BME 445-513

Educational Software

#CUDA Parallelization in MATLAB

ILLINOIS TECH

ILLINOIS INSTITUTE OF TECHNOLOGY

Ignacio Hidalgo Power

GPU Parallelization in MATLAB

MATLAB supports GPU acceleration using NVIDIA graphics cards. This can be achieved either through high-level parallelization using special data types like gpuArray, or through low-level GPU programming using custom CUDA kernels via MEX interfaces.

In this software, we leverage low-level GPU parallelization using NVIDIA CUDA. This requires a CUDA compatible hardware and the installation of the NVIDIA CUDA toolkit [https://developer.nvidia.com/cuda-toolkit]

When do we use GPU parallelization in MATLAB

- **Large computations:** When we need to execute this in parallel
- **Matrix-heavy or vectorized algorithms:** Such as FFT, matrix multiplication, or numerical solvers
- **Repetitive, element-wise operations:** This can be distributed across thousands of GPU cores

Voltage Experiment GPU Parallelization with CUDA

We have made the GPU module for the Voltage experiment. This is called 'voltage_kernel.cu'. To measure the performance of the code we have made a temporary backend module that uses both methods (GPU and CPU) and shows the running time.The structure of this cuda code is:

Device functions: To substitute the support functions. These are here so that the Kernel can calculate them since it can not call for the Matlab functions declared in the 'Support_Function' Module.

```
__device__ double alpha_m_d(double V) {
    if (fabs(25.0 - V) < 1e-9) {
        return 1.0;
    }
    return 0.1 * (25.0 - V) / (exp((25.0 - V) / 10.0) - 1.0);
}

__device__ double beta_m_d(double V) {
    return 4.0 * exp(-V / 18.0);
}

__device__ double alpha_h_d(double V) {
    return 0.07 * exp(-V / 20.0);
}

__device__ double beta_h_d(double V) {
    return 1.0 / (exp((30.0 - V) / 10.0) + 1.0);
}
```

Kernel Code: This is the code that substitutes the for loop to execute it in the GPU through the use of CUDA.

```
__global__ void voltageBE_kernel(
    int num_time_steps,
    double DT,
    int tdel1_idx,
    int tdel2_idx,
    double V_REST,
    double G_NA,
    double G_K,
    double E_K,
    double E_NA,
    double V_clamp1,
```

```
// Determine the voltage at the current time step based on indices
if (i <= tdel1_idx) {
    V = V_REST;
} else if (i >= tdel2_idx) {
    V = V_clamp1;
} else {
    V = V_clamp0;
}

// Calculate steady-state values and time constants for the current voltage V
double m_inf = alpha_m_d(V) / (alpha_m_d(V) + beta_m_d(V));
double h_inf = alpha_h_d(V) / (alpha_h_d(V) + beta_h_d(V));
double n_inf = alpha_n_d(V) / (alpha_n_d(V) + beta_n_d(V));

double tau_m = 1.0 / (alpha_m_d(V) + beta_m_d(V));
double tau_h = 1.0 / (alpha_h_d(V) + beta_h_d(V));
double tau_n = 1.0 / (alpha_n_d(V) + beta_n_d(V));
```

Mex wrapper: This is the part of the code that wraps the CUDA code into a usable Matlab function. It calculates the initial variables and prepares the resource allocation for the code in the GPU (Block size, grid, etc..). It also handles the memory interaction between the GPU and the Matlab program.

```
// MEX function entry point
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    // --- Input Parsing ---
    // Expected inputs:
    // prhs[0]: V_clamp1 (double)
    // prhs[1]: Nae (double)
    // prhs[2]: tdel1 (double)
    // prhs[3]: tend (double)
    // prhs[4]: use_modified_flag (double, 0 or 1)
    // prhs[5]: V_clamp0 (double, required if use_modified_flag is 1)
    // prhs[6]: tdel2 (double, required if use_modified_flag is 1)

    if (nrhs < 5 || nrhs > 7) {
        mexErrMsgTxt("Incorrect number of input arguments.");
    }
    if (nlhs < 5 || nlhs > 6) {
        mexErrMsgTxt("Incorrect number of output arguments.");
    }
```

To use it we need to run the command ‘mexcuda -v voltage_kernel.cu NVCC_FLAGS="-allow-unsupported-compiler"’ in the Matlab terminal, in the location where the .cu file is located.

```
>> cd GPU_Modules\
>> mexcuda -v voltage_kernel.cu NVCC_FLAGS="-allow-unsupported-compiler"
Trying MEX options 'C:\Program Files\MATLAB\R2024b\toolbox\parallel\gpu\extern\src\mex\win64\nvcc_msvcpp2022.xml'...SUCCESS
mex -largeArrayDims -f C:\Program Files\MATLAB\R2024b\toolbox\parallel\gpu\extern\src\mex\win64\nvcc_msvcpp2022.xml NVCC_FLAGS="" -v voltage_kernel.cu NVCC_FLAGS="-allow-unsupporte
Verbose mode is on.
... Looking for compiler 'NVIDIA CUDA Compiler' ...
... Looking for environment variable 'ProgramFiles(x86)' ...Yes ('C:\Program Files (x86)').
... Looking for file 'C:\Program Files (x86)\Microsoft Visual Studio\Installer\vswhere.exe' ...Yes.
... Executing command '"C:\Program Files (x86)\Microsoft Visual Studio\Installer\vswhere.exe" -version "[17.0,18.0)" -products Microsoft.VisualStudio.Product.Enterprise -property i
... Looking for environment variable 'ProgramFiles(x86)' ...Yes ('C:\Program Files (x86)').
... Looking for file 'C:\Program Files (x86)\Microsoft Visual Studio\Installer\vswhere.exe' ...Yes.
... Executing command '"C:\Program Files (x86)\Microsoft Visual Studio\Installer\vswhere.exe" -version "[17.0,18.0)" -products Microsoft.VisualStudio.Product.Professional -property
... Looking for environment variable 'ProgramFiles(x86)' ...Yes ('C:\Program Files (x86)').
```

```
link /nologo  /DLL  /EXPORT:mexFunction /EXPORT:mexfilerequiredapiversion C:\Users\ihida\AppData\Local\Temp\mex_193691924974148_25300\voltage_kernel.obj C:\Users\ihida\AppData\Loca
   Creating library voltage_kernel.lib and object voltage_kernel.exp

del "voltage_kernel.exp" "voltage_kernel.lib" "voltage_kernel.ilk"
MEX completed successfully.
```

To run and see the difference between the GPU run and non GPU run we just need to execute the program as normal. ‘VoltageGUI’ has already been modified to make use of ‘VoltageBEG.m’ which will run the code the like ‘VoltageBE.m’ would do but it also measures the computing time of each method

```
>> demo
Simulation Time using NON-CUDA 0.0081
Simulation Time using CUDA 0.0033
```

Resources

MATLAB GPU

https://www.mathworks.com/help/parallel-computing/gpu-computing-in-matlab.html

https://www.youtube.com/watch?v=F2z5iP9EcnI&t=147s

CUDA

https://docs.nvidia.com/cuda/cuda-c-programming-guide/

https://www.youtube.com/watch
v=GgpFMclkZE8&pp=ygURY3VkYSBoZWxsbyB3b3JsZCA%3D

General Concepts

https://learning.google.com/experiments/learn-about