Astroinformatics: Day 1

Adam W. Lavely, Ph.D.

Afternoon Seminar A

June 4 2018

Slides/Files available: `https://goo.gl/Wz72vs`

PennState
Institute
for CyberScience

Contents

Contents

Historical Shared Memory

Shared Memory Parallelization:
All of the cores have access to the same memory pool

Previous implementations: Large memory pools available to multiple processors

- *Individual nodes* - High memory nodes with many processors
  - Intrinsic limit of how much RAM you can pack onto a node
- *Special cluster hardware* - Blacklight (2010-2015) at PSC (16 Tbytes RAM available)
  - Limited by hardware capabilities (R.I.P. SGI)

Current Shared Memory

Technology advances!

Current implementations: Memory can be shared in many ways

- *Individual chips* - Multiple cores have access to the same cache and RAM
  - CPU and accelerators
- *Individual nodes* - High memory nodes with many processors
  - Intrinsic limit of how much RAM you can pack onto a node
- *Special cluster hardware* - Blacklight (2010-2015) at PSC (16 Tbytes RAM available)
  - Limited by hardware capabilities (R.I.P. SGI)
- *Software driven sharing* - Languages built to abstract data movement
  - Various types of Global Address Space models available

Confusing Hardware Buzzwords

Multicore Make a core 'fast' and put as many as close together as thermally possible

- Intel's Core i9[1] - 18 cores at 2.6 GHz nominal and 4.2 GHz turbo
- AMD's Ryzen Threadripper[2] - 16 cores at 3.4 GHz nominal and 4 GHz turbo

Manycore Pack as many cores onto a chip as possible with slow clock speeds

- NVIDIA Tesla P100[3] - 3584 cores at 1.328 GHz nominal and 1.48 GHz turbo
- Intel Knight's Landing[4] - 72 cores at 1.5 GHz nominal and 1.7 GHz turbo

[1] urlark.intel.com/products/series/123588/Intel-Core-X-series-Processors

[2] amd.com/en/products/cpu/amd-ryzen-threadripper-1950x

[3] images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[4] intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors/7290.html

Confusing Software Buzzwords

MIMD Multiple instruction, multiple data

- Many different tasks worked on by a set of cores
- Multi-tasking in single or multiple codes

SIMD Single instruction, multiple data

- The same instruction running on multiple cores
- Multi-tasking in a single code (≈vectorized)

Multicore (CPUs): Tend to do best with MIMD, can do SIMD with instructions and/or flags

Manycore (acclerators): Tend to do best with SIMD, MIMD[5] possible in rare circumstances

[5] youtu.be/FZ6efZFlzRQ

Software types

'Physically' available memory: Memory can be accessed by core (processor or node specific RAM)

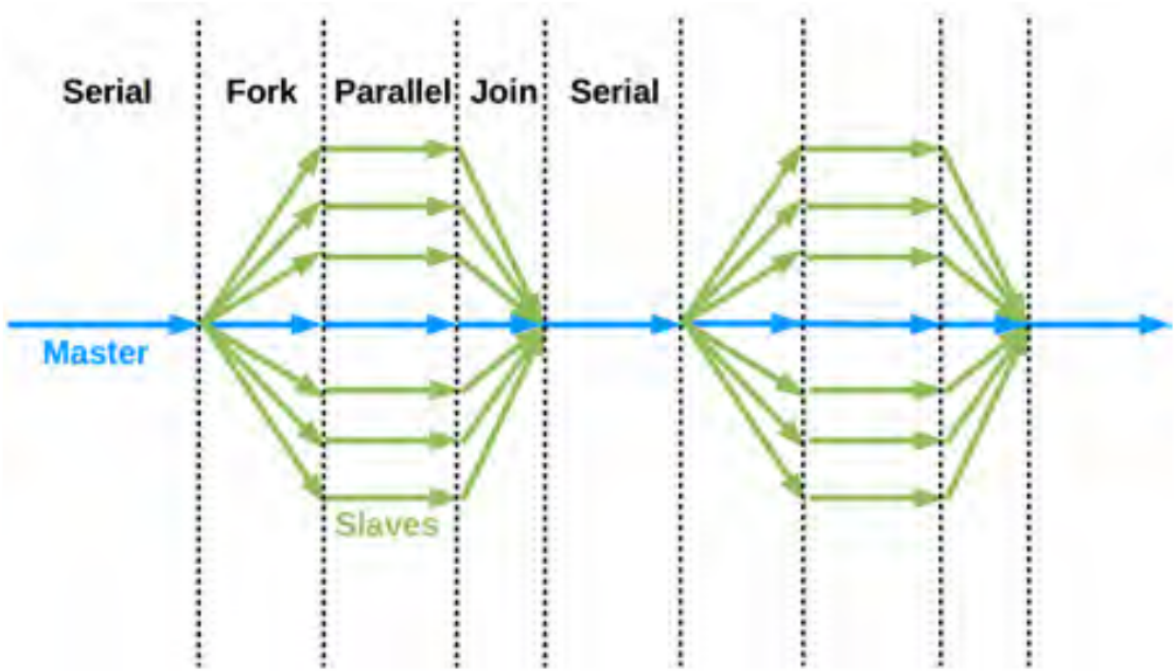
- Controlled: Pthreads, threads, Boost
- Abstracted: Open Multi-Processing (OpenMP), Boost

'Software available' memory: Memory can be accessed by address (global address space)

- Controlled: UPC, UPC++, Fortran 2008
- Abstracted: X10, Chapel, Cluster OpenMP

OpenMP is generally the easiest to implement:

- Fork and join loops in existing code
- Minimal changes to structure



6

[6] Figure taken from Pierre-Yves Taunay's 2014 Astroinformatics Talk

OpenMP[7] codes/compilation/execution common steps:

1. Set up the number of threads to launch using an environment variable (or internally)
   ```
   export OMP_NUM_THREADS = 4
   ```
2. Give access to OpenMP functionality (OpenMP header)
   ```
   #include <omp.h>
   ```
3. Set up parallelization and data protection (next slides)
4. Compile using the correct flag
   ```
   gcc -fopenmp testOpenMP.c
   icc -openmp testOpenMP.c
   ```

[7] Great examples at `openmp.org`

Most common OpenMP loop directives:

parallel — Task is to be done by all threads

```
#pragma omp parallel
printf(‘‘Hello world!\n’’);
```

parallel for — For loop where each loop is done by a thread

```
#pragma omp parallel for
for(i=0;i< num_steps; i++){printf(‘‘Hi!\n’’);}
```

schedule — Schedule task chunk size

Other typical loop directives:
reduction, ordered, nowait, sections

Some things should be done once:

- Do something in parallel loop only once *single*
- Don't share some variables: *private*

Task Synchronization:

- Hold processors until all reach a point: *barrier*

Avoid race conditions:

- Hold based on memory in use: *lock*
- Hold based on code in use: *critical*

OpenMP Synchronization Concepts

	Execution/Access	*Other Threads*
Lock	**Memory** can only be accessed by one process at a time	Run if not requiring accessed memory, wait if requiring memory that is in use
Critical	**Process** executed one at a time	Wait until the code section is available
Single	Process is only executed once	Pass on once a thread starts this process
Barrier	Wait	All processes stop until every process reaches this point

OpenMP Synchronization Code

Lock

```
omp_set_lock(lockA);
{printf(‘‘Hi from threads with unique memory\n’’
omp_unset_lock(lockA);
```

critical

```
#pragma omp critical
{printf(‘‘Hi from only thread here\n’’);}
```

single

```
#pragma omp single
{printf(‘‘Hi from first thread here!\n’’);}
```

barrier

```
#pragma omp barrier
printf(‘‘All threads are here!\n’’);
```

Overhead

Things take time:

- Thread creation/finalization takes time
- Locking/unlocking adds steps
- Barriers and criticals cause idle cores

There is no free lunch.

Even the most simple parallelization is tricky. (sorry)

Contents

N-body problem

OpenMP example on Bridges

Restart at 4 pm

Please stay after if you need help with:

- Shared memory example