Technische Universität München
Lehrstuhl Informatik V
M. Bader, A.Pöppl

WS 2015/16
Example assignments
January, 18th 2016

HPC - Algorithms and Applications WS 15/16

Questions from previous exams

This worksheet contains exercises that have been asked in previous exams.

1 Parallel Sparse-Matrix Computation

In Figure 1, you see two suggestions to distribute the elements of a sparse matrix (a 13×13 matrix with 53 non-zeros) to four parallel partitions to be used for distributed-memory parallelisation.

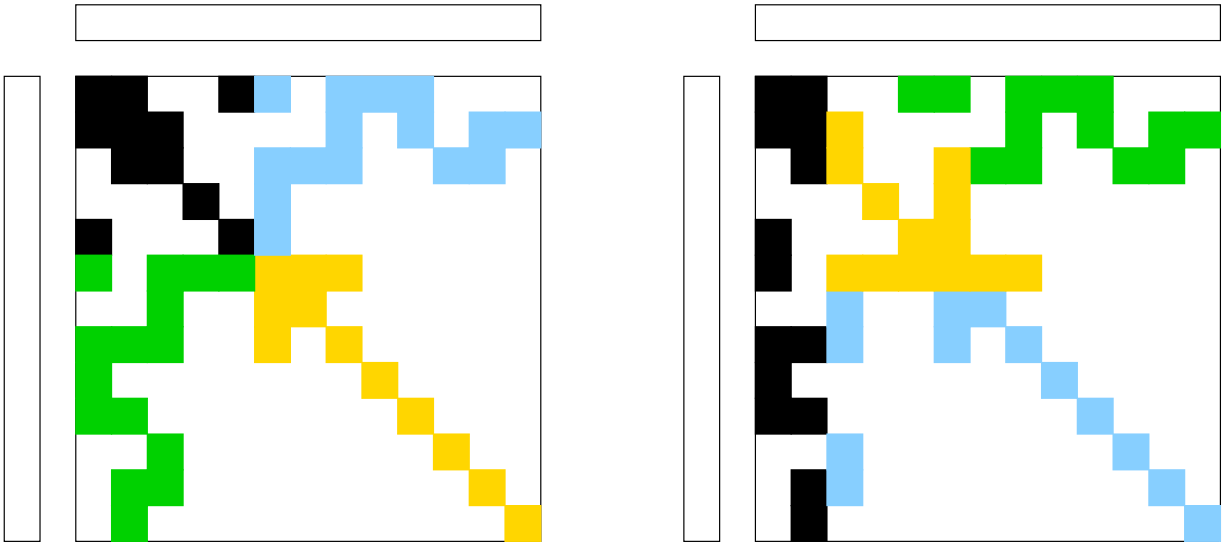

Figure 1: Two suggestions for the partitioning of a sparse matrix

a) For each of the matrix distributions, suggest an efficient(!) distribution of the elements of the input and output vectors – mark the elements in Fig. 1 by the characters 'k' (black), 'b' (blue), 'g' (green), and 'o' (orange/yellow), respectively. If necessary, state which assumptions you make on the underlying problem.

b) State the main goals for efficient parallelisation of sparse-matrix computation. Discuss which of the two suggested partitionings achieves the respective goals more successfully.

2 Spacetree Partitioning

Figure 2 shows a small quadtree grid. The underlying problem for this exercise is to create efficient partitionings for parallelisation with distributed memory.

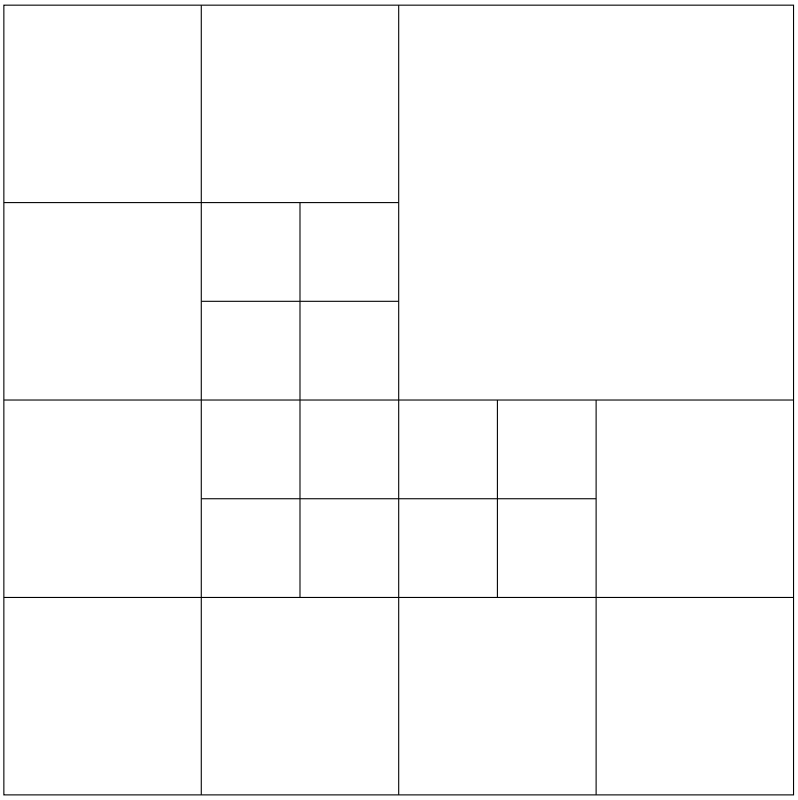

Figure 2: Adaptive quadtree grid for Exercise 2b) – sketch the REFTREE partitions here.

a) Draw the corresponding tree structure of the quadtree grid in Figure 2. The children shall be ordered in a way such that a depth-first traversal of the quadtree leads to a grid traversal in Hilbert order.

b) Shortly describe the main algorithmic steps of the REFTREE algorithm for the partitioning of tree-structured grids. Sketch the resulting partitioning into two partitions in Figure 2.

c) Explain how the partitioning problem can be formulated as a graph partitioning problem. Sketch the respective graph in Figure 3.

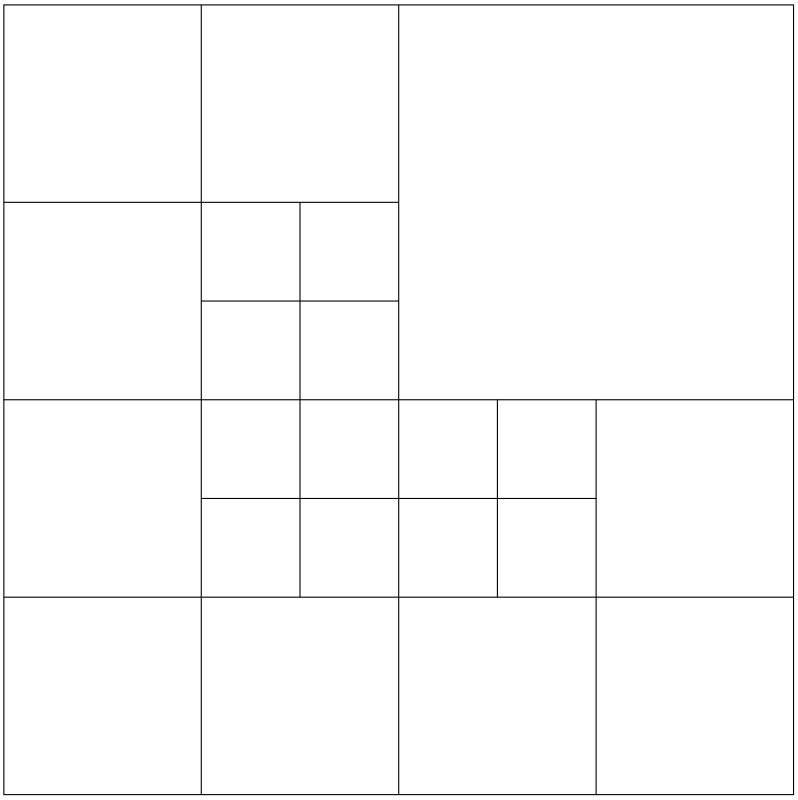

Figure 3: Adaptive quadtree grid for Exercise 2c) – sketch the graph here.

d) Give a short overview of the main steps of the multilevel graph partitioning algorithm discussed in the lecture (roughly two sentences for each of the main algorithmic phases are sufficient).

3 Structured Grids (≈ 5+2+3 = 10 points)

Given is the following relaxation scheme on a 2D Cartesian mesh – the unknowns are located on the Cartesian grid points and stored in an array `u(i,j)` (using column-major order for the storage scheme; `i` is the column index).

```
for (int i=1; i<n; i++)
   for (int j=1; j<n; j++) {
      u(i,j) = u(i,j) - 0.125*( u(i+1,j) + u(i-1,j) +
               u(i+1,j-1) + u(i,j-1) + u(i-1,j-1) +
               u(i+1,j+1) + u(i-1,j+1) + u(i,j+1)
              );
  };
```

a) Analyse the performance of this relaxation scheme in the Roofline Model. Consider a small cache in comparison to the problem size n (i.e., cache size $M \ll n$). Assume that you are running the code on a machine with a memory bandwith of 10 GB/s and a peak floating-point performance of 20 GFlop/s (assume single precision for all variables and Flop/s) .

b) What is the best performance you could achieve by using a (perfect) loop blocking implementation for this relaxation scheme?

c) Explain shortly how the cache line transfers could be reduced for the case of performing multiple relaxations by using the Cache oblivious algorithm by Frigo et al.

4 Band matrices in CUDA

A sparse matrix $\mathbf{B} \in \mathbb{R}^{N \times N}$ is called *band matrix* if its sparsity pattern fulfills the following rule: For certain k_{start}, k_{end} with $-N \ll k_{start} < k_{end} \ll N$:

$$\mathbf{B}_{ij} \neq 0 \text{ only if } j \geq i + k_{start} \text{ and } j < i + k_{end}$$

Here, a band matrix is compressed efficiently into a 1D-array `b[]` of size $N \cdot (k_{end} - k_{start})$, that contains the successive elements $(\mathbf{B}_{ij})_{j=i+k_{start},\ldots,i+k_{end}-1}$ for each row $i \in \{0, 1, \ldots, N-1\}$. Any array element with an invalid column index < 0 or $\geq N$ is assumed 0.
We implement a band matrix-vector product $\mathbf{y} = \mathbf{B}\mathbf{x}$ with $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ in CUDA. The target architecture supports CUDA Compute Capability 2.0 or higher and contains a shared memory but no cache:

```
__global__ void band_matvec(int N, int k_start, int k_end,
                            float* b, float* x, float* y) {

  int i = TILE_SIZE * blockIdx.x + threadIdx.x;

  if (i < N) {
    float dot = 0;

    for (int k = k_start; k < k_end; k++) {
      float val = b[(k_end - k_start) * i + k - k_start];
      int j = i + k;

      if (val != 0) dot += val * x[j];
    }

    y[i] = dot;
  }
}
```

a) Draw a sketch of the band matrix $\mathbf{B}$ and the vector $\mathbf{x}$. Mark the matrix and vector elements, which are read by the kernel for computation of a single vector entry $\mathbf{y}_i$.
You can assume that the condition in line 12 of the kernel always evaluates to true.

(i) How many read accesses to $\mathbf{x}$ does the kernel execute for each block of size `TILE_SIZE`?

(ii) Is this number optimal? What's the smallest number of elements that must be read from $\mathbf{x}$ in order to compute a block of `TILE_SIZE` elements?

b) Describe shortly how usage of shared memory would reduce the amount of read accesses to global memory in the kernel. Write a line of code where you allocate the amount of shared memory required for this improvement.

c) What is coalesced memory access? Are read accesses to the array `b[]` coalesced in the original kernel? Explain your answer.

d) We assume, that the order of elements in the array `b[]` can be chosen freely. Describe in two

sentences, how under these circumstances coalesced memory access to `b[]` is achieved. Which order would you have to choose for `b[]`? Explain, how line 9 in the original code has to be changed accordingly:

```
float val = b[(k_end - k_start) * i + k - k_start];
```

e) In order to measure the compression quality of a sparse matrix format, we count the number of entries that are zero and not stored explicitly by the format. Describe a sparse matrix $\in \mathbb{R}^{N \times N}$, that cannot be compressed by the band matrix format, but is compressed optimally by the ELLPACK format.