Getting Started with cuMAC

Table of contents

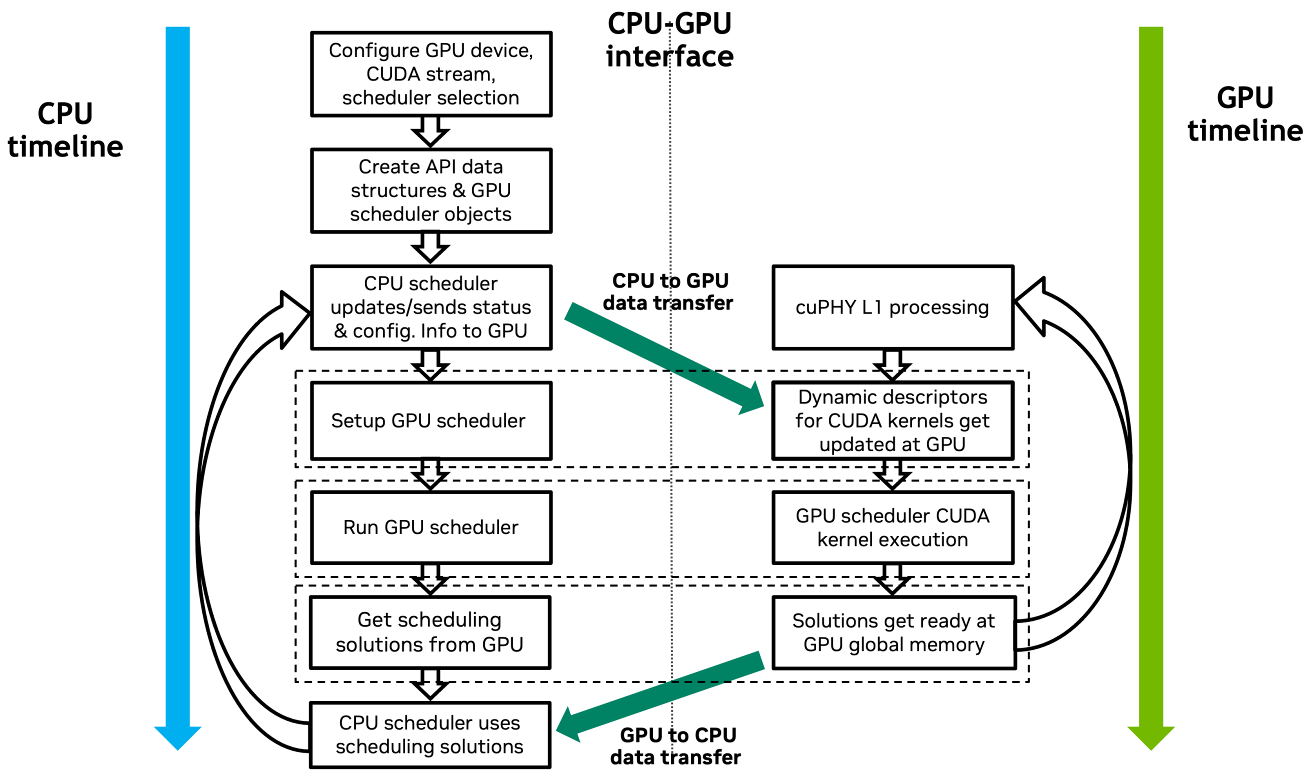
List of Figures

All cuMAC data structures and scheduler module classes are included in the name space cumac

The header files api.h and cumac.h should be included in the application program of cuMAC

Data Flow

A diagram of cuMAC data flow for both CPU MAC scheduler host and GPU execution is given in follwoing figure:


CPU-GPU interface
CPU timeline
GPU timeline
Configure GPU device, CUDA stream, scheduler selection
Create API data structures & GPU scheduler objects
CPU scheduler updates/sends status & config. Info to GPU
CPU to GPU data transfer
cuPHY L1 processing
Setup GPU scheduler
Dynamic descriptors for CUDA kernels get updated at GPU
Run GPU scheduler
GPU scheduler CUDA kernel execution
Get scheduling solutions from GPU
Solutions get ready at GPU global memory
CPU scheduler uses scheduling solutions
GPU to CPU data transfer

cuMAC multi-cell scheduler execution data flow

Each cuMAC scheduler module (UE selection, PRB allocation, layer selection, MCS selection, etc.) is implemented as a C++ class, consisting of constructors with different combinations of input arguments, a destructor, a setup () function to set up the CUDA kernels in each TTI and a run () function to execute the scheduling algorithms in each TTI.

All parameters and data buffers required by the cuMAC scheduler modules are wrapped into three cuMAC API data structures, including *cumacCellGrpUeStatus*, *cumacCellGrpPrms*, and *cumacSchdSol*. Each of these data structures contains a number of constant parameters, and a number of data buffers whose memories are allocated on GPU.

In the initialization phase, the objects of all cuMAC scheduler modules are created using their corresponding constructors. Meanwhile, the above-mentioned three API data structures are also created, with their constant parameters being properly set up and data buffers getting memory allocations on GPU.

In the per-TTI execution, the CPU MAC scheduler host first prepares all the required data in GPU memory for the three API data structures. Then the setup () function of each cuMAC scheduler module is called 1) to pass the required constant parameters and addresses of the data buffer GPU memories from the API data structures to the scheduler module objects, and 2) to complete the internal configuration of the CUDA kernels. Next, the run () function of each schedule module is called to execute the scheduling algorithms and obtain the scheduling solutions. Finally, the CPU MAC host transfers the computed scheduling solutions from GPU to CPU and applies them in the system.

Quick Setup

Prerequisites

1. CMake (version 3.18 or newer)

   If you have a version of CMake installed, the version number can be determined as follows:

   ```
   cmake --version
   ```

   You can download the latest version of CMake from the official CMake website.

2. CUDA (version 12 or newer)

   CMake intrinsic CUDA support will automatically detect a CUDA installation using a CUDA compiler (nvcc), which is located via the PATH environment variable. To check for nvcc in your PATH:

```
which nvcc
```

To use a non-standard CUDA installation path (or to use a specific version of CUDA):

```
export PATH=/usr/local/cuda-12.0/bin:$PATH
```

For more information on CUDA support in CMake, see https://devblogs.nvidia.com/building-cuda-applications-cmake/. (The statement above is equivalent to ” -gencode arch=compute_80,code=sm_80 -gencode arch=compute_90,code=sm_90 “.)

3. cuMAC requires a minimum GPU architecture of Ampere or newer.
4. HDF5 (Hierarchical Data Format 5)

   The cuMAC CMake system currently checks for a specific version (1.10) of HDF5. To install a specific version of HDF5 from a source code archive:

   4.1. Remove the original hdf5 library (if necessary)

   ```
   dpkg -l \| grep hdf5
   ```

   ```
   sudo apt-get remove &lt;name of these libraries&gt;
   ```

   4.2. To build from source:

   ```
   wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.5/src/hdf5-1.10.5.tar.gz
   ```

   ```
   tar -xzf hdf5-1.10.5.tar.gz
   ```

   ```
   cd hdf5-1.10.5
   ```

```
./configure --prefix=/usr/local --enable-cxx --enable-build-mode=production
```

```
sudo make install
```

Getting and building cuMAC

1. To download cuMAC, you can use the following link:

```
git clone --recurse-submodules https://gitlab-master.nvidia.com/gputelecom/cumac
```

2. To build cuMAC, use the following commands:

```
cd cumac
```

```
mkdir build && cd build
```

```
cmake ..
```

```
make
```

Additional CMake options:

Creating a release build (using the default list of target architectures):

```
cmake -DCMAKE_BUILD_TYPE=Release ..
```

Creating a debug build (using the default list of target architectures):

```
cmake .. -DCMAKE_BUILD_TYPE=Debug
```

Specifying a single GPU architecture (e.g., to reduce compile time):

```
cmake .. -DCMAKE_CUDA_ARCHITECTURES="80"
```

Specifying multiple GPU architectures:

```
cmake .. -DCMAKE_CUDA_ARCHITECTURES="80;90"
```

(The statement above is equivalent to ” -gencode arch=compute_80,code=sm_80 -gencode arch=compute_90,code=sm_90 “.)