DTS202TC Foundamentals of Parallel Computing

Lecture 1

Hong-Bin Liu

Xi'an Jiaotong Liverpool University

November 6, 2023

Academic Team - Module Leader

- Hong-Bin Liu is an assistant professor at School of AI and Advanced Computing (AIAC). Before joining XJTLU, he obtained his PhD from James Cook University and Master of Computer Science from RMIT, Australia. And he worked as a full-stack developer for 5 years both in China and Australia. His research interests include Artificial Intelligence, Spatio-Temporal Reasoning, and Computer Vision etc.
- Email: hongbin.liu@xjtlu.edu.cn
- Office: D5004
- Office telephone number: 0512- 88970598
- Office hour: Thursday after lectures

Academic Team - Co-teacher

- Dr. Maruf Hasan is an Associate Professor at the School of AI and Advanced Computing in XJTLU. He has decades of experience in teaching a broad range of computing courses in computer science and information system disciplines. He is a Certified Information System Security Professional. He lived and worked internationally and supervised several postgraduate theses and undergraduate projects. He also received numerous research grants and published his research regularly in Machine Learning, Natural Language Processing, Digital Library, E-learning, Artificial Intelligence and Semantic Web Technologies.
- Email: MdMaruf.Hasan@xjtlu.edu.cn
- Office: D5009
- Office telephone number: 0512-88167118
- Office hour: Tuesday 14:00–16:00 and online by appointment

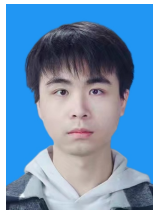
Academic Team - Co-teacher

- Di Zhang is an associate professor at School of AI and Advanced Computing (AIAC). PhD of CS from Communication University of China (directed by CAS) and B.Sc and M.Sc of CS from BUAA. 15+ years of working experience in Institute of Software (CAS), Nokia Research Center, and Noah's Ark Lab. Huawei. Research Domain: Probabilistic Graphical Models, Large-scale Numerical Library. Application Domain: Telecom Big Data.
- Email: di.zhang@xjtlu.edu.cn
- Office: D5026
- Office telephone number: 0512- 89167604
- Office hour: Monday, 9:00 - 10:00

Academic Team - Teaching Assistant


Haoyan Gong
Haoyan.Gong21


Yuzheng Feng
Yuzheng.Feng21


Zhenrong Zhang
Zhenrong.Zhang21

Module Overview

- Prerequisites: *DTS102TC* and *MTH007*
- Learning outcomes:
  - Identify serial and parallel algorithms.
  - Devise and implement parallel algorithms.
  - Identify and solve a computation problem with parallel algorithm.

Module Content

1. Week 8 (Nov. 6-13): Introduction to Parallel and C Programming. } Hongbin Liu
2. Week 9 (Nov. 20-26): Share-Memory Programming with Pthreads } Md Maruf
3. Week 10 (Nov. 13-19): Distributed-Memory Programming with MPI } Di Zhang
4. Week 11 (Nov. 27 - Dec. 3) : Shared-Memory Programming with OpenMP } Hongbin L
5. Week 12 (Dec. 4-10): Intro to CUDA (Course provided by Nvidia) } Xiaodong Yang

Timetable

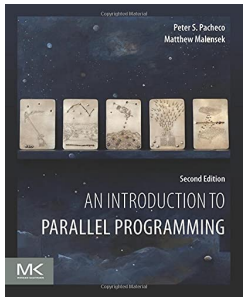
	Monday	Tuesday	Wednesday	Thursday	Friday
9:00					DTS202TC-Comp.Lab-D1/3 TC-D-3001 Week:8-13 Time:9AM - 10:50AM
9:30					
10:00					
10:30					
11:00					
11:30					
12:00					
12:30					
13:00	DTS202TC-Lecture-D1/1 TC-AB-2002 Week:8-12 Time:1PM - 3:20PM	DTS202TC-Seminar-D1/1 TC-AB-2002 Week:13 Time:1PM - 2:50PM; DTS202TC-Tutorial-D1/1 TC-D-2008 Week:8-12 Time:1PM - 1:50PM	DTS202TC-Tutorial-D2/1 TC-D-2008 Week:13 Time:1PM - 2:50PM	DTS202TC-Lecture-D2/1 TC-AB-2002 Week:8-12 Time:1PM - 3:20PM	DTS202TC-Comp.Lab-D1/1 TC-D-3001 Week:8-13 Time:1PM - 2:50PM
13:30					
14:00		DTS202TC-Tutorial-D1/2 TC-D-2008 Week:8-12 Time:2PM - 2:50PM			
14:30					
15:00		DTS202TC-Tutorial-D1/3 TC-D-2008 Week:8-12 Time:3PM - 3:50PM	DTS202TC-Tutorial-D2/2 TC-D-2008 Week:13 Time:3PM - 4:50PM		DTS202TC-Comp.Lab-D1/2 TC-D-3001 Week:8-13 Time:3PM - 4:50PM
15:30					
16:00					
16:30					
17:00		DTS202TC-Tutorial-D2/3 TC-D-2008 Week:13 Time:5PM - 6:50PM			
17:30					
18:00					
18:30					

Assessments

- Group Assessments 1, **30**%, due Sunday *Nov 19th, 2023* @ 11:59pm (Week 9)
- Individual Assessments 2, **70**%, due Saturday *Dec. 23rd, 2023* @ 11:59pm (Week 14)

Text Book and Readings

- Peter Pacheco, An Introduction to Parallel Programming, Second Edition, Morgan-Kaufmann, Paperback ISBN: 9780128046050
- C Programming: A Modern Approach, Second Edition, K. N. King
- Optional reading material will be posted every week.



Important Information

- Submit all team members' information before 13th Nov, 23:59 via LMO
- 5% late penalties per day, 0 mark after 5 days late.
- There is no exam.
- CUDA will be taught by Nvidia DLI, you will get a certificate if you passed the test.
- Please post questions on discussion board.
- Attendance is important!!

Outline

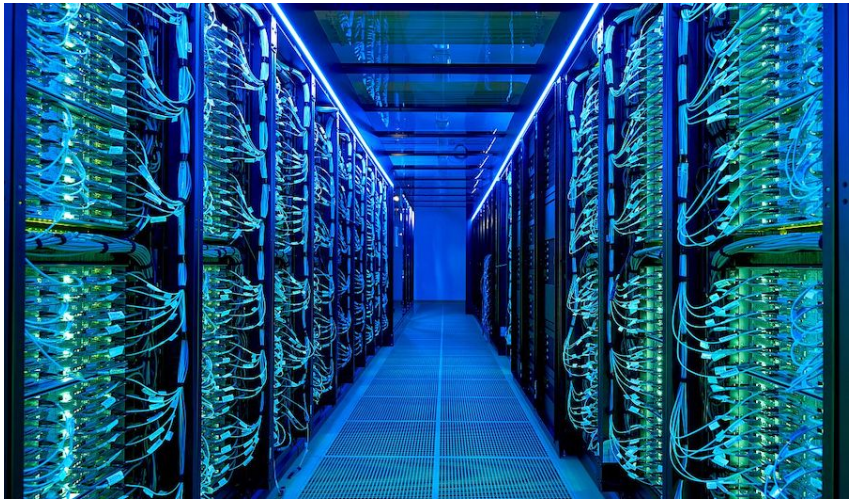
What is Parallel Computing?

Xi'an Jiaotong-Liverpool University 西交利物浦大學

Using multiple processors in parallel to solve problems more quickly than with a single processor.

Why Parallel Computing?

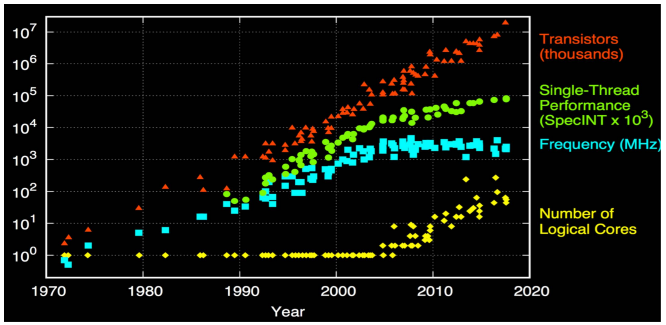
Xi'an Jiaotong-Liverpool University 西交利物浦大學



Why Parallel Computing?

Xi'an Jiaotong-Liverpool University 西交利物浦大學

- From 1986 – 2002, microprocessors were speeding like a rocket, increasing in performance an average of 50% per year.
- Since then, it's dropped to about 20% increase per year.



M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten, and K. Rupp

Why Parallel Computing?

Xi'an Jiaotong-Liverpool University 西交利物浦大學

- Instead of designing and building faster microprocessors, put multiple processors on a single integrated circuit.
- Adding more processors doesn't help much if programmers aren't aware of them...
- ...or don't know how to use them.
- Serial programs don't benefit from this approach (in most cases).

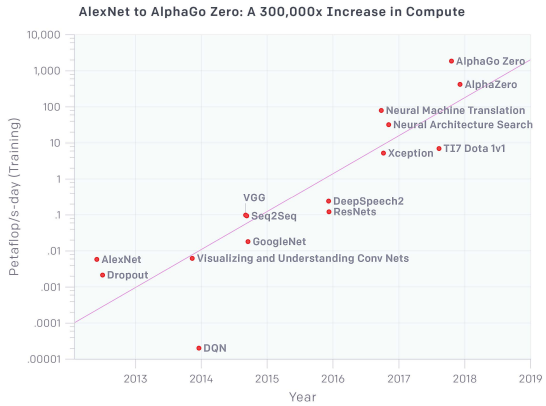
Why Parallel Computing?

- Computational power is increasing, but so are our computation problems and needs.
- Problems we never dreamed of have been solved because of past increases, such as decoding the human genome.
- More complex problems are still waiting to be solved.

Why Parallel Computing?

- Climate models, large-scale, more realistic simulations
- Machine Learning (deep learning)
- Bioinformatics
- Games! VR requires massive computational capabilities

Why Parallel Computing?



https://blog.openai.com/ai-and-compute/

Outline

Approaches to the serial problem

- Rewrite serial programs so that they're parallel.
- Write translation programs that automatically convert serial programs into parallel programs.
  - This is very difficult to do.
  - Success has been limited.

Example

- Compute n values and add them together.
- Serial solution:

```
1  sum = 0;
2
3  for (i = 0; i < n; i++) {
4    x = Compute_next_value(...);
5    sum += x;
6  }
```


Example (cont.)

- We have p cores, p much smaller than n.
- Each core performs a partial sum of approximately n/p values.

```
1  my_sum = 0;
2  my_first_i = ...;
3  my_last_i = ...;
4
5  for (my_i = my_first_i; my_i < my_last_i; my_i ++) {
6    my_x = Compute_next_value(...);
7    my_sum += my_x;
8  }
```

Each core uses it's own private variables and executes this block of code independently of the other cores.

Example (cont.)

- After each core completes execution of the code, is a private variable *my_sum* contains the sum of the values computed by its calls to *Compute_next_value*.
- Ex., 8 cores, $n = 24$, then the calls to *Compute_next_value* return:
  1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

Example (cont.)

- Once all the cores are done computing their private *my_sum*, they form a global sum by sending results to a designated "master" core which adds the final result.

Example (cont.)

```
if (I am the master core) {
  sum = my_x;
  for each core other than myself {
    receive value from core;
    sum += value;
  }
} else {
  send my_x to the master;
}
```


Example (cont.)

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Global sum

8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95

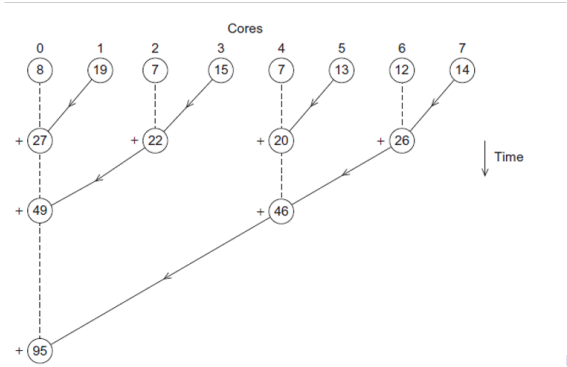
Core	0	1	2	3	4	5	6	7
my_sum	95	19	7	15	7	13	12	14

Better parallel algorithm

- Don't make the master core do all the work.
- Share it among the other cores.
- Pair the cores so that core 0 adds its result with core 1's result.
- Core 2 adds its result with core 3's result, etc.
- Work with odd and even numbered pairs of cores.

Better parallel algorithm (Cont.)

- Repeat the process now with only the evenly ranked cores.
- Core 0 adds result from core 2.
- Core 4 adds the result from core 6, etc.
- Now cores divisible by 4 repeat the process, and so forth, until core 0 has the final result.



How do we write parallel programs?

- Task parallelism
  - Partition various tasks carried out solving the problem among the cores.
- Data parallelism
  - Partition the data used in solving the problem among the cores.
  - Each core carries out similar operations on it's part of the data.

Division of work – data parallelism

```
1  sum = 0;
2
3  for (i = 0; i < n; i++) {
4    x = Compute_next_value(...);
5    sum += x;
6  }
```


Division of work – task parallelism

```
1  program:
2  ...
3  if CPU = "a" then
4      do task "A"
5  else if CPU="b" then
6      do task "B"
7  end if
8  ...
9  end program
```

Coordination

- Cores usually need to coordinate their work.
- Communication – one or more cores send their current partial sums to another core.
- Load balancing – share the work evenly among the cores so that one is not heavily loaded.
- Synchronization – because each core works at its own pace, make sure cores do not get too far ahead of the rest.

What we'll be doing

- Learning to write programs that are explicitly parallel.
- Using the C language.
- Using four different extensions to C.
  - Message-Passing Interface (MPI)
  - Posix Threads (Pthreads)
  - OpenMP
  - Nvidia CUDA

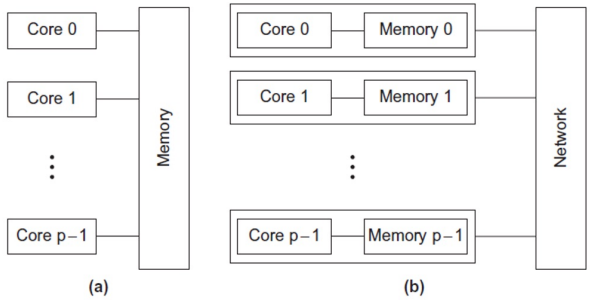
Outline

Terminology

- Concurrent computing – a program is one in which multiple tasks can be in progress at any instant.
- Parallel computing – a program is one in which multiple tasks cooperate closely to solve a problem.
- Distributed computing – a program may need to cooperate with other programs to solve a problem.

Type of parallel systems

- Shared-memory
  - The cores can share access to the computer's memory.
  - Coordinate the cores by having them examine and update shared memory locations.
- Distributed-memory
  - Each core has its own, private memory.
  - The cores must communicate explicitly by sending messages across a network.



Performance Measurement: Speedup and efficiency

- Speedup: Ratio of execution time on one process to that on p processes

$$Speedup = \frac{t_1}{t_p} \tag{1}$$

- Efficiency: Speedup per process

$$Efficiency = \frac{t_1}{t_p \times p} \tag{2}$$

Performance Measurement: Speedup and efficiency

p	1	2	4	8	16
S	1.0	1.9	3.1	4.8	6.2
E	1.0	0.95	0.78	0.60	0.39

Amdahl's law

- Speedup is limited by the serial portion of the code.
  - Often referred to as the serial "Bottleneck"
- Lets say only a fraction f of the code can be parallelised on p processes

$$Speedup = \frac{1}{(1-f) + f/p} \tag{3}$$

Amdahl's law

- Speedup is limited by the serial portion of the code.
  - Often referred to as the serial "Bottleneck"
- Lets say only a fraction f of the code can be parallelised on p processes

$$Speedup = \frac{1}{(1-f)+f/p} \tag{3}$$

$$Speedup = \frac{20s}{(20s-18s)+18s/p} \tag{4}$$

Outline

Designning Parallel Algorithm

- Foster's methodology
- Steps
  - Partitioning
  - Communication
  - Agglomeration
  - Mapping

Partitioning

- Domain decomposition
  - Decompose the data associated with a problem
  - We divide these data into small pieces of approximately equal size if possible.
- Functional decomposition
  - Divide the computation into disjoint tasks

Communication

- Work out messages that are to be sent and received.
- In domain decomposition problems, communication requirements can be difficult to determine.
- In contrast, communication requirements in parallel algorithms obtained by functional decomposition are often straightforward.

Agglomeration

- Combine tasks and communications identified in the first step into larger tasks.

Mapping

- Assign the composite tasks identified in the previous step to processes/threads.
- This should be done so that the communication is minimised, and each processes/threads gets roughly the same amount of work.
- Many algorithms developed using domain decomposition techniques feature a fixed number of equal-sized tasks and structured local and global communication. In such cases, an efficient mapping is straightforward.

An Example - Parallelizing the Trapezoidal Rule

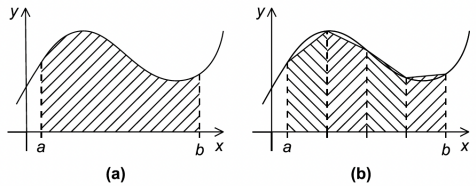
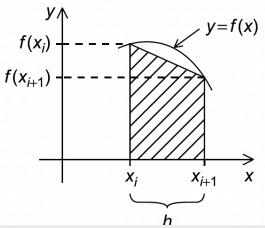
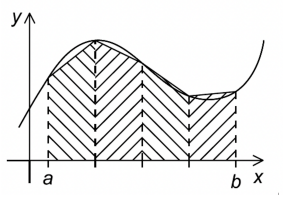


FIGURE 3.3

The trapezoidal rule: (a) area to be estimated and (b) approximate area using trapezoids.



An Example Cont.



- Try to identify all the tasks
- What are the communications between these tasks?
- Aggregate the computation of areas into groups
- Map the composite tasks to cores

More of Foster's Methodology

https://www.mcs.anl.gov/~itf/dbpp/text/node14.html

Summary

- Module introduction
- Why parallel computing?
- An parallelism example
- Terminology and definition

Next Lecture

- C Programming Language