Universitetet i Bergen
Det matematisk-naturvitenskaplige fakultet

Exam in INF236 - Parallel programming,
Monday 28.09.18, 9AM - 12AM
English text

Problem 1, speedup

A. A sequential algorithm spends 1/4 of its running time in an initial preprocessing phase, and 1/4 is spent in a postprocessing phase. Neither the preprocessing nor the postprocessing phase can be parallelized. What is the maximum speedup attainable when moving to a parallel version of the algorithm?

Let A, B, C and D be four programs that each take one minute to execute sequentially and that have to be run in sequence. Thus A must finish before B can start and so on. Assume that porting your programs to a GPU will give A a speedup of 100, B a speedup of 10, C a speedup of 1 and D a speedup of 0.1.

B. What is the combined running time of the programs after porting all of them onto the GPU?

C. Which programs would you recommend to move to the GPU and what is the combined running time of your solution.

Problem 2, GPU questions

Explain what must be done to synchronize the following groups of threads:
A.
I. All the threads in a warp
II. All the threads in a block
III. All the threads in the GPU

B.
Let A and T be two matrices of dimension n x n. Then T is the *transpose* of A if $T_{i,j} = A_{j,i}$ for every pair of values i and j. In the following you are to write a CUDA kernel for computing the transpose T of a matrix A. In the following kernel the variables A and T points to the first elements of the two matrices each containing n^2 elements. Comment on if there are any restrictions on the block and grid dimension that your kernel can be called with.

```
__global__ void transpose(int *A,int *T,int n) {

    // Your code goes here
}
```

Problem 3, BSP, sorting

Let A be an array of size n containing distinct floating point values (float). Thus A[i] and A[j] have different values for all i and j where i is not equal to j. We now want to sort the elements of A and store them in an array B, also of size n. We do this by counting the number of elements that are smaller than each A[i] and then placing A[i] in the correct position of B. The sequential code for this algorithm is as follows:

```
for (int i=0; i<n; i++) {
  int pos=0;
  for (int j=0; j<n; j++)
    if (A[j] < A[i]) pos++;
      B[pos] = a[i];
}
```

We now want to write a parallel algorithm using BSP for executing this algorithm on a distributed memory parallel computer. Initially every process has a part of the input of length np stored in a local array A. When the program is done each process should have np elements of the output stored in B. The elements of B on each processor should be sorted and all elements of B on process i should be smaller than all elements on process j if i < j.

The algorithm works by cyclicly shifting the elements of A between the processes (similarly to what was done in the first part of the obligatory assignment on matrix multiplication). We will use B for this. Thus A is initially copied into B and B is set up for being accessed by the other processes as follows:

```
void bspsort(A,B,np) {
  bsp_begin(P);
  long p = bsp_nprocs();
  long s = bsp_pid();

  for(i=0;i<np;i++)
    B[i] = A[i];

  bsp_push_reg(B,np*sizeof(float));
  bsp_sync();

// Your code goes here

  bsp_end();
}
```

A) Write the code that counts the total number of elements across all processes that are smaller than each A[i]. The number for A[i] should be stored in pos[i] where pos is a local integer array of size np.

B) Assume now that `pos[i]` contains the number of elements across all processes that are smaller than A[i]. Write the code that puts each element of A into its correct place of B on the appropriate process.

The syntax for bsp_put is as follows:

bsp_put(pid, *source, *destination, offset_in_bytes, nbytes);

Problem 4, OpenMP

Consider the following parallel code for transposing an array using OpenMP. Assume that you run the code using two threads.

```
#pragma omp parallel for schedule(runtime)
for(int i=0;i<N;i++)
  for(int j=i+1;j<N;j++) {
     float temp = a[i][j]
     a[i][j] = a[j][i];
     a[j][i] = temp;
  }
```

A. What is the (approximate) best speedup you can expect when using static scheduling?

B. What is the (approximate) best speedup you can expect when using dynamic scheduling?