Aneesh Pandoh

Configuration

Utilized the Linux Lab machines. The lscpu command reveals that the machine has an 11th-gen core i5 processor with a base clock of 2.70 GHz and 6 cores with 2 threads each meaning **12 total threads**. Based on the Intel product page, **the peak memory bandwidth of the processor is 50 GB/s.**

Architecture: x86_64
CPU(s): 12
Model name: 11th Gen Intel(R) Core(TM) i5-11500 @ 2.70GHz
Thread(s) per core: 2
Core(s) per socket: 6

[HOW TO RUN]

Need CMake to build the project configured for the G++ compiler on Linux. **To compile the program go into the build directory and then run make which creates the "P4" executable.** The CMakeLists.txt includes the OpenMP option, src files, and compiler optimization (as "-03" is not enabled by default on g++) seen below:

```
cmake_minimum_required(VERSION 3.5)
project(P4 VERSION 0.1.0 LANGUAGES C CXX)
set(CMAKE_C_COMPILER "icc")
set(CMAKE_CXX_COMPILER "icpc")
include(CTest)
enable_testing()

set(SOURCES
   ${CMAKE_SOURCE_DIR}/main.cpp
   ${CMAKE_SOURCE_DIR}/MatMatMultiply.cpp
   ${CMAKE_SOURCE_DIR}/MatMatMultiply.h
   ${CMAKE_SOURCE_DIR}/Parameters.h
   ${CMAKE_SOURCE_DIR}/Timer.h
   ${CMAKE_SOURCE_DIR}/Utilities.cpp
   ${CMAKE_SOURCE_DIR}/Utilities.h)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O3 -mavx512f -mkl")
add_executable(P4 ${SOURCES})

find_package(OpenMP)
if(OpenMP_CXX_FOUND)
 target_link_libraries(P4 PUBLIC OpenMP::OpenMP_CXX)
endif()
set(CPACK_PROJECT_NAME ${PROJECT_NAME})
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
include(CPack)
```

Results:

**Using the average time of performance runs*

1 Thread

	16x16 block size	**32x32 block size**	**64x64 block size**
1024x1024 matrix resolution	58.281 ms	58.233 ms	53.408 ms
2048x2048 matrix resolution	717.740 ms	518.846 ms	462.382 ms
4096x4096 matrix resolution	7146.616 ms	4382.321 ms	3659.260 ms

12 Threads

	16x16 block size	**32x32 block size**	**64x64 block size**
1024x1024 matrix resolution	22.914 ms	21.476 ms	20.838 ms
2048x2048 matrix resolution	202.519 ms	155.2428 ms	139.327 ms
4096x4096 matrix resolution	1871.162 ms	1211.520 ms	1087.397 ms

From the data, we can see that increased block size generally decreases the average time taken for the MatMul for almost all resolutions, no matter the number of threads. However, we can see that at smaller resolutions such as 1024, it doesn't make too significant of a difference, this is probably because most of the elements in the matrices are cached anyway in the smaller size, meaning blocking will not improve cache misses by much. We can also see that the increase in resolution increases the time to completion, which seems obvious as there is just more memory to load in and more computations necessary, but we can see that the speed up from utilizing 12 threads of parallelism compared to just a single thread is not close to 1/12. While the speed-up is more significant for larger resolutions, it still looks like the 12 threads are not cutting the work evenly. This may be because every thread has its own cache which means they are all

contending for space within the overall cache as well as the overhead of managing more threads and simply the fact that splitting up some of these tasks doesn't make much of a difference if the amount of loaded in memory are smaller than a cache line worth of space. While we didn't see it in the test laid out, there are also drawbacks of bigger block sizes as if they are bigger than the size of some of the smaller caches, which increases the amount of cache misses.