ACC-BioFVM: Accelerating an open source Finite Volume Method for Biological Problems on GPUs

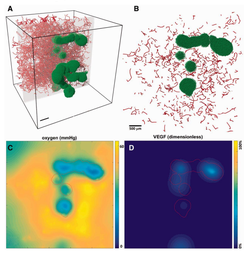
Matthew Stack, Dr Paul Macklin, Dr Sunita Chandrasekaran
Dept. of Computer and Information Sciences and Dept. of Chemistry, University of Delaware
https://crpl.cis.udel.edu/

Find the code on github!

MOTIVATION AND GOAL

MOTIVATION

- BioFVM efficiently models the systems of PDE's for multicellular diffusion, decay, uptake, and release of multiple substrates in 2D and 3D environments.
- BioFVM is parallelized by OpenMP for efficient transport solving on CPU, but could see efficiency gains running on GPUs.
- Scientific possibilities of accelerated transport solving include higher simulation throughput, finer resolution, higher accuracy, and new (previously infeasible) 3D problems.



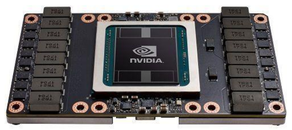
GOAL

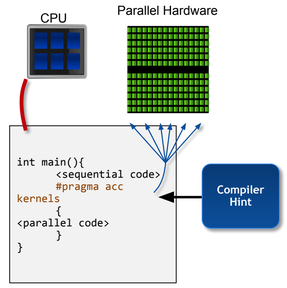
- **Create a GPU accelerated version of BioFVM using OpenACC capable of running on the range of multicore CPU to supercomputer**

USING OpenACC DIRECTIVES

- In order to maintain portability of many different hardware architectures, and to ease the development process of working on a pre-existing code, we decided to use OpenACC to parallelize the code
- OpenACC [3], a directive-based parallel programming model for accelerators

OpenACC
More Science, Less Programming

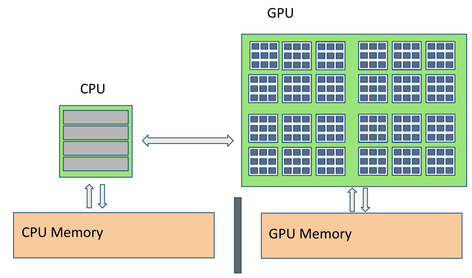




HANDLING DATA

- Data must be explicitly transfer between Host and Device.
- Vector of vectors **p_density_vectors** holds the values of the PDE's.
  - Translation: Converting data from CPU data type (vectors and 2D vectors) to and from GPU data type (arrays and 2D arrays) while maintaining accuracy
  - Transfer: Iterating through data, dereferencing the pointers and using OpenACC data migration pragmas to transfer

Translating and transferring requires communication between the Host (CPU) and the Device (GPU) and is necessary but extremely expensive so the goal is to minimize communication!



PROJECT ROADMAP

May

- BioFVM is tens of thousands lines of code, and for 2D problems the hot spot is x and y diffusion, each are a short but very complex segment of calculations
- Becoming even slightly familiar with the advanced Biology and PDE's took lots of time and help
- Tracking the all the data in the form of pointers, C++ class members, and others was fascinating to watch

$$\frac{\partial \vec{\rho}}{\partial t} = \overbrace{\vec{D}\nabla^2\vec{\rho}}^{\text{diffusion}} - \overbrace{\vec{\lambda}\vec{\rho}}^{\text{decay}} + \overbrace{\vec{S}(\vec{\rho}^* - \vec{\rho})}^{\text{bulk source}} - \overbrace{\vec{U}\vec{\rho}}^{\text{bulk uptake}}$$
$$+ \overbrace{\sum_{\text{cells}k} 1_k(\vec{x}) \left[\vec{S}_k(\vec{\rho}_k^* - \vec{\rho}) - \vec{U}_k\vec{\rho}\right]}^{\text{sources and uptake by cells}} \text{ in } \Omega$$

June

- Converting the gcc compiled code to an intelligent PGI compiler that allowed for profiling with PGPROF
- Maintain accuracy during transition between compilers while exploring the new features of PGI
- Discovered the errors that helped guide the work including the ambiguous "Illegal Address during Kernel Execution"
- Explored PGI feature Managed Memory which bridges the gap between Host and Device memory, but not used as it's not standard across compilers

July

- Our first major roadblock was with C++ STL Vectors
- This data structure obscures the data from us and makes it difficult to manage CPU-GPU data
- We replaced vectors with C-style arrays when possible, and in some cases we could simply use data() to get the underlying pointer for our OpenACC data constructs

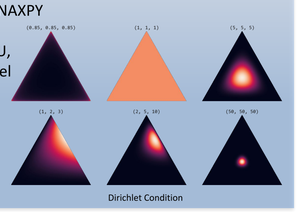
```
vector< vector<double> > cpu_density;
for (i < size) {
gpu_density[i] = cpu_density[i].data()
}
```

```
      #pragma acc enter data
create(this->temp_thomas_cy[0:bin_thomas_cy][0:0])
      for (int i = 0; i < bin_thomas_cy; i ++) {
            int sze = thomas_cy[i].size();
            temp_thomas_cy[i] = thomas_cy[i].data();
            #pragma acc enter data
copyin(this->temp_thomas_cy[i:1][:sze])
      }
```

- OpenACC is directed to "attach" the temp_thomas_cy 2D vector to the C++ object calling this function, allowing correct pointer manipulation in GPU address space
- C++ objects must be shaped and attached using OpenACC pragmas in manual copy implementations
- Very expensive operation in an implementation supposed to have dramatic speedups
- Minimizing the number of times this transfer is called is the key to having a good balance of communication and computation

August

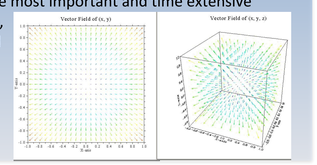
- Making correct copies of the rest of the object data to allow for more time computing on the GPU and less time updating between Host and Device versions of p_density_vector
- Constructing parallelized version of functions used in accelerated region, including AXPY and NAXPY
- Converting mesh dirichlet condition application to GPU, along with porting each voxel dirichlet data
- x-diffusion, y-diffusion, and dirichlet_conditions now running uninterrupted on GPU

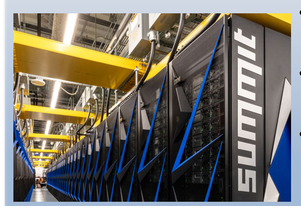


Dirichlet Condition

Future Work

- Moving from 2D -> 3D requires new functions to be ported to Device and introduce new z-diffusion on GPU
- A user will be able to control with ease whether BioFVM should be run parallelized with OpenACC on GPU, or for smaller domain problems/when there's no GPU available run on CPU with the existing OpenMP accelerated code
- Solving diffusion is the most important and time extensive piece of computation, but more background work done on the Device equates to less time wasted on communication





IBM's Summit massively parallel supercomputer

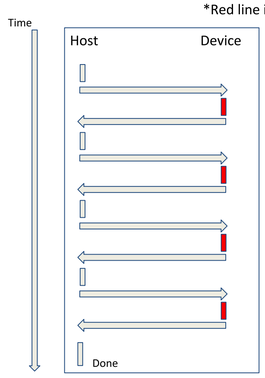
- Full computation on GPU
- One transfer of data to the Device at the beginning of the simulation, and one transfer back to Host at the end
- Full documentation with examples and instructions how to follow the data the user cannot see
- Once PGI Managed memory is a standard feature to all of OpenACC, reassessing the pros and cons of a Managed memory model (very simple, yet slower than manual)

Next Steps

Results

Current results are actually slightly slower than the CPU version! But this is completely expected and correct and here's why:
Until every part of the parallel region (main work loop) is available on GPU, there is slow down from communication between translating and updating the two copies of p_density_vector.

*Red line is the same size!




```
main (){
… init of environment…
transfer(); // all necessary data
while (i < time) {
simulate_bulk_sources_and_sinks(dt)
translate_vector_to_array()
simulate_diffusion_decay(dt)
translate_array_to_vector()
simulate_cell_sources_and_sinks(dt)
i += dt
}
}
```

***Bold**: GPU function

References & Acknowledgements

1. Ahmadreza Ghaffarizadeh, Samuel H. Friedman, Paul Macklin, BioFVM: an efficient, parallelized diffusive transport solver for 3-D biological simulations, Bioinformatics, Volume 32, Issue 8, 15 April 2016, Pages 1256–1258
2. The Portland Group, PGI Compilers and Tools User's Guide, www.pgroup.com April 2018.
3. https://www.openacc.org/

I would like to thank Mathew Colgrove from NVIDIA for his help and examples!