Info

Exam in IN3200 spring semester 2019

Assistance: One two-sided A4 page with hand-written notes, plus a calculator. (A calculator is also available in Inspera.) No other assistance is allowed.

All the exam questions should be answered with keyboard and mouse. No need to use sketching paper.

Weighting of questions:
Questions 1.1, 1.2 (Performance of serial code) 10 points
Question 2.1 (Processing dependent tasks) 10 points
Questions 3.1, 3.2 (OpenMP) 10 points
Question 4.1 (MPI) 10 points
Questions 5.1, 5.2, 5.3 (Counting occurrences of a text pattern) 20 points

1.1 Improving code performance

Explain why the following code will probably run very slow. How will you modify the code such that the same computation is done but the performance is considerably improved?

```
for (int i=0; i<n; i++) {
  c[i] = exp(1.0*i/n)+sin(3.1415926*i/n);
  for (int j=0; j<n; j++)
    a[j][i] = b[j][i] + d[j]*e[i];
}
```

We assume that both a and b are row-major 2D arrays of dimension $n \times n$, while b, c, d, e are 1D arrays of length n.

Fill in your answer here

Format

Words: 0

Maximum marks: 5

1.2 Estimating time usage

Assume that we have a CPU with 40 GB/s as the theoretical memory bandwidth and 100 GFLOP/s (in double precision) as the theoretical peak floating-point performance. Can you estimate how much time the following code needs on this CPU, when $n = 10^{10}$?

```
double s = 0.0;
for (i=0; i<n; i++) {
  s += a[i]*a[i];
}
```

We assume that array a is of length n and contains double-precision values.

Fill in your answer here

Format

Words: 0

Maximum marks: 5

2.1 Processing dependent tasks

The figure below shows the dependency relationship between 16 tasks: $T_{1,1}, T_{1,2}, \ldots, T_{4,4}$. Each directed edge in the graph connects a pair of "source" and "destination" tasks. A destination task cannot be started until all its source tasks are carried out. All the 16 tasks are equally time-consuming, requiring one hour of a worker. (We also assume that it is not possible to let two or more workers collaborate on one task for a faster execution.)

Explain how many hours minimum do 3 workers need to finish all the 16 tasks.

$$\begin{tikzcd}
T_{1,1} \arrow[r] \arrow[d] & T_{1,2} \arrow[r] \arrow[d] & T_{1,3} \arrow[r] \arrow[d] & T_{1,4} \arrow[d] \\
T_{2,1} \arrow[r] \arrow[d] & T_{2,2} \arrow[r] \arrow[d] & T_{2,3} \arrow[r] \arrow[d] & T_{2,4} \arrow[d] \\
T_{3,1} \arrow[r] \arrow[d] & T_{3,2} \arrow[r] \arrow[d] & T_{3,3} \arrow[r] \arrow[d] & T_{3,4} \arrow[d] \\
T_{4,1} \arrow[r] & T_{4,2} \arrow[r] & T_{4,3} \arrow[r] & T_{4,4}
\end{tikzcd}$$

Fill in your answer here

Format

Words: 0

Maximum marks: 10

3.1 OpenMP parallelization

Explain why the following code segment cannot be directly parallelized by inserting "#pragma omp parallel for" before the for-loop. If the computation involved in function "func()" is very time consuming, how will you modify the code such that OpenMP parallelization becomes possible to speed up the entire computation?

```
for (i=0; i<n-1; i++) {
  u[i] = func(u[i+1]);
}
```

Fill in your answer here

Format

Words: 0

Maximum marks: 5

3.2 False sharing

In the context of OpenMP programing, what does **false sharing** mean? Please give a very simple example of false sharing.

Fill in your answer here

Format

Words: 0

Maximum marks: 5

4.1 Understanding an MPI program

What will be the result of running the following MPI program using 8 MPI processes?

```
#include <mpi.h>
#include <stdio.h>
int main (int nargs, char **args)
{
  int own_value, in_value, out_value, i;
  int rank, size;
  int send_to, recv_from;
  MPI_Status recv_status;
  MPI_Request recv_req;

  MPI_Init (&nargs, &args);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  own_value = rank;
  recv_from = (rank+2)%size;
  send_to = (rank-2+size)%size;
  out_value = own_value;

  for (i=0; i<(size/2)-1; i++) {
    MPI_Irecv(&in_value, 1, MPI_INT, recv_from, 0, MPI_COMM_WORLD, &recv_req);
    MPI_Send (&out_value, 1, MPI_INT, send_to, 0, MPI_COMM_WORLD);
    MPI_Wait (&recv_req, &recv_status);
    own_value += in_value;
    out_value = in_value;
  }

  printf("On rank <%d>, own_value=%d\n",rank,own_value);
  MPI_Finalize ();
  return 0;
}
```

Fill in your answer here

Format

Words: 0

Maximum marks: 10

5.1 Function count_occurrence

Write a serial C function with the following syntax:

```
int count_occurrence (const char *text_string, const char *pattern);
```

Note that this function takes two input arguments: **text_string** and **pattern**. The purpose is to find out many times **pattern** appears in **text_string**. Some concrete examples are as follows:

- count_occurrence ("ATTTGCGCAGACCTAAGCA", "GCA") will return 2
- count_occurrence ("AAABCDEFGAAGEREAANMT", "AA") will return 4
- count_occurrence ("ABCDEFGHIJKLMNOPQRSTUVWXYZ", "BTTT") will return 0

Hint: The following two C standard functions from "string.h" can be useful:

- The C library function **size_t strlen(const char *str)** computes the length of the string **str** up to, but not including the terminating null character. (For example, strlen("GCA") returns 3.)
- The C library function **int strncmp(const char *str1, const char *str2, size_t n)** compares at most the first **n** bytes of **str1** and **str2**. In case **str1** and **str2** are identical for the first **n** bytes, the function will return **0**, otherwise the function will return a non-zero value.

Fill in your answer here

Maximum marks: 5

5.2 OpenMP parallelization

Implement an OpenMP parallelization of function 'count_occurrence':
Fill in your answer here

```
```

Maximum marks: 7

5.3 MPI parallelization

Write an MPI version of 'count_occurrence' with the following syntax:

```
  int parallel_count_occurrence (const char *text_string, const char *pattern);
```

This function is to be called by all MPI processes, where the input arrays **text_string** and **pattern** are both empty pointers on all MPI processes except on MPI process with rank 0.

You can asseme that 'MPI_Init' has already been exectuted before function 'parallel_count_occurrence' is called.

Fill in your answer here

Syntax for some of the most important MPI functions:

```
int MPI_Comm_size( MPI_Comm comm, int *size )

int MPI_Comm_rank( MPI_Comm comm, int *rank )

int MPI_Barrier( MPI_Comm comm )

int MPI_Send(const void *buf, int count, MPI_Datatype datatype,
        int dest, int tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,
        int tag, MPI_Comm comm, MPI_Status *status)

int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root,
         MPI_Comm comm )

int MPI_Alltoall(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
          void *recvbuf, int recvcount, MPI_Datatype recvtype,
          MPI_Comm comm)

int MPI_Reduce(const void *sendbuf, void *recvbuf, int count,
         MPI_Datatype datatype,
         MPI_Op op, int root, MPI_Comm comm)

int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count,
           MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
         void *recvbuf, int recvcount, MPI_Datatype recvtype,
         int root, MPI_Comm comm)

int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
         void *recvbuf, int recvcount, MPI_Datatype recvtype,
         int root, MPI_Comm comm)
```

int MPI_Gatherv(void **sendbuf*, int *sendcnt*, MPI_Datatype *sendtype*,
void **recvbuf*, int **recvcnts*,
int **displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

int MPI_Scatterv(void **sendbuf*, int **sendcnts*, int **displs*, MPI_Datatype *sendtype*,
void **recvbuf*, int *recvcnt*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Maximum marks: 8