CHALMERS TEKNISKA HÖGSKOLA
Institutionen för data- och informationsteknik
Avdelningen för datorteknik

Exam in DAT400 (Chalmers) and DIT431 (GU) High Performance Parallel Programming, Friday, October 28th, 2022, 14:00h - 18:00h

---

Teacher/Lärare: Miquel Pericas, tel 7721705. Exam hall visits by teaching staff are planned approx. 1h and 3h after examination start.

Language/Språk: Answers shall be given in English.

Solutions/Lösningar: Solutions will be posted on Thursday, November 4th, 2022 on the Canvas page.

Exam review/Granskning: The review date will be posted on the course Canvas page by the time you receive the email from LADOK.

Aids/Hjälpmedel: Only Chalmers-authorised aids are allowed during examination. This includes pencils, erasers, rulers and dictionaries. Electronic dictionaries or graphing calculators are not allowed.

Grades:

Chalmers				
Points	0-23	24-35	36-47	48-60
Grade	Failed	3	4	5

GU				
Points	0-23	24-41	42-60	
Grade	Failed	G	VG	

Good Luck!

Problem 1 (10 points)

Consider a program that consists of the following main loop:

```
float sum = 0.0;
float a[N], b[N], c[N], d[N+1];

// main loop:
for (auto i = 0; i < N; i++){
    c[i] = a[i] + b[i];
    sum += a[i] * b[i];
    d[i+1]=d[i];
    }
```

Tasks:

(a) Discuss whether this code can be parallelized and/or vectorized. If not, list the code transformations required to enable its parallelization and/or vectorization.

(b) The programmer is now tasked with parallelizing the code over a multicore processor. To select the unit of scheduling, they have been given three options: processes, kernel-level threads or user-level threads. Describe each option (2-3 lines) and list (at least) one advantage and one disadvantage of each approach.

(c) In the previous code, if each iteration is to become an independent scheduling unit, should the iteration be mapped to a distinct process, a distinct kernel-level thread, or a distinct user-level thread? Discuss each option separately from a scheduling perspective.

(d) Finally, from a data perspective, list two downsides of creating a new scheduling unit for each iteration. Given the discussion, suggest a new chunk size that will lead to high performance for a system with P processors and cache line size of 64 bytes.

Problem 2 (10p)

STREAM is a widely used benchmark to measure the memory bandwidth of a system. In this problem you will use it to study the effect of different kinds of OpenMP schedulers on performance. Consider the following code:

```
#pragma omp parallel for
for (int i=0; i<STREAM_ARRAY_SIZE; i++) {
    c[i] = a[i];
}
```

Assume a programmer wants to run the loop on a system with 4 processors. `STREAM_ARRAY_SIZE` is set to 256.

Tasks:

(a) The programmer is considering the following four schedulers:

- Static schedule
- Dynamic schedule: "self-scheduling"
- Dynamic schedule: "chunk scheduling", with chunk size 50
- Guided schedule with minimum chunk size of 15: "GSS(15)"

How many chunks will be generated by each scheduler? And what is the number of iterations in each chunk?

(b) The programmer has found that each iteration executes for 1ns on the system. Based on your analysis in part (a), what are the execution times on the system when using the four scheduling schemes listed above?

(c) Now consider only self-scheduling and chunk scheduling. Assuming the dispatch overhead for one iteration is 0.1 ns, and the dispatch overhead for scheduling a chunk of size 50 is 4.5 ns, which of the two scheduling policies will execute faster?

(d) The programmer decides to use the static scheduling strategy for the STREAM COPY problem and thinks the execution time using static schedule for problem size 256 is satisfactory. Now they are able to access a larger system with 32 processors. However, creating the parallel threads on the new system incurs overhead, which depends on the number of processors (P). The time cost (overhead) is T = 1 ns + P $\times$ 0.5 ns. For simplicity, assume that the new system offers unlimited memory bandwidth. What is the largest problem size `STREAM_ARRAY_SIZE` that can be addressed on the new system without increasing the execution time?

Problem 3 (10p)

Consider an application consisting of the following kernel:

```
#define NDIM 1000

void 2dfunc(float next[NDIM][NDIM],
            float prev[NDIM][NDIM])
{
  for(j=1;j<NDIM-1;j++){
    for(i=1;i<NDIM-1;i++){
      next[i][j] = -4.0*prev[i][j] +
                        prev[i][j-1] + prev[i][j+1] +
                        prev[i-1][j] + prev[i+1][j];
      next[i][j] *= next[i][j];
    }
  }
}
```

Tasks:

(a) What is the arithmetic intensity (AI) of this code? Now consider a multicore chip in which each core has a performance of 4 GFLOPS (single precision) and a shared memory bus of 16GB/s. Using the simplified DRAM roofline model, what will be the performance (execution time) if the code is run on a single core or on 8 cores? Note: a single `float` consists of 4 bytes.

(b) The above kernel is included in an outer loop that evolves the execution over multiple timesteps (see the code below). This outer loop includes a serial component that takes 0.5 ms. What is the upper limit to the speed-up (compared to a single core) according to Amdahl's law? For the above 8 core system, compute the speed-up over a single core. Discuss, individually, the suitability of (i) executing the kernel on even more cores, (ii) optimizing the kernel to achieve a higher AI, or (iii) optimizing the serial part of the computation.

```
float a[NDIM][NDIM];
float b[NDIM][NDIM];

void run()
{
// assume input vectors are initialized
float *tmp, *future=b, *current=a;
for(int ts = 0; ts < NTIMESTEPS; ts++){
  serial_computation(current);
  2dfunc(future, current);
// swap pointers
  tmp = future;
  future = current;
  current = tmp;
  }
}
```

(c) Assume now that the developers are happy with the execution time of the current system. They believe that by replacing the current chip with a larger multicore chip they will be able to solve a larger problem in the same time. In 2-3 lines, discuss the suitability of this proposal.

(d) Part of the `serial_computation()` in (b) consists of the following Matrix-vector multiplication code:

```
int i, j, A[N][N], B[N], C[N];
for (i = 0; i < N; i++) {
  C[i] = 0;
  for (j = 0; j < N; j++) {
    C[i] = C[i] + A[i][j] * B[j];
  }
}
```

Propose two loop transformations (not involving parallelization) to improve the performance of this code on a system with caches. Rewrite the loop code with the proposed transformations.

Problem 4 (10p)

The code in the below listing shows a serial implementation of the Cholesky decomposition. There are three steps to perform a Cholesky decomposition on a matrix: (1) The upper triangle of the matrix is replaced with zeroes, (2) the diagonal elements are calculated, and (3) the elements below the diagonal are computed. The goal of this problem is to develop an OpenMP implementation of the code.

```
double ** cholesky(double ** L, int n) {
    int i, j, k;
    for (j = 0; j < n; j++) {
        #pragma omp parallel
        {
            //Step 1. Set upper triangle to 0
            for (i = 0; i < j; i++)
                L[i][j] = 0;
            //Step 2: Calculate diagonal elements
            //OMP Loop 1
            //-------------------------
            for (k = 0; k < i; k++)
                    L[j][j] = L[j][j] - L[j][k] * L[j][k];
            //Should be done by one thread
            L[j][j] = sqrt(L[j][j]);
            //Step 3: calculate Lower triangular elements
            //OMP Loop 2
            //-------------------------
            for (i = j+1; i < n; i++) {
                for (k = 0; k < j; k++)
                    L[i][j] = L[i][j] - L[i][k] * L[j][k];
                L[i][j] = L[i][j] / L[j][j];
            }
        }
    }
    return L;
}
```

Tasks:

(a) The code `L[j][j] = sqrt(L[j][j])` should be executed by one thread. Show two ways to achieve this using OpenMP.

(b) Write the missing OpenMP pragmas for the loops "OMP Loop 1" and "OMP Loop 2". Explain your choices of shared and private list of variables.

(c) The loop "OMP Loop 1" contains a race condition. (1) Explain what causes the race condition, and (2) provide at least two ways to solve the race condition.

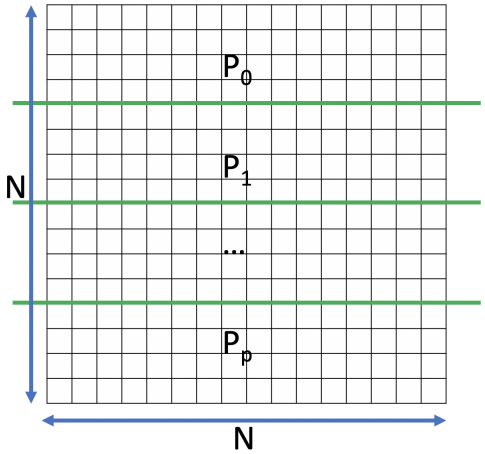
(d) Finally we ask you to consider if the outer j-loop can be parallelized. Explain in less than 3 lines why this is, or why this is not possible.

Problem 5 (10p)

The following code is used to approximate the solution of the Poisson problem $\nabla^2 u = f$ on a square matrix:

```
void compute(float** anew, float** aold, int j_start, int j_end,
             int i_start, int i_end){
    for(int j=j_start; j<j_end; ++j) {
      for(int i=i_start; i<i_end; ++i)
        anew[i][j] = (aold[i][j]+
                      aold[i-1][j] + aold[i+1][j] +
                      aold[i][j-1] + aold[i][j+1]) / 5.0;
    }
}
int main() {
  ...
  for (int time = 0; time < MAX_TIME; time++){
    compute(pnew, pold, j_start, j_end, i_start, i_end);
    pold = pnew; // update pold for next iteration
  }
  ...
}
```

This is a 2D 5-point stencil computation. The size of the matrix is N^2 and it is divided among *P* MPI processes as shown in the figure below.



Domain decomposition among *P* processes

Tasks:

(a) How many matrix elements are exchanged in each iteration? Provide the solution as a function of *N* and *P*.

(b) Add the necessary MPI calls to `main()` in order to exchange information. Make sure it does not cause deadlock.

(c) Now consider that each process checks for the convergence condition on its own region. The computation should stop if all processes agree that the solution has converged. An implementation of this is shown in the code below. Suggest an MPI implementation (in Psuedo code) that breaks the *time loop* (line 17) if all processes indicate convergence.

```
1  int compute(float** anew, float** aold, int j_start,
2              int j_end, int i_start, int i_end){
3      for(int j=j_start; j<j_end; ++j) {
4        for(int i=i_start; i<i_end; ++i)
5          anew[i][j] = (aold[i][j]+
6                        aold[i-1][j] + aold[i+1][j] +
7                        aold[i][j-1] + aold[i][j+1]) / 5.0;
8      }
9      if(delta(anew, aold) < 0.05)
10         return 1; //solution converged
11     else
12         return 0;
13 }
14
15 int main() {
16   ...
17   for (int time = 0; time < MAX_TIME; time++){
18     converge = compute(pnew, pold, j_start, j_end,
19                        i_start, i_end);
20     if(converge) // each process should check convergence
21         break;
22     else
23         pold = pnew; // update pold for next iteration
24   }
25   ...
26 }
```

(d) Add the necessary code to measure the execution time in each iteration. How can you make sure that all processes have completed before measuring the time?

A subset of MPI calls that are useful for this problem are shown below

```
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 dest, int tag, MPI_Comm comm, MPI_Status *status)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Barrier( MPI_Comm comm )
int MPI_Bcast(void* data, int count, MPI_Datatype datatype,
        int root, MPI_Comm communicator)
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_Allgather(const void *sendbuf, int sendcount,
        MPI_Datatype sendtype, void *recvbuf, int recvcount,
        MPI_Datatype recvtype, MPI_Comm comm)
int MPI_Alltoall(const void *sendbuf, int sendcount,
        MPI_Datatype sendtype, void *recvbuf, int recvcount,
        MPI_Datatype recvtype, MPI_Comm comm)
```

Problem 6 (10p)

The CUDA program below computes the outer product of two vectors, `u[]` and `v[]`. The outer product is a specific case of matrix multiplication in which the first matrix is a (column) vector of N elements and the second matrix is the transpose of a vector (also called a row vector) of N elements. When we multiply a N×1 matrix by a 1×N matrix, the result is an N×N matrix, which we call the outer product of the two vectors. The code below calculates the outer product of vector `u[]` with vector `v[]` and returns the answer as matrix `A[][]`.

```
#define BLOCK_DIM 32
__global__ outer_product_kernel (float* u, float* v, float* A,
  unsigned int N){
  /* Perform the outer product of u and v,
  * u is of size N x 1
  * v is of size 1 x N
  * A is of size N x N */
  unsigned int row = blockIdx.y * blockDim.y + threadIdx.y;
  unsigned int col = blockIdx.x * blockDim.x + threadIdx.x;
  if(row<N && col<N){
      A[row * N + col] = u[row] * v[col];
  }
}

void outer_product (float* u, float* v, float* A, unsigned int N)
{
  .....
  // sizeof(float) = 4 bytes
  cudaMalloc((void **) &u_d, N * sizeof(float));
  cudaMalloc((void **) &v_d, N * sizeof(float));
  cudaMalloc((void **) &A_d, N * N * sizeof(float));
  cudaMemcpy(1, 2, N * sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(3, 4, N * sizeof(float), cudaMemcpyHostToDevice);
  dim3 blockDim (5);
  dim3 gridDim (6);
  outer_product_kernel<<<gridDim, blockDim>>>(u_d, v_d, A_d, N);
  cudaMemcpy(7, 8, N * N * sizeof(float), cudaMemcpyDeviceToHost);
  .....
}
```

Tasks:

(a) Considering the CUDA implementation above. Fill in the missing code pieces 1 to 8 in the function `outer_product`. You may find the following API helpful:
`cudaMemcpy(Destination, Source, Size in bytes to copy, Type of transfer)`

(b) Now we consider using tiling and shared memory to optimize the kernel. Each thread block computes a tile of matrix A. Parts of vectors `u[]` and `v[]` are loaded from global memory to shared memory. Assume the size of shared memory on the GPU is 256 bytes. How many elements of vectors `u[]` and `v[]` can be loaded into shared memory?

(c) Rewrite the CUDA kernel (`outer_product_kernel`) to make use of tiling and shared memory as described in part (b).