SCC25 and IndySCC25
Boosting Earth System Model Outputs And Saving PetaBytes in Their Storage Using Exascale Climate Emulators

November 18, 2025

1 Task 1: Preprocessing (10 points)

As the first step in building the Climate Emulator, you will have five years of global surface temperature data in `NetCDF` format. The objective of this task is to remove the mean trend from the data and compute accuracy metrics. This task focuses only on **accuracy** (no performance scores). All you need to report is the generated files.

Data and Software Setup

1. Download and install `ExaGeoStatCPP` by cloning the repository from: `https://github.com/ecrc/exageostatcpp` or (for SCC teams) your fileserver.

   ./configure -e -m –climate-emulator

   ./clean_build.sh

2. You will need the input data from `2017-2021.tar`, available on the shared volume (IndySCC) or your fileserver (SCC).

3. The code will run on a single CPU node (**no GPU required**).

4. Run the `Example_Mean_Trend_Removal` program using the following command:

```
./examples/mean-trend-removal/Example_Mean_Trend_Removal \
--kernel=trend_model  \
--data-path=/home/netcdf_files/  \
--forcing-data-path=/home/forcing_new.csv \
--lb=0.001 \
--ub=0.95  \
--starting-theta=0.9  \
--cores=20  \
--gpus=0   \
--dts=200   \
--resultspath=/home/outcpp/   \
--startyear=2017   \
--endyear=2021   \
--lon=1440   \
--lat=200 \
--max-mle-iterations=30   \
--tolerance=7 \
--mean-trend-removal \
--log
```

`forcing_new.csv` is available in the main directory of the ExaGeoStatCPP repository. You should change the values of `--data-path=`, `--forcing-data-path=`, and `--results-path=` to match the paths of your input data and output directory. Note that `--numlocs=1440` has been modified compared to the practice example. You can now use `--lon=1440` and `--lat=200` (corresponding to 1440 longitudes and 200 latitudes) as shown above.

Expected Output

- The code will generate new measurements after removing the mean trend in the `/home/outcpp/` directory.
- Each output file corresponds to a single hour of data. For the given five `NetCDF` files (2017-2021.tar) (years 2017, 2018, 2019, 2020, and 2021), the total expected number of files is:

$$5 \times 365 \times 24 = 43{,}800$$

- Inside the `/home/outcpp/` directory, a file named `params.csv` will also be generated. This file contains 25 columns and 1440 rows, representing a subset of the grid with 1440 longitudes and 721 latitudes. The data is ordered starting from longitude 0, latitude $0, 1, 2, \ldots, 721$, then longitude 1, latitude $0, 1, 2, \ldots, 721$, and so on. The columns correspond to the spatial parameters in Equation (2) of the paper:
`https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10793148`

$$\rho,\ \sigma^2,\ \beta_0,\ \beta_1,\ \beta_2,\ a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}\ b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}$$

Reporting Requirements

Each team must submit:

1. The generated output consists of 43,801 files representing different realizations of Z, along with a parameter file (`params.csv`).
2. Submit your `params.csv` file and the ten `Z` files, namely `Z_10.csv`, `Z_20.csv`, ..., `Z_100.csv` to the submission server under `ECE/Task1/`.

2 Task 2: Pipelines (10 points)

This task includes running more stages of the Exacale Climate Emulator pipelines.

2.1 Setup

- Download the climate emulator code by cloning `https://github.com/ecrc/hicma-x`, or for SCC teams, a clone of the hicma-x repo is available on your fileserver). (this is the same code you already build from the `climate_emulator` branch of this repo. SCC teams, if rebuilding, are advised to use the hicma-x repo on your fileserver, as the `git submodule update` step has been run there)
- Follow `README.md` and `BUILD.md` to build the project.

Note: Skip this step if you have already done this.

2.2 Data

The input data needed is the same data as you used for practice task 2. They are available at:

- `ece-task2-data.tgz` on the shared volume for IndySCC teams (see README.md in this repo),
- `ece-task2-data.tgz` on your file server for SCC teams (see README.md of this repo)
- Indy teams can alternatively download it from the Globus endpoint (described in the README.md) or from This Google Drive Link.

2.3 Command

From the `build` directory, execute the following command if using OpenMPI.

```
mpirun -np nb_procs --npernode 1 ./tests/testing_climate_emulator  \
--latitude 256 \
--NB 2048 \
--N 65536 \
--gpus 8 \
--verbose 2 \
--mesh_file /home/qcao3/data \
--adaptive_decision 1 \
--adaptive_memory 0 \
--adddiag 1.0e2 \
--time_slots 100
```

Update these arguments as needed:

- Set `--mesh_file` to point to your data directory;
- Adjust `nb_procs` to the desired number of processes;
- Specify `--npernode 1` to run one process per node; and
- Modify `--gpus 8` if there are more than eight GPUs available per node.

The option `--adddiag 1.0e2` is used to prevent non-positive-definite issue due to the identical data for all time slots here. This value should NOT be changed in your tests. Refer to Section 3.6 or `./tests/testing_climate_emulator --help` for more details.

2.4 Expected Submission

Submit the raw program outputs to the submission server under `ECE/Task2/`.

3 Task 3: Performance

This task evaluates system performance by executing the Cholesky factorization under various precision configurations.

3.1 Setup

- Download the climate emulator code by either switching to the `climate_emulator` branch of this repo, or cloning `https://github.com/ecrc/hicma-x`. (for SCC teams, a clone of the hicma-x repo is available on your fileserver).
- Follow `README.md` and `BUILD.md` to build the project.

Note: Skip this step if you have already done this.

Note: For all raw data in Task 3.2, Task 3.3, and Task 3.4, (1) use `--verbose 2` for the raw data of the configuration that attains the highest performance; (2) name output files differently for each task.

3.2 Task 3.1: Benchmarking the Practical Peak Performance (15 points)

Run benchmarks to estimate the practical single-GPU performance using `./tests/testing_gemm_gpu` or `./tests/testing_gemm_gpu_hip`. Evaluate FP64, FP32, and one FP16 variants: `A16B16C32OP32`. Do not consider the data movements and datatype conversion. Take FP64 for example:

```
[ICL:methane build]$ ./tests/testing_gemm_gpu 8192 1
GEMM: FP64 8192 8192 8192 1 0 0 0 : 15.535932 TFLOPS :
16780176.1595091745257378 16780176.1595091745257378 0
GEMM: FP64 8192 8192 8192 1 0 0 0 : 15.816900 TFLOPS :
16780176.1595091745257378 16780176.1595091745257378 0
```

Expected Submission. For each precision, sweep square matrix sizes until the performance reaches a plateau. For each precision variant, submit a performance plot/table over varying matrix sizes, with the x-axis as matrix size and the y-axis as achieved performance (TFLOP/s).

You should submit the plot/tables to the submission server under `ECE/Task3.1/`.

3.3 Task 3.2: FP64 (15 points)

Example Command for 1 process:

```
./tests/testing_potrf_tlr \
--N 40960 \
--NB 2048 \
--verbose 2 \
--adaptive_memory 1 \
--kind_of_cholesky 5 \
--gpus 1 \
--band_dense_dp 1000000
```

`--band_dense_dp` specifies how many matrix bands are stored in FP64 (double precision). If its value is greater than or equal to the number of tiles/blocks along one matrix dimension, the entire matrix is treated as FP64. Use `--verbose 10` to print the matrix/tile precision information.

Expected Submission. Report the performance results for both a single GPU and the entire system. For each configuration: (1) A performance plot/table over varying matrix sizes, with the x-axis as matrix size and the y-axis as achieved performance (TFLOP/s); report only results for `hicma_parsec_cholesky`. (2) The raw program output corresponding to the configuration that attains the highest performance.

You should submit the plot/tables and program output to the submission server under `ECE/Task3.2/`.

3.4 Task 3.3: Mixing FP64 and FP32 (15 points)

Example Command for 1 process:

```
./tests/testing_potrf_tlr \
--N 40960 \
--NB 2048 \
--verbose 2 \
--adaptive_memory 1 \
--kind_of_cholesky 5 \
--gpus 1 \
--band_dense_dp 1 \
--band_dense_sp 1000000
```

`--band_dense_sp` sets the number of additional matrix bands stored in FP32 (single precision) *after* those counted by `--band_dense_dp`. If its value is greater than or equal to the number of tiles/blocks along one matrix dimension, all remaining (non-DP) tiles/blocks are stored in FP32. In the example command, the diagonal tiles/blocks are in FP64, while all others are in FP32. Use `--verbose 10` to print matrix/tile precision details.

Expected Submission. Report the performance results for both a single GPU and the entire system. For each configuration: (1) A performance plot/table across varying matrix sizes, with the x-axis as matrix size and the y-axis as achieved performance (TFLOP/s); report only results for `hicma_parsec_cholesky`. (2) The raw program output corresponding to the configuration that attains the highest performance.

You should submit the plot/tables and best-configuration program output to the submission server under `ECE/Task3.3/`.

3.5 Task 3.4: Mixing FP64 and FP16 (A16B16C32OP32) (15 points)

Example Command for 1 process:

```
./tests/testing_potrf_tlr \
--N 40960 \
--NB 2048 \
--verbose 2 \
--adaptive_memory 1 \
--kind_of_cholesky 5 \
--gpus 1 \
--band_dense_dp 1 \
--band_dense_sp 1 \
--band_dense_hp 1000000 \
--tensor_gemm 11
```

`--band_dense_hp` specifies the number of additional matrix bands stored in FP16 (half precision) *after* those defined by `--band_dense_sp`. If its value is greater than or equal to the number of tiles/blocks along one matrix dimension, all remaining (non-DP/SP) tiles/blocks are represented in FP16. In the given example, the diagonal tiles/blocks are stored in FP64, while all others use FP16 precision. Use `--verbose 10` to display detailed matrix/tile precision information.

Expected Submission. Report the performance results for both a single GPU and the entire system. For each configuration: (1) A performance plot/table across varying matrix sizes, with the x-axis as matrix size and the y-axis as achieved performance (TFLOP/s); report only results for `hicma_parsec_cholesky`. (2) The raw program output corresponding to the configuration that attains the highest performance.

You should submit the plot/tables and best-configuration program output to the submission server under `ECE/Task3.4/`.

3.6 Parameter Explanations

- `--N`: Matrix size. Performance typically increases with `N` until reaching a plateau.
- `--NB`: Tile/block size (tunable parameter); $N\%NB == 0$.
- `--gpus`: Number of GPUs to use per node. If the requested value exceeds the available GPUs on a node, it is capped at the node's maximum. Use `--verbose 2` to display detailed GPU statistics.
- `--adaptive_memory`: Controls memory allocation strategy: either allocate the entire matrix in FP64 or allocate each tile in its target precision. This affects the maximum matrix size that can be executed and may also move the memory register cost out of each kernel.
- `--datatype_convert`: Conversion policy. `0`: receiver converts; `2`: auto strategy (sender or receiver converts). See details at Reducing Data Motion and Energy Consumption.
- `--lookahead`: Controls the execution flow (degree of pipeline depth). See details at Extreme-Scale Task-Based Cholesky Factorization.
- `--band_dist`: Chooses between a two-dimensional block-cyclic distribution and a hybrid band distribution. See details at Extreme-Scale Task-Based Cholesky Factorization.
- `--tensor_gemm`: Bitmask of GEMM compute modes: `0x1 = FP64`, `0x2 = FP32`, `0x4 = TF32`, `0x8 = TF16 A16_B16_C32_OP32`, `0x10 = TF16 A16_B16_C16_OP16`.

3.7 Performance Tips

- **Core configuration:** Set `--cores` to `number_of_cores - 1`, reserving one dedicated core for communication.
- **Processes per node:** For best performance, use one process per node or NUMA node. If you run one process per NUMA node or per GPU, additional configuration (e.g., via Slurm) is required; see examples on Frontier supercomputer in `TESTS.md` via Slurm or tips below.
- **Process-level load balancing:** PaRSEC does not automatically rebalance work across processes at runtime. Tile/block placement on each node is determined by the data descriptor (see this example). Therefore, there may be a load imbalance issue for some SCC teams if each node's computational capacity is not the same. (Note this will not affect IndySCC teams, as your hardware is all the same)

  One solution is providing a custom distribution as described in this paper: A Framework to Exploit Data Sparsity. `descDist` examples: potrf_L_sparse_tlr_dp_balance.jdf and hicma_parsec_sparse_analysis.c.

  Another easier solution is to use one GPU per process. You may use `CUDA_VISIBLE_DEVICES`. Or you can set a process-specific `cuda_mask` in PaRSEC. In current PaRSEC builds, the MCA parameter `--device_cuda_mask` is global (unified) across all processes. As a workaround, you can modify the PaRSEC source to accept a per-process `cuda_mask` (e.g., here). If you take this route, set `--gpus` to the maximum number of GPUs across all nodes. Note that, unlike system-level configurations (e.g., Slurm), this approach will cause CPU cores on a node to be shared among all processes in this node. Of course, you can also provide the CPU core binding and communication thread binding by the mca parameters or by modifying the PaRSEC source code.

SCC teams may, if they choose, modify the source code according to the above tips. If you do this, you must submit the diff with your changes, along with your results.

- **GPU load balancing within one process:** PaRSEC can automatically balance work across GPUs within one process. However, users may also specify explicit tile/block placement per GPU to improve performance. The current strategy is implemented in gpu_load_balance, with alternative strategies in this file. For further details, see `https://www.qingleicao.com/papers/wamta25_main.pdf`.