Optimization

Sergio Oyaga & Carlos Mougan

OpenMP

Submitted To:
Ana Sikora
Parallel Programming
Computer Science
Department

Submitted By :
Sergio Oyaga
Carlos Mougan
MSc in Modelling
CRM

1 Motivation

For this assignment the goal is to improve the execution time of Laplace equation code by using OpenMP.

In order to see the different differences in the performance, we chose various loop (in terms of computational cost) and created multiple threads using OpenMP, which would split the iterations into different processes.

We will measure the impact of making the code parallel by using *perf stat* and *TAU*. We will run the code with different parameters such as:

- Serial, 2, 4, 8
- Different sizes of matrix

2 Parallelisation using OpenMP

First we initialize the number of threads that we are going to use with:

```
1     #pragma omp parallel for num_threads(NUM_THREADS)
```

We will use different threads as Serial, 2, 4 & 8. And we compile the code with 'perfstat' flag to be able to see the speed up when we increase the number of threads. The output has the following form.

```
1     Performance counter stats for './laplace1.test 100 500':
2
3        1108,018799      task-clock (msec)         #    0,972 CPUs utilized
4                 78      context-switches          #    0,070 K/sec
5                  0      cpu-migrations            #    0,000 K/sec
6              3.694      page-faults               #    0,003 M/sec
7      3.456.515.693      cycles                    #    3,120 GHz
   (100,00%)
8      1.129.406.293      stalled-cycles-frontend   #   32,67% frontend cycles idle
   (100,00%)
9      1.012.651.460      stalled-cycles-backend    #   29,30% backend  cycles idle
   (100,00%)
10     4.244.597.098      instructions              #    1,23  insns per cycle
11                                                  #    0,27  stalled cycles per insn
   (100,00%)
12     1.297.715.844      branches                  # 1171,204 M/sec
   (100,00%)
13        26.905.987      branch-misses             #    2,07% of all branches
   (100,00%)
14
15       1,140299001 seconds time elapsed
```

Size=1000	Time [s]
Serial	95
2 Threads	49.4
4 Threads	28.6
8Threads	18

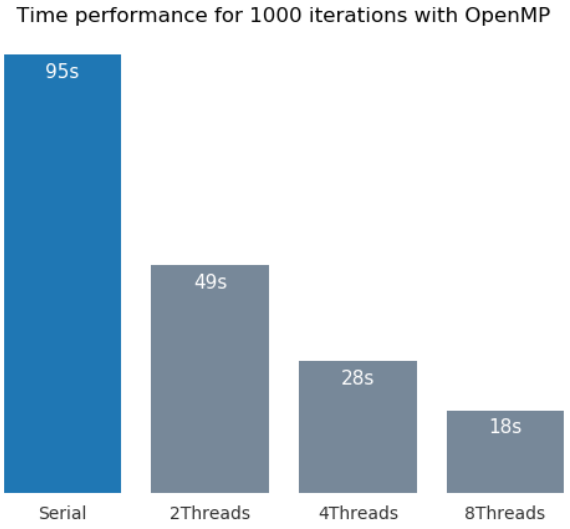

Figure 1: Barplot with the evolution of time performance of OpenMP

It is easy to see that threads and performance are correlated. As more threads we use the more efficient the computation becomes. We can appreciate the great importance of parallelisation and its applications on high-performance computing.

In this figure we can see how the execution times increases proportionally to the mesh size. Since the increasing ratio is extremely high we have decided to plot the logarithmic of the execution time.

This is due to various factors, the most important ones are:

- Rise of number of machines instructions
- Rise of the number of memory access.

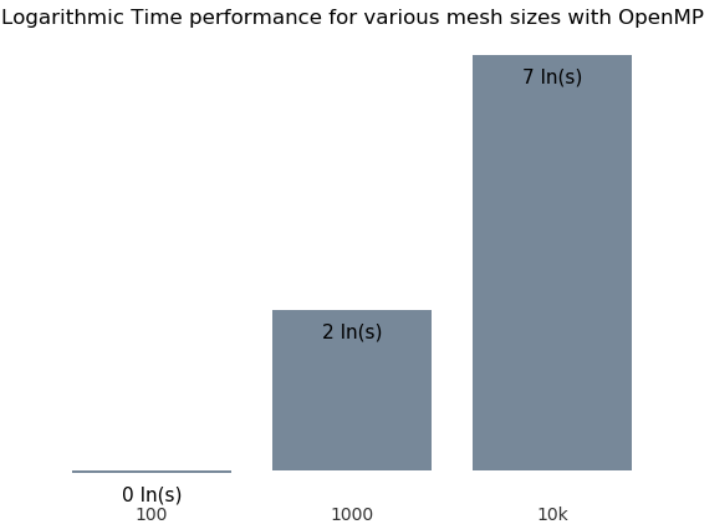

Figure 2: Barplot with the evolution of time performance of OpenMP

3 TAU

In order to use the Tuning and Analysis Utilities (TAU), once we have the package downloaded, we need to export the variables which allow the compiler to find the folders where TAU is located:

```
1    export PATH=\$HOME/my_TAU/\$86\_64/bin:\$PATH
2    export TAU\_MAKEFILE=\$ HOME/my_TAU/\$ 86\_64/lib/Makefile.tau-openmp-opari
3    export TAU\_OPTIONS=-optCompInst
4    export  TAU\_PROFILE=1
```

We also need to export the metrics we want to count. In this case we will use two counters:

```
1    export TAU\_METRICS=PAPI\_L3\_TCM  (counts the total accesses to memory.)
2    export TAU\_METRICS=PAPI\_TOT\_INS (counts the total number of instructions.)
```

Once we have this metrics we can easily use the TAU tool just by compiling and executing as follow:

```
1    tau\_cc.sk -o laplace\_parallel.test laplace\_parallel.c -lm
2    ./laplace_parallel.test 1000 500
```

The analysis can be done for several threads, as we did before. Each thread will give us one profile file ($profile.0.0.*$). To analyze these files the 'paraprof' tool is suitable.

4 Paraprof

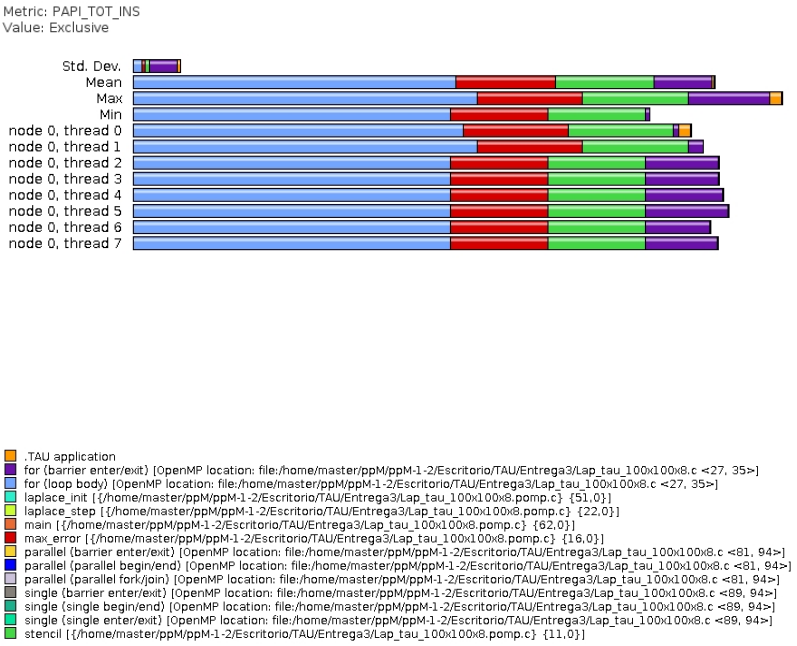

Figure 3: Paraprof Analisis counting the total instructions

Even if the 'Paraprof' tool is very useful, for this pourpose is not worth it. So decided to just push the 'perfstat'.

```
1      Performance counter stats for './laplace1.test 100 500':
2
3         1108,018799      task-clock (msec)         #    0,972 CPUs utilized
4                  78      context-switches          #    0,070 K/sec
5                   0      cpu-migrations            #    0,000 K/sec
6               3.694      page-faults               #    0,003 M/sec
7       3.456.515.693      cycles                    #    3,120 GHz
    (100,00%)
8       1.129.406.293      stalled-cycles-frontend   #   32,67% frontend cycles idle
    (100,00%)
9       1.012.651.460      stalled-cycles-backend    #   29,30% backend  cycles idle
    (100,00%)
10      4.244.597.098      instructions              #    1,23  insns per cycle
11                                                   #    0,27  stalled cycles per insn
    (100,00%)
12      1.297.715.844      branches                  # 1171,204 M/sec
    (100,00%)
13         26.905.987      branch-misses             #    2,07% of all branches
    (100,00%)
14
15        1,140299001 seconds time elapsed
16
17
18
19
20   Performance counter stats for './laplace1.test 1000 500':
21
22        94789,137428      task-clock (msec)         #    0,998 CPUs utilized
23               4.908      context-switches          #    0,052 K/sec
24                   7      cpu-migrations            #    0,000 K/sec
25               4.097      page-faults               #    0,043 K/sec
26     319.337.978.492      cycles                    #    3,369 GHz
    (100,00%)
27     107.295.673.905      stalled-cycles-frontend   #   33,60% frontend cycles idle
    (100,00%)
28      96.402.052.660      stalled-cycles-backend    #   30,19% backend  cycles idle
    (100,00%)
29     352.770.592.298      instructions              #    1,10  insns per cycle
30                                                   #    0,30  stalled cycles per insn
    (100,00%)
31     112.823.446.716      branches                  # 1190,257 M/sec
    (100,00%)
32       3.081.982.180      branch-misses             #    2,73% of all branches
    (100,00%)
33
34        95,021779526 seconds time elapsed
35
36
37   Performance counter stats for './laplace2.test 100 500':
38
39        1366,497351      task-clock (msec)         #    1,577 CPUs utilized
```

```
40                154      context-switches          #    0,113 K/sec
41                  5      cpu-migrations            #    0,004 K/sec
42              3.730      page-faults               #    0,003 M/sec
43      4.096.283.727      cycles                    #    2,998 GHz
    (100,00%)
44      1.383.368.876      stalled-cycles-frontend   #   33,77% frontend cycles idle
    (100,00%)
45      1.148.072.345      stalled-cycles-backend    #   28,03% backend  cycles idle
    (100,00%)
46      5.093.496.128      instructions              #    1,24  insns per cycle
47                                                   #    0,27  stalled cycles per insn
    (100,00%)
48      1.513.491.385      branches                  # 1107,570 M/sec
    (100,00%)
49         27.743.263      branch-misses             #    1,83% of all branches
    (100,00%)
50
51        0,866543161 seconds time elapsed
52
53
54
55   Performance counter stats for './laplace2.test 1000 500':
56
57       98134,690136      task-clock (msec)         #    1,985 CPUs utilized
58              5.017      context-switches          #    0,051 K/sec
59                  7      cpu-migrations            #    0,000 K/sec
60              5.412      page-faults               #    0,055 K/sec
61    317.660.683.254      cycles                    #    3,237 GHz
    (100,00%)
62    105.645.335.035      stalled-cycles-frontend   #   33,26% frontend cycles idle
    (100,00%)
63    121.363.430.667      stalled-cycles-backend    #   38,21% backend  cycles idle
    (100,00%)
64    353.674.493.829      instructions              #    1,11  insns per cycle
65                                                   #    0,34  stalled cycles per insn
    (100,00%)
66    113.051.504.394      branches                  # 1152,003 M/sec
    (100,00%)
67      3.031.100.362      branch-misses             #    2,68% of all branches
    (100,00%)
68
69       49,435932303 seconds time elapsed
70
71
72  Performance counter stats for './laplace4.test 100 500':
73
74        1843,367064      task-clock (msec)         #    2,183 CPUs utilized
75                217      context-switches          #    0,118 K/sec
76                  7      cpu-migrations            #    0,004 K/sec
77              3.795      page-faults               #    0,002 M/sec
78      5.865.769.995      cycles                    #    3,182 GHz
```

```
    (100,00%)
 79      2.053.663.103      stalled-cycles-frontend   #   35,01% frontend cycles idle
    (100,00%)
 80      1.516.617.272      stalled-cycles-backend    #   25,86% backend  cycles idle
    (100,00%)
 81      6.757.888.245      instructions              #    1,15  insns per cycle
 82                                                   #    0,30  stalled cycles per insn
    (100,00%)
 83      1.936.352.668      branches                  # 1050,443 M/sec
    (100,00%)
 84         49.649.682      branch-misses             #    2,56% of all branches
    (100,00%)
 85
 86        0,844574656 seconds time elapsed
 87
 88
 89  Performance counter stats for './laplace4.test 1000 500':
 90
 91     112563,559604      task-clock (msec)         #    3,934 CPUs utilized
 92             5.945      context-switches          #    0,053 K/sec
 93                55      cpu-migrations            #    0,000 K/sec
 94             5.612      page-faults               #    0,050 K/sec
 95   364.234.565.627      cycles                    #    3,236 GHz
    (100,00%)
 96   132.734.800.179      stalled-cycles-frontend   #   36,44% frontend cycles idle
    (100,00%)
 97   130.636.428.868      stalled-cycles-backend    #   35,87% backend  cycles idle
    (100,00%)
 98   354.694.597.844      instructions              #    0,97  insns per cycle
 99                                                   #    0,37  stalled cycles per insn
    (100,00%)
100   113.307.642.770      branches                  # 1006,610 M/sec
    (100,00%)
101     4.689.464.634      branch-misses             #    4,14% of all branches
    (100,00%)
102
103       28,610388706 seconds time elapsed
104
105 Performance counter stats for './laplace4.test 10000 500':
106
107  10647454,034856      task-clock (msec)         #    3,988 CPUs utilized
108           550.608      context-switches          #    0,052 K/sec
109             1.210      cpu-migrations            #    0,000 K/sec
110             8.606      page-faults               #    0,001 K/sec
111 34.574.718.144.033      cycles                    #    3,247 GHz
    (100,00%)
112 11.543.987.592.264      stalled-cycles-frontend   #   33,39% frontend cycles idle
    (100,00%)
113 12.154.214.333.683      stalled-cycles-backend    #   35,15% backend  cycles idle
    (100,00%)
114 35.321.241.059.830      instructions              #    1,02  insns per cycle
```

```
115                                                 #    0,34  stalled cycles per insn
    (100,00%)
116  11.302.117.744.722      branches                # 1061,485 M/sec
    (100,00%)
117     470.216.907.157      branch-misses           #    4,16% of all branches
    (100,00%)
118
119      2669,794667554 seconds time elapsed
120
121  Performance counter stats for './laplace8.test 100 500':
122
123         4363,324391      task-clock (msec)       #    4,643 CPUs utilized
124               1.012      context-switches        #    0,232 K/sec
125                  18      cpu-migrations          #    0,004 K/sec
126               3.925      page-faults             #    0,900 K/sec
127      12.991.247.131      cycles                  #    2,977 GHz
    (100,00%)
128       8.143.708.885      stalled-cycles-frontend #   62,69% frontend cycles idle
    (100,00%)
129       1.862.883.156      stalled-cycles-backend  #   14,34% backend  cycles idle
    (100,00%)
130      10.076.592.840      instructions            #    0,78  insns per cycle
131                                                 #    0,81  stalled cycles per insn
    (100,00%)
132       2.780.403.402      branches                #  637,221 M/sec
    (100,00%)
133          50.428.228      branch-misses           #    1,81% of all branches
    (100,00%)
134
135         0,939698894 seconds time elapsed
136
137  Performance counter stats for './laplace8.test 1000 500':
138
139       139289,018265      task-clock (msec)       #    7,718 CPUs utilized
140              11.274      context-switches        #    0,081 K/sec
141                  18      cpu-migrations          #    0,000 K/sec
142               6.958      page-faults             #    0,050 K/sec
143     450.834.259.223      cycles                  #    3,237 GHz
    (100,00%)
144     270.178.431.620      stalled-cycles-frontend #   59,93% frontend cycles idle
    (100,00%)
145      67.343.708.967      stalled-cycles-backend  #   14,94% backend  cycles idle
    (100,00%)
146     358.697.718.647      instructions            #    0,80  insns per cycle
147                                                 #    0,75  stalled cycles per insn
    (100,00%)
148     114.321.458.480      branches                #  820,750 M/sec
    (100,00%)
149       2.041.619.707      branch-misses           #    1,79% of all branches
    (100,00%)
150
```

```
151        18,047719717 seconds time elapsed
152
153  Performance counter stats for './laplace8.test 10000 500':
154
155     13940998,035297      task-clock (msec)         #    7,710 CPUs utilized
156           1.586.604      context-switches          #    0,114 K/sec
157               5.475      cpu-migrations            #    0,000 K/sec
158               6.749      page-faults               #    0,000 K/sec
159  43.025.230.125.559      cycles                    #    3,086 GHz
     (100,00%)
160  25.762.609.864.363      stalled-cycles-frontend   #   59,88% frontend cycles idle
     (100,00%)
161   6.229.123.843.158      stalled-cycles-backend    #   14,48% backend  cycles idle
     (100,00%)
162  35.340.752.831.504      instructions              #    0,82  insns per cycle
163                                                    #    0,73  stalled cycles per insn
     (100,00%)
164  11.305.860.885.399      branches                  #  810,979 M/sec
     (100,00%)
165     195.898.344.843      branch-misses             #    1,73% of all branches
     (100,00%)
166
167      1808,271352942 seconds time elapsed
```

5 Code

5.1 A* strategies

6 Results

7 Conclusions