CS 378: Programming for Performance
Assignment 4: Fast matrix multiply
Due: April 3rd

April 3, 2015

Late submission policy: Submission can be at the most 2 days late. There will be a 10% penalty for each day after the due date (cumulative).

In class, we described the structure of the optimized MMM code produced by ATLAS. The "computational heart" of this code is the mini-kernel that multiplies an NBxNB block of matrix A with an NBxNB block of matrix B into an NBxNB block of matrix C, where NB is chosen so that the working set of this computation fits in the cache. The mini-kernel itself is performed by repeatedly calling a micro-kernel that multiplies an MUx1 column vector of matrix A with a 1xNU row vector of matrix B into an MUxNU block of matrix C. The values of MU and NU are chosen so that the micro-kernel can be performed out of the registers. Pseudocode for the mini-kernel is shown below (note that this code assumes that NB is a multiple of MU and NU).

```
//mini-kernel
for (int j = 0; j < NB; j += MU)
  for(int i = 0; i < NB; i += NU)
    load C[i..i+MU-1, j..j+NU-1] into registers
      for (int k = 0; k < NB; k++)
        //micro-kernel
        load A[i..i+MU-1,k] into registers
        load B[k,j..j+NU-1] into registers
        multiply A's and B's and add to C's
    store C[i..i+MU-1, j..j+NU-1]
```

For each optimization below:

- Compile your code in ICC with flags '-O3 -fp-model precise'.
- Submit your run to the job scheduler on Stampede at TACC - use the 'serial' queue. Since the values you obtain will depend a lot on the machine you use, you must use Stampede for the numbers you report.

- Report the performance of your code in GFLOPS. The number of floating point operations in matrix multiplication is $2 * N^3$, where NxN is the size of each matrix and so, FLOPS is given by $2 * N^3/time$ (1 GFLOP = 10^9 FLOPS).
- Run the code to multiply matrices of various sizes (at least 5) and plot a graph of GFLOPS vs. matrix size. Explain your results briefly.

Note: Each optimization is cumulative, i.e., you implement one optimization on top of another.
As a reference, for 4096x4096 size matrices, the performance of ijk matrix multiplication version on Stampede is around 0.29 GFLOPS, the peformance of matrix multiplication using BLAS is around 22 GFLOPS.

Register-blocking 25 points

To measure the impact of register-blocking without cache-blocking, implement register-blocking by writing a function for performing MMM, using the mini-kernel code with NB = N (you should verify that this implements MMM correctly). You can use the results in the Yotov et al. study of the ATLAS system to determine good values for MU and NU, or you can experiment with different values to find good values.
Note: Use values of N that are multiples of your values for MU and NU; otherwise, you will have to write some clean-up code to handle leftover pieces of the matrices.

Cache-blocking 25 points

Modify the above code to implement both register-blocking and cache-blocking. You will have to wrap three loops around the mini-kernel to get a full MMM. Use any method you wish to determine a good value for NB.
Note: Use values of N that are multiples of your values for NB; otherwise, you will have to write some clean-up code to handle leftover pieces of the matrices.

Data copying 25 points

You can improve the performance of your kernel by copying blocks of data into contiguous storage as explained in the Yotov et al. paper. Modify the above code to implement data copying (along with register-blocking and cache-blocking).

Vectorization 25 points

Modify the above code (along with register-blocking, cache-blocking, and data copying) to use vector registers and vector intrinsics instead of scalar registers.

Implementation notes

- In the C programming language, a 2-dimensional array of floats is usually implemented as a 1-dimensional array of 1-dimensional arrays of floats. For our computations, it is better to allocate one big contiguous block of storage to hold all the floats, and then create a 1-dimensional row vector that holds points to the start of each row. You can use the following code for this purpose: it takes the number of rows (x) and columns (y) as parameters, and it returns a pointer to the row vector.

```
float **Allocate2DArray_Offloat(int x, int y)
   {
          int TypeSize = sizeof(float);
          float **ppi            = malloc(x*sizeof(float*));
          float *pool            = malloc(x*y*TypeSize);
          unsigned char *curPtr = pool;
          int i;
          if (!ppi || !pool)
          {  /* Quit if either allocation failed */
                  if (ppi) free(ppi);
                  if (pool) free(pool);
                  return NULL;
          }
          /* Create row vector */
          for(i = 0; i < x; i++)
          {
                  *(ppi + i) = curPtr;
                  curPtr += y*TypeSize;
          }
          return ppi;
   }
```

- Refer PHiPAC coding guidelines for writing "portable assembly language programs" in the C language.

Deliverables

Submit (to canvas) one source code containing all optimizations (cumulative) and a report containing all plots and analysis. Briefly describe a way to verify correctness of your code for your choice of problem size. You will not be given any points if we are not able to verify correctness.

Conflict misses

What are conflict misses? What is cache associativity and how does that impact conflict misses? Briefly answer this in the same report.