Small Scale Parallel Programming

Problem Scenario

The task at hand is the development of a sparse matrix-vector product kernel. You are required to implement a kernel capable of computing

$$y \leftarrow Ax$$

where A is a sparse matrix stored in the

1. CSR
2. ELLPACK

storage formats (see below). The kernel shall be parallelized to exploit available computing capabilities. The code for both formats shall be developed in C/C++, in both OpenMP and CUDA versions, and shall be tested for correctness against a serial reference implementation. You shall need to develop auxiliary functions to preprocess the matrix data and represent it in the desired format. The code shall be tested for performance on a specified set of matrices; performance tests shall be carried out on the Crescent computing facility.

Test Matrices

The test matrices shall be obtained from the Suite Sparse Matrix Collection at the website `https://sparse.tamu.edu/`. It is recommended to download the matrices in the MatrixMarket format; software for reading matrices from file into memory is available at `http://math.nist.gov/MatrixMarket/`. The matrices are characterized by a number of rows M, number of columns N and number of nonzero entries NZ. Typically, for symmetric matrices only the upper (or lower) triangle is stored on file; for the purposes of this assignment, you should assume that matrices in computer memory are always stored in full, therefore upon reading from file you should reconstruct the missing part of the matrix, and adjust the actual value of NZ accordingly. Some of the matrices are marked in their storage file as “pattern”; for these matrices all of the nonzero coefficient values are implied to be equal to 1.0, and therefore those values are not stored explicitly on file to save disk space.

The test set shall include at least the following matrices:

cage4	Cube_Coup_dt0	FEM_3D_thermal1
mhda416	ML_Laplace	thermal1
mcfe	bcsstk17	thermal2
olm1000	mac_econ_fwd500	thermomech_TK
adder_dcop_32	mhd4800a	nlpkkt80
west2021	cop20k_A	webbase-1M
cavity10	raefsky2	dc1
rdist2	af23560	amazon0302
cant	lung2	af_1_k101
olafu	PR02R	roadNet-PA

although testing with other matrices is also encouraged.

Performance measurements

All performance data shall be obtained by repeated invocation of the kernel for a certain number of times for each matrix, resulting in a measurement of the average and/or median time per kernel invocation.

For all cases, the measure of performance in MFLOPS or GFLOPS will be obtained as

$$FLOPS = \frac{2 \cdot NZ}{T}$$

where NZ is the *number of nonzero entries* in the matrix, adjusted as necessary in the case of symmetric storage on file, and T is the (average/median) time per kernel invocation.

The measurements shall only include the time needed to perform the matrix-vector product operation; input/output from file and data structure preprocessing shall *not* be included in the timings.

For the OpenMP version the code shall be tested with varying number of threads, from 1 up to the maximum number of available cores within a single Crescent node.

Storage formats

The storage formats will be described with code in Matlab.

When translating the codes into C/CUDA, you should pay attention to the actual data access pattern from memory and the resulting data structure adjustments that may be necessary, keeping in mind that Matlab employs storage by columns, whereas C and C++ employ storage by rows.

CSR

Compressed Storage by Rows format. An $M \times N$ matrix with NZ non-zero entries is described by the following data:

M Rows;

N Columns;

IRP(1:M+1) Array of pointers to row start;

JA(1:NZ) Array of column indices;

AS(1:NZ) Array of coefficients;

so that the matrix-vector product in Matlab would be implemented by the following code:

```
for i=1:m
  t=0;
  for j=irp(i):irp(i+1)-1
     t = t + as(j)*x(ja(j));
  end
  y(i) = t;
end
```

As an example, the matrix

$$\begin{pmatrix} 11 & 12 & 0 & 0 \\ 0 & 22 & 23 & 0 \\ 0 & 0 & 33 & 0 \\ 0 & 0 & 43 & 44 \end{pmatrix}$$

would be stored in CSR (assuming 1-base indexing as in Matlab) as:

$$\begin{aligned} M &= 4 \\ N &= 4 \\ IRP &= [1,3,5,6,8] \\ JA &= [1,2,2,3,3,3,4] \\ AS &= [11,12,22,23,33,43,44] \end{aligned}$$

ELLPACK

ELLPACK storage format. An $M \times N$ matrix with NZ non-zero entries and such that $MAXNZ$ is the maximum number of nonzero entries per row across all rows is described by the following data:

M Rows;

N Columns;

MAXNZ Max nonzeros per row

JA(1:M,1:MAXNZ) 2D array of column indices

AS(1:M,1:MAXNZ) 2D array of coefficients;

so that the matrix-vector product in Matlab would be implemented by the following code:

```
for i=1:m
t=0;
for j=1:maxnzr
t = t +  as(i,j)*x(ja(i,j));
end
y(i) = t;
end
```

Rows with a number of nonzeros less than $MAXNZ$ shall be padded with appropriate values, e.g. zeros in AS. As an example, the matrix

$$\begin{pmatrix} 11 & 12 & 0 & 0 \\ 0 & 22 & 23 & 0 \\ 0 & 0 & 33 & 0 \\ 0 & 0 & 43 & 44 \end{pmatrix}$$

would be stored in ELLPACK (assuming 1-base indexing as in Matlab) as:

$$\begin{aligned} M &= 4 \\ N &= 4 \\ MAXNZ &= 2 \\ JA &= \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 3 \\ 3 & 4 \end{pmatrix} \\ AS &= \begin{pmatrix} 11 & 12 \\ 22 & 23 \\ 33 & 0 \\ 43 & 44 \end{pmatrix} \end{aligned}$$

Note that describing the ELLPACK format in Matlab, which has storage by columns, implies that coefficients in the same column are adjacent in memory; reproducing the same data layout in a C-like language means that the matrix would appear to be transposed. This has an effect in the CUDA version: most matrices have a relatively small number of nonzeros per row, so it makes sense to assign only one thread per row. If we store a two-dimensional matrix by rows and then access it by columns with one thread per row, accesses will be uncoalesced; so, the layout in memory should be adjusted to bring consecutive threads to access consecutive entries, i.e. the matrix should be transposed in memory (rows of the C data structures should correspond to columns shown above in Matlab).

Note that for some matrices the amount of padding needed may be so large that they would not fit into the available memory, and that would make it impossible to use the ELLPACK format on those matrices; such matrices (if any) should be clearly identified in the project report.

Deliverables

The students shall have to submit:

- The project code, with appropriate makefiles; the code must compile and run on Crescent;
- A project report, describing the data structures employed; the report shall also include:
- A brief summary design specification and test plan, including correctness and performance testing;
- An analysis of the performance data;

Marking Scheme

40 Software quality & performance;

30 Overall Project Report quality;

10 Design specification and test plan;

20 Performance analysis;

Submission deadline full time students: 9:30 am, February 27th, 2023
Submission deadline part time students: 9:30 am, March 13th, 2023