INF3380: Summary

Outline

INF3380: an introduction to *parallel programming*

- Why?
  - We want to solve *larger* scientific problems *faster*
  - Parallel hardware is now widespread
- How?
  - Given a problem, identify parallelism
  - Design & analysis of parallel algorithms
  - Implementation using OpenMP and/or MPI programming
- Concrete examples

Note: This summary is very general, you should read the book for details!

Decomposition (Chap. 3)

- Decomposition is the first step of developing a parallel algorithm
- A given problem may be decomposed into tasks, in many different ways
- A decomposition can be represented by a task dependency graph:
  - Nodes correspond to tasks
  - Edges indicate that the result of one task is required for processing the next

Granularity and degree of concurrency (Chap. 3)

- Granularity depends on the number of tasks from decomposition
  - fine-grained decomposition
  - coarse-grained decomposition
- Degree of concurrency: the number of tasks can be executed in parallel
  - may change as the execution proceeds
  - finer granularity $\rightarrow$ increased concurrency

Limits on parallel performance (Chap. 3)

- It would appear that the parallel time can be made arbitrarily small by making the decomposition finer in granularity.
- There is, howeer, an inherent bound on how fine the granularity of a computation can be.
- Concurrent tasks often have to exchange data with other tasks. This results in communication overhead.
- There is a tradeoff between the granularity of a decomposition and associated overheads.

Processes and mapping (Chap. 3)

- In general, the number of tasks from a decomposition exceeds the number of processing elements available
- A parallel algorithm thus must also provide a mapping of tasks to processing elements
- Appropriate mapping is important for parallel performance
  - load balancing
  - interaction minimization
  - assigning tasks on critical path to processing elements as soon as possible

Point-to-point communication cost

A simple cost model:

$$t_{\mathrm{comm}} = t_s + m t_w$$

t_s — startup time
t_w — per-word transfer time
m — amount of data transferred

Group communication (Chap. 4)

- Many interactions in practical parallel programs occur in well-defined patterns involving groups of processors.
- Group communication operations are built using point-to-point messaging primitives.
- The actual cost of a group communication depends on
  - the type of communication
  - the number of processors involved
  - the communication network used

Analytical modeling (Chap. 5)

- The parallel runtime T_P of a program depends on the input size the parallel system and p
- T_S: serial time.
- Total overhead: $T_o = T_{all} - T_S = pT_P - T_S$

Speedup (Chap. 5)

$$S(p) = \frac{T_S}{T_P(p)}$$

- Always consider the best sequential program as the baseline
- Speedup is normally bounded by p, but can have exceptions (superlinear speedup)
- Parallel efficiency: $E = S(p)/p$

Cost optimality (Chap. 5)

- Total cost of a parallel system: $p \times T_P$
- A parallel system is said to cost-optimal if $p \times T_P$ is asymptotically identical with T_S
- Parallel efficiency $E = O(1)$ for cost-optimal systems

Scaling (Chap. 5)

- Efficiency:

$$E = \frac{S}{p} = \frac{T_S}{pT_P} = \frac{1}{1 + \frac{T_o}{T_S}}$$

- Note: total overhead T_o is typically an increasing function of p
- To maintain a constant level of E, we have to increase the problem size as the same time as p increases
  - if yes $\rightarrow$ scalable parallel system

Maintaining parallel efficiency (Chap. 5)

- At what rate should the problem size be increased, with respect to p, if we want to maintain a constant parallel efficiency?
- This rate determines the scalability of the system, the slower the better
- Problem size W: the asymptotic number of operations associated with the best serial algorithm to solve the problem

Isoefficiency metric (Chap. 5)

- Recall

$$T_P = \frac{W + T_o(W,p)}{p}$$

$$S = \frac{W}{T_P} = \frac{pW}{W + T_o(W,p)}$$

- Therefore,

$$E = \frac{S}{p} = \frac{W}{W + T_o(W,p)} = \frac{1}{1 + T_o(W,p)/W}$$

More about isoefficiency metric

- From $E = \frac{1}{1+T_o(W,p)/W}$ we can get

$$\frac{T_o(W,p)}{W} = \frac{1-E}{E} \quad \Rightarrow \quad W = \frac{E}{1-E}T_o(W,p)$$

- For a desired efficiency E, we will have a constant $K = E/(1-E)$, which tells us that W must grow as fast as

$$W = KT_o(W,p)$$

- That is, W can usually be obtained as a function of p for maintaining efficiency — isoefficiency function

Serial fraction (Chap. 5)

Suppose

$$\begin{aligned} W &= T_{ser} + T_{par} \\ T_P &= T_{ser} + \frac{T_{par}}{p} = T_{ser} + \frac{W - T_{ser}}{p} \end{aligned}$$

Then we can define the serial fraction as

$$f = \frac{T_{ser}}{W}$$

Therefore

$$T_P = f \times W + \frac{W - f \times W}{p} = W \times \left(f + \frac{1-f}{p} \right)$$

MPI programming (Chap. 6)

- MPI is the de-facto standard of message passing programming
- Assumption: each process's own memory is not directly accessible by other processes
- Collaboration between the processes is through sending and receiving messages between the processes
  - a message is an array of predefined data types
  - point-to-point communication
  - collective communication
- The global data structure is normally divided among the processes (as little duplication as possible)

MPI basics (Chap. 6)

- The working units are called MPI processes
- An MPI communicator is group of processes
- Each process within a communicator has a unique rank, between `0` and `#procs-1`
- Carelessly programmed MPI communications may deadlock
- Non-deterministic features of an MPI program
  - Between communications, the different processes may proceed at different paces
  - If a process is expecting two messages from two senders, the order of arrival is normally not known beforehand
- Synchronization
  - explicit – `MPI_Barrier`
  - implicit – collective commands or matching `MPI_Send` and `MPI_Recv`

Overlap communication with computation (Chap. 6)

- Performance may be improved on many systems by overlapping communication with computation
- Use of non-blocking communication and completion routines
- For example, initiate the communication with `MPI_Isend` and `MPI_Irecv`, continue with computation, finish with `MPI_Wait`

OpenMP programming (Chap. 7)

- OpenMP is the most user-friendly thread programming standard
- Thread programming is a natural model for shared-memory architecture
  - Execution unit: thread
  - Many threads have access to shared variables
  - Information exchange is (implicitly) through the shared variables

The programming model of OpenMP (Chap. 7)

- Multiple cooperating threads are allowed to run simultaneously
- The threads are created and destroyed dynamically in a **fork-join** pattern
  - An OpenMP program consists of a number of parallel regions
  - Between two parallel regions there is only one master thread
  - In the beginning of a parallel region, a team of new threads is spawned
  - The new threads work simultaneously with the master thread
  - At the end of a parallel region, the new threads are destroyed

The memory model of OpenMP (Chap. 7)

- Most variables are shared between the threads
- Each thread has the possibility of having some private variables
  - Avoid race conditions
  - Passing values between the sequential part and the parallel region
- Very important to decide: which variables should be shared? which should be private?

Practicalities

- First step: identify parallelism in a sequential algorithm
    - find out the operations that can be done simultaneously
- Good work division is important
    - even distribution of the work load among computational units
    - keep the overhead of resulting communication low
- On distributed memory, data should be divided as well
- Be aware of needed synchronizations (both MPI and OpenMP)
- Be aware of possible deadlocks (both MPI and OpenMP)
- Be aware of possible racing conditions (OpenMP)

Matrix-vector multiplication (Chap. 8)

- Multiply a dense $n \times n$ matrix A with an $n \times 1$ input vector x to yield an $n \times 1$ result vector y.
- Rowwise 1D partitioning
- 2D block partitioning

Matrix-matrix multiplication (Chap. 8)

- $C = A \times B$, where A, B and C are al square $n \times n$ matrices
- 2D block data partitioning, each block is an $(n/q) \times (n/q)$ submatrix.
  - Simple parallel algorithm
  - Cannon's algorithm
  - The DNS algorithm

Solving a system of linear equations (Chap. 8)

- $Ax = b$ where A is an $n \times n$ square (dense) matrix
- Parallelization of a simple Gaussian elimination algorithm

```
1.    procedure GAUSSIAN_ELIMINATION ($A$, $b$, $y$)
2.    begin
3.       for $k := 0$ to $n-1$ do                  /* Outer loop */
4.       begin
5.          for $j := k+1$ to $n-1$ do
6.             $A[k,j] := A[k,j]/A[k,k]$;   /* Division step */
7.          $y[k] := b[k]/A[k,k]$;
8.          $A[k,k] := 1$;
9.          for $i := k+1$ to $n-1$ do
10.         begin
11.            for $j := k+1$ to $n-1$ do
12.               $A[i,j] := A[i,j] - A[i,k] \times A[k,j]$; /* Elimination step */
13.            $b[i] := b[i] - A[i,k] \times y[k]$;
14.            $A[i,k] := 0$;
15.         endfor;             /* Line 9 */
16.      endfor;                /* Line 3 */
17.   end GAUSSIAN_ELIMINATION
```

Parallel sorting (Chap. 9)

- Odd-even transposition
- Shellsort
- Quicksort

Parallel graph algorithms (Chap. 10)

- Minimum spanning tree: Prim's algorithm
- Single-source shortest paths: Dijkstra's algorithm
- All-pairs shortest paths: Dijkstra's algorithm & Floyd's algorithm

About the exam

- 4-hour written exam
- One A4-sheet (two-sided) with handwritten notes is allowed at the exam