Assignment 2 | SPP

In the last assignment, we implemented a `Cache-Oblivious` **Divide and Conquer** Algorithm for Matrix Multiplication. In this assignment, we parallelise our code by using the `OpenMP` library to further improve the performance.

Improvements made to the Previous Code

`-02` optimisation flag

The `-02` optimisation flag being added to the `gcc` compiler made a significant improvement to the speed.

Multi-threading

Parallelism has been achieved through multithreading.

```
omp_set_num_threads(16);
#pragma omp parallel
{
    #pragma omp single
    {
      dnc(current_prod, prev_prod, arr[i+1], current_dim_x, dim[i+1][0], dim[i+1][1], 0, 0, 0);
    }
}
```

We have used two classic OpenMP directives involving the following

Task Parallelisation

In the recursive function for matrix multiplication, the larger matrix is split into smaller ones (along the horizontal axis for first matrix and along the vertical axis for the second matrix) and the smaller parts are multiplied recursively. As these multiplications are independent and operate on different parts of the resultant matrix, these can be executed in parallel. Note that the case when the matrix is split into four parts ($C = A_1B_1 + A_2B_2$) cannot be parallelised, as both function calls operate on the same part of the resultant matrix.

```
#pragma omp task
{
  dnc(c, a, b, rows1, cols1, b2, starting_col_c, starting_col_a, starting_col_b);
}
#pragma omp task
{
  dnc(c, a, b, rows1, cols1, cols2 - b2, starting_col_c + b2, starting_col_a, starting_col_b + b2);
}
#pragma omp taskwait
```

Loop Parallelisation

The `i` loop (outermost loop) in the base case for matrix multiplication is parallelised using the `omp parallel for` directive. This can be done without any additional code as the iterations are independent (no forward dependence) and can proceed in parallel. A similar parallelisation of the `j` and `k` loop can also be done, however, the overhead added is not worth the performance benefit, and thus, the time increases.

```
#pragma omp parallel for
    for(int i=0; i<rows1; ++i){
        for ( int k = starting_col_a; k < starting_col_a +cols1; ++k){
            for ( int j = starting_col_c; j < starting_col_c+cols2; ++j){
                c[i][j] += a[i][k] * b[k-starting_col_a][j];
            }
        }
    }
```

Vectorisation

The computation happening in the `j` loop (innermost loop) has no forward dependency. Thus, the operation can be applied on the entire array together rather than on each element. Vectorisation is used here using the `omp simd` directive.

```
#pragma omp simd
for ( int j = starting_col_c; j < starting_col_c+cols2; ++j){
    c[i][j] += a[i][k] * b[k-starting_col_a][j];
}
```

Performance Analysis and Comparisions

The performance of both serial and parallel versions is analysed using various tools and the results are documented below:

Note: The following analysis and comparision is done with -O2 optimisation for both serial and parallel codes.

Perf

Serialised

```
Samples: 30K of event 'cycles', Event count (approx.): 9279194148
  Children      Self  Command  Shared Object        Symbol
+   86.48%    86.39%  a.out    a.out                [.] dnc
+   11.47%     0.00%  a.out    libc-2.31.so         [.] __libc_start_main
+    8.91%     0.47%  a.out    libc-2.31.so         [.] __isoc99_scanf
+    7.47%     7.24%  a.out    libc-2.31.so         [.] __vfscanf_internal
+    7.29%     0.00%  a.out    [unknown]            [.] 0x0000000000000005
+    6.46%     0.00%  a.out    [unknown]            [.] 0x0000000000000004
+    5.38%     0.00%  a.out    [unknown]            [.] 0x0000000000000007
+    5.10%     0.00%  a.out    [unknown]            [.] 0x0000000000000006
+    4.98%     0.00%  a.out    [unknown]            [k] 0000000000000000
+    4.66%     0.00%  a.out    [unknown]            [.] 0x0000000000000009
+    4.53%     0.00%  a.out    [unknown]            [.] 0x0000000000000008
+    3.72%     0.00%  a.out    [unknown]            [.] 0x0000000000000002
+    3.53%     0.00%  a.out    [unknown]            [.] 0x000000000000000a
+    2.58%     0.00%  a.out    [unknown]            [.] 0x0000000000000003
+    2.32%     0.08%  a.out    libc-2.31.so         [.] __printf_chk
+    1.82%     0.00%  a.out    [unknown]            [.] 0x0000000000000001
+    1.44%     1.41%  a.out    libc-2.31.so         [.] __vfprintf_internal
+    1.28%     1.26%  a.out    libc-2.31.so         [.] __GI_____strtoll_l_internal
+    1.06%     0.00%  a.out    [unknown]            [.] 0x0000561f490c1380
+    1.03%     0.00%  a.out    [unknown]            [.] 0x0000561f49a920c0
+    1.02%     0.00%  a.out    [unknown]            [.] 0x0000561f49e25f00
+    1.00%     0.00%  a.out    [unknown]            [.] 0x0000561f49cf4a40
+    0.99%     0.00%  a.out    [unknown]            [.] 0x0000561f492eb0f0
+    0.96%     0.00%  a.out    [unknown]            [.] 0x0000561f48e5ea00
+    0.96%     0.00%  a.out    [unknown]            [.] 0x0000561f4999b760
+    0.96%     0.00%  a.out    [unknown]            [.] 0x0000561f4982b8a0
+    0.96%     0.00%  a.out    [unknown]            [.] 0x0000561f4926dcf0
+    0.95%     0.00%  a.out    [unknown]            [.] 0x0000561f48de3550
+    0.95%     0.00%  a.out    [unknown]            [.] 0x0000561f4954da70
+    0.95%     0.00%  a.out    [unknown]            [.] 0x0000561f4a103d30
+    0.94%     0.00%  a.out    [unknown]            [.] 0x0000561f4a1f8740
+    0.94%     0.00%  a.out    [unknown]            [.] 0x0000561f49bc3580
+    0.94%     0.00%  a.out    [unknown]            [.] 0x0000561f4a17f1e0
+    0.94%     0.00%  a.out    [unknown]            [.] 0x0000561f48ceeb40
+    0.94%     0.00%  a.out    [unknown]            [.] 0x0000561f49fd2870
+    0.94%     0.00%  a.out    [unknown]            [.] 0x0000561f493a1100
+    0.93%     0.00%  a.out    [unknown]            [.] 0x0000561f496fa3e0
+    0.93%     0.00%  a.out    [unknown]            [.] 0x0000561f49dac9a0
+    0.93%     0.00%  a.out    [unknown]            [.] 0x0000561f499202b0
+    0.92%     0.00%  a.out    [unknown]            [.] 0x0000561f48f8fec0
```

Parallelised

```
Samples: 36K of event 'cycles', Event count (approx.): 9086862388
  Children      Self  Command  Shared Object       Symbol
+   84.68%    84.37%  a.out    a.out               [.] dnc._omp_fn.0
+   59.42%     0.01%  a.out    libgomp.so.1.0.0    [.] GOMP_parallel
+   12.88%     0.00%  a.out    libc-2.31.so        [.] __libc_start_main
+    9.94%     0.49%  a.out    libc-2.31.so        [.] __isoc99_scanf
+    9.03%     0.00%  a.out    [unknown]           [k] 0000000000000000
+    7.98%     7.67%  a.out    libc-2.31.so        [.] __vfscanf_internal
+    7.24%     0.00%  a.out    [unknown]           [.] 0x000055db2958f090
+    7.24%     0.00%  a.out    [unknown]           [.] 0x000055db28a12980
+    7.14%     0.00%  a.out    [unknown]           [.] 0x000055db28de80c0
+    7.14%     0.00%  a.out    [unknown]           [.] 0x000055db28269290
+    7.11%     0.00%  a.out    [unknown]           [.] 0x000055db297784b0
+    7.11%     0.00%  a.out    [unknown]           [.] 0x000055db28a13150
+    7.07%     0.00%  a.out    [unknown]           [.] 0x000055db291bc850
+    7.07%     0.00%  a.out    [unknown]           [.] 0x000055db28a119e0
+    7.06%     0.00%  a.out    [unknown]           [.] 0x000055db28bfeca0
+    7.06%     0.00%  a.out    [unknown]           [.] 0x000055db28268ac0
+    6.99%     0.00%  a.out    [unknown]           [.] 0x000055db28fd14e0
+    6.99%     0.00%  a.out    [unknown]           [.] 0x000055db28269a60
+    6.92%     0.00%  a.out    [unknown]           [.] 0x000055db293a5c70
+    6.92%     0.00%  a.out    [unknown]           [.] 0x000055db28a121b0
+    6.77%     0.00%  a.out    [unknown]           [k] 0x000055db282682f0
+    6.77%     0.00%  a.out    [unknown]           [.] 0x000055db28a15880
+    2.70%     0.06%  a.out    libc-2.31.so        [.] __printf_chk
+    2.29%     0.00%  a.out    [unknown]           [.] 0xfffffffffffa1c3
+    2.26%     0.00%  a.out    [unknown]           [.] 0xfffffffffd305547
+    2.25%     0.00%  a.out    [unknown]           [.] 0x000000000139001f
+    2.23%     0.00%  a.out    [unknown]           [.] 0x0000000001c2b489
+    2.23%     0.00%  a.out    [unknown]           [.] 0xffffffffffff0b06
+    2.21%     0.00%  a.out    [unknown]           [.] 0xfffffffffe7e1dff
+    2.20%     0.00%  a.out    [unknown]           [.] 0x000000000235109c
+    2.10%     0.00%  a.out    [unknown]           [.] 0xfffffffffffff958d
+    2.07%     0.00%  a.out    [unknown]           [.] 0x0000000000695fad
+    2.06%     0.00%  a.out    [unknown]           [.] 0xffffffffffffed642
+    2.01%     0.00%  a.out    [unknown]           [.] 0x0000000000013b69
+    1.99%     0.00%  a.out    [unknown]           [.] 0x0000000001cc34be
+    1.96%     0.00%  a.out    [unknown]           [.] 0x000000000002f589
+    1.92%     0.00%  a.out    [unknown]           [.] 0x00000000037ed12d
+    1.90%     0.00%  a.out    [unknown]           [.] 0xffffffffffff5fdc
+    1.77%     1.71%  a.out    libc-2.31.so        [.] __vfprintf_internal
+    1.75%     0.00%  a.out    [unknown]           [.] 0xfffffffff8416af
```

We can clearly see the `GOMP_parallel` takes a significant time during the runtime. This depicts the overhead in the creation of multiple threads. To reduce this thread, we coarsen our alogorithm to 128 from 32.

Valgrind

Serialised

```
pavani@pavani-Asus:~/SPP-assignment$ valgrind --tool=cachegrind ./a.out < 90.txt > 90_out.txt
==29955== Cachegrind, a cache and branch-prediction profiler
==29955== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==29955== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==29955== Command: ./a.out
==29955==
--29955-- warning: L3 cache found, using its data for the LL simulation.
==29955== brk segment overflow in thread #1: can't grow to 0x483f000
==29955== (see section Limitations in user manual)
==29955== NOTE: further instances of this message will not be shown
98980359

==29955==
==29955== I   refs:      22,134,386,347
==29955== I1  misses:             1,348
==29955== LLi misses:             1,335
==29955== I1  miss rate:           0.00%
==29955== LLi miss rate:           0.00%
==29955==
==29955== D   refs:      10,454,952,988  (9,907,353,454 rd   + 547,599,534 wr)
==29955== D1  misses:        41,957,351  (   41,017,237 rd   +     940,114 wr)
==29955== LLd misses:         3,348,410  (    2,469,886 rd   +     878,524 wr)
==29955== D1  miss rate:            0.4% (          0.4%     +         0.2%  )
==29955== LLd miss rate:            0.0% (          0.0%     +         0.2%  )
==29955==
==29955== LL refs:           41,958,699  (   41,018,585 rd   +     940,114 wr)
==29955== LL misses:          3,349,745  (    2,471,221 rd   +     878,524 wr)
==29955== LL miss rate:             0.0% (          0.0%     +         0.2%  )
pavani@pavani-Asus:~/SPP-assignment$
```

Parallelised

```
pavani@pavani-Asus:~/SPP-assignment$ valgrind --tool=cachegrind ./a.out < 90.txt > 90_out.txt
==29751== Cachegrind, a cache and branch-prediction profiler
==29751== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==29751== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==29751== Command: ./a.out
==29751==
--29751-- warning: L3 cache found, using its data for the LL simulation.
==29751== brk segment overflow in thread #1: can't grow to 0x483f000
==29751== (see section Limitations in user manual)
==29751== NOTE: further instances of this message will not be shown
==29751==
==29751== I   refs:      14,986,773,863
==29751== I1  misses:             2,154
==29751== LLi misses:             2,124
==29751== I1  miss rate:           0.00%
==29751== LLi miss rate:           0.00%
==29751==
==29751== D   refs:       4,602,345,720  (3,281,064,805 rd   + 1,321,280,915 wr)
==29751== D1  misses:       386,679,026  (  385,793,884 rd   +       885,142 wr)
==29751== LLd misses:         3,530,125  (    2,647,542 rd   +       882,583 wr)
==29751== D1  miss rate:            8.4% (         11.8%     +           0.1%  )
==29751== LLd miss rate:            0.1% (          0.1%     +           0.1%  )
==29751==
==29751== LL refs:          386,681,180  (  385,796,038 rd   +       885,142 wr)
==29751== LL misses:          3,532,249  (    2,649,666 rd   +       882,583 wr)
==29751== LL miss rate:             0.0% (          0.0%     +           0.1%  )
pavani@pavani-Asus:~/SPP-assignment$
```

The cache hits in the parallel version are less compared to that of the serialised version. This is primarily due to the coarsening of the algorithm in parallel version to optimise cache performance.

GProf

Serialised

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total           
 time   seconds   seconds    calls  Ts/call  Ts/call  name    
101.17      2.07     2.07                             dnc
```

```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.48% of 2.07 seconds

index % time    self  children    called     name
                               98301             dnc [1]
[1]    100.0    2.07    0.00       0+98301   dnc [1]
                               98301             dnc [1]
-----------------------------------------------
```

Parallelised

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total           
 time   seconds   seconds    calls  Ts/call  Ts/call  name    
100.53      1.07     1.07                             frame_dummy
```

```
Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.94% of 1.07 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    1.07    0.00                 frame_dummy [1]
-----------------------------------------------
                                 380             dnc [5]
[5]      0.0    0.00    0.00       0+380     dnc [5]
                                 380             dnc [5]
-----------------------------------------------
```

SpeedUp Graph