CS200 LECTURE 9

DISTRIBUTED ALGORITHMS

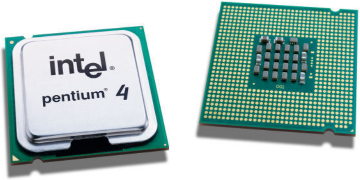
Po Hao Chen

June 4, 2023

BOSTON UNIVERSITY

Why Distributed Algorithms?

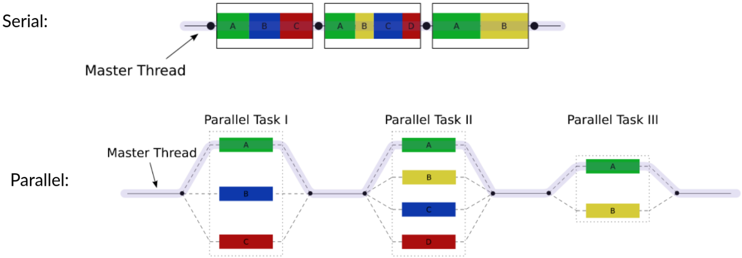
Historical Context



- So far, sequential algorithms
- CPUs used to be single core, not anymore!
- Distributed (parallel) algorithms make better use of hardware

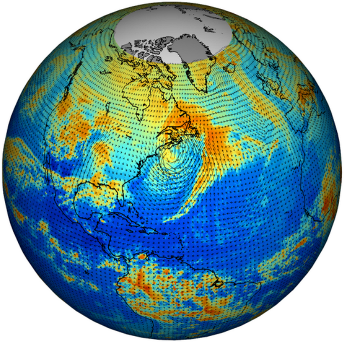
Parallelism

So, what is ~~paralism~~ anyways?
***parallelism**



Why Parallelism?

Example: Climate Simulation Model



- Modeling the exchange of matter and energy over time.
  - Wind and water currents
  - Atompheric pressure
  - Weather tracking
- How?
  - Cut globe into square grid
  - Equation characterize energy movement
  - Solving equations repeatedly

4 steps: **Decomposition, Assignment, Orchestration, Mapping**.

Speedup

How much of a speedup can we actually get?

- Depends on code itself
- Amdahl's Law -

$$\frac{1}{1 - p + \frac{p}{s}}$$

- `p` - fraction of original execution time that can be optimized
- `s` - speedup of optimized code over original code

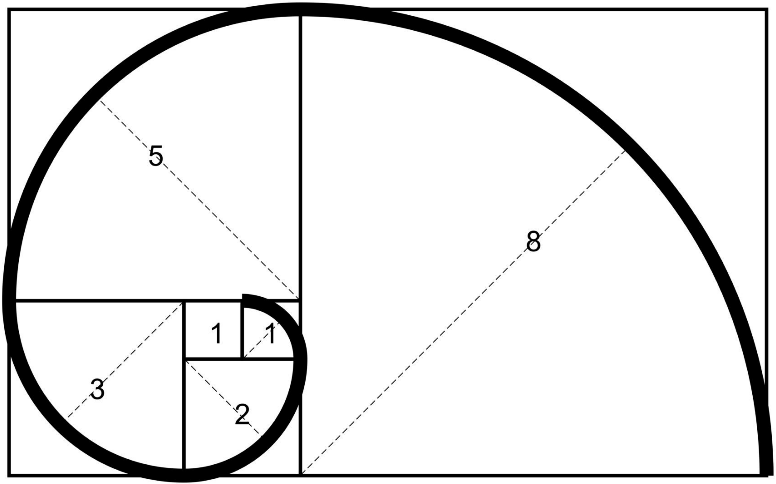
This throws practical applications in the face of theory

- distinction of `O(2n)` and `O(n)`

What are parallelizable?

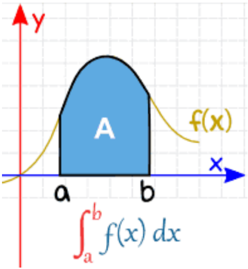
- Parallel execution is possible if the execution is tolerant of iteration reordering.
- Strategy: Look for decomposition oppurtunity in which parallel tasks can perform similar operation in the array.

Examples: Fibonacci Number

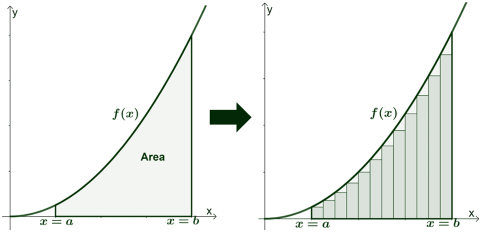


$$F_n = F_{n-1} + F_{n-2}$$

Example: Riemann Sums



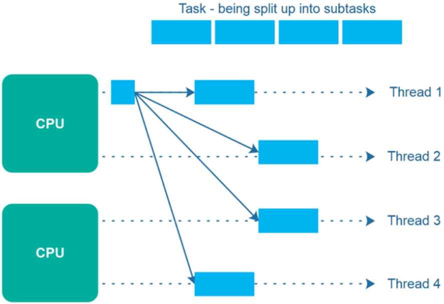
- Remember Riemann Sums?



$$\int_a^b f(x)d(x)$$

Thread Creation

- Programs/process usually start with 1 “master/main” thread
- Program creates additional threads to execute a specified function/code block in parallel
- Operating System schedules threads to run on available CPUs



OpenMP

API supports shared-memory parallel programming in C, C++, and Fortran.

- High-Level abstracted directives for application programmers and scientists
- easy to use!

OpenMP

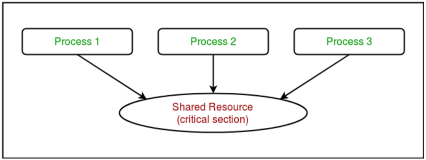
- **#include <omp.h>**
  - Imports the OpenMP library
- **omp_set_num_threads(<int NUM_THREADS>);**
  - Inside your main function, this command sets the number of threads you want openMP to use. Usually the same number as the number of CPU cores.
- **#pragma omp parallel**
  - Runs the following {block of code} in each thread
- **#pragma omp parallel for**
  - Include before for loop to divide the iterations of the loop into separate threads

- **omp_get_thread_num();**
  - Returns the current thread inside a parallel block
- **omp_get_num_threads();**
  - Returns the number of threads
- **#pragma omp master**
  - The following {} block will only run in 1 thread
  - Place inside a parallel code block
- **collapse(<int number of loops>)**
  - Example: #pragma omp parallel for collapse(3)
  - Parallelize n nested loops

Race Condition

Race Conditions (and how to use reduction)

- When Parallel threads need to modify the same data, errors can occur




```
#pragma omp parallel for private(x, i) reduction(+:integral)
for (i = 0; i < num_boxes; i++) {

    x = i*delta;
    // add the area of box to running sum
    integral += height(x)*delta;

}
```

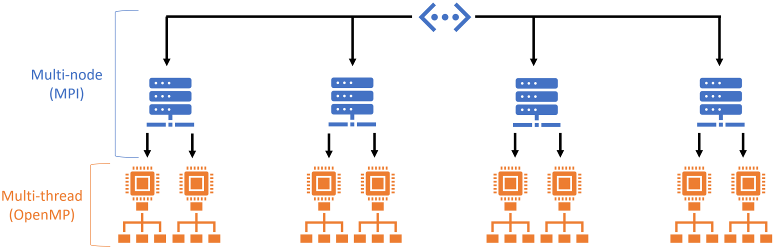
(calc_pi.cpp)

Let's Try It

We will approximate π by calculating the area of the unit circle.

- Integrate a quadrant of the circle, i.e, $\int_0^1 \sqrt{1-x^2}dx = \pi/4$
- We discretise the integral into n vertical boxes, each with $\Delta x = 1/n$ and height $= \sqrt{1-x_i^2}$ where $x_i = i\Delta x$

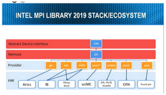
Levels of Parallelism



Message Passing Interface (MPI)

Message Passing Interface (MPI)

- MPI is a standard designed to allow for parallel functionality in a cluster
- Every implementation has to follow this standard
- Looks like function calls in code (library)
- Pass data between processes as messages
  - 1:1 transmit and receive



Hello World Example

MPI has a whole slew of new concepts

- process - instance of the program being run
- rank - unique identifier for a process (usually related to thread count)
- communicators - groups together MPI processes (Ex. `MPI_COMM_WORLD`)
- finalization - clean up performed at end of MPI program

Does MPI Work?

Let's take a look at the example of bitonic sort.

Bitonic sorter

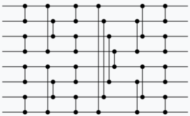
From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.
Find sources: "Bitonic sorter" – news · newspapers · books · scholar · JSTOR (October 2017) (*Learn how and when to remove this template message*)

Bitonic mergesort is a parallel algorithm for sorting. It is also used as a construction method for building a sorting network. The algorithm was devised by Ken Batcher. The resulting sorting networks consist of $O(n\log^2(n))$ comparators and have a delay of $O(\log^2(n))$, where n is the number of items to be sorted.[1]

A sorted sequence is a monotonically non-decreasing (or non-increasing) sequence. A *bitonic* sequence is a sequence with $x_0 \leq \cdots \leq x_k \geq \cdots \geq x_{n-1}$ for some $k, 0 \leq k < n$, or a circular shift of such a sequence.

Bitonic sorter



Bitonic sort network with eight inputs.

Class	Sorting algorithm
Data structure	Array
Worst-case performance	$O(\log^2(n))$ parallel time
Best-case performance	$O(\log^2(n))$ parallel time
Average performance	$O(\log^2(n))$ parallel time
Worst-case space complexity	$O(n\log^2(n))$ non-parallel time

Contents [hide]

Complexity [edit]

Let $p = \lfloor \log_2 n \rfloor$ and $q = \lceil \log_2 n \rceil$.

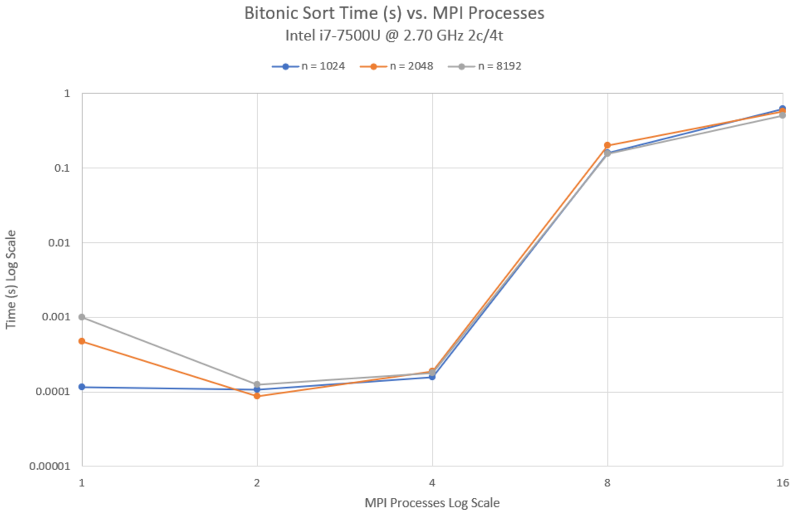
It is obvious from the construction algorithm that the number of rounds of parallel comparisons is given by $q(q+1)/2$.

It follows that the number of comparators c is bounded $2^{p-1} \cdot p(p+1)/2 \leq c \leq \lfloor n/2 \rfloor \cdot q(q+1)/2$ (which establishes an exact value for c when n is a power of 2).

Merge Sort Runtime: `O(nlogn)`

Parallel Bitonic Sort Runtime: $O(\log^2 n)$

Data for Bitonic Sort



Calculating Pi Example

Mathematicians are very concerned (perhaps to a worrying degree) about finding all the digits of π.

To date, 62.8 trillion digits of π have been calculated.

Leibniz formula:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - ... = \frac{\pi}{4}$$

We can calculate this in a distributed fashion!

Parallelization in Python

Using Ray for Highly Parallelizable Tasks

- For example, we may have 100,000 time series to process with exactly the same algorithm, and each one takes a minute of processing. (Ref. Ray Tutorial)
- Let's calculate π by throwing darts. (Monte Carlo)

Reference and Additional Materials

Special thanks to Carlton Knox and Benjamin Li for the contribution to the material.

For those who are interested for more:
https://github.com/buhpc/buhpc-workshops/tree/master/openmp