Exercise for Course

Parallel High-Performance Computing

Dr. S. Lang

Return: 17. December 2014 at the beginning of the exercise or earlier

---

Task 17 MPI: Communication in the ring (5 points)

With this task we want to perform first steps with MPI. Implement a communication of 8 processes in the ring. Each process shall sent its rank within a message once in the ring and terminate, if it again receives its rank within a message. Use synchronous sending resp. receiving and one of the techniques presented in the lecture, to avoid deadlocks, e.g. coloring of the edges. Each process shall in each send/receive step print its ranks and the just received message. Test your program in the pool and hand in an output of the communication sequence.

More details for using of MPI in the pool is on the Homepage. Helpful for the right syntax are the manpages about MPI (e.g. `man MPI_Comm_rank`).

Task 18 Parallel Computing of π with MPI (5 points)

From the identity $\pi = 4(\arctan 1)$ one gets by usage of the derivative of the arctan, $(\arctan x)' = 1/(1+x^2)$, a formula for calculating π:

$$\pi = \int_0^1 \frac{4}{1+x^2} dx.$$

By division of the interval into n equidistant partial pieces the integral can be evaluated with the midpoint rule. You can find a sequential program in the file `piseq.c` on the homepage. We want to parallelize it with MPI. The strategy is:

- process 0 reads the number of partial intervals and passes it to all other processes,
- The `for` loop over the partial intervals will be parallelised, each process calculates a local partial sum. The results wil be collected by process 0 with a reduction operation `MPI_Reduce` and the partial sums are added.

First determine the convergence order of the midpoint rule with the sequential program. Establish a double-logarithmic plot with the integration error over the interval length h. The steepness of the line give the order. Now implement a parallel version. Compare the accuracy in the calculations (last digits) with the sequential solution and the exact value (short discussion).

Optional task additional 5 credit points

The number of valid digit positions can be enhanced by using the *Gnu Multiprecision Arithmetic Library* (GMP, you can find at www.gmplib.org, for the most Linux distributions there is a package). Thus π could be calculated e.g. up to 40., 60. or 80 positions. Of course the choosen method with quadratic convergence is much to slow for that, this means a up to 80. digit exact program would run nearly forever. Implement a version of the sequential or parallel program, that uses the GMP. If you want to gain high accuracy, you need to use a better method than the midpoint rule. Otherwise you can test which accuracy you can achieve with GMP and the midpoint rule in a meaningful computing time. You can get the reference value of *pi* by internet.

Task 19 Simple Parallelisation of the Jacobi method with MPI (10 points)

For linear equation systems $A\mathbf{x} = \mathbf{b}$, $A \in \mathbb{R}^{n\times n}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$, $n \in \mathbb{N}$, direct solution methods are mostly inefficient for large n. Therefore one often uses iterative methods like the *Jacobi method*. You get the

method by additive spliting of the system matrix A in an upper and low triangular matrix U and L and the diagonal matrix D, $A = D+L+U$. This leads to the fixpoint iteration $\mathbf{x} = D^{-1}(\mathbf{b}-(A-D)\mathbf{x})$, that can be solved under certain circumstances: $\mathbf{x}^{(m+1)} = D^{-1}(\mathbf{b}-(A-D)\mathbf{x}^{(m)})$, Index $m \in \mathbb{N}$ is the iteration step. The i.th equation for the $(m+1)$. step of the method is then called

$$x_i^{(m+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(m)}}{a_{ii}}.$$

As starting vector $\mathbf{x}^0$ you can use each arbitrary vector. The calculation of the residual $\mathbf{r}$ provides a termination condition in m.th step. $\mathbf{r}^m := \mathbf{b} - A\mathbf{x}^{(m)}$: Is an adequate norm of the residual (e.g. the maximum norm $||\mathbf{r}||_\infty$) smaller than a given tolerance $\epsilon \in \mathbb{R}_+$, the iteration stops. The calculation of the iteration depend only on the previous solution and therrre are no data dependencies between the newly calculated values x_i. Thus the Jacobi methods is easy to parallelize.

We want to use the Jacobi method, to calculate the discrete solution of the poisson equation

$$\begin{aligned} -\triangle u &= f \quad \text{in } \Omega = (0,r)^2, \\ u &= 0 \quad \text{auf } \partial\Omega \end{aligned}$$

on a square with side length r. The source f be $f(x,y) := 2\pi^2 \sin(\pi x) \cdot \sin(\pi y)$. Then the analytic solution is given by $u(x,y) = \sin(\pi x) \cdot \sin(\pi y)$. The unit square is covered from a grid with n^2 points, see Figure 0.5 left with $r = 4$ and $n = 9$. The distance h between two points (the „grid resolution“) is $h = r/(n-1)$. In the Figure to the right a source f is shown.

An equation system in matrix shape is gotten by approximation of the second derivative through a central difference quotient at each inner grid point (i,j) mit $i,j = 1 \ldots n-1$. At the boundary points with $i = 0$ or $i = n$ (also for j) the solution is predefined. The grid points can be enumerated consecutively by $k = i \cdot n + j$, thus each index pair (i,j) is mapped onto an index k in a unique way. Then we can consider the grid function

$$u^h := (u_{01}, u_{02}, \ldots, u_{0n}, u_{11}, \ldots, u_{1n}, \ldots, u_{n1}, \ldots u_{nn})^T = (u_1, \ldots u_N)^T$$

with $N = n^2$. After the approximation of the second derivative the equation system for the unknown grid function u^h (details to the derivation soon) reads:

$$\frac{1}{h^2} \begin{pmatrix} 4 & -1 & \cdots & -1 & & & & & \\ -1 & 4 & -1 & \cdots & -1 & & & & \\ & \ddots & \ddots & \ddots & & \ddots & & & \\ & & & \ddots & -1 & & \ddots & & \\ & & & -1 & 4 & -1 & \cdots & -1 & \\ & & & & -1 & \ddots & & & \\ & & & & & & \ddots & -1 & \\ & & & & \cdots & & -1 & 4 & \end{pmatrix}_{N \times N} \cdot \begin{pmatrix} u_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} f_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_N \end{pmatrix}.$$

The values f_1 to f_N are evaluations of the source f at the grid points. Each line correspoinds to a grid point x_{ii}. The matrix contains on the diagonal the 4 as entry of u_{ii}, and in the direct neighbors left and right the entry -1 at the indices u_{i-1j} and u_{i+1j}. Furthermore there are in each row two further entries -1. This are exactly the one of the neighbors above and below u_{ij+1} and u_{ij-1}. Therefore the distane between the -1 right beneath a 4 and the next -1 just n.

At a boundary point the assigned matrix line has to be substituted through a null line with a 1 on the diagonal entry, the solution u has to be set in this case to 0 as well as the right side also to 0.

Task

1. Develop a MPI-parallel variant of the Jacobi method. A possible strategy is to split the matrix A and the vectors in stripes of size (row or column count) α, each processor works with one stripe.

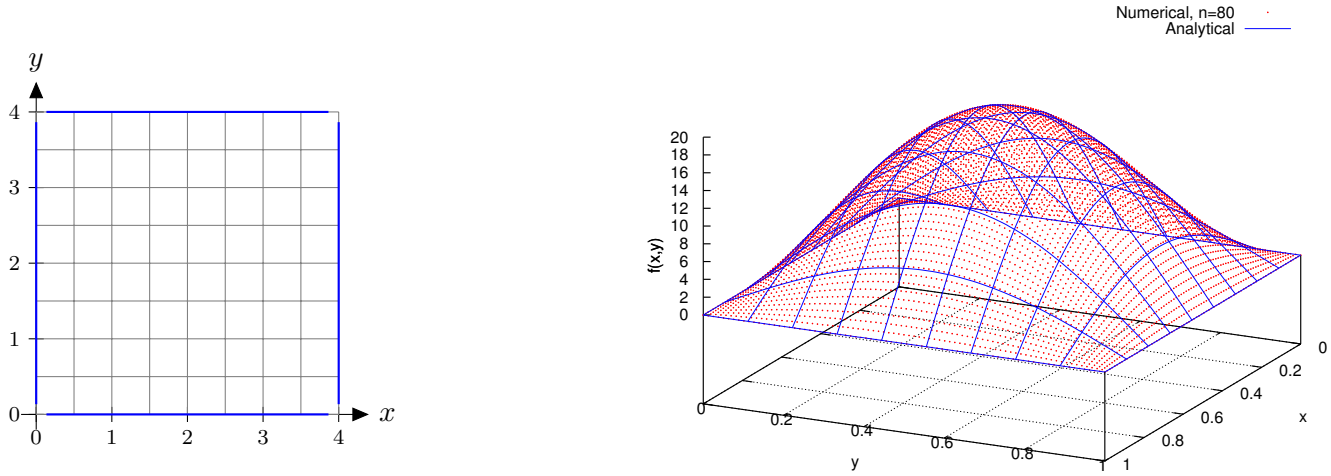

Abbildung 0.5: Left: area Ω with discretization, right: source $f(x, y)$.

Each processor gets in the Jacobi step m a copy of the previous solution $\mathbf{x}^{(m-1)}$, to calculate the new values x_i of a stripe. Then in each iteration each process hat to communicate its new values of its partial domain to all other processes, e.g. using multi broadcast.

2. Initialize the initial solution u^h with 0.0. Set $r = 1.0$ and use a tolerance for $\epsilon = 10^{-4}$. Don't forget when assembling the matrix and the initialisation of the vectors the treatment of the boundary points! Test with your code the convergence of the error $\|u - u^h\|_\infty$ between analytic u and numerical solution u^h at the grid points for $n = 4, 8, 16, 32$ and 64. Establish a plot of the error over the grid width h. Can you determine the consistency order?

3. Measure the speedup in the pool against the sequential version ($P = 1$) for different problem sizes $n \leq 32$ and processor counts P.

Hints

- If you like you can implement another strategy, that you can find in a textbook on numerical solution methods for linear equation systems.

- If you have any difficulties you can make the task easier and choose a another matrix without physical application. For convergence of the Jacobi method the matrix A needs to be strictly diagonal dominant: It should apply for each line $\sum_{j=1;j\neq i}^{n} |a_{ij}| < |a_{ii}|$, $\forall i \in \{1, .., n\}$ (the absolute value of diagonal element of each line should be greater than the sum of absolute values of the off diagonal entries).