Exam Assignments V01

chengchengguo 183090

1. Describe how **parallelism** differs from **concurrency**

concurrency: A system is **concurrent** if it can support two or more actions *in progress* at the same time. Concurrency means executing multiple tasks at the same time but not necessarily simultaneously.

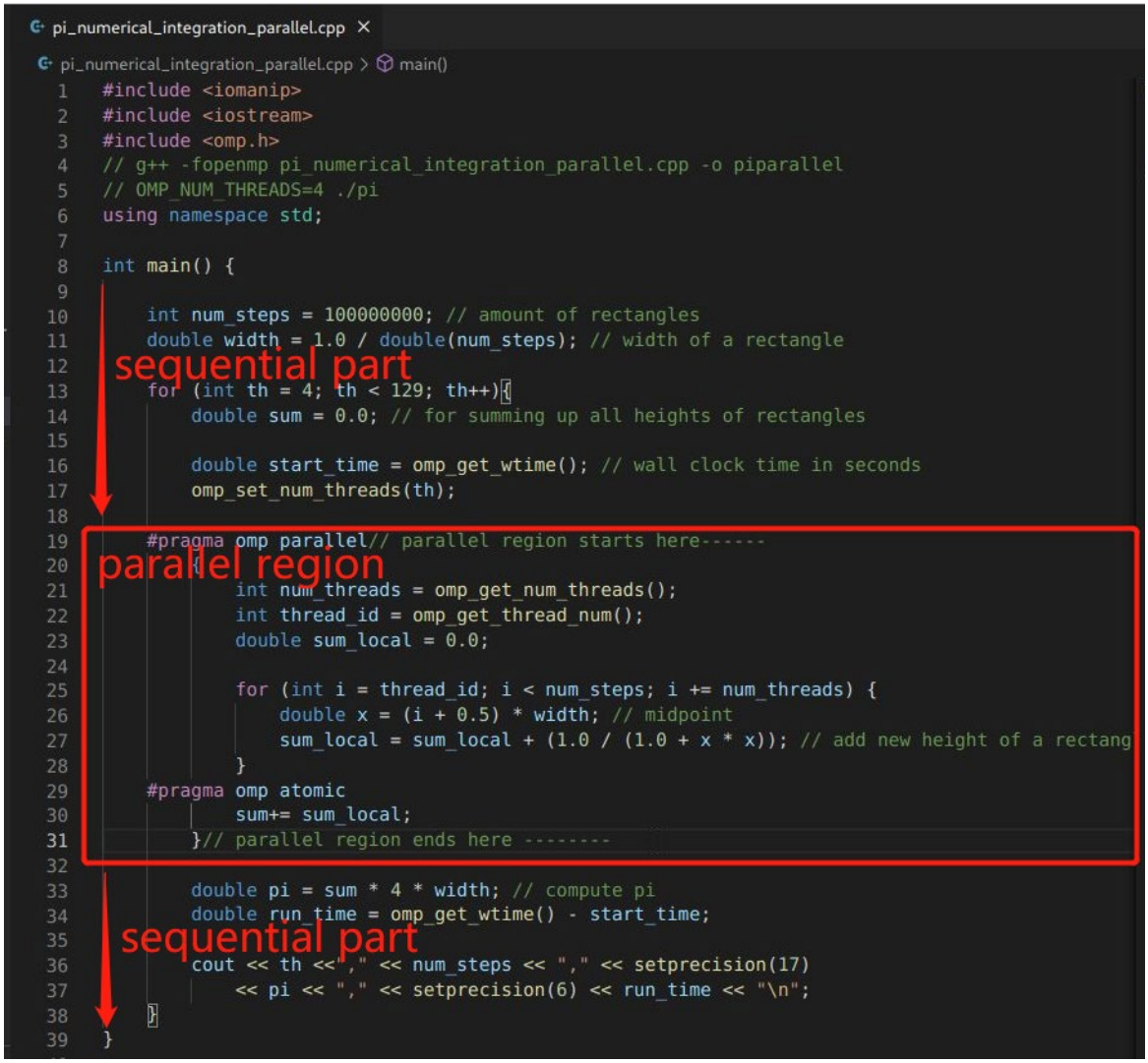
parallelism: A system is **parallel** if it can support two or more actions executing simultaneously. Parallelism is a specific kind of concurrency where tasks are really executed simultaneously.[1]

2. What is fork-join parallelism?

Fork/join parallelism is a style of parallel programming useful for exploiting the parallelism inherent in divide and conquer algorithms on shared memory multiprocessors.

The idea is quite simple: a larger task can be divided into smaller tasks whose solutions can then be combined. As long as the smaller tasks are independent, they can be executed in parallel.

[1] Concurrency vs. Parallelism — A brief view | by Madhavan Nagarajan | Medium



3. Read Chapter 1 from Computer Systems: A Programmer's Perspective[2]. Discuss one thing you find particularly interesting. (google it to find more information)

I found the part **1.9,1 Concurrency and Parallelism** is most interesting, because we have been exposed to this concept in class. This section explains it in more detail. It introduces 3 levels of parallelism from highest to lowest.

1) Thread-Level Concurrency

This form of concurrency allows multiple users to interact with a system at the same time, such as when many people want to get pages from a single web server. It also allows a single user to

[2] book.dvi (cmu.edu)

engage in multiple tasks concurrently, such as having a web browser in one window, a word processor in another, and streaming music playing at the same time.[3]

uniprocessor system.*:* allows a single user to engage in multiple tasks concurrently

multiprocessor system*:* a system consisting of multiple processors all under the control of a single operating system kernel

Multi-core processors have several CPUs (referred to as "cores") integrated onto a single integrated-circuit chip. Each core has its own L1 and L2 caches, but sharing the higher levels of cache as well as the interface to main memory.

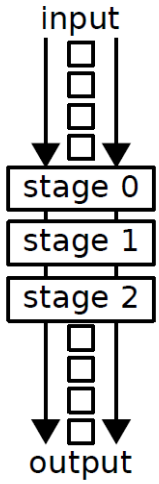
Hyperthreading, sometimes called *simultaneous multi-threading* is a technique that allows a single CPU to execute multiple flows of control. (As an example, the Intel Core i7 processor can have each core executing two threads, and so a four-core system can actually execute eight threads in parallel.)

The use of multiprocessing can **improve system performance** in two ways. **First**, it reduces the need to simulate concurrency when performing multiple tasks. **Second**, it can run a single application program faster, but only if that program is expressed in terms of multiple threads that can effectively execute in parallel. Thus, although the principles of concurrency have been formulated and studied for over 50 years, the advent of multi-core and hyperthreaded systems has greatly increased the desire to find ways to write application programs that can exploit the thread-level parallelism available with the hardware.

2) Instruction-Level Parallelism

At a much lower level of abstraction, modern processors can execute multiple instructions at one time, a property known as **instruction-level parallelism**.

it introduces the concept of ***pipelining***, where the actions required to execute an instruction are partitioned into different **steps**, and the processor hardware is organized as a series of **stages**, each performing one of these steps. The stages can operate in parallel, working on different parts of different instructions.



- Idea: Split complex operation into several simpler (faster) stages
- Benefit: If we process the stages in parallel, we are able to increase the throughput
- Drawback: Pipeline has to be filled to be efficient start-up latency
- Pipelining applied to an instruction implements Instruction Level Parallelism (ILP)
- Pipelining is also applied "high-level", e.g., for the parallelization of neural networks

[3] book.dvi (cmu.edu)

Example: Floating Point Multiplication:

$c = a \bullet b$

$a = s_1 \bullet 0.m_1 \bullet 10^{e1}$

$b = s_2 \bullet 0.m_2 \bullet 10^{e2}$

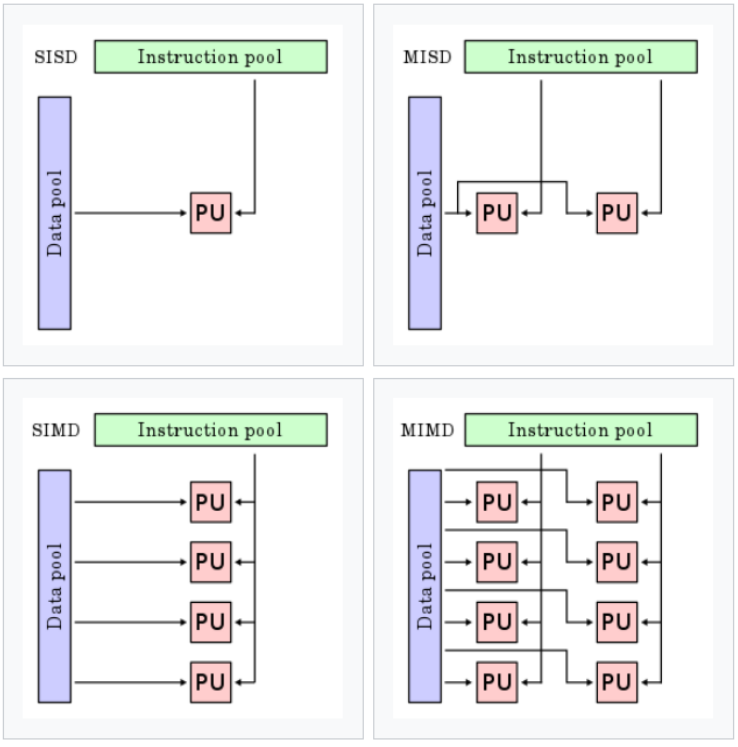
- Goal: Multiply to floating point values and given in a "normalized" representation
- Normalized:
- Sign bit (-1 or 1): s_1, s_2
  - Mantissa with non-zero leading digit: m_1, m_2
  - Exponent with positive or negative integer: e_1, e_2

3) Single-Instruction, Multiple-Data (SIMD) Parallelism

This is one of the Flynn classifications(below), it means: A single instruction is simultaneously applied to multiple different data streams. Instructions can be executed sequentially, such as by pipelining, or in parallel by multiple functional units. [4]

Flynnsche Klassifikation

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD



[4] Flynn's taxonomy - Wikipedia

4. Read the paper There's plenty of room at the Top:[5] What will drive computer performance after Moore's law?

Explain in detail the figure Performance gains after Moore's law ends. (on the first page)

Moore's law is the observation that the number of transistors in a dense integrated circuit doubles about every two years.

- room at the "Top.": opportunities for growth in computing performance by improvement of software, algorithms, and hardware architecture.
- room at the "Bottom.": means semiconductor miniaturization

Moore's Law predicts exponential growth, and clearly exponential growth of computer performance can't continue forever.

" Performance gains after Moore's law ends. In the post-Moore era, improvements in computing power will increasingly come from technologies at the "Top" of the computing stack, not from those at the "Bottom", reversing the historical trend.**"** means:

through processor simplification, where a complex processing core is replaced with a simpler core that requires fewer transistors. The freed-up transistor budget can then be redeployed in other ways. —for example, by increasing the number of processor cores running in parallel, which can lead to large efficiency gains for problems that can exploit parallelism.

In the post-Moore era, performance improvements from software, algorithms, and hardware architecture will increasingly require concurrent changes across other levels of the stack.

efficient cache utilization : restructure software to more predictable access patterns
Vectorization: use vector units for SIMD work

[5] Science Journals — AAAS (microsoft.com)