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 "hw5" 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(hw5 VERSION 0.1.0 LANGUAGES C CXX)
set(CMAKE_C_COMPILER "icc")
set(CMAKE_CXX_COMPILER "icpc")
set(SOURCES
        ${CMAKE_SOURCE_DIR}/CSRMatrix.h
    ${CMAKE_SOURCE_DIR}/CSRMatrixHelper.h
    ${CMAKE_SOURCE_DIR}/Laplacian.cpp
    ${CMAKE_SOURCE_DIR}/Laplacian.h
    ${CMAKE_SOURCE_DIR}/main.cpp
    ${CMAKE_SOURCE_DIR}/DirectSolver.cpp
    ${CMAKE_SOURCE_DIR}/DirectSolver.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(hw5 ${SOURCES})
find_package(OpenMP)
if(OpenMP_CXX_FOUND)
 target_link_libraries(hw5 PUBLIC OpenMP::OpenMP_CXX)
endif()
set(CPACK_PROJECT_NAME ${PROJECT_NAME})
include(CPack)
```

Part A Setup:

Parameters.h

```
#define XDIM 64
#define YDIM 64
#define ZDIM 64
```

No Permutation

```
  //setup own permutations
  std::vector<int> perm(n);
  for (int i=0; i < n; i++) {
      perm[i] = i;
  }
```

Part A Results:

	Nested Dissection:	Minimum Degree Algorithm:	"No-permutation" Permutation:
sparsity of the L factor (# of non-0s)	93089648	188357569	901970427
operations needed for the factorization	243.207535	985.514343	3449.491455
bandwidth achieved during factorization	173.289169	162.913467	6.495938
run-time for solve	0.038997 s	0.079928 s	17.025986 s

Part B Setup:

Parameters.h

```
#define XDIM 64
#define YDIM 64
#define ZDIM 64
```

Duplication for b and x

```
MKL_INT k = 100;

float* flattenedx = &x[0][0][0];
float* flattenedb = &f[0][0][0];

float* bMultiple = new float[n * k];
float* xMultiple = new float[n * k];

#pragma omp parallel for
for (MKL_INT iter = 0; iter < k; iter++) {
    for (MKL_INT niter = 0; niter < n; niter++) {
        bMultiple[iter*n + niter] = flattenedb[niter];
        xMultiple[iter*n + niter] = flattenedx[niter];
    }
}
```

Part B Results:

k = 100

Time spent in direct solver at solve step (solve): 10.254684 s

k = 10

Time spent in direct solver at solve step (solve): 4.002065 s

k = 1

Time spent in direct solver at solve step (solve): 0.860459 s

Summary:

From the results we can see the significant impacts of reordering, as the no permutation version was almost 400x slower than the optimized nested dissection method of ordering. We can see that this is both because the bandwidth decreases, presumably, because the the reordering allows for the CSR matrix to be more easily parallelized. We can also see that the sheer amount of operations decreases as well, which is where we can mainly see the advantage of nested dissection over the min degree algorithm. In Part B we can examine the fact that the scale of the slow down is not linear as the amount of right-hand-sides goes up, as expected, with the given the explanation found in the write-up, “PARDISO can take advantage of the extra right-hand-sides to make the execution more compute-bound”.