An intro to OpenACC

or

How to speed up your code in ten lines or less

Rachel Smullen
Steward Code Coffee
23 Oct 2018

IHPCSS

- International High Performance Computing Summer School
  - http://www.ihpcss.org/
    - 2019 in Kobe, Japan
  - Summer school to learn HPC tools
    - ALL expenses paid
  - You don't need to be experienced to apply!

What is OpenACC and why should you care?

- OpenACC is a way to use a GPU to help speed up your code
  - Without having to write in CUDA!
- Uses compiler hints (pragmas) to tell the compiler where to use a GPU
  - Means that the code is runnable in serial or parallel
  - We (the users) don't need to know how a GPU works to use one!

OpenACC

More Science, Less Programming

A Few Cases

Reading DNA nucleotide sequences

Shanghai JiaoTong University

4 directives

16x faster

Designing circuits for quantum computing

UIST, Macedonia

1 week

40x faster

Extracting image features in real-time

Aselsan

3 directives

4.1x faster

HydroC- Galaxy Formation

PRACE Benchmark Code, CAPS

1 week

3x faster

Real-time Derivative Valuation

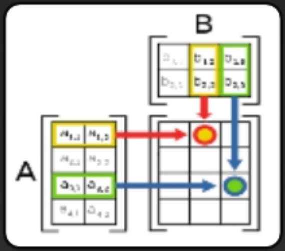
Opel Blue, Ltd

Few hours

70x faster

Matrix Matrix Multiply

Independent Research Scientist



4 directives

6.4x faster

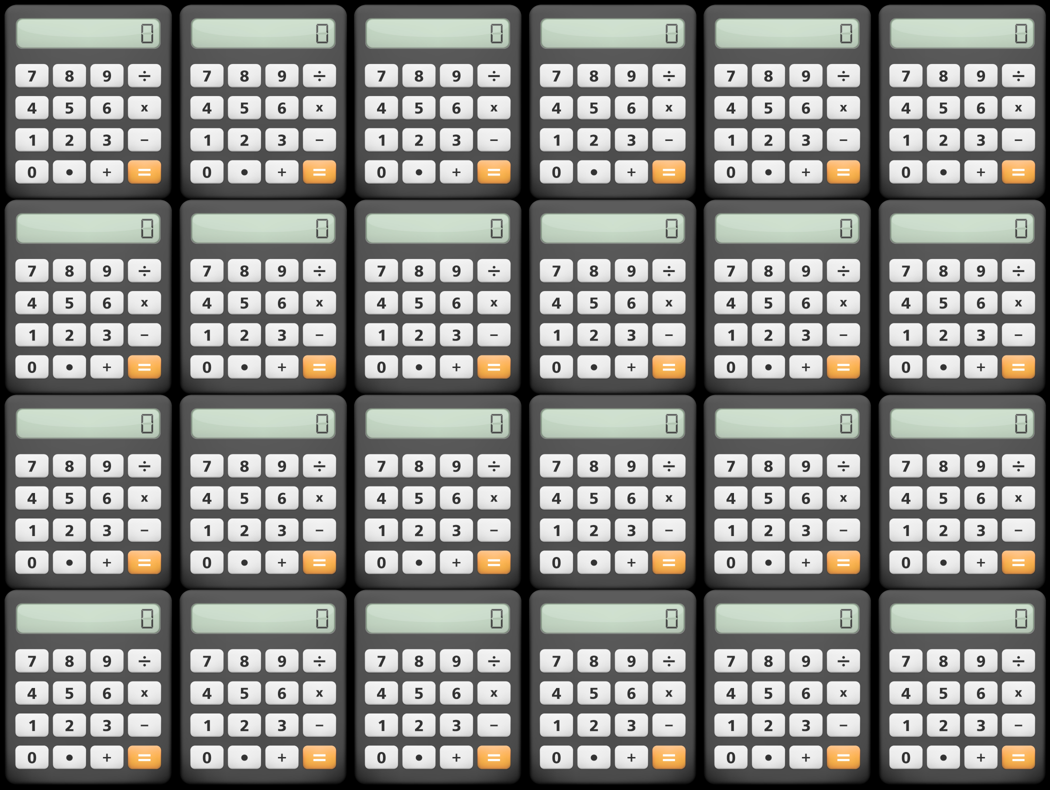
John Urbanic, PSC

When should you use OpenACC?

- When you have independent loops or highly parallelizable regions
  - When a small chunk of data that one processor may have doesn't depend on the chunk that another has
  - Are you using MPI or OpenMP?
    - You can probably use OpenACC
  - a[i] = b[i] + c[i]
- Do NOT use OpenACC (or other acceleration methods) if you have significant data dependencies
  - a[i] = a[i-1]+b[i]+c[i]
    - a[i] depends on something that happened elsewhere

What is a GPU?

CPU (Central Processing Unit)	**GPU** (Graphics Processing Unit)
Few expensive cores	Many cheap cores
Very smart	Pretty stupid
Somewhat parallelizable (threading)	*Very parallelizable*



Where are GPUs on campus?

Computers are here
Speedway Boulevard
Computer Center
Highland Avenue
1st Street
1st Street
Vine Avenue
Warren Avenue
Martin Avenue
2nd Street
2nd Street
Second St. Garage
Education
Steward Observatory
You are here
Hawthorne Street
Student Union Memorial Center
Admin
Modern Languages
Psychology
Sonett
Flandrau
Kuiper
Gittings
Lapan
UTC
Pima
NLK
Beal
Bartlett
Pacheco

- UA HPC resources:
  - El Gato (old)
  - Ocelote (new)
- El Gato has many (140) old GPUs
  - CUDA 6-9
- Ocelote has a few (46) new GPUs and updated software
  - Nvidia Telsa P100 GPUs
  - PGI compilers (best for OpenACC)
  - GCC 6+, CUDA 7-8

Rixin and I gave a Code Coffee presentation on HPC resources last year: information can be found here

How to access the GPUs?

- UA HPC docs: https://docs.hpc.arizona.edu
- Open OnDemand: https://ood.hpc.arizona.edu
  - Provides a nice web interface for:
    - Looking at files on the HPC cluster
    - Checking on jobs
    - Gaining shell access to both Ocelote and El Gato
      - Can run interactive nodes, but no graphics (X-forwarding)
    - Using an interactive desktop environment
    - Running Jupyter notebooks

How to access the GPUs?

- Need to submit a job that requests GPU resources
  - For an interactive node:

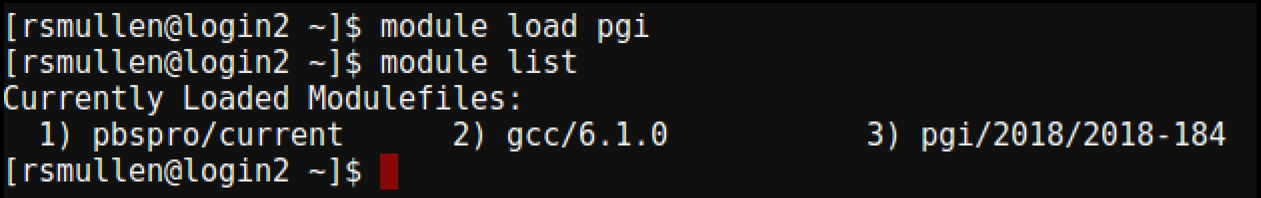
```
qsub -I -W group_list=kkratter -q standard -l select=1:ncpus=28:mem=168gb:ngpus=1 -l walltime=4:0:0
```

  - Submit a job
  - Interactively
  - To the group *kkratter*
    - for me—use the *va* command to find the groups you belong to
  - To the queue *standard*
    - Can also use *windfall*, but do NOT use *oc_standard* or *oc_windfall* for GPUs
  - That gives me 1 *node* with 28 *CPUs* and 168 GB *memory* and 1 *GPU*
  - That will run for 4 hours 0 min 0 sec
  - Can also include -X for X-forwarding if available

Now back to OpenACC

- The PGI compiler (installed on Ocelote) is the easiest compiler to use for OpenACC
  - C++ and Fortran compatible
  - GCC 6+ supports some OpenACC commands, but is behind the times and isn't nearly as well supported
    - You can figure that one out yourself… :)

- To load PGI in your interactive session
  - *module load pgi*

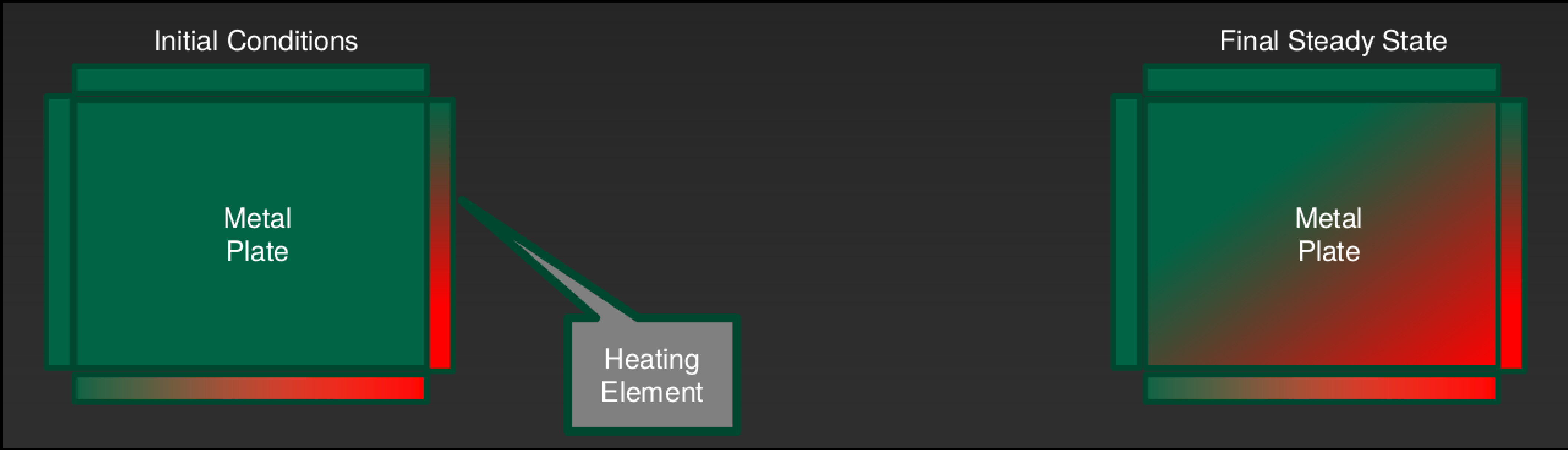


```
[rsmullen@login2 ~]$ module load pgi
[rsmullen@login2 ~]$ module list
Currently Loaded Modulefiles:
  1) pbspro/current      2) gcc/6.1.0          3) pgi/2018/2018-184
[rsmullen@login2 ~]$
```

Aside: the test problem

- I've provided a test case that solves the Laplace equation

$$\nabla^2 f(x,y) = 0$$



- Using the iterative average of neighboring cells, we recover the time evolution and steady state solution

How do we use OpenACC

- With PGI, we can use the *kernels* directive to interact with the GPU
  - It automatically chooses the best way to disperse your code to GPU cores
  - We can have many parallel regions with different kernel calls (no race conditions)

```
#pragma acc kernels
{
    code to parallelize
}
```

  - Try this yourself in the provided code!

Let's compile and run our code!

- Serial:
  - Compile:
    ```
    pgcc -o serial.out laplace_acc.c
    ```
  - Run: `./serial.out`

- OpenACC
  - Compile:
    ```
    pgcc -acc -ta=tesla:cc60 -Minfo=accel -o gpu.out laplace_acc.c
    ```
  - Run: `./gpu.out`
    - *-acc*: use OpenACC
    - *-ta*: target the tesla GPUs
    - *-Minfo*: provide some output on what is being parallelized

Timings

- Serial: ~16.5 s
- Accelerated: ~27 s

Timings

- Serial: ~16.5 s
- Accelerated: ~27 s

...uhhhh, what?

I thought the code was supposed to be faster?!?!

Timings

- Serial: ~16.5 s
- Accelerated: ~27 s

...uhhhh, what?

I thought the code was supposed to be faster?!?!

Let's profile our code.

In the terminal, type
export PGI_ACC_TIME=1
and rerun

Data Copy

- From the output, we can see that something called copyin and copyout took a large part of our run time
- The compiler was being conservative and making sure that the arrays on the GPU and CPU were synced every step

- If you remember nothing else, remember that I/O is always slow, and minimize it where you can!

Data Copy

- We can use the following pragmas to tell the compiler when and where to copy our code
  - #pragma acc data copy(my_array)
    - Copy my_array in at start of block and out at end of block
  - #pragma acc data create(my_array)
    - Create an empty my_array on the GPU—no data transfer
  - #pragma acc data copyin(my_array), copyout(my_array)
    - Copy my_array to GPU at beginning of block
    - Copy my_array to CPU at end of block
  - #pragma acc update host(my_array[1:1000])
    - Force an update of my_array to the host (CPU) or devide (GPU)

Try adding in the correct data copy commands and recompiling/rerunning your code.

Did it work?

My solution (runs in <1s)

```
#pragma acc data copy(Temperature_last), create(Temperature)
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
        // main calculation: average my four neighbors

        #pragma acc kernels
        for(i = 1; i <= ROWS; i++) {
                for(j = 1; j <= COLUMNS; j++) {
                        Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
                                Temperature_last[i][j+1] + Temperature_last[i][j-1]);
                }
        }
        dt = 0.0; // reset largest temperature change

        // copy grid to old grid for next iteration and find latest dt
        #pragma acc kernels
        for(i = 1; i <= ROWS; i++){
                for(j = 1; j <= COLUMNS; j++){
                        dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
                        Temperature_last[i][j] = Temperature[i][j];
                }
        }
        // periodically print test values
        if((iteration % 500) == 0) {
        // not formally needed, but this is how you would update the CPU copy for temporary output
                #pragma acc update host(Temperature)
                track_progress(iteration);
        }
        iteration++;
}
```

Voila!

- You can now apply OpenACC to your own code!
- There is A LOT more that I haven't covered
  - You can mesh OpenACC and MPI (best) or OpenMP (not great) to use multiple GPUs
  - There are lots of ways for you to help the compiler out to get the right result
    - Wait, async, atomic, etc.
  - I've included some PowerPoints from the workshop I went to to provide some examples

Happy coding!