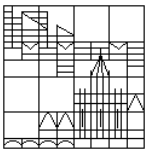

Universität Konstanz
FB Mathematik & Statistik
Prof. Dr. M. Junk
Dr. Z. Yang

Ausgabe: 02. Mai; SS08

Parallele Numerik

Blatt 1

As a first step, we consider two basic problems. Hints for the realization in MPI, OpenMP, Java, and Matlab are given below. Please familiarize yourselves with the commands used in the example programs for your group (using, for example, the literature available on the web site). You should be able to explain your code. It would be nice, if you could also provide some more information on the syntax or usage of the language constructs whenever you think it is needed.

Problem 1: Hello world!

Write a baby parallel program where each process prints

```
Hello world!  I am process Pid of Nproc
```

on the screen. Here Pid is the process number (also called identifier or rank) and Nproc is the total number of processes.

Problem 2:

Use N processes to calculate a new vector $\boldsymbol{c} \in \mathbb{R}^m$ from two given vectors $\boldsymbol{a}$, $\boldsymbol{b} \in \mathbb{R}^m$ by $c_k = a_k b_k$. The entries of the vector $\boldsymbol{c}$ are divided into N consective groups,

c_1	$c_{p(1)+1}$	.....	$c_{p(N-1)+1}$
c_2	$c_{p(1)+2}$		$c_{p(N-1)+2}$
...	...		...
$c_{p(1)}$	$c_{p(2)}$		c_m

Each process deals with one group. Finally, a master process should print the entries of the vector $\boldsymbol{c}$.

Hints for MPI realization

Before working with MPI, you should once execute the command

```
>cp /software/rhosts .rhosts
```

in your home directory (the leading dot in .rhosts is required).

1. An examplary C-program which implements Task 1 using MPI is

```
1   #include <mpi.h>
2   #include <stdio.h>
3
4   int main(int argc, char **argv)
5   {
6   int rank, size;
7   MPI_Init(&argc, &argv);
8   MPI_Comm_size(MPI_COMM_WORLD, &size);
9   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
10  printf("Hello World! I am process %d of %d.\n", rank, size);
11  MPI_Finalize();
12  return 0;
13  }
```

   Save this (or your own program) in a file hello.c and use the following commands to compile and run it with an (almost) arbitrary number of processes (note: if you want to work with more than 8 processes, do not compile on bart.phyma but use another phyma computer instead).

```
> mpicc -o hello hello.c                 compile hello.c
> mpirun -np 3 hello                     run hello with 3 processors
Hello World! I am process 0 of 3         possible result after running hello
Hello World! I am process 2 of 3
Hello World! I am process 1 of 3
>
```

   Understand and explain the MPI concept and syntax for:
   (i) initialization and finalization
   (ii) process information

2. Write your own MPI program for the second problem. Explain how to use the MPI message-passing commands and how to assign different tasks to the processes.

   Hint 1: The calculation of vector $\boldsymbol{c}$ can be realized by a loop with Pid-depending limits from *p(Pid)* to *p(Pid+1)*.

   Hint 2:, The MPI message-passing commands

```
int MPI_Send(.....)
int MPI_Recv(.....)
```

   are required, when the master processor collects all the entries of the vector $\boldsymbol{c}$ for output.

Hints for OpenMP realization

The best machine for OpenMP implementations is bart.phyma because it has 8 CPUs.

1. An exemplary C-program which implements Task 1 using OpenMP is

```
1   #include <omp.h>
2   #include <stdio.h>
3
4   main( ) {
5   int nthreads, tid;
6   #pragma omp parallel private(tid)
7   {
8   nthreads = omp_get_num_threads( );
9   tid = omp_get_thread_num( );
10  printf("Hello World! I am process %d of %d.\n", tid, nthreads);
11  }
12  }
```

   Save this (or your own program) to the file hello.c and use the following commands to compile and run it with an (almost) arbitrary number of processes

```
> gcc -fopenmp -o hello hello.c          compile hello.c
> NUM_OMP_THREADS = 3                    set 3 processors
> ./hello                                run hello
Hello, World! I am process 1 of 3        possible result after running hello
Hello, World! I am process 0 of 3
Hello, World! I am process 2 of 3
>
```

   Understand and explain the OpenMP concept and syntax for:
   (i) compiler directives and process information
   (ii) the fork-join excution model

2. Write your own OpenMP C-program for the second problem. Explain how to use the OpenMP **for directive**, and how the work is assigned to each process. Can the user decide about the workload of each process?

Hints for Java realization

The best machine for Java implementations is bart.phyma because it has 8 CPUs.

1. An examplary Java-program which implements Task 1 is

```
1   public class HelloWorld extends Thread
2   {
3   private int Pid, Nproc;
4
5   public HelloWorld(int Pid, int Nproc)
6   {
7   this.Pid = Pid;
8   this.Nproc = Nproc;
9   }
10
11  public void run()
12  {
13  System.out.println("Hello world! I am process " + Pid + " of " + Nproc);
14  }
15
16  public static void main(String[] args)
17  {
18  int Nproc = 12;  // choose number of threads
19  HelloWorld[] Threadarray;
20  Threadarray = new HelloWorld[Nproc];
21  for (int i = 0; i<Nproc; i++)
22  {
23  Threadarray[i] = new HelloWorld(i,Nproc);
24  Threadarray[i].start();
25  }
26  }
27  }
```

   Save this (or your own program) to the file HelloWorld.java and use the following commands to compile and run it

```
> javac HelloWorld.java                compile
> java HelloWorld                      run HelloWorld
Hello World! I am process 0 of 12      possible result after running HelloWorld
Hello World! I am process 1 of 12
.....
>
```

2. Write your own Java-program for the second problem. The master process has to wait for the threads to finish before it can output the final result. Explain the Java construction used to wait for the termination of the threads.

Hints for Matlab realization

To begin with, you should use an ***interactive pmode*** session to understand and explain the Matlab parallel computing tools. The commands to start and end a pmode session are given at the usual Matlab prompt as follows. A detailed description can be found in the handbook of the distributed computing toolbox available on the web site.

```
>> pmode start local 3        open a pmode session with 3 labs
>> doc pmode                  open a pmode document window
>> pmode exit
>>
```

1. An examplary, interacive Matlab-program which implements Task 1 is obtained by typing the following Matlab commands at the pmode prompt

   P >> labindex
   P >> numlabs
   P >> fprintf("Hello, World! I am process %d of %d.\n", labindex, numlabs);

   Understand and explain the results in each lab, and familiarize yourself with the possible configurations of the parallel command window.

2. Use *parfor*, *darray*, *gather*, etc. to calculate the vector $\boldsymbol{c}$ in the pmode labs and to plot $\boldsymbol{c}$ in the Matlab client.

3. A non-interactive version of the parallel programs is also possible using the commands introduced above. Before running the corresponding Matlab-program you should start an appropriate number of parallel sessions. This is accomplished with the following commands

```
>> matlabpool open 2          set two processors
>> run you own program
>>
```