GPU Enablement of MICM Chemistry Solver

Qina Tan, Jian Sun, John Dennis, Matthew Dawson
National Center for Atmospheric Research

1. Model-Independent Chemistry Module (MICM)

- Software to enable study of time-dependent behavior of atmospheric chemistry
- Being developed in C++ currently with about 2000 lines of codes and 96% testing coverage
- Computationally intensive yet capable of being executed concurrently in parallel
- Explored three GPU-accelerated strategies (via CUDA) & compared their performances

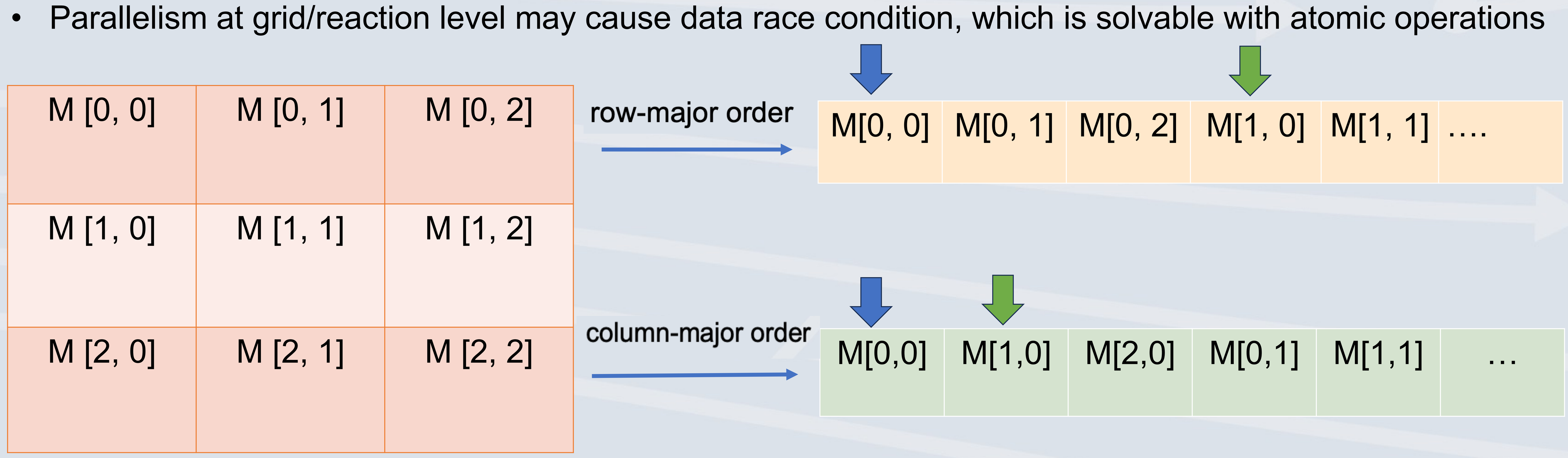
2. GPU Programming in CUDA

- GPUs enable parallel computation of large data sets with dense array of cores
- CUDA is a parallel computation platform and programming model created by NVIDIA
  - Requires manual configuration and launch of kernel function
  - Requires memory management between host (CPU) and device (GPU)

```
//allocate device memory
double d_pointer;
size_t d_bytes = sizeof(double)*200;
cudaMalloc(&d_point, d_bytes);
//transfer data from host memory to device memory
cudaMemcpy(d_pointer, h_pointer, d_bytes, cudaMemcpyHostToDevice);
```

3. Implementation: AddForcingTerms( )

- Computes the rate of change in atmospheric composition associated with rate constants and reactant concentrations of a set of chemical reactions occurs in the atmosphere
- Data are organized in matrix: rows as grid boxes in 3D climate model, columns as rate constants of reactions
- Matrix data are transformed into linear vector in C++ using row-major order and column-major order
- Stride memory access pattern in row-major order and contiguous memory access pattern in column-major order
- Parallelism at grid/reaction level may cause data race condition, which is solvable with atomic operations



Snippets for threads accessing data organized by **row-major order (left)** and **column-major order (right)**

```
//thread index
int tid_idx = blockIdx.x * blockDim.x + threadIdx.x;
//grid-parallel : loop through number of reactions in a mechanism
for (size_t i_rxn = 0; i_rxn < rxn_count; i_rxn++){
    double rate = rate_constants[i_rxn * tid_idx + row_count];
}
```

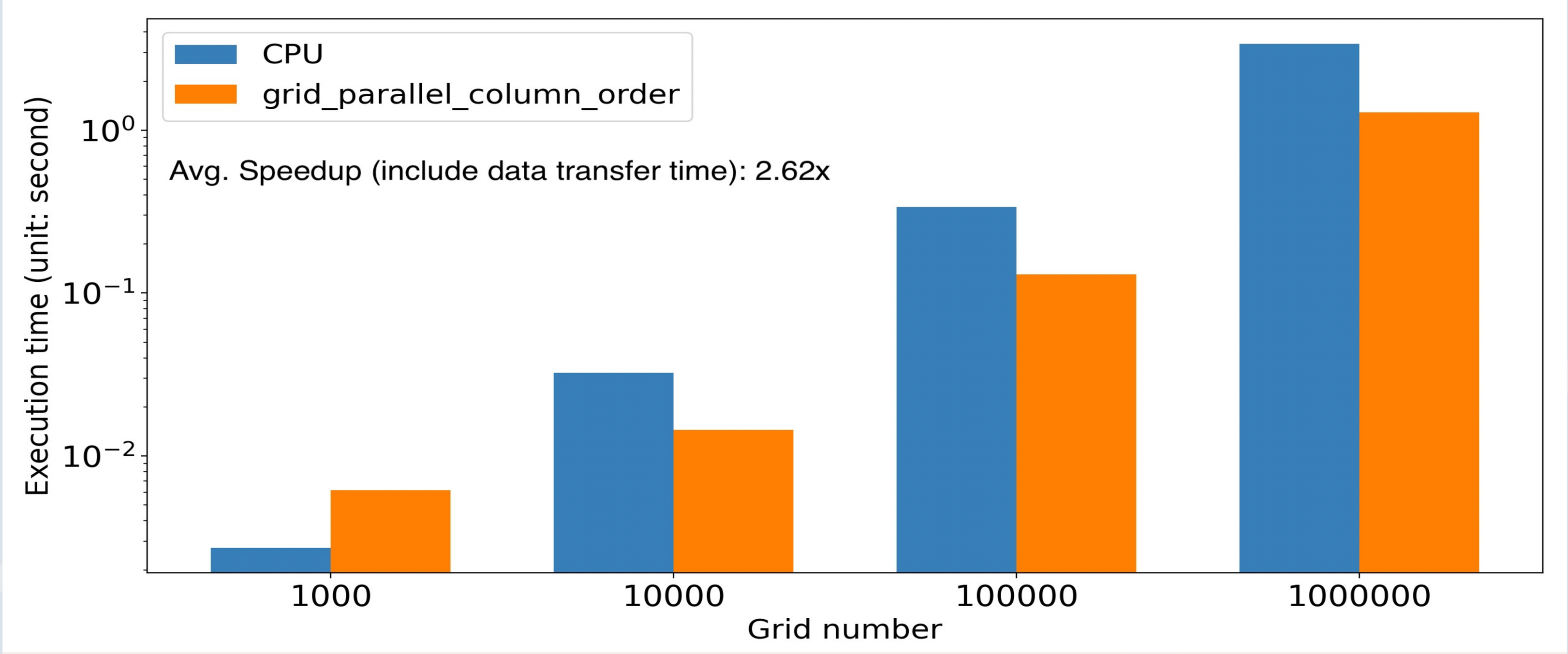
```
//thread index
int tid_idx = blockIdx.x * blockDim.x + threadIdx.x;
//grid-parallel : loop through number of reactions in a mechanism
for (size_t i_rxn = 0; i_rxn < rxn_count; i_rxn++){
    double rate = rate_constants[i_rxn * row_count + tid_idx];
}
```

4. Experiment

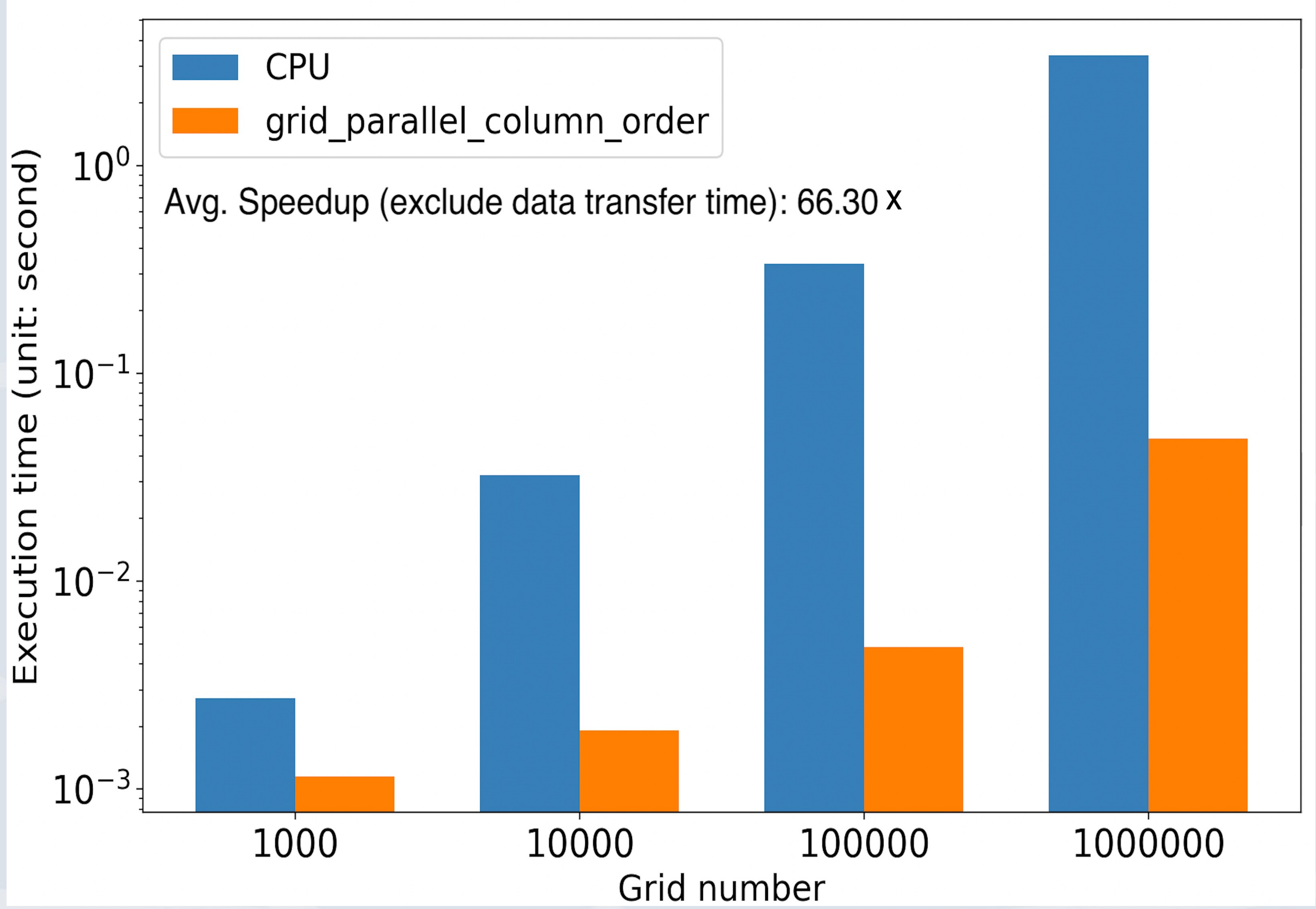
- Machine: Gust
- Compiler: nvhpc/23.5
- Bit for Bit Accuracy of CPU code against GPU code
- CPU Performance: 1 CPU
- GPU Performance: 1 NVDIA A100 GPU (w/ and w/o data transfer time)
- 3 CUDA implementation versions:
  - Parallelism at grid level with row-major order memory layout
  - Parallelism at grid level with column-major order memory layout
  - Parallelism at grid/reaction level with column-major order memory layout

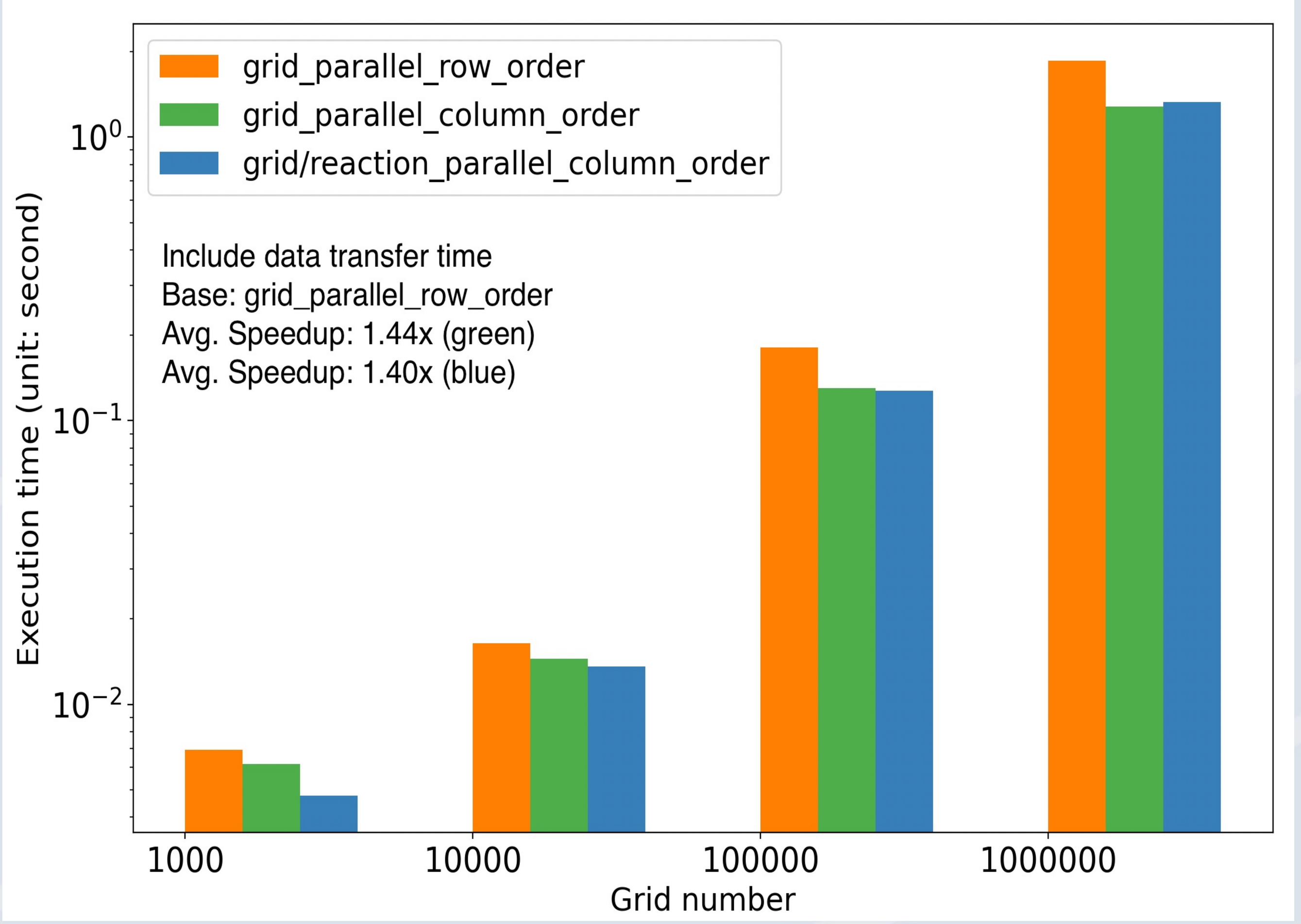
5. Time Performance

Performance comparisons between **CPU vs GPU (top left, bottom left)** and between **GPU-accelerated strategies (bottom right)**



- Constant input for all three graphs: 500 reactions, 400 chemical species





6. Conclusion & Future Work & Acknowledgment

- We ported AddForcingTerms( ) function to GPU via CUDA with different implementations
- Performance testing shows increasing speedups with increasing problem size
- Future works: port more function to GPU via similar approach
- My sincere gratitude to NCAR and SIParCS mentors Jian Sun, John Dennis and Matthew Dawson