Application form LUMI-BE (Belgian share of LUMI)

Instructions

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 or a EuroHPC regular or extreme access application on LUMI. Maximum 500 CPU.kH or 25 GPU.kH. Preparatory projects run for 4 months from the date of creation unless a motivated request for a longer period (at most one year) is given *at application time* (as we understand that true development can take more time). No extension is possible after the granting of the project.
* LUMI-BE Regular: standard application for large scale production runs. Maximum 10 CPU.MH or 500 GPU.kH. Regular projects run for 1 year from the date of project creation and cannot be extended.

If you want to apply for a project that uses both the CPU and GPU sections of LUMI, the sum of the number of CPU hours applied for divided by the maximum allowed for the project type and the number of GPU hours applied for divided by the maximum allowed for the project type should not be larger than 1. Or in other words, if you request x% of the maximum CPU hours for the project type, you can request at most (100-x)% of the maximum allowed GPU hours for the project type.

Applications will be evaluated by a panel of academic and technical staff. We aim to send the result to applicants one month after the application deadline.

# General remarks

* The LUMI-BE regular and preparatory tracks can only be used for open and publishable research.
* 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.
* LUMI is different from your regular HPC cluster. Rules and restrictions are also different on such machines than on a smaller local cluster. First check the LUMI documentation on [docs.lumi-supercomputer.eu](https://docs.lumi-supercomputer.eu/) to ensure that LUMI is suitable for the work you intend to do.
* LUMI is not a machine meant to learn the art of HPC. It is each applicant’s responsibility to ensure that all members of the project are sufficiently trained in working on HPC systems first and then as much as possible try to take one of the courses offered by the LUMI User Support Team and local BE team to learn what is different on LUMI, or at least carefully study the LUMI documentation at [docs.lumi-supercomputer.eu](https://docs.lumi-supercomputer.eu/).
* *LUMI is a new architecture. Expect 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.*

# Restrictions of LUMI

These restrictions are also mentioned in the LUMI documentation at [docs.lumi-supercomputer.eu](https://docs.lumi-supercomputer.eu/) but spread over various places. Without being exhaustive, we mention these restrictions in particular:

* The ***CPU section*** of LUMI, called LUMI-C consists mostly of AMD-based nodes with 128 cores and 256 GB of memory of which 212 GB is available to users. There are 128 nodes with 512 GB per node and 32 nodes with 1 TB per node, but a single job cannot use more than 4 of those nodes simultaneously.
Projects that need nodes with more than 256 GB of RAM for most of their computations will not be granted as LUMI does not offer a sufficient number of suitable resources for those applications.
* The ***GPU section*** of LUMI, called LUMI-G, is advertised as nodes with one 64-core processor and 4 MI250x GPUs. Due to the internal structure of the MI250x GPU, each GPU should effectively be treated as 2 separate GPUs for most applications and the node should be treated as one 64-core node with 8 GPUs with 64 GB of memory each.

Also note that one of the 64 cores is reserved for OS and device driver tasks, leaving only 63 cores available to the user.

The number of nodes available for sub-node allocation is also limited. LUMI is really meant for applications that can use full GPU nodes (though you may work around this limitation by organizing multiple jobs on the same node yourself, outside the control of the Slurm scheduler).

* The ***LUMI-D partition*** with 8 nodes for visualization (using NVIDIA GPUs) and 8 nodes with 4 TB of memory each for interactive data analytics work is not yet available (even though the documentation mentions some of these nodes already as the data analytics nodes were available temporary before a system upgrade), and it is not clear when they will become available.

Note that these NVIDIA GPUs are not meant for compute jobs!

Note that resources on the 4 TB nodes are limited so they should not be the only compute resource of your project but instead be used for data-intensive pre- and postprocessing steps.

* ***Shared storage:*** The LUMI file systems are optimized for working with relatively large files and show poor performance for very small files. Any job that opens and closes a lot of files in a short amount of time puts a high strain on the metadata servers of the parallel file system. Therefore, there are rather strict policies on the number of files a user can have in each file system. See the “[data storage options](https://docs.lumi-supercomputer.eu/runjobs/lumi_env/storing-data/)” page in the [LUMI documentation – Running jobs section](https://docs.lumi-supercomputer.eu/runjobs/).

Users with large conda and/or Python installations are asked to containerize these installations or use the “[Container wrapper](https://docs.lumi-supercomputer.eu/software/installing/container-wrapper/)” tool provided on LUMI that puts the installation in a single file which is then mounted in a minimal container and provides wrapper scripts for commands found in the bin subdirectory to ease the use without confronting users with singularity commands. Note that due to security concerns LUMI offers no support for building containers on LUMI itself.

The object storage system is not yet fully available or documented, but that support might be there by the time projects in this round starts.

* ***Local storage:*** There is no local storage on the regular compute nodes and GPU nodes.
* ***Scheduler***
	+ The scheduler on the system is Slurm. No wrappers for compatibility with other schedulers are provided.
	+ The limits on the maximum wall time for jobs are strict. No extensions are granted, not even for a single job. See also the [overview of Slurm partitions](https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/partitions/) in the [LUMI documentation – Running jobs section](https://docs.lumi-supercomputer.eu/runjobs/).
	+ The scheduler is meant to schedule jobs that are substantial in size and hence the number of jobs any user can have in the queue is limited. It is not meant to be used as a fine-grained scheduler in a capacity computing job as doing that on a system the size of LUMI would reduce the responsivity of the scheduler for other users. In those cases you need an additional workflow manager inside the job.
* ***MPI on LUMI:***
	+ The MPI implementation on LUMI is based on MPICH. We can currently not yet support Open MPI and that support will come in phases.
		- We expect to be able to support Open MPI later this year but that depends on updates of the scheduler to communicate properly with the Open MPI process starter. Older versions of Open MPI may work with the current scheduler setup.
		- It is unclear when GPU support can be offered. The network interface of LUMI does not support UCX (only OFI/libfabric) while Open MPI still requires UCX for GPU-enabled MPI.

Tests have shown that for many programs doing MPI transfers directly from the GPU rather than through the host is important for scalability.

* + There is no mpirun or mpiexec command on LUMI. Parallel jobs are started through the Slurm srun command.
* ***Compilers:*** LUMI supports the GNU compilers (though without support for offloading to GPU currently), Cray Compiling Environment (Clang/LLVM based C/C++ and Cray’s own Fortran compiler with an LLVM-based backend), and the AMD CPU (aocc) and GPU (rocm) compilers. These compilers are accessed through wrapper scripts that automatically add the necessary options for the math libraries and MPI. They are different from what you are used to on most clusters of the VSC and CÉCI. Only Lucia has a similar environment though with different compilers.

Note that the Intel compiler is not supported on LUMI. Install and use at your own risk.

Supported GPU programming models:

* + The preferred programming models are AMD HIP and OpenMP 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. It is possible to combine HIP with the GNU compilers though.
	+ 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. We are aware that there is work ongoing on OpenACC support in the Clang/LLVM community but neither Cray nor AMD have announced that they will support this so it is unclear if we will ever be able to support this on LUMI.
	+ 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 we have also done some very limited testing with the Data Parallel C++ of Intel (which is not the one delivered in binary form in the oneAPI installers as that compiler only supports the Intel GPUs, but a public domain version made available in source form by Intel).
	+ 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.
* ***Software:***
	+ Very little software is installed centrally on LUMI as this creates a maintenance nightmare on a cluster with a user base the size of LUMI, so take this into account when requestion storage.
	+ 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 (BLAS, LAPACK, ScaLAPACK). 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).
	+ The Cray Programming Environment also includes the following commonly used libraries: HDF5, netCDF, FFTW. These libraries are automatically linked by the Cray compiler wrapper when the ad hoc module is loaded.
	+ 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.
	+ See also the note above on the restrictions of the storage and applications that install tens of thousands of small files.
* ***Availability***
	+ LUMI-C is available to users.
	+ LUMI-G is available to users.
	+ The installation of LUMI-D, the data analytics / visualization nodes, is planned for 2023 but no date is set yet.
	+ The object storage is nearing availability, but no date is fixed.
	+ 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 partition will likely be cancelled.

# About CPU, GPU and storage billing on LUMI

On LUMI-C billing is done based on (physical) core hours consumed. However, on those nodes that support multiple jobs of multiple users running simultaneously (a subsection of the 256 GB nodes and all 512 GB and 1 TB nodes), memory is distributed proportional to the number of cores to ensure that all jobs get memory that is physically close to the cores to ensure predictable performance. If you need more memory per core than available on the node type, additional cores will be billed. See also the “[Billing policy](https://docs.lumi-supercomputer.eu/runjobs/lumi_env/billing/)” page in the [LUMI documentation – Run jobs section](https://docs.lumi-supercomputer.eu/runjobs/).

On LUMI-G usage is billed purely on the GPUs used. However, CPU memory and cores are also allocated proportionally to the number of GPUs requested, again as a proper mapping between GPU dies, CPU dies and memory is essential for good performance on nodes used by multiple users and jobs. 1 GPU hour corresponds to one hour of a full MI250x GPU, so basically 2 hours of use of what the scheduler reports as one GPU which is half of an MI250x package or 1 GPU compute die. Use of a full GPU node of LUMI-G costs 4 GPU hours per hour. Note that no CPU hours are billed when using LUMI-G.

On LUMI two ways are used to control storage use. LUMI uses file and block quota on all volumes, but storage use is also billed on all file systems except the home file system (but the latter is fixed size and no extension will ever be granted, nor can the directory be made readable to more people). Storage is not billed based on the quota but based on the actual use measures in Tbyte hours. Storing 1 TB for 1 hour will cost you 1 TB hour, as will storing 10 GB for 100 hours. On the flash file system however, storage is billed at 10 times this rate (also corresponding to the real cost per TB of flash storage versus disk-based storage), and on the object file system it will be billed at half the rate when that system becomes available.

Do not be shy to ask a reasonable amount of storage billing units for your projects. It is perfectly normal that you need a certain amount of storage for the whole project, e.g., for the software installation or input data used for the whole project. We also fully understand that it is not possible to transfer all output immediately after the job to storage at your home institution. However, LUMI is not meant as a system for long-term data storage. The Belgian storage allocation on LUMI is large enough that any reasonable and well-motivated request can be granted.

Note that jobs cannot run if you run out of storage billing units in your project.

# Remarks on the summary page

The ultra-short description here is not the same as the short description asked in question 2 for the regular projects and is mandatory for all project times. It is used at the time of project creation, as LUMI-BE has to provide a very short (on the order of 300 characters) description of each project.

OECD Field-of-Science: Use the categories from the table below (a slight variant of the [2007 classification](https://www.oecd.org/science/inno/38235147.pdf)):

|  |  |  |
| --- | --- | --- |
| 1.1 Mathematics1.2 Computer and information sciences1.3 Physical sciences1.4 Chemical sciences1.5 Earth and related environmental sciences1.6 Biological sciences1.7 Other natural sciences | 2.1 Civil engineering2.2 Electrical engineering, electronic engineering, information engineering2.3 Mechanical engineering2.4 Chemical engineering2.5 Materials engineering2.6 Medical engineering2.7 Environmental engineering2.8 Systems engineering2.9 Environmental Biotechnology2.10 Industrial Biotechnology2.11 Nano-technology2.12 Other engineering and technologies | 3.1 Basic medicine3.2 Clinical medicine3.3 Health sciences3.4 Health biotechnology3.5 Other medical sciences |
| 4.1 Agriculture, forestry, and fisheries4.2 Animal and dairy science4.3 Veterinary science4.4 Agricultural biotechnology4.5 Other agricultural sciences | 5.1 Psychology5.2 Economics and business5.3 Educational sciences5.4 Sociology5.5 Law5.6 Political Science5.7 Social and economic geography5.8 Media and communications5.9 Other social sciences | 6.1 History and archaeology6.2 Languages and literature6.3 Philosophy, ethics and religion6.4 Art (arts, history of arts, performing arts, music)6.5 Other humanities |

Please ensure that the requested resources mentioned here correspond with the motivation and plan in question 4.

The list of simulation codes is important as the technical panel does some effort to detect compatibility problems with LUMI in advance.

The applicant of the project will automatically receive a user id on LUMI. Please mention the other members also as this smoothens the process of creating the project. However, only mention people who will effectively compute on LUMI and access the project portal as any open account is a risk and as inactive accounts are blocked for security reasons and can pose problems at later project creations. So do not add your supervisor or department head if they don’t plan to access LUMI.

# Detailed questions

The questions follow the same order for both the regular and preparatory access application forms which is why some questions are missing. Some questions have a different form for the regular and preparatory access forms.

*Question 1:* Unlike EuroHPC we do not do a thorough scientific review of projects, but we do expect such review has been done by another source.

*Question 2:* No further remarks

*Question 3:* For regular projects, this question is mostly about the readiness and suitability for LUMI of the applications that will be used. In this question you also show that the application will use LUMI resources efficiently. You can copy the template table 1 from the appendix in this document. For preparatory projects it is obvious that this information is not available yet as one of the possible goals of such a project is to generate that data for other applications, so information more in line with the preparatory nature is requested.

*Question 4:* This question is used to judge if the requested resources are adequate and reasonable, and if you take into account the particular weaknesses of LUMI. For a preparatory project it is not possible to give this information in full detail, but we ask you to give a reasonable and motivated estimate based on current experience.

Question 5: Only for preparatory projects: Motivate why you need a longer duration allocation than just the standard duration. Note that this is perfectly acceptable for projects that involve a lot of development as it is clear that this cannot be done in 4 months.

# Appendix

Example Table 1 (question 3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 (question 3)



Example Table 2 (question 4, for CPU-only jobs)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 (question 4, for GPU jobs)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 TB.hours 30.000 files  |
|  Summary |   |   |   |   | Sum of GPU-hours applied for = …  |   |  |   | Sum of TB.hours + number of associated files at any given time= … |