Matrix Multiplication - Activision CTS: Internship Test

Talha Saruhan

1 Project

In this project, I've implemented multiple methods for multiplying matrices, and relevant utilities. My prime focuses were:

- Cache locality, memory access patterns.
- SIMD, ways to help compiler generate vectorized code without using intrinsics (as ASM was forbidden)
- Cache friendly multithreading

I didn't implement the Strassen's algorithm as I had only heard of the basic idea and didn't know the exact relations (between C11, ... C22, M1, M2, ... M7 and A, B) or how to derive them. I felt like looking it up would be considered as cheating.

2 How to run

Navigate to *x64\Release* and run this command or call "run.bat". If you don't have "tee" command, just delete the last part or install GnuWin32 CoreUtils.

```
for /l %x in (1, 1, 100) do echo %x && (MatrixGenerator.exe && printf "Generated valid ...
    output. Testing...\n" && MatrixMulTester.exe matrixAB.bin MatrixMult.exe matrixA.bin ...
    matrixB.bin matrixAB-out.bin && printf \n\n ) | tee -a out.txt
```

(Note that I've changed the default valid output generator to transposed B method, since default naïve method took too long for big matrices.)

3 A benchmark for large matrices

On my machine (6 core i7-8700K), I've compared my implementation against multithreaded python-numpy which uses C/C++ backend and Intel MKL BLAS library. (Note that both implementations use close to 100% CPU)

```
>>> import numpy as np
>>> a = np.random.randn(10000, 10000)
>>> import time
>>> start = time.time(); b=np.dot(a, a); end=time.time();
>>> end-start
8.877262115478516
```

```
MatrixGenerator.exe && echo generated && MatrixMulTester.exe matrixAB.bin MatrixMult.exe ...
    matrixA.bin matrixB.bin matrixAB-out.bin
a: [10000 10000] | b: [10000 10000]
Generation w/ tranposed mult. took: 161384008 microseconds.
generated
Queued!
Done!
Runtime: 20676630 microseconds.
Correct.
```

My multithreaded implementation is only 2.3 times slower than a professional BLAS package (20.7 seconds vs 8.9 seconds). If I'd implemented Strassen's algorithm, assuming same constants, the program would run $(10^4)^{3-2.8} \approxeq 6.3$ times faster. Obviously Strassen's constant is perceptibly larger, but I think it's safe to assume it would improve the overall performance to a more comparable level.

4 Code details:

4.1 Functions for multiplying matrices

```
const Mat TransposeMat(const Mat& mat)
const Mat ST_NaiveMatMul(const Mat& matA, const Mat& matB)
const Mat ST_TransposedBMatMul(const Mat& matA, const Mat& matB)
const Mat ST_BlockMult(const Mat& matA, const Mat& matB)
const Mat MTMatMul(const Mat& matA, const Mat& matB)
```

I've tried to address vectorization and cache locality in every function, even algorithm wise naïve ones. For more details, each function has a block of comment that explains how it's designed and why so.

4.2 Multithreading utilities

```
Namespace QueryHWCores,
HWLocalThreadPool<NumOfCoresToUse, NumThreadsPerCore>
```

I've also implemented a hardware local thread pool for multithreaded *MTMatMul* function. The pool runs every thread corresponding to a job on the same physical core. Idea is that, on hyperthreaded systems such as mine, 2 threads that work on contiguous parts of memory should live on the same core and share the same L1 and L2 cache.

> Each job is described as an array of N functions. (N=2)
>
> For each job, N threads are created and assigned respective functions.
>
> For a given job, all threads are guaranteed to be on the same physical core.
>
> No two threads from different jobs are allowed on the same physical core.

Basically, when traversing AB matrix in a block by block fashion, we split each block in two and create two threads on a single physical core, each handling half of the block.

4.3 MSVC2017 Build options (over default x64 Release build settings)

- Maximum optimization: /O2
- Favor fast code /Ot
- Enable function level linking: /Gy
- Enable enhanced instruction set: /arch:AVX2
- Floating point model: /fp:fast
- Language: /std:c++17 (for several "if constexpr"s. otherwise can be compiled with C++ 11)

4.4 Links

Code for this project can be found on my GitHub page: github.com/talhasaruhan