CS 6230

<u>Term Project</u>

Due 11:59pm, Friday, 12/8/2023

The term project can be done either alone or in teams of two. When grading, some consideration will be given to whether it was a solo effort or a two-person effort, with higher expectations for projects done by two-person teams.

Numerical libraries implement efficient routines for matrix-matrix multiplication (commonly called GEMM – GEneral Matrix Multiplication), where four variants are implemented:

i) **C=AB**, ii) **C=A^{T}B**, iii) **C=AB^T**, and iv) **C=A^{T}B^T**.

For the term project, these four variants of GEMM are to be optimized by you: 1) Using OpenMP for multi-core CPUs, and 2) Using CUDA for GPUs.

A primary difference between previous optimization exercises for programming assignments and the term project is that of addressing *generality*. For the programming exercises with variants of matrix multiplication, the exercises only involved the "square" case (equal values for all three size parameters for matrix multiplication) and the evaluation involved a single problem size. The term project highlights the challenge of developing high-performance numerical libraries: high performance is desired across a range of different problem sizes and the specific matrix sizes used by an application program invoking the library routine is not known a priori. Thus, adaptivity with respect to the problem size parameters in the call can be beneficial. The adaptivity may involve dynamically choosing parameters such as tile sizes, shapes/sizes of thread blocks, or even different code versions, e.g., representing different loop permutations and/or degrees of loop unrolling. If multiple code versions are used, an automatic dynamic selection mechanism must be implemented that chooses one of the versions based on problem size parameters.

The code versions to be optimized are:

1) **Base Matrix-Matrix Multiplication (C=AB):**

```
for (i=0;i<Ni;i++)
 for (j=0;j<Nj;j++)
  for (k=0;k<Nk;k++)
// C[i][j] += A[i][k]*B[k][j];
   C[i*Nj+j] += A[i*Nk+k]*B[k*Nj+j];
```

2) **Transposed-Nontransposed Matrix-Matrix Multiplication (C=A^{T}B):**

```
for (i=0;i<Ni;i++)
 for (j=0;j<Nj;j++)
  for (k=0;k<Nk;k++)
// C[i][j] += A[k][i]*B[k][j];
   C[i*Nj+j] += A[k*Ni+i]*B[k*Nj+j];
```

3) **Nontransposed-Transposed Matrix-Matrix Multiplication (C=AB^T):**

```
for (i=0;i<Ni;i++)
 for (j=0;j<Nj;j++)
  for (k=0;k<Nk;k++)
// C[i][j] += A[i][k]*B[j][k];
   C[i*Nj+j] += A[i*Nk+k]*B[j*Nk+k];
```

4) **Transposed-Transposed Matrix-Matrix Multiplication (C=A^{T}B^T):**

```
for (i=0;i<Ni;i++)
 for (k=0;k<Nk;k++)
  for (j=0;j<Nj;j++)
// C[i][j] += A[k][i]*B[j][k];
   C[i*Nj+j] += A[k*Ni+i]*B[j*Nk+k];
```

Like the programming assignments, template codes will be provided, along with a number of test cases with varying sizes of the matrices. Here are examples of problem-size variation for test cases:

Ni	Nj	Nk
8*1024	8*1024	16
4096	4096	64
2048	2048	256
1024	1024	1024
256	256	16*1024
64	64	256*1024
16	16	4*1024*1024

Ni	Nj	Nk
9*999	9*999	37
3*999	3*999	111
999	999	999
333	333	9*999
111	111	81*999
37	37	729*999