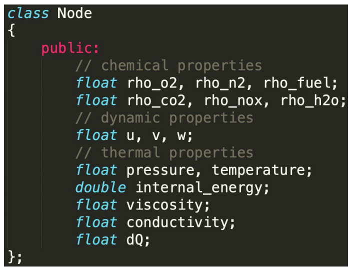
Fuel Ignition Simulation Final Report

Summary

We implemented a CFD model of flame ignition, combustion, and propagation using finite difference methods to approximate Navier-Stokes equations, both sequentially on the cluster CPUs and in parallel using CUDA on the NVIDIA 1080 GPUs in the Gates cluster computers. We also generated a visualizer for the output of the CFD using the GPU.

Background

Our project simulated a CFD model by maintaining a 3-D matrix of nodes, each of which maintained a list of physical properties which were updated with each time step according to finite difference method approximations of the Navier-Stokes equations. Each simulation step consisted of two stages: combustion and advection/conduction. During the combustion stage, each node checks whether it reaches the threshold energy to perform combustion of its fuel and if so adjusts its chemical composition according to combustion reaction and stores energy released during combustion for use in the advection/conduction stage. During this stage, there are no dependencies between the nodes, so it is fairly easy to parallelize as each node can be modified in place. However, this step accounts for only 3% of computation time in the sequential version, so the benefit from parallelizing it is severely limited by Amdahl's law. Its primary effect on runtime is as a synchronization barrier between advection/conduction steps.



```
class Node
{
    public:
        // chemical properties
        float rho_o2, rho_n2, rho_fuel;
        float rho_co2, rho_nox, rho_h2o;
        // dynamic properties
        float u, v, w;
        // thermal properties
        float pressure, temperature;
        double internal_energy;
        float viscosity;
        float conductivity;
        float dQ;
};
```

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho u) + \frac{\partial}{\partial y}(\rho v) + \frac{\partial}{\partial z}(\rho w) = 0$$

conservation of mass equation

$$\rho \frac{Du}{Dt} = \rho f_x - \frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\left[\frac{2}{3}\mu\left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} - \frac{\partial w}{\partial z}\right)\right] + \frac{\partial}{\partial y}\left[\mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] + \frac{\partial}{\partial z}\left[\mu\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)\right]$$

$$\rho \frac{Dv}{Dt} = \rho f_y - \frac{\partial p}{\partial y} + \frac{\partial}{\partial x}\left[\mu\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)\right] + \frac{\partial}{\partial y}\left[\frac{2}{3}\mu\left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x} - \frac{\partial w}{\partial z}\right)\right] + \frac{\partial}{\partial z}\left[\mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)\right]$$

$$\rho \frac{Dw}{Dt} = \rho f_z - \frac{\partial p}{\partial z} + \frac{\partial}{\partial x}\left[\mu\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)\right] + \frac{\partial}{\partial y}\left[\mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)\right] + \frac{\partial}{\partial z}\left[\frac{2}{3}\mu\left(2\frac{\partial w}{\partial z} - \frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right)\right]$$

conservation of momentum equation

$$\tau_{xx} = \frac{2}{3}\mu\left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} - \frac{\partial w}{\partial z}\right)$$

$$\tau_{yy} = \frac{2}{3}\mu\left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x} - \frac{\partial w}{\partial z}\right)$$

$$\tau_{zz} = \frac{2}{3}\mu\left(2\frac{\partial w}{\partial z} - \frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right)$$

$$\tau_{xy} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) = \tau_{yx}$$

$$\tau_{xz} = \mu\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right) = \tau_{zx}$$

$$\tau_{yz} = \mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) = \tau_{zy}$$

$$\frac{\partial E_t}{\partial t} - \frac{\partial Q}{\partial t} - \rho(f_x u + f_y v + f_z w) + \frac{\partial}{\partial x}(E_t u + pu - u\tau_{xx} - v\tau_{xy} - w\tau_{xz} + q_x)$$
$$+ \frac{\partial}{\partial y}(E_t v + pv - u\tau_{xy} - v\tau_{yy} - w\tau_{yz} + q_y)$$
$$+ \frac{\partial}{\partial z}(E_t w + pw - u\tau_{xz} - v\tau_{yz} - w\tau_{zz} + q_z) = 0$$

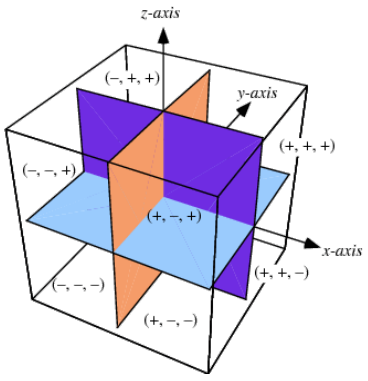
conservation of energy equation

During the advection/conduction stage, each node updates its properties based on the conservation of mass, momentum, and energy equations presented to the left. The ideal gas approximation was also used to solve these. To approximate the first and second partial derivatives of the node properties, we used the central difference formula as follows:

$$\frac{du_i}{dx} = \frac{u_{i+1} - u_{i-1}}{2\Delta x}, \qquad \frac{d^2u_i}{dx^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2},$$

$$\frac{d^2u_{i,j}}{dxdy} = \frac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{(2\Delta x)^2}$$

. Nearly all operations involved in this step are either taking computing one of these differences or adding the change in a property over time to the property. That is, most of the step's computation is repeating the same central difference operation on a different Node field and then adding up all the differences. This makes it seem like a good SIMD candidate. Using these central differences also causes each node to depend on their adjacent and diagonal neighbors in the xy, xz, and yz planes. The nodes depend on their adjacent and diagonal neighbors within the xy, yz, and xz planes. For illustration, in the figure shown to the right, the node at the origin would depend on the nodes colored by one of the planes assuming each of the planes extends only 1 unit from the origin in each direction. This causes each node to have dependencies on 18 nearby nodes, far more than in the combustion stage. This makes it harder to parallelize than the combustion stage. However, since it makes up 97% of the program time, parallelizing it is necessary to obtain significant speed-up. Since there is



strong locality in the dependencies in the 3-D space, we chose to store the grid as an array so that neighbors in the x-dimension would enjoy good locality. By performing strides in this dimension, we should generally benefit from locality for the neighbors in other dimensions since 14 / 18 of them should have been loaded by the previous node's calculation. However, benefits from the cache may be limited due to the large size of Node class, which occupies 64 bytes.

The program's input is a file that includes the deltax, deltay, deltaz, deltat, numIterations, xDim, yDim, zDim, list of initial node values for the grid, length of the spark list, start time for the spark plug, end time for the spark plug, and the spark list of nodes affected by the spark plug. The output is the values of the nodes after numIterations iterations. This is in line with an actual CFD in that generally they will simulate tiny timesteps, but only output the results at large intervals of time steps.

Approach

We used CUDA to take advantage of the NVidia GPUs on the Gates machines. The Gates machines have NVidia 1080s with 484 GB/s of bandwidth and around eight gigabytes of global memory. They also have a Quadro K620 with around two gigabytes of global memory, but we only used the 1080 for our computations.

For the simulator, we approximated different parts of the fluid using discrete nodes and mapped one thread in the graphics card to one node (or particle) in our simulation. We used three-dimensional thread blocks to take advantage of the locality in the warps of the GPU, since every node was only affected by the adjacent nodes in 3d space. We specifically used a shared array within each thread block and placed adjacent nodes in the same thread block so that each thread would be able to access its neighbors relatively quickly.

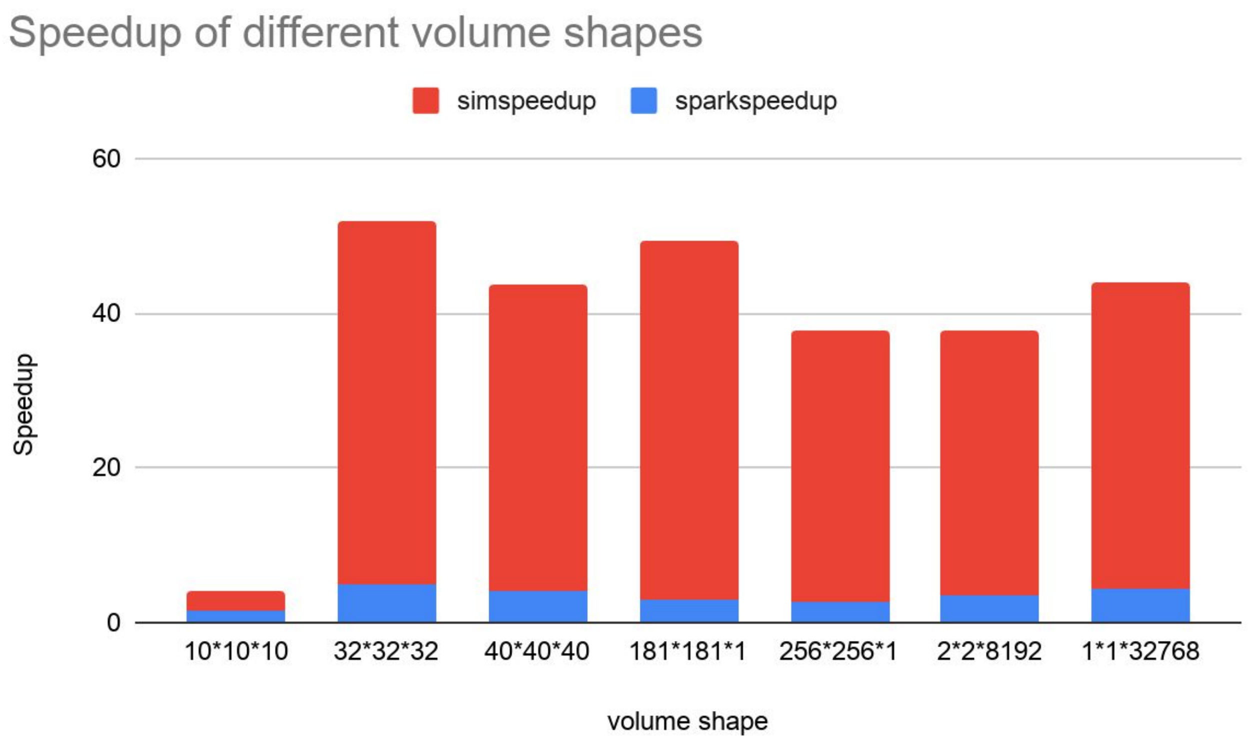
The original serial algorithm used vectors which don't really exist in CUDA, so we had to change the representation of the data to arrays in memory instead of vectors. But, other than that, the sequential implementation was written to be adapted to CUDA, so we didn't have to change too much about the original implementation. Since much of the computation was done by macros, we didn't have to change that much either.

Our optimizations largely centered around the sizes of the thread blocks. We originally had the threads in the block be 16 by 16 by 4 because we wanted to maximize the number of threads per block while still maintaining a three-dimensional block. However, this ended up with issues when we had a very long, thin area to simulate (like, for example, a 1 by 1 by 32768 set of nodes). We then realized, for two of our kernel functions(one where we calculated if a node was about to combust, and one where we copied the new nodes over to the old nodes), the actual position of the node in three-dimensional space didn't actually matter, so we could safely launch those kernels in a one-dimensional grid and not have to worry about computing the node index from the block. We wanted to still take advantage of shared memory so we launched the step simulation kernel in an eight by eight by eight block. This made computations a bit easier when we were trying to get the spatial location of a node, and it made it easier to reason about the shared memory among each thread block.

We used large portions of the code for assignment 2's renderer to handle images for our visualizer, and most of our structure for the CUDA portion of the assignment was adapted from the code in assignment 2. We wrote the actual code in our functions from scratch, with the exception of a few utility functions (such as writing to a .ppm image, clamping numbers, etc.) Almost all of the non-cuda functions and files were written from scratch.

Results

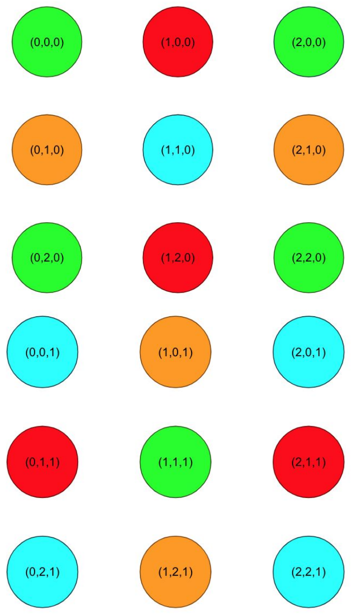
We measured our performance for the simulation step by speedup. Our experimental setup used a python program to generate different text files that gave the parameters for the simulation. The first few lines of the input file defined things like the granularity of the simulation (for the computed derivatives) as well as the dimensions of the container we were using. The next few thousand lines of the file defined the initial values of the node, which were usually some uniform value. Finally, the last few lines of code defined the positions of the initial sparks, which were the cause of the fuel ignition itself.



The above chart shows the speedup on different volumes of fluid simulated when compared to the sequential code on a CPU. Larger cubic volumes seemed to have a good amount of speedup, which makes sense because they were the ones who took advantage of the shared memory within blocks the most. However, the speedup for very small cubes was very small, mostly owing to the size of the block and grid dimensions. Since the block size was eight by eight by eight, the smaller cubes launched 6 unnecessary thread blocks in every iteration, which led to a large amount of load imbalance and unnecessary work for the GPU. In addition, the flatter and narrower volumes also had relatively little speedup.

Also, the speedup of the combustion step was minimal compared to the speedup of the simulation step. As the number of nodes got larger, the speedup of the combustion calculation slightly grew, but, for all cases, the speedup was mostly in regards to the second step of the calculation, where we calculate the effect of each node on its neighbors. This step has both a large amount of computation and a large amount of communication.

One part that limited our speedup was a large number of dependencies in the algorithm. In the 3-D grid, each node in our grid depended on 18 of its neighboring nodes, as it depended on all its adjacent or diagonal neighbors in each of the three planes. This prevented us from being able to concurrently update all nodes in place and instead forced us to either update a second grid with new values or experiment with patterns that updated the nodes in group stages, similar to the red-black ordering presented in class when discussing the ocean simulation. The latter could be done to take advantage of the fact that nodes along the 3-dimensional diagonals are independent to 4-color the grid's dependency graph as shown in the example to the right. Due to time constraints, we chose the former.

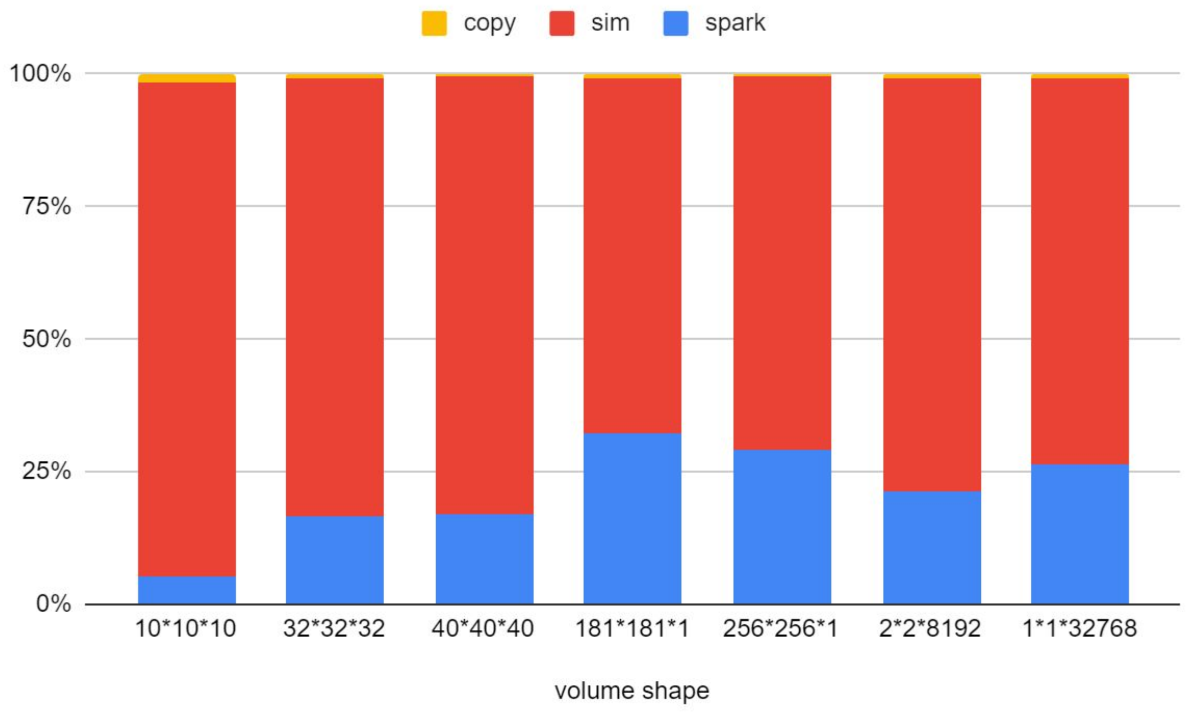


4-Coloring of dependency graph for 3x3x2 instance of grid

This, however, was in contention with our memory constraints. Each node in the grid required 64 bytes to store, so even a grid of only 128x128x128 nodes required roughly 128MB just to store the grid being operated on. This forced us to consider either only processing small parts of the grid at a time. This would have introduced extra complexity because we would have had to then consider the dependencies between nodes at the borders of these regions and whether to include some sort of ghost cells around each region. We did not ultimately implement this due to time constraints.

Each simulation step contained two stages of computation, combustion and advection/conduction. The computation includes a large amount of divergence in both stages that made the SIMD instructions for the GPU warps less effective. For the advection stage, there is divergence for nodes near the boundaries of the grid, as the simulation had to perform modified finite differences to account for boundary conditions.

This reduces the effectiveness of SIMD instructions because the warp would have to perform both the instruction set for the boundary nodes and the instruction set for the non-boundary nodes' asynchronously, increasing the total cycles the warp needs to run its threads. We speculate that it may be beneficial that the advection step first dedicate some warps to computing only boundaries and remaining warps to include only non-boundary nodes to reduce divergence. This would have to be done in a way that preserves the locality of nodes each warp computes. The combustion stage of the simulation also experiences divergence since during each step some nodes experience combustion and others don't, depending on whether they possess the activation energy to ignite. This divergence causes threads for combusting nodes to perform several cycles of computation, while threads for non-combusting nodes to simply stall until the end of the combustion stage. Unlike the divergence for advection, this divergence is between routines that do basically nothing vs. several cycles of computation and limits speedup not because the divergence requires running multiple instruction sets but because many threads simply lay idle for the entire combustion stage rather than performing useful work. The distribution of the combustion nodes within the grid is not known until computation time, so reducing this divergence would require some potentially costly dynamic mapping of threads to combusting nodes. However, physically we would expect it that after the initial spark, combustion would occur in localized clusters since nodes tend to ignite their neighbors as the flame propagates, which could potentially reduce the effect of divergence during the combustion stage.



The above graph shows the breakdown of time between simulating the change in the node from its neighbors, calculating if a node was about to combust, and copying nodes over from the new array to the original array. This suggests that most of the optimizations will be in regards to how the program calculates how nodes affect their nearby nodes. This makes sense because that step of the algorithm is the step that not only involves the most computation but also the most communication.

Something else to consider is the shape of the volumes simulated. For example, the simulation steps took up the largest proportion in cubelike volumes, such as the 10 by 10 by 10 volumes. The neighboring simulation steps took relatively little time in a flatter and narrower volumes, even with approximately the same amount of nodes (and therefore the same amount of total work). This may be because of the communication cost between different nodes: nodes in cubic volumes tend to have more neighbors and therefore will tend to have more communications between threads. Our major optimizations involved decreasing the communication between these threads.

The GPU ended up working out relatively well for this specific project because we had to do a large amount of calculations on memory that depended on nearby pieces of memory. There wasn't too much conditional logic involved and there was a bunch of computation, which played to the GPU's strengths. However, part of where the GPU didn't end up working well is the relative scarcity of memory. We had to limit the number of nodes in our computation so that the GPU could have enough memory to handle all of the information needed to simulate the different properties of the fluid. The CPU, by contrast, didn't have any trouble handling the large amount of memory needed for the fluid simulation.

References

Anderson, Dale A. (Dale Arden), Pletcher, Richard H., and Tannehill, John C. *Computational Fluid Mechanics and Heat Transfer* . 3rd ed. Boca Raton, Florida ;: CRC Press, 2013. Print.