**Just doing parts I can since im still working on getting it to run
Can the GPU give a boost to the parallelism in the Jacobi Iteration?
By: Michael Plekan

Abstract: The GPU is made up of a lot of simple cores, Jacobi is a problem with alot of simple math problems. We can get a decent boost in performance just by parallelizing on the CPU but we might be able to get an even better one on the GPU. Inorder to parallelize, OpenMP is being used, and with the right setup in the code and everything needed for compilation, it gives a big boost in performance.

intro

Throughout this class we have done the Jacobi Iteration problem in many different ways. This project is taking that one step further and parallelizing it on the GPU. There is a big performance gain to be had with doing Jacobi with the GPU. There are a lot of issues that can pop up while trying to get this done. There is also a lot to be learned from this troubleshooting and from these unexpected problems.

skipping results section

issues I ran into

The issues that came during this project were sometimes unexpected and difficult to get around. After a bit of research, I made a Jacobi version that should run on the gpu. I went to compile it, it works fine but it isn't running on the GPU. In openMP the is a target directive, it is how a programmer can use other hardware components while using OpenMP, in this case the GPU. My target wasn’t set when it compiled, so it went to default, the CPU. At this point I was running mingw on windows. I downloaded the nvidia cuda toolkit to see if that had the libraries I needed. The only issue was gcc ran in ming and nvcc ran in windows command prompt, but neither in both. This was only the first issue I ran into, next I went to WSL to see if that would help.

The next step in the process was moving to WSL (Windows Subsystem for Linux). I installed the Ubuntu WSL because Nvidia had a toolkit for the Ubuntu WSL. I get the basics downloaded (gcc, git/gh, etc). I then go to get the Nvidia drivers and toolkit, and more issues come up. The driver always fails to install, all the other parts of the installation complete without a problem except for the key piece. After a lot of searching and trying things I finally figured it out. I only had WSL 1 when I thought I had WSL 2. I update WSL and the driver finally installs. I go to test out my code to see if it will compile but I get a bunch of linking errors.(shown below)

```
gcc -o jacobi timer.o jacobi.o -lm -fopenmp -L/usr/local/cuda/lib64 -lcudart
/usr/bin/ld: jacobi.o:jacobi.c:(.text+0x33): undefined reference to `__mingw_vfprintf'
/usr/bin/ld: jacobi.o:jacobi.c:(.text+0x72): undefined reference to `__imp___acrt_iob_func'
/usr/bin/ld: jacobi.o:jacobi.c:(.text+0x83): undefined reference to `__mingw_vfprintf'
/usr/bin/ld: jacobi.o:jacobi.c:(.text+0xb0): undefined reference to `__mingw_strtod'
/usr/bin/ld: jacobi.o:jacobi.c:(.text+0xed): undefined reference to `__main'
/usr/bin/ld: jacobi.o:jacobi.c:(.text+0x12d): undefined reference to `__imp___acrt_iob_func'
/usr/bin/ld: jacobi.o:jacobi.c:(.text+0x2ee): undefined reference to `___chkstk_ms'
/usr/bin/ld: jacobi.o:jacobi.c:(.text+0x401): undefined reference to `___chkstk_ms'
collect2: error: ld returned 1 exit status
make: *** [Makefile:6: jacobi] Error 1
```

I looked at the errors and they seem to have nothing to do with Nvidia or gpu's at all. From here I start to wonder if WSL will compile and run anything. So I downloaded my original Jacobi

```
mike@DESKTOP-UU1Q7AP:~/jacobi-c-proj-MikePlekan$ make
gcc -g -Wall -c jacobi.c
gcc -g -Wall -o jacobi timer.o jacobi.o -lm
mike@DESKTOP-UU1Q7AP:~/jacobi-c-proj-MikePlekan$ ./jacobi 200 10000 0.000001
Completed 10000 iteration pairs, last maxDiff 0.000009, 5.530851 seconds
mike@DESKTOP-UU1Q7AP:~/jacobi-c-proj-MikePlekan$
```

program. Fortunately and unfortunately it worked perfectly. So I hit a dead end at this point with WSL. I next tried out running and compiling it on the PSC Bridges 2 supercomputer since I know they will have the correct setup for it.

expected results

The results from running on a GPU can be surprising. If you don't use the teams directive it may end up being slow because it will only use a small part of the GPU. If you do use the right directives you can get a speedup of 3-4 times faster compared to the CPU.

OpenMP directives/ GPU techniques learned

- Target- This allows for offloading of work to other devices on the computer
- Teams- Splits threads up into teams so they can run as as separate thread blocks
- Distribute- Distributes works throughout the teams so work isn't being done redundantly

Conclusion