(HPSC **5576** ELIZABETH JESSUP)

HIGH PERFORMANCE SCIENTIFIC COMPUTING

:: Homework / 14

:: Student / Florian Rappl

1 problem / **10** points

OpenMP Performance

References:

http://openmp.org/wp/

The following is a tutorial on how to use OpenMP.

https://computing.llnl.gov/tutorials/openMP/

The following is the section of the tutorial that contains examples similar to the task assigned in this lab.

https://computing.llnl.gov/tutorials/openMP/#Combined

References:

- http://icl.cs.utk.edu/hpcc/
- http://www.netlib.org/utk/people/JackDongarra/faq-linpack.html#_What_is_HPL?
- http://netlib.org/utk/people/JackDongarra/PAPERS/hplpaper.pdf

Procedure:

1. Run the serial code provided below using the compile line also below to establish a baseline on your test machine. Compile using

```
gcc -O3 -fopenmp matmat.c
```

2. Create and test a parallel version of the code using the OpenMP Sections directive. (`#pragma omp sections`). Note that doing so implicitly hardcodes the number of threads; make sure to divide the work.
3. Create a parallel version of the code using the OpenMP for directive (`#pragma omp for schedule`) with ordering not enforced. Choose one of the following schedule options to test: *static*, *dynamic*, *guided*, or *auto* (version 3.0 requires gcc 4.3.1 or better).
4. Test this parallel version for 1 thread, 2 threads, 4 threads, and so on for a few choices, up to the number of cores in the node you are working on. (You don't have to do all powers of two. If you are on Blacklight, consider using two nodes -- it's a shared-memory architecture!)

On Trestles you can go into interactive node mode via:

```
qsub -I -q normal -l walltime=1:00:00 -l nodes=1
```

Solutions

Task:

Write a paragraph or two (or explain to an instructor) analyzing your results. Include in this analysis a compare and contrast of why different codes performed as they did and any parameters you used that might have influenced the performance of your code.

Answer:

The Sections code did already show a speedup by splitting up the two calculations into two threads. However starting threads takes some time as seen in Np=1. We have to admit that we will roughly lose ~a second by starting the thread. We can therefore see a roughly linear speedup from 1 to 4 threads when we take into consideration that some thread startup takes some time and that the shared memory handling has some price as well. After a certain number of threads we do not profit any more since the problem seems to be too small (thread starting time bigger than solving time).

	Serial	Sections	Np=1 (dynamic)	Np=2 (dynamic)	Np=4 (dynamic)	Np=16* (dynamic)
Runtime (s)	5.388934	4.657954	9.176876	5.716748	3.869767	2.672197
MFlops**	3188	3688	1872	3005	4440	6429
Speedup	1	1.16	0.59	0.94	1.39	2.02

* 16 and 32 had the same result. I guess the matrix was not sized big enough for 32 threads to be more efficient than 16 threads.

** calculated over $2 \cdot 8 \cdot 1024^3$ (2 times the matrix calculation with 8 operations per iteration in 3 for loops with 1024 iterations each.

I worked with the following code:

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <stdlib.h>
double calctime(struct timeval start, struct timeval end) {
	double time;
	time = end.tv_usec - start.tv_usec;
	time = time/1000000;
	time += end.tv_sec - start.tv_sec;
	return time;
}
void main() {
	int i, j, k, n = 1024;
	double *A, *B, *C, time;
	struct timeval start;
	struct timeval end;
	A = malloc(n*n*sizeof(double));
	B = malloc(n*n*sizeof(double));
	C = malloc(n*n*sizeof(double));
	for(i = 0; i < n; i++)
		for(j = 0; j < n; j++)
		{
			A[i*n + j] = 0.0;
			B[i*n + j] = i + j*n;
			C[i*n + j] = i*n + j;
		}
	gettimeofday(&start, NULL);
	for (i = 0; i <n; i++)
		for(j = 0; j < n; j++)
			for(k = 0; k < n; k++)
				A[i*n + j] += B[i*n + k] * C[j*n + k];
	for (i = 0; i <n; i++)
		for(j = 0; j < n; j++)
			for(k = 0; k < n; k++)
				A[i*n + j] += B[i*n + k] * C[j*n + k];
	gettimeofday(&end, NULL);
	time = calctime(start, end);
	printf("mat-mat time: %lf\n", time);

}
```

Start of Block to parallelize

Speed differences in the usage of sections and for schedule are by done by the different declarations of private and shared. We have to take two main thoughts into consideration: i.) do both have to work on the variable on the same time (shared required) and ii.) do both just need to work with a copy of it (private would be better). This is somehow architecture dependent because some CPU have a shared memory as well as own core memory – therefore private might speed up while shared might slow down – this means we should only use shared if shared is really required.