Uppsala University
Department of Information Technology
Scientific Computing

# Programming of Parallel Computers, 10hp, 2015-03-19

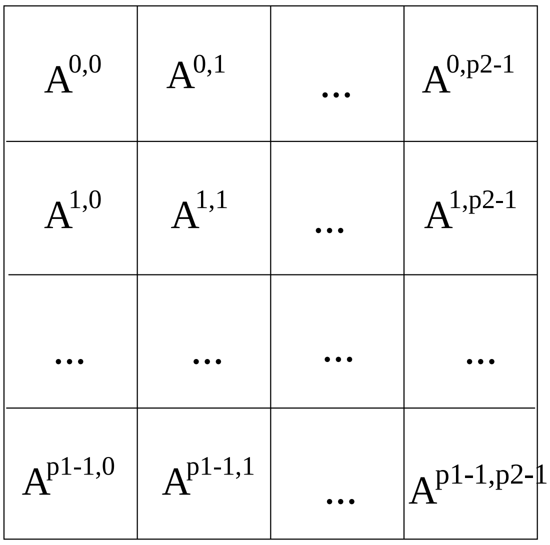
**Time**: $08^{00} - 13^{00}$
**Help**: None
Each of the six problems below can give up to five points. For maximum points, you must give detailed answers and motivate your assumptions. Grade 3: 12p, Grade 4: 18p, Grade 5: 24p

1. Write a code section to distribute a $M \times N$ matrix to $P$ processors using MPI function calls. Your code should implement the following requirements:

   a) Allocate the matrix in processor $P_0$ as a one dimensional array.

   b) Create a $p1 \times p2$ cartesian grid communicator with the factorization $P = p1 \times p2$. The code should do the factorization automatically and as evenly as possible.

   c) Partition the matrix as show in Figure 1 and distribute the matrix blocks with non-blocking point-to-point communication (or with a collective communication call). For simplicity you can assume that $\frac{M}{p1}$ and $\frac{N}{p2}$ are integers.



Figur 1: Data partitioning strategy

2. a) Explain the differences between blocking and non-blocking communication in MPI. Consider $MPI\_Send()$ and $MPI\_Isend()$ as examples.

   b) In the MPI lab you have measured the communication bandwidth using synchronous point-to-point communication. Explain the measurements, the results and the conclusions of the experiments.

3. In the course, we have used Pthreads to write parallel programs using shared memory. In Pthreads, *mutexes* and *condition variables* are central. Describe these two concepts, how they work, and in what situations they are used.

4. Gaussian smoothing is common in image processing for creating a blurred effect. The algorithm calculates the new pixel value as an average of a surrounding block of pixels, weighted by a Gaussian block. The listing below shows a CUDA kernel implementing this for a 5-by-5 Gaussian block.

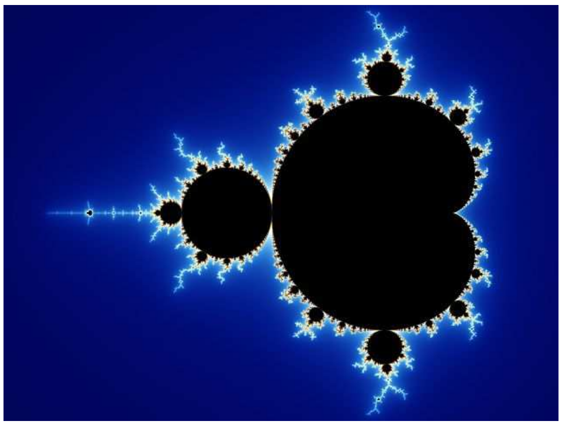
```
__global__ void gaussian_kernel(float *Aout, const float *Ain)
{
  int i = threadIdx.x + blockIdx.x*blockDim.x;

  if(i>1 && i<Height-2) { // only compute in interior
    for(int j=2; j<Width-2; j++) {
      float tmp=0;
      for(int ii=-2; ii<2; ii++) {
        for(int jj=-2; jj<2; jj++) {
          tmp += Weight[ii*5+jj]*Ain[(i+ii)*Width + j+jj];
        }
      }
      Aout[i*Width + j] = tmp;
    }
  }
}
```

   Discuss how this implementation could be optimized by a) increasing the parallelism, and b) using shared memory. In both cases, explain what the problem is, why it would be improved, and sketch an implementation. You do not have to get the syntax right.

5. In parallel computing we have different programming models with respect to the memory space: *(a) local address space*, *(b) global address space*, and *(c) partitioned global address space*. Explain the differences between these and discuss their pros and cons.

6. The Mandelbrot set is defined for points in the complex plane for which the sequence $z_{k+1} = z_k^2 + c$ is bounded, where $z_0 = 0$ and $c$ is a complex number representing the position in the complex plane. Figure 2 shows the resulting Mandelbrot set in the complex plane.

   The Mandelbrot set can be computed with the code listing below. Make an efficient parallelization of the algorithm using OpenMP directives. Consider different settings of the SIZE variable and the number of threads (i.e., how does the parallelization differ when SIZE is small and the number of threads is large compared to when SIZE is large and the number of threads is small).



Figur 2: The Mandelbrot set.

```
int inSet(double ix, double iy)
{
 int iterations = 0;
 double x = ix, y = iy;
 double x2 = x*x, y2 = y*y;
 while ((x2+y2<4) && (iterations<1000))
 {
   y = 2*x*y + iy;
   x = x2-y2+ix;
   x2 = x*x;
   y2 = y*y;
   iterations++;
 }
 return iterations;
}

int main()
{
  int Matrix[SIZE][SIZE];
  int x,y;
  double xv,yv;

  for ( x=0; x<SIZE; x++ )
    for ( y=0; y<SIZE; y++ )
       {
         xv = ((double)x-(SIZE/2))/(SIZE/4);
         yv = ((double)y-(SIZE/2))/(SIZE/4);
         Matrix[x][y]=inSet(xv,yv);
       }

  // Plotting the Mandelbrot set is not part of the task and omitted.
}
```