PHYS-E0412 Computational Physics :: Homework 9

Due date 19.3.2019 at 10 am

Parallel Sparse Matrix-Vector Products

As the development of modern processors has shifted from increasing clock frequencies to increasing the number of cores and reducing power consumption, it has become increasingly important to design computational algorithms that utilize parallelism. In this exercise we will study the matrix-vector product algorithm for sparse matrices and its parallelization.
For the parallelization task we use the OpenMP API, which consists of a number of compiler directives and library routines. The examples are provided in C++, but it is also possible to use OpenMP in a Fortran program. We will mostly be concerned with the simplest example where a for-loop is parallelized over the iterations:

```
#pragma omp parallel for
for (int i=0; i<N ; ++i )
{
// do something
}
```

Here the #pragma compiler directive tells the compiler that it should share the iterations of the loop among several threads (similarly to parfor in matlab). The maximum number of threads can be set by calling

```
omp_set_num_threads ( NThreads ) ;
```

or by setting the environment variable OMP_NUM_THREADS. It should be noted that this simple way of parallelizing loops only works if the iterations are completely independent. For example, if the same global variable is written in all of the iterations, the result can be almost anything, as the iterations can be executed simultaneously or in any order. (Reading the same variable is ok, however.) As an example problem we use the matrix-vector product, which is an important ingredient in many higher level algorithms. We measure the so-called speedup factor, which is defined as T_s/T_p , where T_s is the time taken by the serial implementation and T_p the time for the parallelized one. Note that, for the measurements to make sense, you should use some machine with at least two cores and preferably no other cpu intensive jobs running.

a) See the program mvmultiply_sparse.cpp. Your first task is to implement a discretization using a sparse matrix for the operator $(-\Delta + 2I)$ in the problem

$$\begin{cases} -\Delta u(x,y) + 2u(x,y) = f(x,y), & (x,y) \in [0,1] \times [0,1] \\ u(x,y) = 0, & \text{on the boundary} \end{cases}$$

Use the five-point difference stencil for the Laplacian operator and the COO format to store your matrix (remember to start by counting the number of non-zero entries in the matrix). The program will convert your matrix into CRS format. You can use material form homework 7. (2 p.)

b) Next, implement the sparse matrix multiplication for the COO and CRS matrix formats in the functions matvec_coo and matvec_crs. Note that you should always get the same result vector from both implementations. This can be checked using the function CompareVectors. (2 p.)

c) The matrix-vector product for the COO format is simple to program, but one of its disadvantages is that it is not so trivial to parallelize. Explain why. Use the compiler directive

```
#pragma omp parallel for
```

to parallelize the implementation for the CRS format, and check if you get any speedup. (1 p.)

d) How many hours did you spend working on this exercise?

Hint: See e.g. the page http://www.mathcs.emory.edu/ cheung/Courses/ 561/Syllabus/3-C/sparse.html and the lecture slides for explanation of the matrix formats.

Compiling C++ codes with OpenMP

On Linux systems the codes can be compiled using the GNU compiler as follows:

```
g++ -O3 -fopenmp mvmultiply_sparse.cpp -o mvmultiply_sparse
```

Here the flag -fopenmp enables the OpenMP support of the compiler. If you forget this flag, the program may still compile, but all OpenMP #pragmas are ignored and parallelization does not work. The -O3 flag enables level 3 compiler optimizations. When benchmarking code it is usually a good idea to keep the optimizations on, because they would typically be enabled in any real world application as well, and they often affect the speed substantially. The OpenMP standard is supported by other compilers also, such as the Microsoft Visual C++ compiler, so it should be possible to do the exercise on Windows as well. If you would like to study the C++ programming language a bit more, there are numer- ous tutorials on the internet, such as this one:
https://en.wikiversity.org/wiki/C++
Return your solutions to MyCourses. Please remember to return your codes as well.