High Performance Computing

Project - Homework #7

Due: Monday Nov 20 2017 before 11:59 PM (Midnight)

Email-based help Cutoff: 5:00 PM on Fri, Nov 17 2017

! The 10 points allocated for this performance analysis report is earned only if the program is functionally correct when operating with multiple threads. If the program is not operating correctly, conducting performance analysis on an incorrect program is meaningless and consequently you will not earn points for the report. So ensure you thoroughly test the program first

Name: Kai Li

Experimental Platform

The experiments documented in this report were conducted on the Red Hawk cluster with the following configuration:

Component	*Details*
CPU Model	***Intel(R) Xeon(R) CPU X5550 @ 2.67GHz***
CPU/Core Speed	***1596.000 MHZ***
Main Memory (RAM) size	***24591648 kB***
Operating system used	***Linux mualhpcp01.hpc.muohio.edu 2.6.32-642.el6.x86_64 #1 SMP Tue May 10 17:27:01 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux***
Interconnect type & speed (if applicable)	*n/a*
Was machine dedicated to task (`yes/no`)	Yes
Name and version of C++ compiler (if used)	GCC 4.9.2
Name and version of Java compiler (if used)	n/a
Name and version of other non-standard software tools & components (if used)	usr/bin/time

Parallelization strategy

Briefly describe the program you have developed. Discuss (at least 8 sentences) the region of the program that you have parallelized using OpenMP and your rationale for parallelizing that region of your program.

This program is an image searching program which uses a mask image (a small portion of the original image in .png format) to search for matching regions in the original image. Once all the mask images matches are found then they will be enclosed with a rectangle box drawn around it.

The parallelizing strategy boils down to understanding how to allocate individual tasks (or the number of images / region matches) each thread has to do in the original pictures. Two nested for-loops are used to iterate the original image and a region of the original image is "cropped" as big as mask image, and the cropped region is moved one pixel at a time row by row and column by column. Since the cropped region is compared against the mask image in column by column manner whenever the inner for-loop is executing, this we could visualized as one thread may be responsible one or more of these column iterations in the inner for-loop at a particular column. Depending on the number of threads, each thread may be assigned to different number of rows of iterations moving the cropped images column by column in that row. In this case, OpenMP for directive will implicitly deal with the intricacies of allocating number of rows of pixels to the thread to move the cropped image column by column. OpenMP for directive will also take care of the index bounds accordingly while allocating the number of rows or columns to each thread. Since the cropped region and mask image are always the same or constant during the iterations of the beginning two nested for-loops when used to calculate background averages and same shade matching, we would just leave them as is. However, the matching pixel regions need be stored in a list and such list is shared data among all the threads, therefore we would create critical sections for the list whenever it is being accessed or modified.

Test Case #1

The following performance statistics were collated by using the supplied sample data files `Mammogram.png` and `Cancer_mask.png` shown in the images further below. The image match was conducted with 75% match with color tolerance of 32

The program was compiled using the following command line:

```
$ g++ -g -Wall -std=c++14 -O3 -fopenmp -lpng *.cpp -o Homework7
```

The program was run and timings were collected using the following key lines in the PBS-job script.

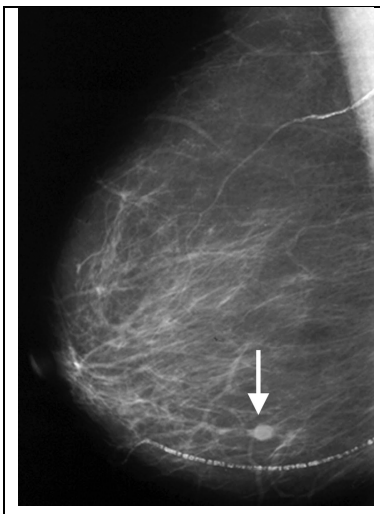
```
/usr/bin/time -v ./Homework7 Mammogram.png Cancer_mask.png
Mammo_result.png true 75 32
```

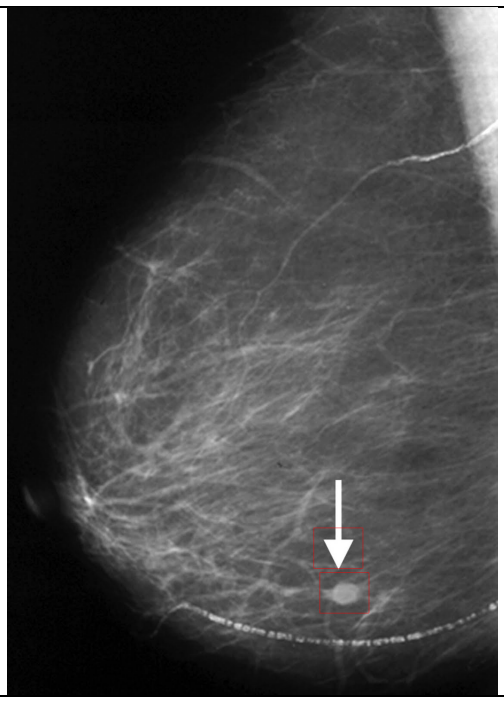
Textual Output verification:

Expected Output	Your Output
sub-image matched at: 1211, 714, 1306, 829 sub-image matched at: 1315, 727, 1410, 842 Number of matches: 2	sub-image matched at: 1211, 712, 1306, 827 sub-image matched at: 1315, 726, 1410, 841 Number of matches: 2

Visual output verification:

Search Image	Mask Image	Output Image





Performance statistics:

#Threads	User Time	Elapsed Time	%CPU
1	1036.14±*17.48151038*	1036.16±17.51674257	99.8%
4	1086.616±0.708810165	272.554±0.158645425	398%
8	1088.382±1.004375061	137.586±0.884869323	791%

Test Case #2

The following performance statistics were collated by using the supplied sample data files `TestImage.png` and `and_mask.png` shown in the images further below. The image match was conducted with 75% match with color tolerance of 16

The program was compiled using the following command line:

```
$ g++ -g -Wall -std=c++14 -O3 -fopenmp -lpng *.cpp -o Homework7
```

The program was run and timings were collected using the following command line:

```
/usr/bin/time -v ./Homework7 TestImage.png and_mask.png
testimage_and_mask_result.png true 75 16
```

Textual Output verification:

Expected Output	Your Output
sub-image matched at: 73, 630, 85, 660 sub-image matched at: 120, 310, 132, 340 sub-image matched at: 202, 677, 214, 707 sub-image matched at: 226, 864, 238, 894 sub-image matched at: 274, 67, 286, 97 Number of matches: 5	sub-image matched at: 73, 630, 85, 660 sub-image matched at: 120, 310, 132, 340 sub-image matched at: 202, 677, 214, 707 sub-image matched at: 226, 864, 238, 894 sub-image matched at: 274, 67, 286, 97 Number of matches: 5

Visual output verification:

Search Image	Mask Image	Output Image
		

Performance statistics:

#Threads	User Time	Elapsed Time	%CPU
1	35.942±0.013599567	35.958±0.023880055	99%
4	38.77±0.426005217	10.026±0.109853865	386.4%
8	40.144±0.372294617	5.402±0.028309756	743%

Test Case #3

The following performance statistics were collated by using the supplied sample data files `MiamiMarcumCenter.png` and `WindowPane_mask.png` shown in the images further below. The image match was conducted with 50% match with color tolerance of 64.

The program was compiled using the following command line:

```
$ g++ -g -Wall -std=c++14 -O3 -fopenmp -lpng *.cpp -o Homework7
```

The program was run and timings were collected using the following command line:

```
/usr/bin/time -v ./Homework7 MiamiMarcumCenter.png
WindowPane_mask.png marcum_result.png true 50 64
```

Textual Output verification:

Expected Output	Your Output
sub-image matched at: 567, 818, 601, 859 sub-image matched at: 567, 1021, 601, 1062 sub-image matched at: 568, 619, 602, 660 sub-image matched at: 568, 1226, 602, 1267 sub-image matched at: 578, 1791, 612, 1832 sub-image matched at: 582, 1996, 616, 2037 sub-image matched at: 590, 2198, 624, 2239 sub-image matched at: 605, 817, 639, 858 sub-image matched at: 605, 1020, 639, 1061 sub-image matched at: 606, 618, 640, 659 sub-image matched at: 607, 1225, 641, 1266 sub-image matched at: 616, 1791, 650, 1832 sub-image matched at: 620, 1995, 654, 2036 sub-image matched at: 627, 2197, 661, 2238 sub-image matched at: 816, 1209, 850, 1250 Number of matches: 15	sub-image matched at: 567, 818, 601, 859 sub-image matched at: 567, 1021, 601, 1062 sub-image matched at: 568, 619, 602, 660 sub-image matched at: 568, 1226, 602, 1267 sub-image matched at: 578, 1791, 612, 1832 sub-image matched at: 582, 1996, 616, 2037 sub-image matched at: 590, 2198, 624, 2239 sub-image matched at: 605, 817, 639, 858 sub-image matched at: 605, 1020, 639, 1061 sub-image matched at: 606, 618, 640, 659 sub-image matched at: 607, 1225, 641, 1266 sub-image matched at: 616, 1791, 650, 1832 sub-image matched at: 620, 1995, 654, 2036 sub-image matched at: 627, 2197, 661, 2238 sub-image matched at: 816, 1209, 850, 1250 Number of matches: 15

Visual output verification:

Search Image	Mask Image	Output Image
	▪▪▪	

Performance statistics:

#Threads	User Time	Elapsed Time	%CPU
1	644.924±0.401283259	645.076±0.402720971	99%
4	684.906 ±0.594453031	175.498±0.23099256	390%
8	688.518±0.565473312	91.192±0.182963462	755%

Inferences

Briefly (4-5 sentences is sufficient) describe how you concluded that your parallelization is operating correctly from the observations? How can you tell if your parallelization is operating effectively as number of threads/cores is increased?

From the observed outputs, we could conclude that parallelization with OpenMP did not affect the output of the program since the actual output resulted the same as the expected for all three tests (except for the comparison between Mammogram.png and Cancer_mask.png, there is only one or two pixels off the expected output in the actual output hence the difference is minute and negligible). To check whether parallelization is operating effectively can be concluded from the observed collected data from elapsed time and the CPU% used. The elapsed time has obviously reduced proportional to the number of threads being used for each tests. In the case of Mammogram.png vs. Cancer_mask.png, the reduction in elapsed time is obvious as the percentage of CPU increases when more threads are allocated to do the same operations on different data chunks (regions of pixels). The User Time for all tests remain close to elapsed time in serial time. As the number of threads increase, the User Time may bump up a little bit since more time is needed to create threads and allocate tasks to the new threads. The User Time remain almost constant for different tests regardless of the number of threads are used is because the amount of time to execute the instructions and the time to create threads are the same hence User Time remain almost the same as the serial version.