Programming Assignment Report 4

Matrix Multiplication with Linux Performance Tools

Approach:

The instructions were very clear, and all the instructions were followed step by step:

1. First, the perf tools were installed
2. Next, the CPU counters were observed
3. Then the naïve and interchange algorithms were run with the performance tools for matrix size 500 and 1000
4. Next, the hotspots were accessed, and the highest cost functions were recorded (anything in red since that meant the largest time)

Insight Gained and Problems:

The biggest problem was having root permission on fusion. Since I had access to the student account, the perf record and perf report steps would not produce the right symbols; this was overlooked since enough information was there to navigate to the right process and see function costs.

Some insights gained were that this tool is very helpful in quantifying how good the optimization of the algorithms is. Even though I knew row-based matrix access in the inner loop is better than column-based since the cache is loaded rows when it loads in bulk, seeing the number of misses is reduced to a quantifiable amount was interesting. There were more CPU counters on the machine we could have used to do further analysis which was not used in this lab (per instruction). I would like to try a few and see more quantifiable results. Also, there exists a triangular optimization for the multiplication which I would also like to see would pan out versus interchange.

Optimizations:

The code was already provided so no optimizations were done. Three algorithms were provided: naïve, Interchange and Triangular. Only naïve and Interchange were used:

```
#ifdef NAIVE
//NAIVE: 2D matrix-matrix multiplication
__attribute__((noinline)) void compute_naive(doul
  for (int i = 0 ; i < matrix_size; i++) {
    for (int j = 0;  j < matrix_size; j++) {
      for (int k = 0; k < matrix_size; k++) {
  C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
}
```

```
#elif INTERCHANGE
//INTERCHANGE: 2D matrix-matrix multiplication
__attribute__((noinline)) void compute_interchange(doul
  for (int i = 0 ; i < matrix_size; i++) {
    for (int k = 0; k < matrix_size; k++) {
      for (int j = 0;  j < matrix_size; j++) {
  C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
}
```

Results & Analysis:

Exercise 1

```
jjoyson@fusion2:~/cs546$ perf list

List of pre-defined events (to be used in -e):

  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  ref-cycles                                         [Hardware event]

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]

  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  LLC-load-misses                                    [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-store-misses                                   [Hardware cache event]
  LLC-stores                                         [Hardware cache event]
  branch-load-misses                                 [Hardware cache event]
  branch-loads                                       [Hardware cache event]
  dTLB-load-misses                                   [Hardware cache event]
  dTLB-loads                                         [Hardware cache event]
  dTLB-store-misses                                  [Hardware cache event]
  dTLB-stores                                        [Hardware cache event]
  iTLB-load-misses                                   [Hardware cache event]
  iTLB-loads                                         [Hardware cache event]
  node-load-misses                                   [Hardware cache event]
  node-loads                                         [Hardware cache event]
  node-store-misses                                  [Hardware cache event]
  node-stores                                        [Hardware cache event]

  branch-instructions OR cpu/branch-instructions/    [Kernel PMU event]
  branch-misses OR cpu/branch-misses/                [Kernel PMU event]
  bus-cycles OR cpu/bus-cycles/                      [Kernel PMU event]
  cache-misses OR cpu/cache-misses/                  [Kernel PMU event]
  cache-references OR cpu/cache-references/          [Kernel PMU event]
  cpu-cycles OR cpu/cpu-cycles/                      [Kernel PMU event]
  cstate_core/c3-residency/                          [Kernel PMU event]
  cstate_core/c6-residency/                          [Kernel PMU event]
  cstate_core/c7-residency/                          [Kernel PMU event]
  cstate_pkg/c2-residency/                           [Kernel PMU event]
  cstate_pkg/c3-residency/                           [Kernel PMU event]
  cstate_pkg/c6-residency/                           [Kernel PMU event]
  cstate_pkg/c7-residency/                           [Kernel PMU event]
  cycles-ct OR cpu/cycles-ct/                        [Kernel PMU event]
  cycles-t OR cpu/cycles-t/                          [Kernel PMU event]
  el-abort OR cpu/el-abort/                          [Kernel PMU event]
  el-capacity OR cpu/el-capacity/                    [Kernel PMU event]
  el-commit OR cpu/el-commit/                        [Kernel PMU event]
  el-conflict OR cpu/el-conflict/                    [Kernel PMU event]
  el-start OR cpu/el-start/                          [Kernel PMU event]
  instructions OR cpu/instructions/                  [Kernel PMU event]
  intel_bts//                                        [Kernel PMU event]
  intel_cqm/llc_occupancy/                           [Kernel PMU event]
  intel_cqm/local_bytes/                             [Kernel PMU event]
  intel_cqm/total_bytes/                             [Kernel PMU event]
  intel_pt//                                         [Kernel PMU event]
  mem-loads OR cpu/mem-loads/                        [Kernel PMU event]
  mem-stores OR cpu/mem-stores/                      [Kernel PMU event]
  msr/aperf/                                         [Kernel PMU event]
  msr/mperf/                                         [Kernel PMU event]
  msr/smi/                                           [Kernel PMU event]
  msr/tsc/                                           [Kernel PMU event]
  tx-abort OR cpu/tx-abort/                          [Kernel PMU event]
  tx-capacity OR cpu/tx-capacity/                    [Kernel PMU event]
  tx-commit OR cpu/tx-commit/                        [Kernel PMU event]
  tx-conflict OR cpu/tx-conflict/                    [Kernel PMU event]
  tx-start OR cpu/tx-start/                          [Kernel PMU event]
  uncore_imc_0/cas_count_read/                       [Kernel PMU event]
  uncore_imc_0/cas_count_write/                      [Kernel PMU event]
  uncore_imc_0/clockticks/                           [Kernel PMU event]
  uncore_imc_1/cas_count_read/                       [Kernel PMU event]
  uncore_imc_1/cas_count_write/                      [Kernel PMU event]
  uncore_imc_1/clockticks/                           [Kernel PMU event]
  uncore_imc_2/cas_count_read/                       [Kernel PMU event]
  uncore_imc_2/cas_count_write/                      [Kernel PMU event]
  uncore_imc_2/clockticks/                           [Kernel PMU event]
  uncore_imc_3/cas_count_read/                       [Kernel PMU event]
  uncore_imc_3/cas_count_write/                      [Kernel PMU event]
  uncore_imc_3/clockticks/                           [Kernel PMU event]
  uncore_imc_4/cas_count_read/                       [Kernel PMU event]
  uncore_imc_4/cas_count_write/                      [Kernel PMU event]
  uncore_imc_4/clockticks/                           [Kernel PMU event]

  rNNN                                               [Raw hardware event descrip
  cpu/t1=v1[,t2=v2,t3 ...]/modifier                  [Raw hardware event descrip
```

Exercise 2

1.

```
jjoyson@fusion2:~/cs546$ g++ -g -fno-omit-frame-pointer -O3 -DNAIVE matmul_2D.cp
p -o mm_naive.out
```

2.

3.

```
jjoyson@fusion2:~/cs546$ perf stat -e cpu-cycles,instructions,L1-dcache-loads,L1-dc
ache-load-misses,L1-dcache-stores ./mm_naive.out 500
using matrix size:500

 Performance counter stats for './mm_naive.out 500':

     5,777,835,295      cpu-cycles

    10,056,070,359      instructions              #    1.74  insns per cycle

     3,765,289,941      L1-dcache-loads

     1,506,723,905      L1-dcache-load-misses     #   40.02% of all L1-dcache hits

     1,258,020,173      L1-dcache-stores


       2.142155310 seconds time elapsed
```

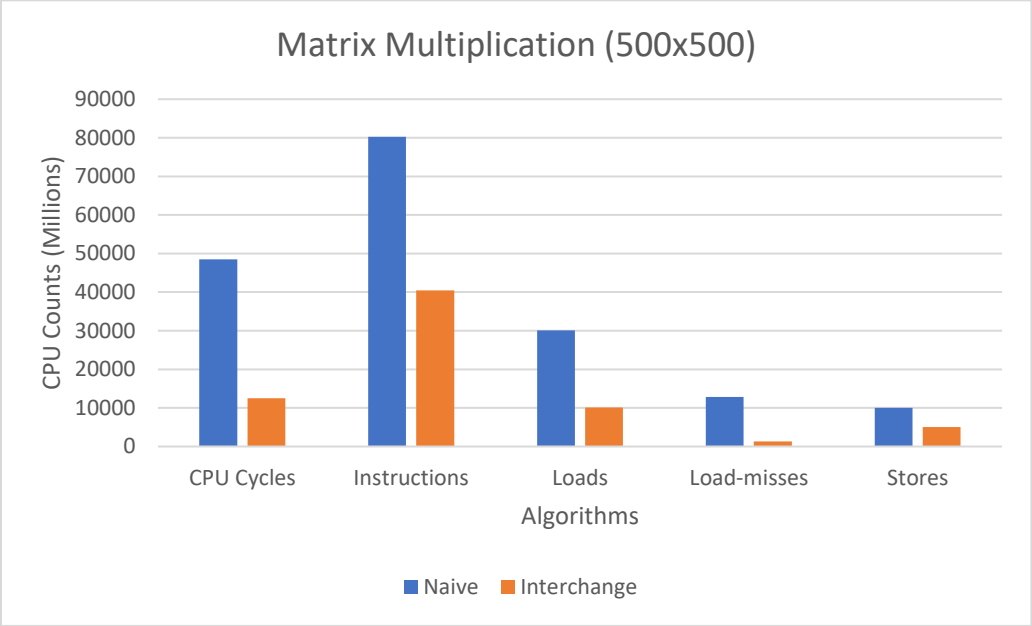
4.

```
jjoyson@fusion2:~/cs546$ g++ -g -fno-omit-frame-pointer -O3 -DINTERCHANGE matmul_2D
.cpp -o mm_interchange.out
```

5.

```
jjoyson@fusion2:~/cs546$ perf stat -e cpu-cycles,instructions,L1-dcache-loads,L1-dc
ache-load-misses,L1-dcache-stores ./mm_interchange.out 500
using matrix size:500

 Performance counter stats for './mm_interchange.out 500':

     1,556,858,964      cpu-cycles

     5,123,307,068      instructions              #    3.29  insns per cycle

     1,276,915,404      L1-dcache-loads

       162,814,710      L1-dcache-load-misses     #   12.75% of all L1-dcache hits

       632,429,038      L1-dcache-stores


       0.741409403 seconds time elapsed
```

6. Compare the numbers for both cases.



The CPU cycles are much lower (27% of naïve version) for the interchange version. Which leads to lower elapsed time. The number of instructions is much lower (51% of naïve version) for the interchange version. Therefore, there are fewer loads. There is much lower (11% of naïve version) LL1 load misses, which indicates better data locality; the loads and stores are significantly less due to this.

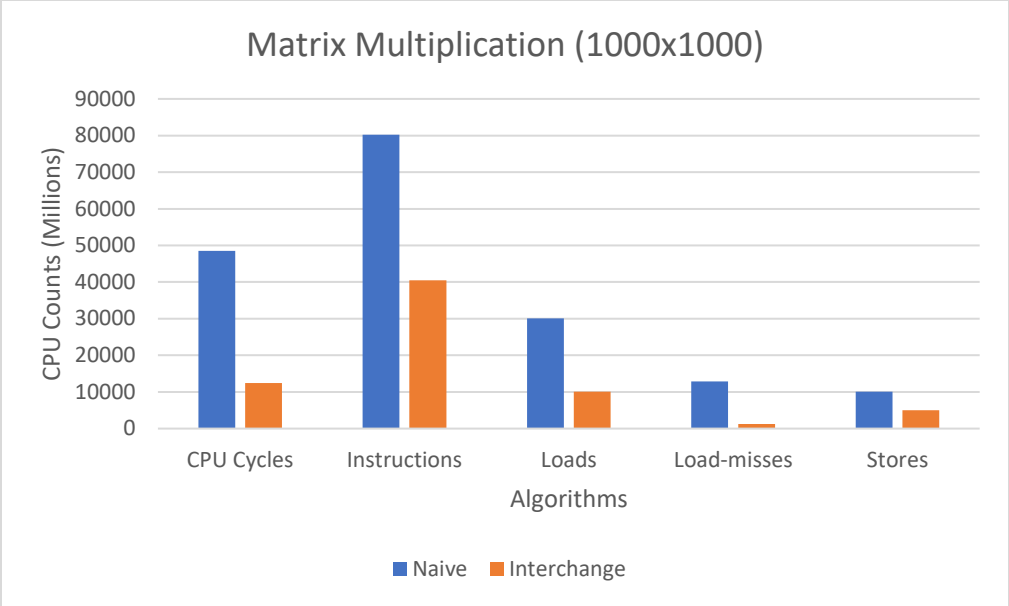
Based on the code, the interchange function has the inner most loop access one value (A[i][k]) and 2 matrix size values (B[k][j] and C[i][j]). Thus, A[i][k] is always accessed since the inner loop only increments j (impacting B[k][j] and C[i][j]). The cache doesn't need to clear A[i][k] and can have better locality since the row of the matrix access on B & C is static (in B[k][j] and C[i][j] only j changes in the inner loop).

The naïve functions also has the innermost loop access 2 matrix size values (A[i][k] and B[k][j]) and on value (C[i][j]). Once again C[i][j] is always accessed since the inner loop only increments k (impacting A[i][k] and B[k][j]). The difference here is that even though there is row-based access on A (in A[i][k] only the column is impacted by the iterations) for a better locality, but access on B is horrible (B[k][j] has its row being varied in each access). This is because when the matrix is loaded into the cache, the temporal locality is increased when the elements are the same rows are access since the rows are loaded together. When the rows are varied, we cannot take advantage of temporal locality and more misses are produced (the cache needs to be cleared for new loading).

Exercise 3

```
jjoyson@fusion2:~/cs546$ perf stat -e cpu-cycles,instructions,L1-dcache-loads,L1-dc
ache-load-misses,L1-dcache-stores ./mm_naive.out 1000
using matrix size:1000

 Performance counter stats for './mm_naive.out 1000':

    48,514,796,915      cpu-cycles

    80,242,238,703      instructions              #    1.65  insns per cycle

    30,068,332,707      L1-dcache-loads

    12,876,570,536      L1-dcache-load-misses     #   42.82% of all L1-dcache hits

    10,037,570,578      L1-dcache-stores


      16.368324196 seconds time elapsed
```

```
jjoyson@fusion2:~/cs546$ perf stat -e cpu-cycles,instructions,L1-dcache-loads,L1-dc
ache-load-misses,L1-dcache-stores ./mm_interchange.out 1000
using matrix size:1000

 Performance counter stats for './mm_interchange.out 1000':

    12,471,941,652      cpu-cycles

    40,491,854,272      instructions              #    3.25  insns per cycle

    10,109,058,912      L1-dcache-loads

     1,277,546,763      L1-dcache-load-misses     #   12.64% of all L1-dcache hits

     5,031,206,643      L1-dcache-stores


       4.371965026 seconds time elapsed
```



The CPU cycles are much lower (26% of naïve version) for the interchange version. Which leads to lower elapsed time. The number of instructions is much lower (50% of naïve version) for the interchange version. Therefore, there are fewer loads. There is much lower (10% of naïve version) LL1 load misses, which indicates better data locality; the loads and stores are significantly less due to this.

As the matrix sizes increased, the percent of the difference between naïve and interchange are the same. This means the Interchange algorithm optimization is scalable.

Exercise 4

```
jjoyson@fusion2:~/cs546$ perf record -g ./mm_interchange.out 500
WARNING: Kernel address maps (/proc/{kallsyms,modules}) are restricted,
check /proc/sys/kernel/kptr_restrict.

Samples in kernel functions may not be resolved if a suitable vmlinux
file is not found in the buildid cache or in the vmlinux path.

Samples in kernel modules won't be resolved at all.

If some relocation was applied (e.g. kexec) symbols may be misresolved
even with a suitable vmlinux or kallsyms file.

Cannot read kernel map
Couldn't record kernel reference relocation symbol
Symbol resolution may be skewed if relocation was used (e.g. kexec).
Check /proc/kallsyms permission or run as root.
using matrix size:500
[ perf record: Woken up 2 times to write data ]
[ perf record: Captured and wrote 0.253 MB perf.data (2982 samples) ]
```

Not root user so symbol naming problems but doesn't matter in cost calculations

```
Samples: 2K of event 'cycles:pp', Event count (approx.): 1591130641
  Children      Self  Command          Shared Object        Symbol
+   99.32%     0.00%  mm_interchange.  mm_interchange.out   [.] main
+   99.32%     0.00%  mm_interchange.  libc-2.23.so         [.] __libc_start_main
+   99.32%     0.00%  mm_interchange.  [unknown]            [.] 0x09f6258d4c544155
-   98.36%    98.30%  mm_interchange.  mm_interchange.out   [.] compute_interchange
   - 98.30% 0x9f6258d4c544155
        __libc_start_main
        main
        compute_interchange
   + 0.06% compute_interchange
+    0.39%     0.00%  mm_interchange.  libc-2.23.so         [.] 0xffff808957022786
+    0.29%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff81867b78
+    0.28%     0.28%  mm_interchange.  mm_interchange.out   [.] init_matrix_2D
+    0.27%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff8106f062
+    0.23%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff8106edd4
+    0.20%     0.20%  mm_interchange.  libc-2.23.so         [.] random
+    0.18%     0.18%  mm_interchange.  libc-2.23.so         [.] random_r
+    0.18%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811f14fa
+    0.18%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811cdee2
+    0.14%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811a42b9
+    0.12%     0.12%  mm_interchange.  [unknown]            [k] 0xffffffff81419807
+    0.08%     0.08%  mm_interchange.  libc-2.23.so         [.] rand
+    0.08%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff81865bf7
+    0.08%     0.08%  mm_interchange.  [unknown]            [k] 0xffffffff81003c23
+    0.08%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff81865fd4
+    0.07%     0.00%  mm_interchange.  ld-2.23.so           [.] 0xffff8089568580db
+    0.07%     0.00%  mm_interchange.  ld-2.23.so           [.] 0xffff80895686d8ad
+    0.07%     0.00%  mm_interchange.  libc-2.23.so         [.] 0xffff808957112ab3
+    0.07%     0.07%  mm_interchange.  libc-2.23.so         [.] 0x0000000000172ab3
+    0.06%     0.06%  mm_interchange.  [unknown]            [.] 0xffffffff81865ca7
+    0.06%     0.06%  mm_interchange.  [unknown]            [k] 0x00007f76a90e2786
+    0.06%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff8105552e
+    0.06%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff8186887b
+    0.05%     0.00%  mm_interchange.  ld-2.23.so           [.] 0xffff80895686079d
+    0.05%     0.00%  mm_interchange.  libc-2.23.so         [.] 0xffff808957112ab7
+    0.05%     0.05%  mm_interchange.  libc-2.23.so         [.] 0x0000000000172ab7
+    0.05%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811e177d
+    0.05%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811c9516
+    0.05%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811ca85c
+    0.05%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811d28d2
+    0.05%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811d4cbd
+    0.05%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811d537a
+    0.05%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff81864f9b
+    0.05%     0.00%  mm_interchange.  libc-2.23.so         [.] brk
+    0.05%     0.05%  mm_interchange.  [unknown]            [k] 0xffffffff811aab8f
+    0.04%     0.00%  mm_interchange.  [unknown]            [.] 0000000000000000
+    0.04%     0.00%  mm_interchange.  [unknown]            [.] 0xffffffff811cbedb
```

```
compute_interchange  /export/home/jjoyson/cs546/mm_interchange.out
                       addsd  (%rdx),%xmm0
                       movsd  %xmm0,(%rdx)
                  114: cmp    $0x1,%r12d
                     ↓ je     1a8
  0.21               movsd  (%rcx),%xmm1
  0.07               mov    -0x48(%rbp),%rax
                     }

                     #elif INTERCHANGE
                     //INTERCHANGE: 2D matrix-matrix multiplication
                     __attribute__((noinline)) void compute_interchange(double **A, double **B, doubl
                       for (int i = 0 ; i < matrix_size; i++) {
                       xor    %edi,%edi
  0.04               unpckl %xmm1,%xmm1
  0.04               lea    (%rsi,%rax,1),%r15
                     xor    %eax,%eax
                     nop
                       for (int k = 0; k < matrix_size; k++) {
                         for (int j = 0;  j < matrix_size; j++) {
                           C[i][j] += A[i][k] * B[k][j];
 15.39  138:         movupd (%r15,%rax,1),%xmm0
 14.13               add    $0x1,%edi
 15.50               mulpd  %xmm1,%xmm0
 19.85               addpd  (%r8,%rax,1),%xmm0
 14.41               movaps %xmm0,(%r8,%rax,1)
 10.51               add    $0x10,%rax
  0.07               cmp    %edi,%r11d
  8.47             ↑ ja     138
                     cmp    %r12d,-0x4c(%rbp)
  0.07             ↓ je     178
                     #elif INTERCHANGE
                     //INTERCHANGE: 2D matrix-matrix multiplication
                     __attribute__((noinline)) void compute_interchange(double **A, double **B, doubl
                       for (int i = 0 ; i < matrix_size; i++) {
                         for (int k = 0; k < matrix_size; k++) {
                           for (int j = 0;  j < matrix_size; j++) {
                       movslq -0x50(%rbp),%rax
                             C[i][j] += A[i][k] * B[k][j];
                  163: movsd  (%rsi,%rax,8),%xmm0
                       lea    (%rdx,%rax,8),%rdi
                       mulsd  (%rcx),%xmm0
                       addsd  (%rdi),%xmm0
                       movsd  %xmm0,(%rdi)
  0.07 178:          add    $0x1,%r9
  0.04               mov    %r10,%rcx

                     #elif INTERCHANGE
```

Ordering these by cost (most costliest to least):

1. addpd (19.85)
2. mulpd (15.50)
3. movupd (15.39)
4. movaps (14.41)
5. add (14.13, 10.51)
6. ja (8.47)

All the other functions are below 1.00 so the cost is negligible.