Introduction to Shared Memory Programming

University of Wyoming
Advanced Research Computing Center (ARCC)
arcc-info@uwyo.edu
(307) 766-7748
https://arcc.uwyo.edu/

Parallel Programming

- Why parallel programming?
  - CPUs have reached there maximum clock rates with respect to heat and power to be and effective speed up mechanism
  - Many applications can experience some method of parallelism
  - As hardware shrinks, more of the hardware in same form factor

- Shared Memory Programming

- Distributed Memory Programming

- HPC systems today are not always the large mainframes that computers used to be, they are traditionally clusters of general computers, therefore not a common memory space usually

Levels of Parallel Execution

- There are several levels that can experience parallel execution

- Distributed Memory Parallelism
  - Message Passing Interface
  - OpenSHMEM
  - Others?

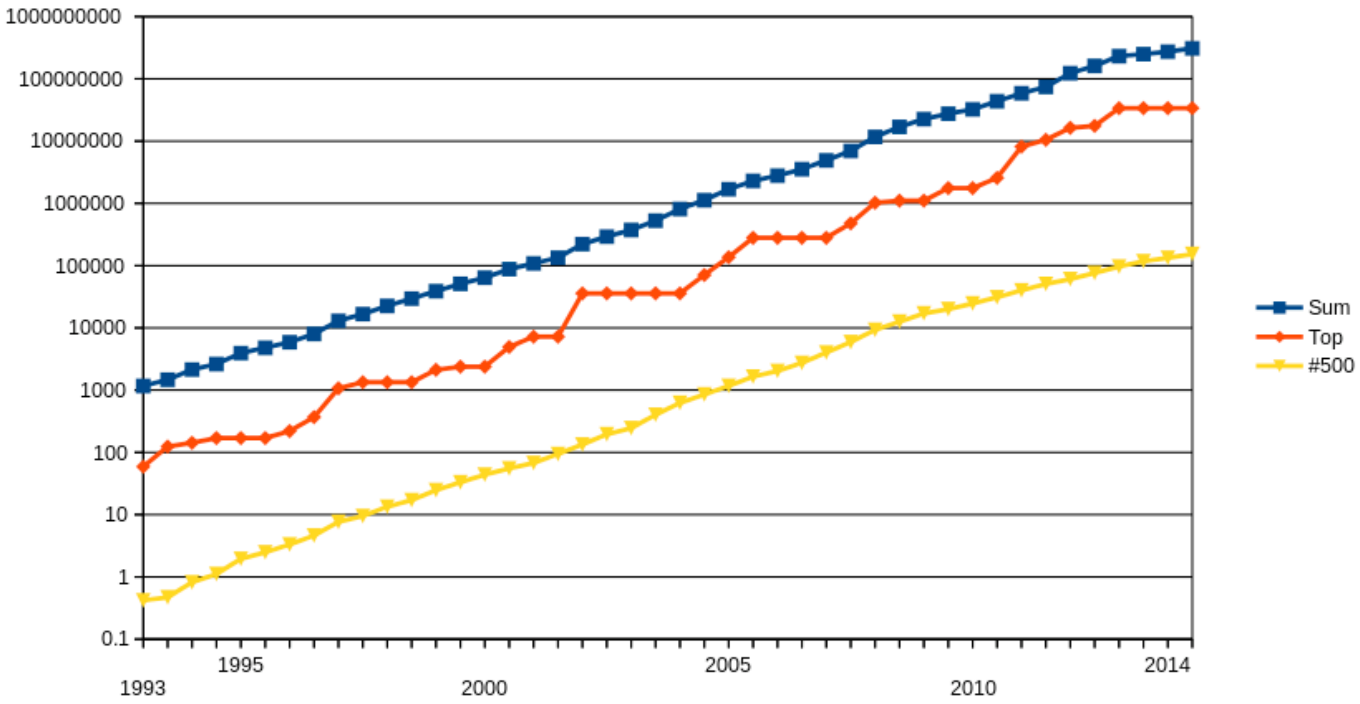
- Shared Memory Parallelism
  - Threading
  - Multiprocessing

- Chip Level Parallelism
  - Vector units
  - Pipelining

History of supercomputers

FLOPS: Floating point Operations Per Second

Steady growth over the last 5 decades



"Supercomputers-history" by AI.Graphic - Own work. Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Supercomputers-history.svg#/media/File:Supercomputers-history.svg

Computing Today

- Generic CPU with multiple cores and vector units
- Graphics Processing Units (GPUs) for extreme parallelism
- Coprocessors (Xeon Phi) for extreme parallelism
- High performance Fabric / Interconnect
    - InfiniBand
    - Ethernet
    - TrueScale/Omnipath
    - Others
- High performance storage
- Memory heirarchies

Hello World

Environment Variable:
OMP_NUM_THREADS

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]) {

        #pragma omp parallel
        {
                printf(“Hello World! I’m thread %d out of %d total threads.\n”,
                        omp_get_thread_num(),
                        omp_get_num_threads());
        }

        return 0;
}
```

Scalar Multiply on a Vector

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]) {

        // Include code here to initialize a vector and a multiplying scalar

        #pragma omp parallel for shared(Vector,Scalar,N)
        for(int i=0;i<N;i++){
                Vector[i] = Vector[i] * Scalar;
        }

        return 0;

}
```

Reductions

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]) {

        // Perform a dot product on two vectors in parallel
        int *A, *B;
        int N;

        // Write the code you need here, pragma given
        #pragma omp parallel for shared(A,B,N) reduction(+:s)

        // Don’t forget to free your memory
        return 0;
}
```

Work on the laplace.c

Ask questions if needed ☺

University of Wyoming
Advanced Research Computing Center (ARCC)
arcc-info@uwyo.edu
(307) 766-7748
https://arcc.uwyo.edu/