CS 491/591: High Performance Computing Project 2

Performance Optimization via Cache Reuse

Due date: 11:59 pm, February 24th, 2019

Note: You need to upload a pdf report for the project into Blackboard Learn system. Please also upload all your source codes and a makefile as a tar file into Blackboard Learn system so that our grader can verify what you achieved in your report. Please send an email to Jiannan Tian (jtian10@crimson.ua.edu) if you have any issues with your Pantarhei account.

Suppose your data cache has 60 lines and each line can hold 10 doubles. You are performing a matrix-matrix multiplication (**C=C+A*B**) with square matrices of size **10000X10000** and **10X10** respectively. Assume data caches are only used to cache matrix elements which are doubles. The cache replacement rule is ***least recently used first***. Assume no registers can be used to cache intermediate computing results. One-dimensional arrays are used to represent matrices with the row major order.

```
/* ijk – simple triple loop algorithm with simple single register reuse*/
for (i=0; i<n; i++)
   for (j=0; j<n; j++) {
         register double r=c[i*n+j];
         for (k=0; k<n; k++)
                 r += a[i*n+k] * b[k*n+j];
         c[i*n+j]=r;
}
```

```
/* ijk – blocked version algorithm*/
for (i = 0; i < n; i+=B)
   for (j = 0; j < n; j+=B)
         for (k = 0; k < n; k+=B)
         /* B x B mini matrix multiplications */
         for (i1 = i; i1 < i+B; i1++)
                 for (j1 = j; j1 < j+B; j1++) {
                         register double r=c[i1*n+j1];
                         for (k1 = k; k1 < k+B; k1++)
                                 r += a[i1*n + k1]*b[k1*n + j1];
                         c[i1*n+j1]=r;
                 }
```

Part 1. (25 points) When matrix-matrix multiplication is performed using the *simple triple-loop* algorithm with single register reuse, there are 6 versions of the algorithm (ijk, ikj, jik, jki, kij, kji). Calculate the **number** of read cache misses for **each** element in **each** matrix for **each** version of the algorithm when the sizes of the matrices are **10000X10000** and **10X10** respectively. What is the percentage of read cache miss for each algorithm?

Part 2. (25 points) If matrices are partitioned into block matrices with each block being a 10 by 10 matrix, then the matrix-matrix multiplication can be performed using one of the 6 ***blocked version algorithms*** (ijk, ikj, jik, jki, kij, kji). Assume the multiplication of two blocks in the inner three loops uses the same loop order as the three outer loops in the blocked version algorithms. Calculate the **number** of read cache misses for **each** element in **each** matrix for **each** version of the blocked algorithm when the size of the matrices is **10000**. What is the percentage of read cache miss for each algorithm?

Part 3. (25 points) Implement the algorithms in part (1) and (2). Report your execution time on our Pantarhei cluster. Adjust the block size from 10 to other numbers to see what the optimal block size is. Compile your code using the default compiler (gcc-7.3.0) on Pantarhei without optimization tag. Compare and analyze the performance of your codes for n=2048. Please always verify the correctness of your code.

Part 4. (25 points) Improve your implementation by using both cache blocking and register blocking at the same time. Optimize your block sizes. Compile your code using both the default compiler and gcc-5.4.0 with different optimization flags (-O0, -O1, -02, and -O3) respectively. Compare and analyze the performance of your codes for n=2048. Highlight the best performance you achieved. Please always verify the correctness of your code. Note that you can use “module swap gnu7/7.3.0 gnu/5.4.0” to replace the default compiler by gcc-5.4.0.