COEN 145 : Parallel Computing

Lab 3 : OPENMP

Part 0 : Request HPC resources

```
#!/bin/bash

# Syntax : #SBATCH --parameter=value

#SBATCH --job-name=pl-lab3

#SBACTH --partition=cpu

# CPU Cores = 2*omp_get_max_threads()

#SBATCH --cpus-per-task=X

#SBATCH --mem-per-cpu=8G

#SBATCH --nodes=1

#SBATCH --output=pl-lab3-%j.out

#SBATCH --time=10:00

#SBATCH --mail-type=ALL

#SBATCH --mail-user=USERNAME@scu.edu


# Environment Variables

export OMP_NUM_THREADS=X

export OMP_PLACES=cores

export OMP_PROC_BIND=true


# Build and Run

make clean && make

./mat_vec_mult.out > out.txt

./mat_vec_mult.out in_mat.txt in_vec.txt 1000 5000 5000 > out.txt
```

- Use the provided ***run.sh*** shell script to run ***sum.out*** from last week. Also do it in an ***interactive*** session

```
$ sbatch run.sh
```

```
$ srun --nodes=1 --ntasks=1 --cpus-per-task=2 --mem=8G --pty /bin/bash &&
./sum.out 0 1 2 3 4 5 6 7 8 9
```

```
> sum = 45
```

```
> time-serial = X microseconds
```

Part 1 : Thread (Affinity/Pinning/Binding)

```
# specify number of OMP threads

$ export OMP_NUM_THREADS=2

# enable thread binding and print out info on thread affinity

$ export OMP_PROC_BIND=true

# bind each thread to a core

$ export OMP_PLACES=cores
```

Part 2 : Linking library inside Makefile

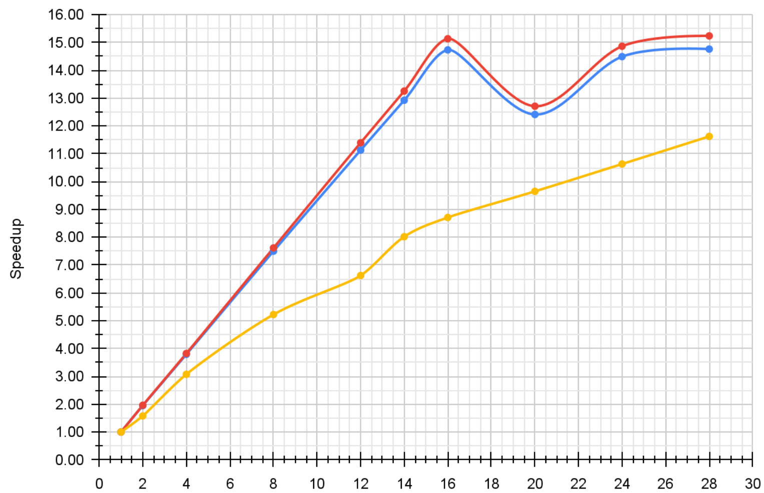
- Update our Makefile to include OPENMP using the compiler flag ***-fopenmp***
- Print the ***num_threads*** available on the working machine using the functions *`omp_get_num_threads()`* and *`omp_get_max_threads()`*

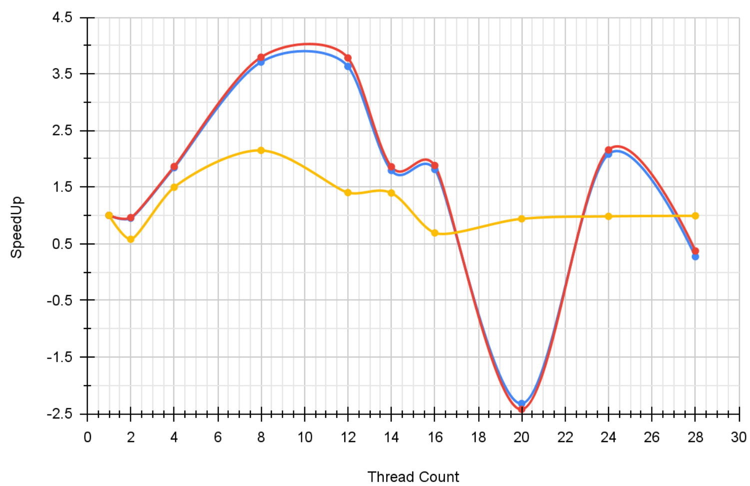
```
> num_threads = X
```

```
> max_threads = Y
```

Part 3 : Measuring Performance

$$speedup = \frac{execution\text{-}time_{serial}}{execution\text{-}time_{parallel}}$$





Part 4 : Parallelize ***sum.c***

- Using the decorators discussed in class modify your code to perform the summation reduction in parallel and report the output in a log file.

```
> sum = 45
```

```
> time-serial = X microseconds
```

```
> time-parallel = Y microseconds
```

```
> speedup = Z.xx (up to 2 decimal places)
```

Part 5 : Parallelize ***mat_vec_mult.c***

- Write a program to perform a matrix vector multiplication in C (not C++) from randomly initialized values as in lab1. Also it should accept optional command-line arguments and read data from a text file as in lab1 with each element on a newline
- Implement using naive method with 2D array
- Compare performance against a single 1D buffer
- Parallelize your code using the directives covered in class and redirect (**>**) the output of the calculation to ***out.txt*** and create **figures** for speedup

```
$ ./mat_vec_mult.out {in_mat.txt | in_vec.txt | nrow | ncol1 | ncol2} > out.txt
```

```
> time-serial (2D) = X microseconds
```

```
> time-parallel (2D) = XZ microseconds
```

```
> time-serial (1D) = Y microseconds
```

```
> time-parallel (1D) = YZ microseconds
```

```
> speedup (2D) = A.xx (up to 2 decimal places)
```

```
> speedup (1D) = B.xx (up to 2 decimal places
```