This is the title of the Proposal

Principal Investigator[1], First Collaborator[2], and Last Collaborator[1,2]

[1]Affiliation of the PI
[2]Other affiliation

Abstract

This is the abstract of the proposal: in this document we provide a template for CSCS Production Project Submission with guidelines, focusing in particular on sections **Representative benchmarks and Scaling**, **Performance Analysis** and **Resource Justification**.

Background and Significance

The project proposal should be no longer than **10 A4 pages** including graphs and references, and must contain the following information:

- Abstract
- Background and significance
- Scientific goals and objectives
- Research methods, algorithms and code parallelization (including memory requirements)
- Representative benchmarks and scaling
- Performance analysis
- Resource justification (annual node hours and disk space)
  - ⋆ Visualisation, pre- and post-processing needs
  - ⋆ Development and debugging requirements
- Project plan: tasks and milestones
- Previous results

Please follow the structure used in this template, which reflects the requirements reported on Production Projects Submission.

Scientific Goals and Objectives

...

Research Methods, Algorithms and Code Parallelization

Please insert in this section a description of the methods and algorithms of all the codes adopted for your computational study, justifying your choices and describing possible alternatives. Furthermore, please specify whether you are the main developer, a contributing developer or a user of the code. You should include a brief list of the main scientific libraries employed and a description of the parallelization approach, with specific memory and I/O requirements as well. In general community codes publish this information on their web sites. For instance, the CP2K home page reports that *CP2K is written in Fortran 2003 and can be run efficiently in parallel using a combination of multi-threading, MPI, and CUDA*. However, please clearly state if you use your own modified version of a community code and briefly describe why you are not using the public release of the software.

If you don't use a community code, please report if the code employs MPI distributed parallelism or hybrid MPI/OpenMP, which type of MPI communication has been implemented and if it makes use of shared memory parallelism, GPU accelerators or OpenACC/CUDA, with specific memory and I/O requirements as well.

Validation, Verification, Uncertainty Quantification, State of the art

Please explain how to validate your method against experiments or other established reference data and verify the numerical consistency of your model, citing the relevant references to peer reviewed papers. Provide parametric sensitivity analysis of your method, with estimates of the uncertainty of your predictions. Data driven uncertainty quantification is encouraged and for multiphysics/multiscale problems, please estimate the uncertainty of the full methods and software. Place the proposed research in the context of competing work, explaining the advantages and drawbacks of your approach.

Representative Benchmarks and Scaling

Please report in this section the results of the mandatory strong scaling tests performed with the selected code: you should report scaling data and plot for every representative system of your project. The goal is to choose the most efficient job sizes to run the performance analysis on your representative systems.

You should select meaningful job sizes to simulate the representative systems, compatible with reasonably short runtimes: the lowest number of nodes is determined in general by memory and wall time constraints, while the highest node counts should let you identify the job size at which you reach $\sim 50\%$ of the parallel efficiency with respect to ideal scaling. If possible, please provide weak scaling tests as well, in addition to the required strong scaling data.

Table 1 reports the wall time in seconds and the corresponding speed-up for a single representative system. Figure 1 shows the scaling plot: we started the scaling test on a single node, taking this runtime as a reference to compute the speed-up of larger job sizes. We then proceed doubling the number of nodes and checking the corresponding speed-up, until we are sure to have reached the $\sim 50\%$ limit in parallel efficiency (16 nodes in the small example below).

Nodes	Wall time (s)	Speed-up	JobID
1	1164	1.00	...
2	669	1.74	...
4	365	3.19	...
8	214	5.44	...
16	141	8.26	...

Table 1: Wall time, speed-up, JobID

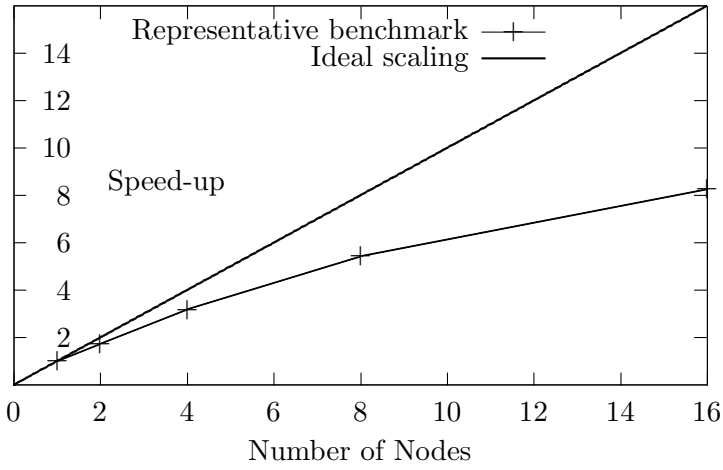

Figure 1: Strong scaling of the representative benchmark

The ratio of the speed-up values (benchmark vs. ideal scaling) for each job size gives the corresponding parallel efficiency.

Please select the optimal job size that you plan to use in production simulations, which should also yield a parallel efficiency (ratio of achieved speed-up vs. linear speed-up) above the 50% limit for your representative system: you will use this job size in the performance analysis.

You are also requested to provide some key parameters of the batch jobs planned on the scratch filesystem during the development of the project, listed below:

- the number of jobs that you plan to run simultaneously
- the wall clock time of your typical submission in hours
- the expected job size of your typical job in nodes
- the memory per node requested by the typical job in GB
- the maximum number of input files read by a job
- the maximum number of output files written by a job
- the largest file size in GB
- the maximum number of files accessed simultaneously during a job
- the library used for I/O
- the maximum number of files on scratch at the same time
- the maximum size of all files on scratch at the same time

The data above will help defining the workflow of your project: if the projects will develop in multiple tasks with different workflows, then you should report the data for each task of the project separately. The I/O patterns are relevant for the scratch filesystem and should be specified explicitly, indicating the maximum number of files accessed simultaneously on the scratch filesystem (Table 2), toghether with the library used for I/O (e.g.: HDF5 / NetCDF / MPIIO / POSIX / ...). Please note that the last two lines of Table 2 refer to the overall sum of file number and size, not to individual jobs.

Table 2 provides a template of how the workflow parameters should be presented in the proposal. In our example, a template project will develop in two tasks that will be described in section **Project**

Plan, therefore we provide two sets of workflow parameters in two columns, one for each task of the project. The overall usage of the scratch filesystem is reported for the two tasks together.

	First task	Second task
Number of simultaneous jobs	4	8
Typical wall clock time (hours)	6	12
Typical job size (nodes)	16	16
Memory per node (GB)	12	18
Maximum number of input files in a job	4	6
Maximum number of output files in a job	8	14
Largest file size (GB)	0.5	1.2
Maximum number of files accessed simultaneously in a job	16	16
Library used for I/O	HDF5	HDF5
Maximum number of files on scratch at the same time	200	
Maximum size of all files on scratch at the same time (GB)	680	

Table 2: Workflow parameters of the two tasks of a template project, providing the overall scratch filesystem usage

Performance Analysis

Please report here a summary of the performance analysis conducted on each representative system at the optimal job sizes selected in the previous section, which should be above the 50% parallel efficiency: you should run the performance analysis using the executable instrumented with *Cray Performance and Analysis Tools (CrayPAT)*.

A successful performance analysis job will create within the working folder a new directory named after our executable, followed by a plus sign (`+`) and few digits: e.g. `cp2k.psmp+17271-0s` as we have used the executable `cp2k.psmp`. This directory contains the report folder `rpt-files` with the report text file `RUNTIME.rpt`: we enclose this file at submission time and we insert within this section of the proposal a summary of the performance data extracted from the report text file, using the following command

```
grep -A 20 CrayPat/X RUNTIME.rpt | grep .
```

The summary should look like the example below:

```
CrayPat/X:  Version 20.08.0 Revision 28ef35c9f  07/08/20 20:40:20
Experiment:                  lite  lite-samples
Number of PEs (MPI ranks):    96
Numbers of PEs per Node:       6  PEs on each of  16  Nodes
Numbers of Threads per PE:     3
        Thread count includes 1 helper thread.
Number of Cores per Socket:    12
Execution start time:  Tue Sep  8 14:08:53 2020
System name and speed:  nid00000  2.601 GHz (nominal)
Intel Haswell    CPU  Family:  6  Model: 63  Stepping:  2
DRAM:  64 GiB DDR4-2400 on 2.6 GHz nodes
Avg Process Time:            147.99 secs
High Memory:                28,815.9 MiBytes     300.2 MiBytes per PE
Observed CPU clock boost:     116.4 %
```

```
Percent cycles stalled:        19.3 %
Instr per Cycle:               0.98
I/O Read Rate:              8.413001 MiBytes/sec
I/O Write Rate:            27.102315 MiBytes/sec
```

Together with the summary data, we also report within the proposal the `SLURM_JOB_ID` of the performance job, as it is not provided by `perftools`.

Resource Justification

The request of the annual amount of node hours should be clearly linked with the node hours used by the representative benchmarks: the number of node hours consumed by a simulation is computed multiplying the number of nodes by the wall time expressed in hours. CrayPAT adds an overhead to the wall time, therefore you cannot use that timing to justify your request.

In the small example used throughout this template, the optimal job size of the representative benchmark is 16 nodes and the corresponding wall time reported in Table 1 is 141 s, which is equivalent to ~ 0.6267 node hours, as a result of the following product:

$$0.6267 \text{ node hours } = 16 \text{ nodes } \times \frac{141s}{3600 \frac{s}{hour}}$$

The benchmark is short and represents in general a small number of iterations (cycles, timesteps or an equivalent measure), while in a real production simulation we will need to extend it.

Therefore we will estimate how many iterations should be necessary to complete a simulation in production. Furthermore, the project plan might contain multiple tasks, each of them requiring several sets of simulations to complete: the annual resource request will sum the corresponding node hours obtained multiplying all the factors reported in Table 3.

	First task	Second task
Simulations per task	2	4
Iterations per simulation	5000	10000
node hours per iteration	0.6267	0.6267
Total node hours	6267	25068

Table 3: Justification of the resource request

The small example above will request a total of 31335 annual node hours, summing the node hours estimated to complete the first and the second task of the project (Table 3), in agreement with the **Project Plan**.

You should present in this section your request for long term storage as well, explaining your needs based on the I/O pattern of the representative benchmarks reported by the performance analysis.

Visualisation, pre- and post-processing

Please insert in this subsection the optional requirements for visualisation, pre- and post-processing.

Development and debugging

Please insert in this subsection the optional requirements for development and debugging.

Project Plan: Tasks and Milestones

Please report tasks and milestones of your project. When describing your project development, aside from laying out the tasks that take you from beginning to end, please mark key dates as well. An easy way to do this graphically is through the use of a Gantt chart: milestones charts are also useful to determine more accurately whether or not a project is on schedule.

Results from Previous Allocations

Please list here your past requests, granted projects and used allocations (if applicable). You should also include a list of research publications that resulted from past allocations.

References

...