## Application form LUMI-BE (Belgian share of LUMI)

Applications can be of two types:

* LUMI-BE Preparatory: limited resources to either check the scalability of a code/workflow on the LUMI system or perform development to port a code/workflow on the LUMI system, in view of a future application for regular access. Maximum 500 CPU.kH or 25 GPU.kH.
* LUMI-BE Regular: standard application for large scale production runs. Maximum 10 CPU.MH or 500 GPU.kH.

Applications will be evaluated by a panel of academic and technical staff. Results will be sent to applicants one month after the application deadline.

Important remarks:

* Any application that does not scrupulously follow the application template will be rejected without evaluation.
* Multiple preparatory applications should not be used instead of a regular application: in case more than one preparatory application is necessary for a given research project, this must be explicitly justified.
* Multiple regular applications linked to the same research project should be carefully justified: applicants are encouraged to apply for compute time on the EU share of LUMI by submitting a EuroHPC proposal instead.
* *This is the first call that includes the GPU nodes of LUMI which are expected to become available to users in early November. However, as this is very new technology do expect that there will be technical hickups, and don’t expect that the LUMI User Support Team has solutions for all problems, nor can it do complicated software installations due to limited human resources. We suggest only more experienced users apply for GPU time in this call. Also be aware that the GPUs are AMD GPUs. Software that requires NVIDIA GPUs will not run.*

|  |
| --- |
| Type of application:  ☐ LUMI-BE Preparatory  ☐ In view of LUMI-BE Regular application?  ☐ In view of EuroHPC application?  ☐ LUMI-BE Regular  ☐ In view of EuroHPC application?  Reference number of related LUMI-BE Preparatory application, if any: ……  Applicant LAST NAME, first name:  Institution:  Research group / department:  E-mail address:  Core-hours (CPU.kH) applied for:  GPU-hours (GPU.kH) applied for:  Target platform (LUMI-C or LUMI-G):  Amount of scratch disk required (in TB.hours):  List of simulation codes and their version numbers (+license if required):  List of the members to be invited to the project (name and email address): |

1. [For regular applications only] Research project within the framework of which computing time is applied for: title, PI, financing institution or channel (RW, FNRS, FWO, VLAIO, EU, …). Mention the project/grant number. Attach a letter of approval of your own institution in case the project has not gone through a scientific approval process.
2. Include a short description of your research project, in layman’s terms wherever possible, with a view to dissemination. Explicitly mention the scientific questions that you are planning to address and the overall scientific goals of the project. (max. 1/2 A4 in Arial 12).
3. [For regular applications only] Justify the readiness of the simulation codes that will be used e.g., by:
   * referring to already relevant published results or previous projects about the scaling of the code,
   * providing relevant benchmark tests performed on an equivalent architecture (Tier-1 or Tier-0), preferably on LUMI itself.

The justifications should account for the potential jumps in system/problem sizes within the intended computational tasks. Provide tables and graphs. (Replace the dummy values in Table 1 and Plot 1 by real values).

If such data are not available, a “preparatory” project should be submitted first.

[For preparatory projects only] Explain the preparatory nature of the project: What are the goals, what is the hoped-for outcome, what are the expected next steps after finishing the project.

1. Overview and management of requested resources:

* Justify the number of core.hours (CPU.kH) and/or GPU.hours (GPU.kH), and storage volume (TB.H) applied for.
* Describe your planned computational tasks and the sequence in which these tasks will be performed. Will you use a task/workflow manager?
* Provide resource estimates (wall clock time, number of nodes/cores/GPUs, estimate of memory requirement (not the target node memory), storage. It should be based on the results of actual calculations. (Replace the dummy values in Tables 2 and/or 3 by real values.)
* Please present how you will manage your data. This is especially important since the time the storage is being used, will be charged. Describe how the transfer of files to/from LUMI will be managed and automated. Describe if data reduction and/or compression of files will be performed. If available, provide information about IOPS.

*Note that per requested GPU-hour on LUMI, you will automatically receive 64 core-hours on the CPU cores of the node containing that GPU unit.*

1. Can this proposal in its entirety be made public by FWO/FNRS or VSC/CECI e.g., as an example or inspiration for other researchers?

☐ Yes

☐ No

Don’t hesitate to consult [lumi-be-support@enccb.be](mailto:lumi-be-support@enccb.be) when you are preparing a LUMI application.

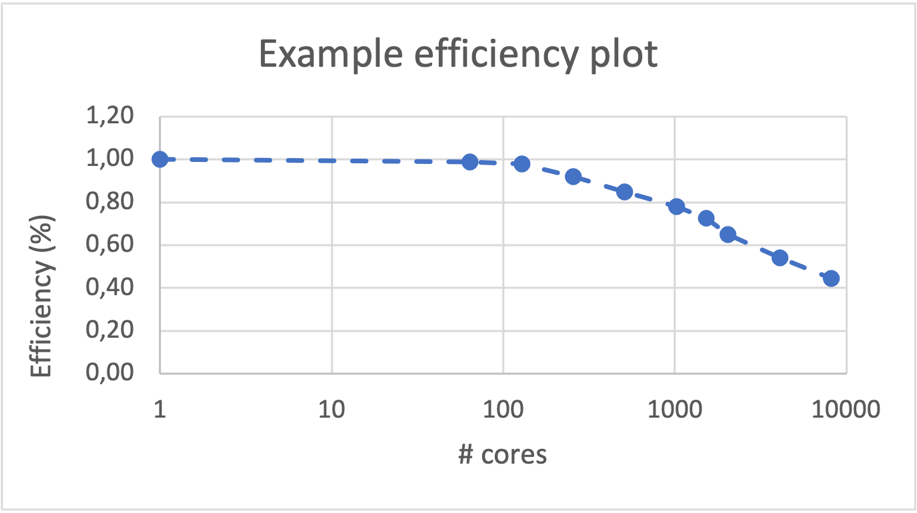
Appendix

Example Table 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of nodes | Total number of cores | Wall clock time (s) | Speed-up  (w.r.t. baseline) | Efficiency |
| *Abaseline* | *Bbaseline* | *Cbaseline* | *1.00* | *1.00* |
| *A1* | *B1* | *C1* | *Cbaseline/C1* | *(Bbaseline\*Cbaseline)/(B1\*C1)* |
| *A2* | *B2* | *C2* | *Cbaseline/C2* | *(Bbaseline\*Cbaseline)/(B2\*C2)* |
| *Baseline = minimal configuration with which your computational task can be carried out on LUMI.* | | | | |
| *Wall clock time is difference between start/end of the computational task, including any I/O operations as part of that task.* | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of nodes | Total number of cores | Wall clock time (s) | Speed-up  (w.r.t. baseline) | Efficiency |
| 1 | 1 | 200000,0 | 1,00 | 1,00 |
| 1 | 64 | 3160,9 | 63,27 | 0,99 |
| 1 | 128 | 1596,9 | 125,24 | 0,98 |
| 2 | 256 | 850,0 | 235,29 | 0,92 |
| 4 | 512 | 460,0 | 434,78 | 0,85 |
| 8 | 1024 | 250,0 | 800,00 | 0,78 |
| 12 | 1536 | 180,0 | 1111,11 | 0,72 |
| 16 | 2048 | 150,0 | 1333,33 | 0,65 |
| 32 | 4096 | 90,0 | 2222,22 | 0,54 |
| 64 | 8192 | 55,0 | 3636,36 | 0,44 |

Example Plot 1



Example Table 2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Core-hour calculation | |  |  |  | Memory usage | Type of computation | Storage |
| Computational task | Number of such jobs | Wall clock time (in hours) per job | Number  of  nodes per job | Number  of cores per node per job | Total core-  hours per task | Estimate of memory usage (GiB) per node  per job | OpenMP / MPI / OpenMP + MPI  (hybrid) / etc. | volume (TB.hours) + number of files |
| Task   * software X * parameters/conditions * system/mesh size * … | A | B | C | D | = A x B x C x D |  |  |  |
| Task example CP2K   * CP2K – MD * 200 ns runs * PBE functional * 1 -> 20 molecules | 20 | 12 | 10 | 128 | 307.200 | 128 | MPI | 100 TB.hours  20.000 files |
| Summary |  |  |  |  | Sum of core-hours applied for = … |  |  | Sum of TB.hours + number of associated files at any given time  = … |

Example Table 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Core-hour calculation | |  |  |  | Memory usage | | Type of computation | Storage |
| Computational task | Number of such jobs | Wall clock time (in hours) per job | Number  of  nodes per job | Number  of GPUs per node per job | Total GPU-  hours per task | Estimate of CPU memory usage (GiB) per node  per job | Estimate of GPU memory usage (GiB) per GPU  per job | OpenMP / ... | volume (TB.hours) + number of files |
| Task   * software X * parameters/conditions * system/mesh size * … | A | B | C | D | = A x B x C x D |  |  |  |  |
| Task example Pytorch   * Pytorch * Batch size … * Fp32 training | 20 | 12 | 16 | 4 | 15.360 | 128 | 96 | RCCL (ROCm Communication Collectives Library) & OpenMP | 1.500 TiB.hours  30.000 files |
| Summary |  |  |  |  | Sum of GPU-hours applied for = … |  |  |  | Sum of TB.hours + number of associated files at any given time  = … |

Restrictions of LUMI

* LUMI provides the GNU compilers, Cray compilers (Clang-based C/C++ and Cray’s own Fortran) and AMD compilers (AOCC for the CPU nodes, ROCm for the GPU nodes). There is no support for Intel oneAPI or other versions of the Intel compilers and libraries and there is no guarantee that Intel MPI would work on the Cray SlingShot interconnect. Intel MKL is known to have performance and correctness problems on AMD in versions released since 2019. The preferred BLAS/LAPACK/ScaLAPACK libraries are the ones provided by HPE Cray. Users can install other libraries (and the support team provides instructions for some alternatives), but it implies that the user should take care of conflicts with already installed libraries and packages that are compiled for the Cray Scientific Library.
* The scheduler on the system is Slurm. No wrappers for compatibility with other schedulers are provided.
* Only HPE Cray’s own version of MPICH is fully supported as an MPI library. The interconnect in the GPU partition and final hardware in the CPU-based compute partition does not support UCX, only libfabric. Open MPI is currently not supported, though we do have build instructions for Open MPI on libfabric for the CPU nodes only with the GNU compilers (with no guarantee of optimal performance).
* Regular CPU partition:
  + Most nodes have 2 GB RAM per (physical) core (and 128 cores per node).
  + 128 nodes have 4 GB per core and 32 nodes have 8 GB per core. However, jobs cannot span more than 4 of those nodes and projects should not monopolise these resources.
  + There are some nodes with even more memory. These are not meant to be the main compute resource of your project but are meant for postprocessing and analysing the data.
* GPU partition
  + The GPU nodes consist of four AMD MI250X GPUs with 128 GB HBM2e memory and one 64 core CPU. Each MI205X GPU contains two dies (with each die connected to 64 GB of memory) and is shown as two GPUs to the user, so a single process that is not multi-GPU-aware can use only one eighth of the node.
  + The preferred programming models are AMD HIP and Open MP offload using the AMD ROCm or Cray compilers. It is not clear if or when the GNU compilers will fully support the GPU architecture of LUMI.
  + There is support for OpenACC in the Cray Fortran compiler only, not in other Fortran compilers or in C/C++. The Cray Fortran compiler is very strict when it comes to language compliance. Codes that use GNU extensions don’t work.
  + The degree of support for OpenCL is unclear (it is not a high priority for AMD). We currently have an experimental build of hipSYCL available, but it has not been widely tested, and it is still unclear if and when we will be able to offer support for Data Parallel C++ (via third party software).
  + As the system has AMD GPUs, it is not compatible with CUDA. CUDA code should be ported to HIP which is not a responsibility of the LUMI team.
  + There are 8 nodes with NVIDIA GPUs in the system. These are meant for visualization only and not as the compute resource of your project, and are in a partition that is not yet available to users (but may become during the course of a regular project granted in this call). CUDA is not installed on them currently.
* Storage: See <https://docs.lumi-supercomputer.eu/storage/>
  + On each partition, volume quota is flexible up to the maximum stated on <https://docs.lumi-supercomputer.eu/storage/>. The quota on the number of files is strict. Large supercomputers with large parallel file systems are not built to operate on millions of small files. Use data formats developed for large supercomputers instead, such as HDF5, netCDF, SIONlib, …
  + When computing your storage needs, take into account that the flash-based storage (project fast storage) is billed at ten times the rate of disk based storage. Hence, for using 1 TB of storage on the flash-based file system for one year you need to apply for 10 x 1 TB x 365 days x 24 hours/day TBhours of storage billing units.
  + Regular compute nodes and GPU nodes have no local storage.
  + Data retention policies will likely be enforced later in 2022.
  + *Note: The URL for the storage information in the LUMI documentation may change soon. The likely new URL is* [*https://docs.lumi-supercomputer.eu/ runjobs/lumi\_env/storing-data/*](https://docs.lumi-supercomputer.eu/%20runjobs/lumi_env/storing-data/)*.*
* Software:
  + There is limited support for Singularity containers. Note that containers requiring MPI should be able to work with Cray MPI which supports the MPICH ABI. As both the interconnect driver architecture and possibly the kernel module for intra-node communication is different from your typical Mellanox InfiniBand-based cluster, other MPI libraries will likely fall back to slower communication protocols (e.g., TCP/IP for internode communication). The same holds for containers with GPU software relying on the RCCL library. For optimal performance on LUMI, the RCCL library needs a plugin offering support for libfabric which is not yet in standard distributions. There is currently no support for building containers on LUMI, and support for other container runtimes is currently not planned either as they require features to be enabled in the Linux kernel for which there are currently too many severe security vulnerabilities.
  + The preferred use of the programming environment on LUMI is through the Cray compiler wrappers. These wrappers also offer support for MPI and easy linking of the right version of the Cray Scientific Library. Programs that expect the MPI wrappers to be called mpicc, mpif77 and so on will need to be patched, and there is no mpirun or mpiexec on the systems. Distributed memory programs are started through the Slurm process manager (srun command).
  + There is support for EasyBuild and limited support for Spack to install software on the system. Note that EasyBuild on Cray does not work with the regular common toolchains provided by the EasyBuild community though. Build recipes need to be adapted to work with the Cray Programming Environment-based toolchains.
  + Software requiring tens of thousands of small files such as Python and R should be installed in containers or be installed by a wrapper tool currently supporting conda and pip as this reduces the load on the file system and speeds up those packages in many scenarios.
* Availability
  + LUMI-C is available to users.
  + LUMI-G is expected to become available to regular users in November 2022.
  + The installation of the data analytics / visualization nodes is planned for late 2022 or 2023.
  + There is no date set yet for the availability of the Kubernetes partition so projects requested in this round should not rely on it. This partition is also mostly meant to provide services to the regular compute partitions, not to be the main compute resource of your project.
  + The availability date for the object storage is also not yet set.