take place on this system in the next years. A larger local volume implies more floating point operations to be performed by each node, and is expected to yield higher Flop/s. In all cases the fifth dimension is fixed to 16 sites by the benchmark and is not distributed across nodes.

Finally, we compile and run Grid with the known performance-optimal settings that were determined during the technical commissioning of the system. Each node runs 4 MPI processes, optimally pinned to cores and NUMA domains to maximise the access speed to the 4 GPUs. Additionally NVIDIA GPUDirect RDMA was used for multi-node communications. The complete set of compilation and runtime options can be found in the supplemental data associated with this report.

2.2 GPU frequency control and power monitoring

In this section we describe the different tools used in this study to control the GPU clock frequency and monitor the power consumption of the system.

2.2.1 GPU frequency control

The frequency of the NVIDIA A100-40 GPU is variable, and can be limited to a maximum frequency ranging from 210 MHz to 1410 MHz, by increments of 15 MHz. By default there is no limit in place (which is equivalent to a limit of 1410 MHz). Frequency limits can be set using the NVIDIA SMI[5] interface, which is part of the CUDA toolkit. NVIDIA SMI features a command-line interface, and for example the command

```
nvidia-smi -ac 1215,1110
```

sets all GPUs of the current node with a clock limit of 1110 MHz, and a memory clock limit of 1215 MHz. On the A100-40 GPU the memory clock limit is not adjustable, and 1215 MHz is the only authorised value. This command needs to be executed with root privileges. In all the benchmarks performed in this study, the `nvidia-smi` command was used, invoked through `sudo`. When using a large number of nodes, this process was automatised by retrieving the user password from a safe location and piping it in the `sudo` command, and the resulting command was broadcasted using SSH to all nodes.

We strongly advise against using the setup described here in production, as it would require giving root privileges to users and storing passwords on the file system. Instead, one should expose

[4] https://github.com/paboyle/Grid/tree/188d2c7a4dc77807b545f5f2813cdb589b9e44ca
[5] https://developer.nvidia.com/nvidia-system-management-interface

the GPU frequency limit as a user option in job scripts and configure the system appropriately. In the case of the Slurm workload manager this can be done using the `--gpu-freq` option[6].

2.2.2 GPU power monitoring

On a given node, the GPU power consumption in watts can be monitored using the NVIDIA SMI interface described above. We used the command

```
nvidia-smi dmon -o DT
```

which performs real-time monitoring with the addition of timestamps to the output. We used the maximum sampling frequency allowed by this tool, which is one sample per second (this is the default setting). This tool can run with user permissions.

2.2.3 XH2000 rack power monitoring

We additionally used tools provided by ATOS to monitor power consumption in watts at the whole rack level. The reported figures include all elements plugged on a specific rack, including network equipment and full node consumption. These tools do not provide continuous monitoring at regular interval, and we implemented a monitoring loop as part of the test scripts. We found that the maximum achievable sampling frequency was around one sample every 7 seconds, and time intervals between samples were recorded to take into account possible variations.

2.3 GPU frequency limit scan procedure

Here we describe the different steps of the two tests performed in this study. The benchmarks were performed on 2 XH2000 racks, each containing 24 GPU nodes. The total 48 nodes are partitioned in two 16-node jobs, named “16A” and “16B”, and two 8-node jobs, similarly named “8A” and “8B”. These blocks are statically allocated to insure maximum interconnect performance, according to the network layout established with ATOS during the validation of the system. The two racks used were reserved for the test, and no other jobs were allowed to run on this partition. With this setup, the test sequence is as follows

1. Start of the XH2000 rack monitoring loop. Each sample is collected approximately every 7 seconds, and is timestamped with the UNIX epoch at which the monitoring command was invoked. The resulting measurement is inserted in an SQLite database.
2. Start of the clock frequency limit loops iterating over the 81 possible frequencies between 210 MHz and 1410 MHz.
3. The 4 jobs 8A, 8B, 16A, and 16B are submitted. Each job set the frequency limits on all nodes, and monitor the GPU power draw of the master node using `nvidia-smi dmon` for the duration of the execution. The result of each NVIDIA SMI output is parsed and stored in an individual SQLite table.
4. Once all four jobs have completed, the test proceed with the next loop iteration.
5. Once the frequency loop is finished, the rack power monitoring is interrupted.

[6] `https://slurm.schedmd.com/gres.html`

The whole procedure is repeated for both the C0 and loc32 problem sizes. Two comments are in order. Firstly, one can notice that the GPU power monitoring is only running on the master node. The workflow benchmarked here is exclusively using synchronous parallelism, and up to negligible fluctuation all GPUs are expected to behave identically to the master node. Therefore we do not expect that this approximation has any impact on our results. Secondly, in principle the 8-node jobs and 16-node jobs can have different execution times depending on scaling efficiency. Here we remind the reader that the local problem size is kept constant across jobs, so differences in execution times are related to weak scaling efficiency. It is known that Tursa has close to perfect weak scaling running Grid on these problem sizes for more than 8 nodes. In practice we observed that the 8-node jobs were faster than the 16-node ones only by a negligible fraction of the execution time, confirming the previous statement.

The whole set of run scripts and monitoring data is available as part of the supplemental data released with this report.

3 Results

We present in this section the results obtained following the protocol described above.

3.1 Raw data examples

In this section, we give examples of the raw data captured by the two power monitoring tools described in Sec. 2.2.

In Fig. 1, we show the GPU power draw and activity in function of the execution time, as recorded by NVIDIA SMI. This example is for the two problem sizes, with a GPU clock limit of 1020 MHz. In this figure, we clearly see the two phases of the benchmark, i.e. first the Dhop operator benchmark followed by the DhopEO operator. The short periods of inactivity correspond to the required initialisation sequences for the two phases, but the execution time is nonetheless dominated by GPU activity. We also observe that the DhopEO phase is about 2 times faster than the Dhop one, this is expected as the DhopEO sparse matrix operates on a checkerboarded lattice, which has half the volume of the full one. We see that both operators generate a similar level of activity in the GPUs, which is essentially the maximum level of activity for the loc32 problem size, and around 60% of the maximum activity for C0. The smaller problem size C0 is more sensitive to bottlenecks from the interconnect, and we are confident that this explains the lower activity levels. In Fig. 2, we plot racks 1 and 2 power draw in function of time for a full scan of the 81 possible GPU clock frequency limits. We clearly see the power draw increasing as we allow for higher frequencies.

Let us now discuss the data analysis to study how the power draw depends on the GPU clock frequency limit.

3.2 Power consumption

The total energy E of a job is given by

$$E = \int_{t_a}^{t_b} \mathrm{d}t\, p(t)\,, \qquad (1)$$

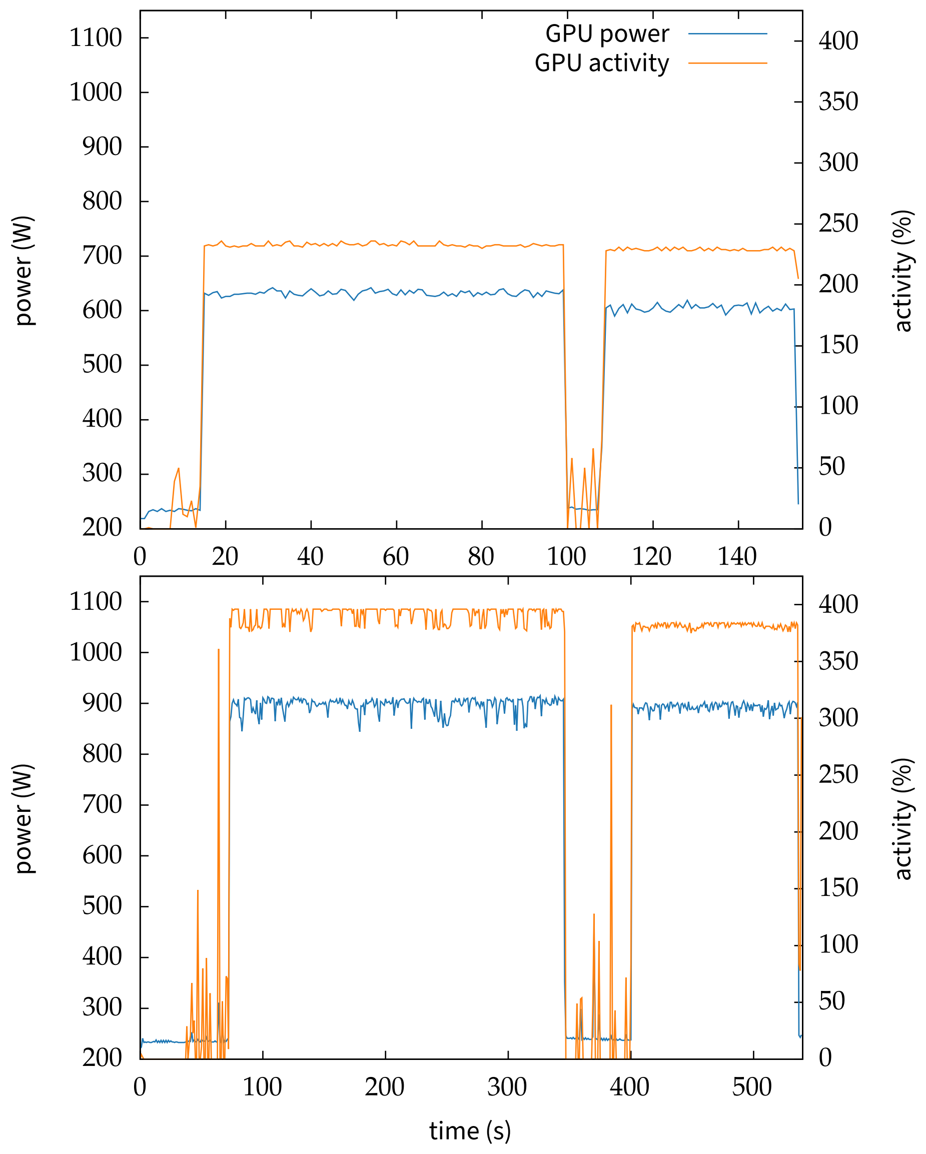

Figure 1: Example of the GPU activity in function of the execution time on the master node of a 16-node job, with a clock limit of 1020 MHz. The upper plot is the activity for the C0 problem size, and the lower plot for the loc32 size. The left vertical axis indicates the 4 GPUs power draw in watts as measure by the NVIDIA SMI. The right vertical axis indicates the GPUs activity, 400% being the maximum possible value.

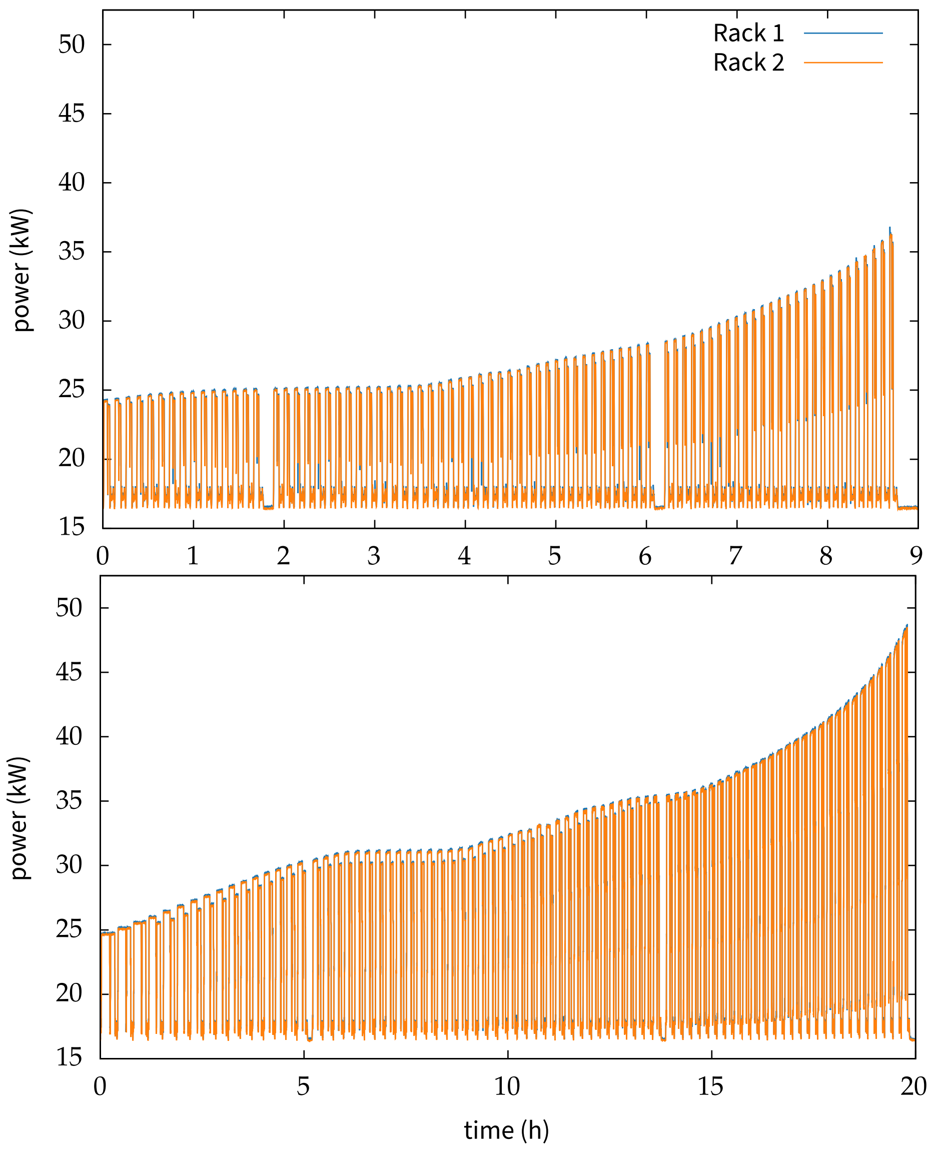

Figure 2: Rack power in function of the test time on individual racks, as measured by the ATOS XH2000 monitoring tool. The upper plot is the activity for the C0 problem size, and the lower plot for the loc32 size. In both cases, the test loop executes 81 sets of benchmarks populating the whole 2 racks, increasing the GPUs clock limit from 210 MHz to 1410 MHz by increments of 15 MHz.

where t_a and t_b are respectively the start and end time of the job, and $p(t)$ is the power draw at time t. The average power draw $\overline{p}$ is then given by

$$\overline{p} = \frac{E}{t_b - t_a}\,. \tag{2}$$

In practice, we collect power samples at discrete values t_n of the execution time. The NVIDIA SMI time collects sample at each second, and we found it is sufficient to compute the average power as the average of the power samples

$$\overline{p}_{\mathrm{GPU}} \simeq \frac{1}{N}\sum_{n=1}^{N} p_{\mathrm{GPU}}(t_n)\,, \tag{3}$$

where $t_1 = t_a$ and $t_N = t_b$. For the rack monitoring, the samples are collected at a lower frequency (approximately every 7 seconds), and the interval between two samples potentially varies by a few seconds. To take into account the irregularity of the measurements, and to mitigate the coarseness of the sampling, we compute the energy using a trapezoidal rule

$$E_{\mathrm{rack}} \simeq \frac{1}{2}\sum_{n=1}^{N-1}(t_{n+1}-t_n)[p_{\mathrm{rack}}(t_{n+1}) + p_{\mathrm{rack}}(t_n)]\,, \tag{4}$$

where it is understood that the number of samples N and the sample times t_n are different than in the GPU case. The average rack power $\overline{p}_{\mathrm{rack}}$ is then computed using Eq. (2).

In Fig. 3, we show the observed average power draw in function of the GPU clock limit. Thanks to the two monitoring in place, we can deduce the average power draw of the non-GPU elements, simply defined as

$$\overline{p}_{\text{non-GPU}} = \overline{p}_{\mathrm{rack}} - \overline{p}_{\mathrm{GPU}}\,. \tag{5}$$

A first possible observation is that the rack average power measurement for the C0 size is noisier than in the loc32 case. This is due to the coarseness of the rack power monitoring sampling and the shorter execution time in the case of C0. Another more important observation is that in both cases the change in power draw is essentially entirely due to the GPUs, and the non-GPU elements consume a constant power of approximately 25 kW. Through a measurement of the power draw with both racks idle, we found that this figure is also consistent with the idle power draw of the non-GPU elements.

We now move to discussing the energy efficiency of the benchmarks, which is the core objective of this study.

3.3 Energy efficiency

We start this section by showing the performances of the various benchmarks, as plotted in Fig. 4. A number of observations can be made. Firstly, the performance observed at maximum clock limit (the default setting) are in excellent agreement with Grid benchmarks performed in production and during the technical commissioning of the system. Secondly, the Dhop and DhopEO performances are very similar, and for the sake of simplicity we will call “average performances” the average of the two operators. Thirdly, the A100 GPU has a peak performance of 19.5 TFlop/s[7], which means the GPU peak performance for this cluster is 78 TFlop/s/node. From this we can see that our benchmarks achieve 6% and 13% of the peak performances, for the C0 and loc32 problem sizes, respectively.

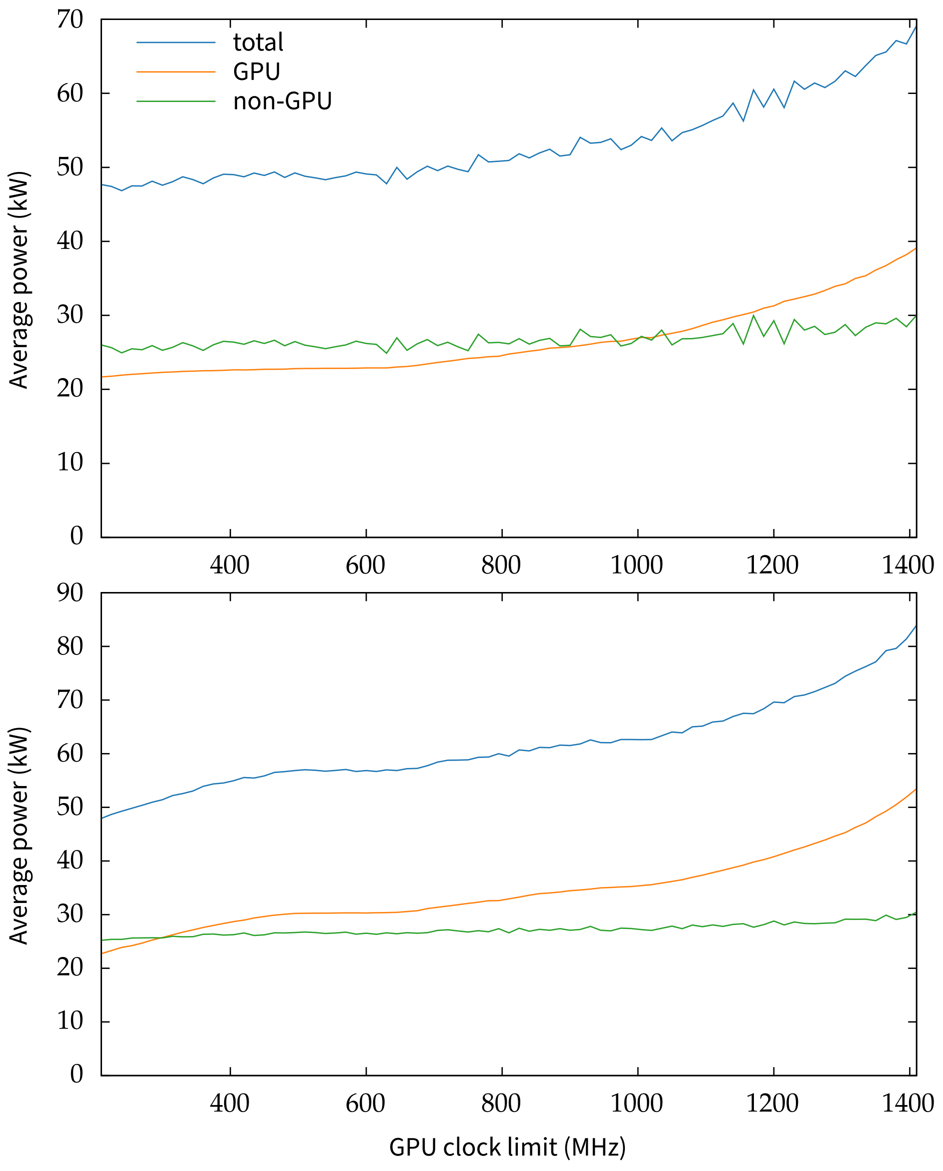

Figure 3: Average power draw in kilowatts, in function of the GPU clock limit. As previously, the upper plot is for the C0 problem size, and the lower plot for the loc32 size.

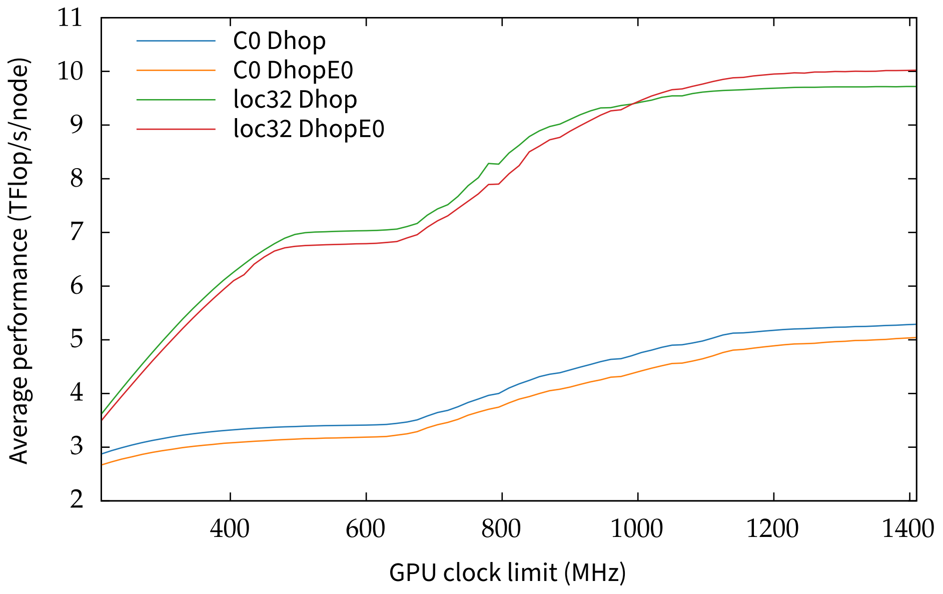

Figure 4: Measured performances in TFlop/s for the four different benchmarks, as a function of the GPU clock limit.

This is considerable considering the operators involved are mainly bandwidth-bound, and is the result of the high level of optimisation of the Grid routines.

Now, the energy efficiency of a given application can be measured as an amount of work per unit of energy. Here we adopt the units used in the Green500 list[8], *i.e.* GFlop/s/W, which is equivalent to GFlop/J. In Fig. 5 we show the measured energy efficiency of the Grid benchmarks for the two problem sizes. Firstly, as a reference Tursa is ranked 27 in the June 2022 Green500 list[9], with an energy efficiency of 21.431 GFlop/J. This figure is significantly higher than the ones shown here because the Green500 benchmark is based on LINPACK, which has a much higher intensity than the operators used here. Secondly, we start to observe that the default, maximum limit of 1410 MHz is less energy-efficient than lower clock limits. This is emphasised further in Fig. 6, where we plot the relative energy cost of the benchmarks in function of the GPU frequency limit. The notion of relative cost is defined as the ratio to the energy consumed for the default limit of 1410 MHz. In Fig. 6 we can read the optimal energy cost for the two problem sizes: with a clock limit around 1 GHz, a same amount of work can be performed for an energy cost reduced by 16% and 24%, for respectively C0 and loc32. These gains are computely solely on the basis of energy costs, but clearly lower clock speeds implies longer execution times, which might be undesirable by users. This tradeoff is the topic of the next section.

4 Decision-making

In this section we discuss a strategy for decision-making regarding the tradeoff between performances and energy costs mentionned in Sec. 3.3.

The issue at hand is a rather simple optimisation problem due to its low dimensionality. There

[7]https://www.nvidia.com/en-gb/data-center/a100/
[8]https://www.top500.org/lists/green500/
[9]https://www.top500.org/lists/green500/list/2022/06/

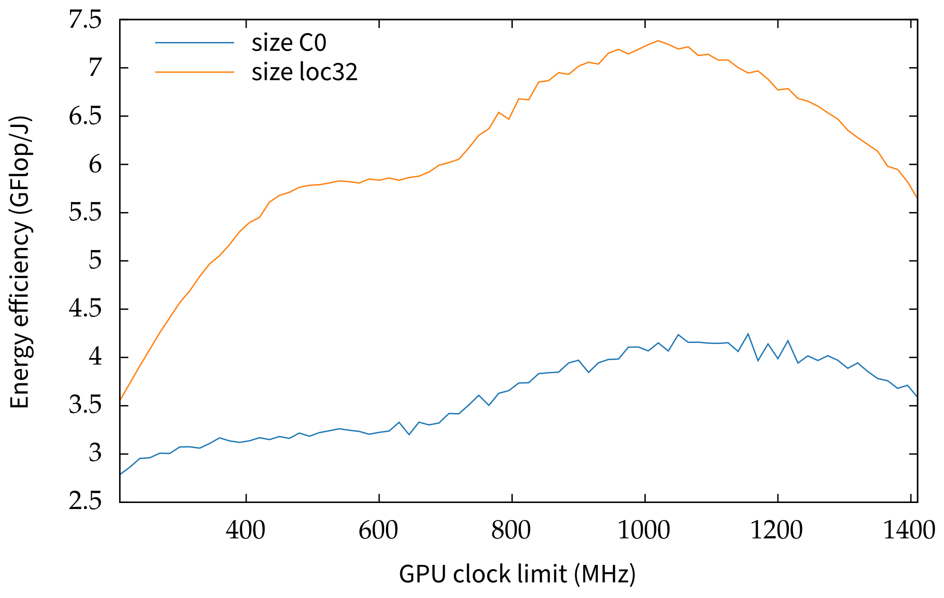

Figure 5: Energy efficiency in GFlop/J, as a function of the GPU clock limit.

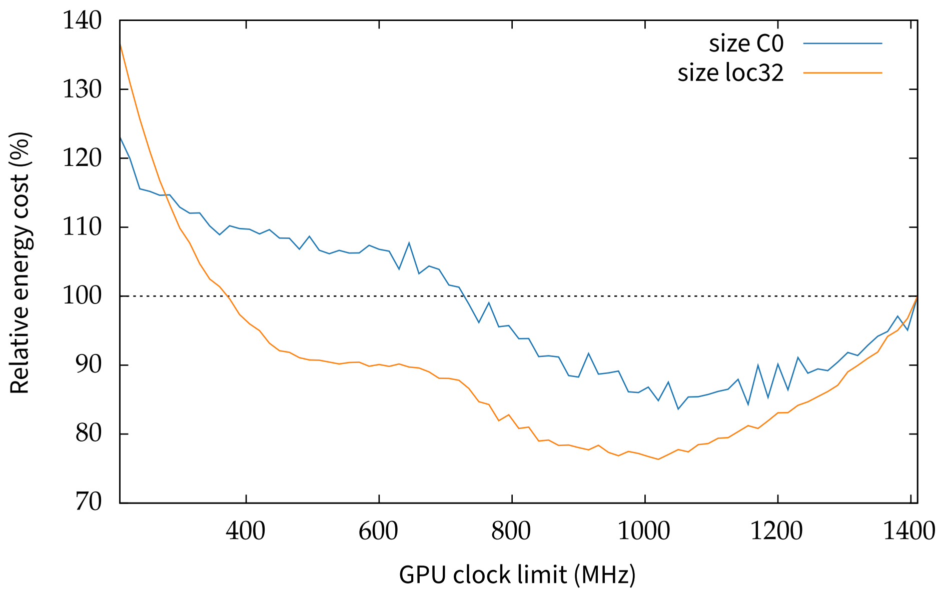

Figure 6: Relative energy cost as a function of the GPU clock limit. Here by “relative” we mean as the ratio to the measurements obtained at the maximal GPU clock limit of 1410 MHz.

is one dimension of benefit, which is the floating-point performance, and one dimension of cost, the total energy consumed by a fixed amount of work. In practice we want to minimise the energy cost while maximising the performances. In Fig. 7 we represent a landscape of all possible options. The result from this figure is very positive: it appears that the optimal energy settings lie in a region where performances will be negatively affected by no more than 10%. Let us quantify that more formally.

We propose to solve this problem using an ε-constraint optimisation on the performances. This is equivalent to answering the following question: "what GPU frequency limit minimises the energy cost, with a maximum negative impact on performances of ε relatively to the maximum clock performances?". In Fig. 9 we plot the GPU frequency limits solving this problem as a function of ε, for the two problem sizes. This figure confirms what could be seen already in Fig. 5: if an impact on performances of 10% is acceptable, then a frequency limit close to 1 GHz lead to an energy-efficient setup for both problem sizes, achieving energy savings from 16% to 24%.

A more radical approach can be taken in the eventuality a minimum level of power cut needs to be insured, regardless of efficiency. We can look at such constraint by considering an ε-constraint on the relative reduction of the median power draw against floating-point performances. This is equivalent to answering the following question: "what GPU frequency limit maximises floating-point performances while imposing a median power cut of at least ε relatively to the maximum frequency power draw?". Fig. 10 shows the solution of this optimisation problem. We can observe that at most 30% of power reduction on all problem sizes, due to the constant power draw of non-GPU elements, and the GPU idle draw (*cf.* Sec. 3.2 and Fig. 3).

5 Conclusion

In this study, we demonstrated that the default, maximum frequency setting of the NVIDIA A100 GPU is energy-inefficient for lattice calculations at problem sizes typically used in production, and even more on larger problem sizes to be used in the short-term future. We determined that a GPU frequency limit around 1 GHz yield 16% to 24% more energy-efficient calculations depending on the problem size, which is substantial. These figures take into account all elements in the XH2000 racks, and we also found comparing rack and GPU power measurements that the non-GPU elements have a power draw similar to their idle draw independently of the GPU activity, and constitute essentially half of the total power consumption.

To conclude, we strongly recommend computing centres to involve their users directly in assessing the energy-efficiency of their workflows. High-performance computing hardware is generally designed so that the highest level of performances is achieved under load, without direct considerations for the energy efficiency. It is likely that in general the maximum "turbo" frequencies of computing elements are energy-inefficient, albeit delivering higher performances. Informations and incentives should be provided to users so that projects make a priority to assess their energy efficiency. Possible strategies are for example to allocate kWh instead of wall-clock hours to projects, and/or to include energy-efficiency as part of the projects technical assessment. In any case, it is crucial to expose to users hardware settings which impact strongly the power draw, typically GPU and CPU frequencies.

Finally, the methodology used here is generic and can be extended to other workflows. For lattice simulations, it would be instructive to extend this study to other libraries beyond Grid, for example the NVIDIA library QUDA[10].

[10] https://github.com/lattice/quda

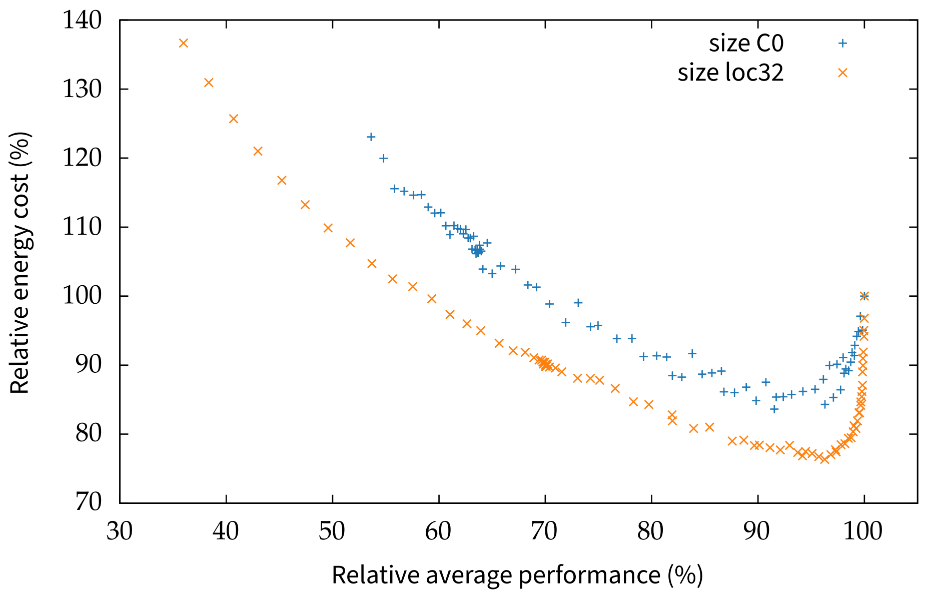

Figure 7: Relative energy cost versus the relative average performance for the two problem sizes. Here by "relative" we mean as the ratio to the measurements obtained at the maximal GPU clock limit of 1410 MHz.

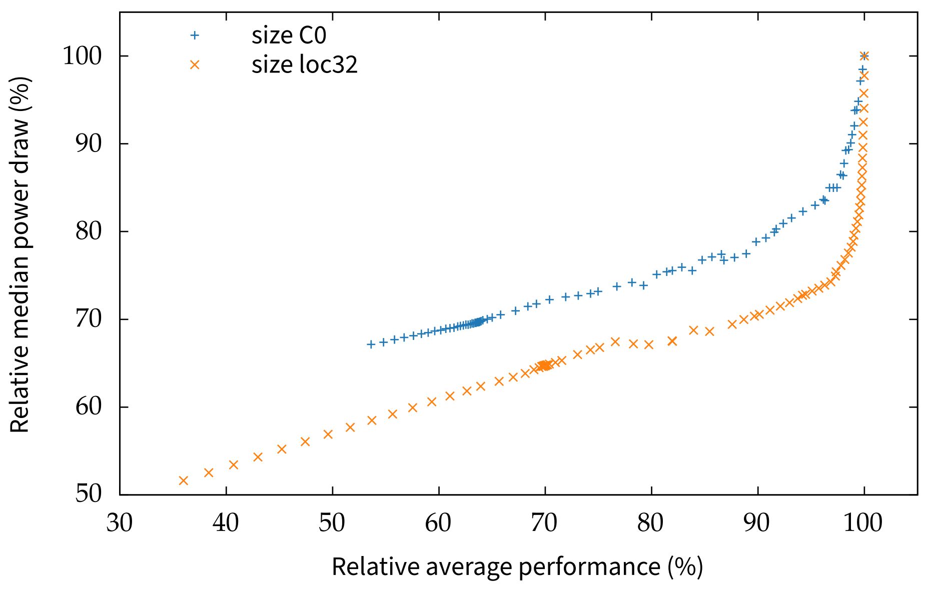

Figure 8: Relative median power draw versus the relative average performance for the two problem sizes. Here by "relative" we mean as the ratio to the measurements obtained at the maximal GPU clock limit of 1410 MHz.

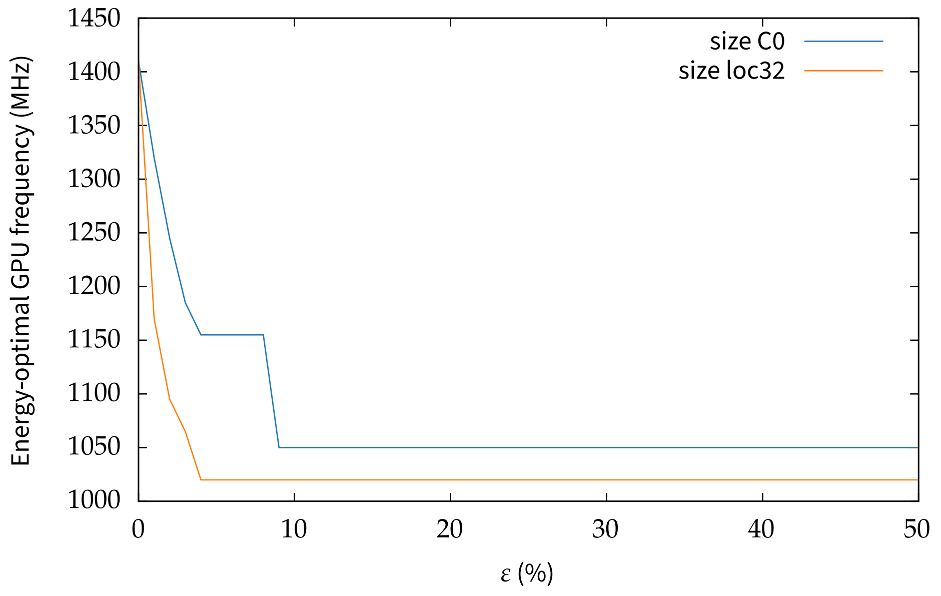

Figure 9: Energy-optimal GPU frequency limits solving an ε-constraint on performances, *i.e.* the GPU frequency that consumes the less energy while tolerating a maximum reduction of ε in performances relatively to the maximum frequency setting.

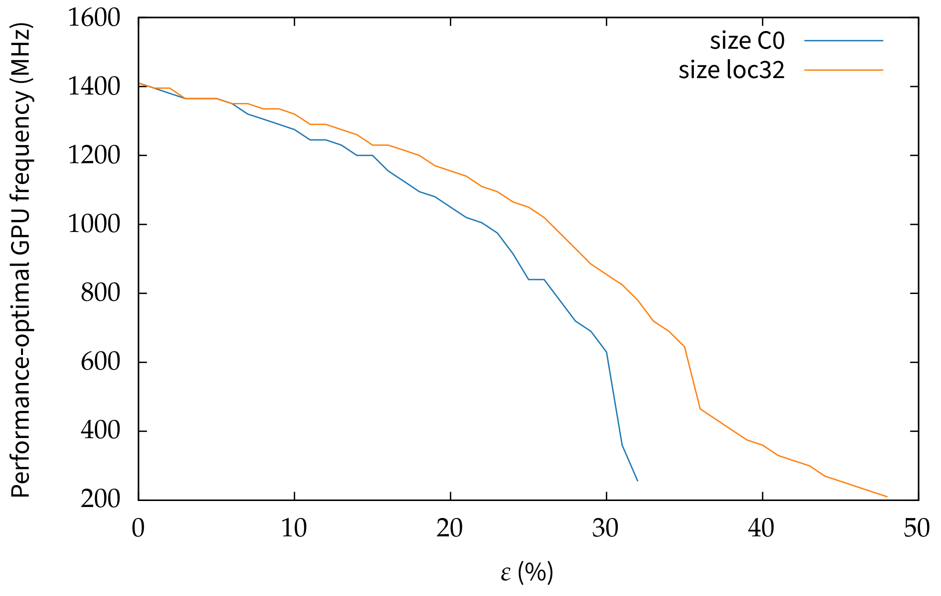

Figure 10: Energy-optimal GPU frequency limits solving an ε-constraint on median power draw reduction, *i.e.* the GPU frequency that achieves the maximum performances while imposing a median power draw cut of at least ε.