Introduction to high performance computing (HPC)

Oguz Kaya

Assistant Professor
Université Paris-Saclay and ParSys Team of LISN, Gif-sur-Yvette, France

université
PARIS-SACLAY

Introduction
oooo

Parallel computing, why?
oooo

Basics of parallel computing
oooooooooooo

Types of parallelism
ooooo

Parallel architecture
ooooooooooooooo

Outline

1 Introduction

2 Parallel computing, why?

3 Basics of parallel computing

4 Types of parallelism

5 Parallel architecture

Outline

Objectives

- Get familiar with parallel and high performance computing (HPC)
- Discover applications that require HPC
- Explore modern parallel computer achitectures
- Present major parallel programing models
- Introduce fundamental concepts in HPC
- Provide advices for efficient parallel programming and code optimization

Sequential programs

Traditionally, programs are based on a **sequential** execution:
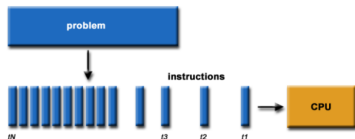
- A problem is split into a series of basic instructions.
- These instructions are executed **sequentially** one after the other.
- Instructions are executed by a **single processor**
- At a given time unite (clock tick), **a single instruction** is executed.
- Runtime of the program (or performance) mostly depends on the **frequency (Hz)** of the processor.

Introduction
○○○●

Parallel computing, why?
○○○○

Basics of parallel computing
○○○○○○○○○○○○

Types of parallelism
○○○○○

Parallel architecture
○○○○○○○○○○○○○○

Parallel programs

A **parallel program** enables the utilization of multiple processing units to solve a problem:
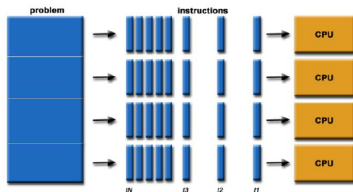

- A problem is split into subproblems that can be executed simultaneously.
- Each subproblem is split into basic instructions.
- Instructions in each subproblem are executed **in parallel** by using one processor per subproblem.
- The performance is mostly determined by:
  - frequency (Hz)
  - number of processors
  - degree of parallelization of the problem
  - and other characteristics (load balancing, overheads due to parallelism)

Outline

Applications of parallel computing

Numerous applications require immense compute power in diverse domains:
bigskip
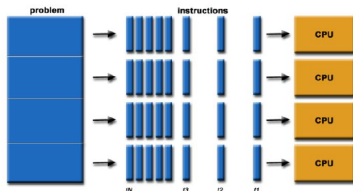
- Scientific computing: Simulations in physics, chemistry, biologie, . . .
- Machine learning: Neural networks, big data analysis, . . .
- Computer graphics: Rendering, games, video editing, . . .
- Operating systems: Linux, Android, . . .
- and many others (computational finance, CAD, medicine, . . .)

Introduction
0000

Parallel computing, why?
0000

Basics of parallel computing
000000000000

Types of parallelism
00000

Parallel architecture
0000000000000

Maximum frequency of a CPU in 2002

What is the maximum frequency of a CPU in 2002?

- 3.06 GHz

Intel Pentium 4, Northwood

Maximum frequency of a CPU in 2020

What is the maximum frequency of a CPU in 2020?

Intel Core i9-10900K, Comet Lake

- 5.3 GHz
- What about peak performance? 5.3 operations/s?
- No! $\approx$1.7Tops/s (1700Gflops/s) How?
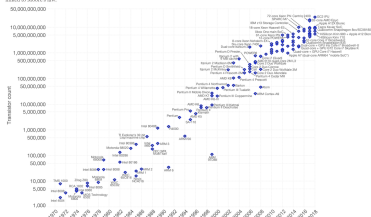- Thanks to the parallelism inside the CPU (10 cores, each having two 8-wide vector units)

Outline

Moore's Law

The number of transistors in integrated circuits double every two years.

Moore's Law

- What is it's importance?
- More transistors $\rightarrow$ more performance potential
- Moore's Law is still mostly valid today (yet slowed down slightly)

Dennard Scaling

If the feature size is reduced by 30% (0.7x) in each generation of transistors, it yields

40 Years of Microprocessor Trend Data
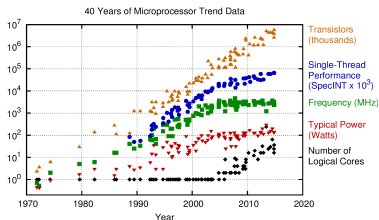
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Dennard Scaling

- Circuit size reduction of 50%
- Latency reduction of 30%
- As a result, frequency increase of 40% ($10/7 \approx 1.4$).
  - Higher frequency difficult to attain due to quantum effects!
  - Resort to multi-cores, or "wider" cores.
- $energy \sim frequence^2$
  - Use more cores at a lower frequency $\rightarrow$ more performance in the same power window.

universite
PARIS-SACLAY

Acceleration and efficiency

Acceleration/Speedup: How much faster does a parallel program run?
Efficiency: How well are computing resources used?

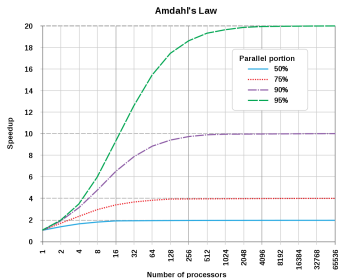Cray-1 Supercomputer (1975) with peak performance of $\approx$ 160Mflops/s

- Sequential execution time: $T(1)$
- Parallel execution time using N processors: $T(N)$
- **Acceleration**: $S(N) = T(1)/T(N)$
- **Efficiency**: $E(N) = S(N)/N$

Amdahl's Law

How much faster can a program run? What's the limit?

Amdahl's Law

- Introduced in 1967 by Gene Amdahl
- Let s and p be **sequential** and **parallel** franctions of a program ($s + p = 1.0$).
- Attainable acceleration is limited by
  $$A(N) = \frac{T(1)}{T(N)} \leq \frac{T(1)}{pT(1)/N + sT(1)} = \frac{1}{p/N + s}$$
- **Strong scaling:** Parallel scaling for a fixed problem size with increasing #processors.
- Loss of hope for parallel computing?
  - Typically, s diminishes with the problem size for almost all problems.

Example: Sum of an array
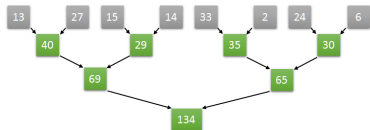
Parallel sum

$\sum_{i=0}^{N-1} A[i]$

- $N - 1$ additions
- For $N = 8$, 7 additions where 3 must be sequential, $S \leq 7/3 = 2.34$
- For $N = 16$, 15 additions where 4 must be sequential, $S \leq 15/4 = 3.75$
- In general, $N - 1$ additions where $\log N$ must be sequential, $S \leq (N - 1)/\log N$

Introduction
oooo

Parallel computing, why?
oooo

Basics of parallel computing
oooooo●ooooo

Types of parallelism
ooooo

Parallel architecture
ooooooooooooooo

Gustafson's Law

How much acceleration can be obtained if the problem size increases proportionally with the #processors?
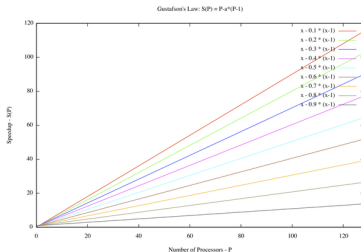
Gustafson's Law

- Amdalh: Difficult to accelerate a fixed computation with more processors.
- Gustafson: Possible to solve a bigger problem in a similar amount of time with more processors.
- Which one is more important nowadays?
- **Weak scaling:** Parallel scaling with proportional increase in the problem size and #processors.

université
PARIS-SACLAY

Introduction
oooo

Parallel computing, why?
oooo

Basics of parallel computing
ooooooo●oooo

Types of parallelism
ooooo

Parallel architecture
oooooooooooooo

# Flynn's classification of parallel machines

A 2×2 matrix to classify parallel machines

S I S D	S I M D
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
M I S D	M I M D
Multiple Instruction stream Single Data stream	Multiple Instruction stream Multiple Data stream

Flynn's classification

- Popular classification, used since 1966
- X axis: single/multiple instruction
- Y axis: single/multiple data
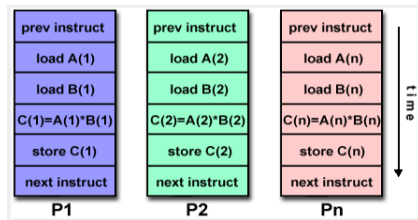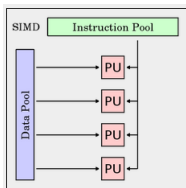
Single Instruction, Single Data (SISD)
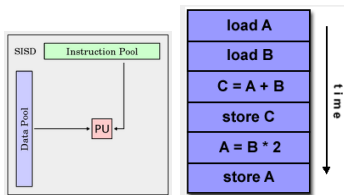
SISD processor

- Completely sequential and scalar machine
- "single" instruction: A single instruction is executed at each clock cycle
- "single" data: A single data element is used/modified (a.k.a. scalar).
- Execution is deterministic.
- Most ancient CPU architectures (e.g., basic computer achitecture course).
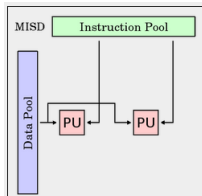
Single Instruction, Multiple Data (SIMD)

- An example of a parallel machine
- "single" instruction: All compute units execute the same instruction at each clock cycle
- "multiple" data: Each compute unit operates on a different data piece
- Useful for numerous problems and applications (matrix computations, image processing, ...)

Multiple Instruction, Single Data (MISD)

- A rare example of a parallel machine
- "single" data: A single data element is used/modified (a.k.a. scalar) by all compute units.
- "multiple" instructions: Each compute unit operates on the same data but executes a different instruction.
- A few example use cases:
  - applying multiple frequency filters on the same signal simultenously
  - multiple cryptography algorithms to decipher a single message
  - redundant operation for safety.
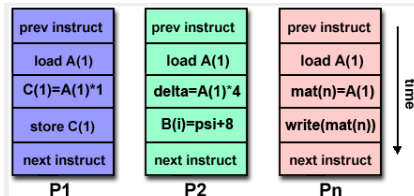
Multiple Instruction, Multiple Data (MIMD)

- A more modern example of a parallel machine
- "multiple" instructions: Each compute unit executes a different instruction.
- "multiple" data: Each compute unit operates on a different data piece
- Execution can be synchronous or asynchronous, deterministic or non-deterministic.
- Most processors today are in this category.
- Good for exploiting instruction level parallelism, and accelerate single threaded execution.
- Most MIMD architectures involve many SIMD components inside.
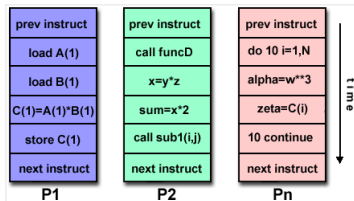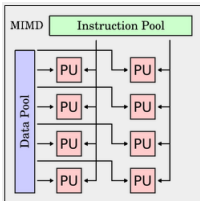
MIMD	Instruction Pool
Data Pool	→ PU ← ← PU ←
	→ PU ← ← PU ←
	→ PU ← ← PU ←
	→ PU ← ← PU ←

	P1	P2	Pn
	prev instruct	prev instruct	prev instruct
	load A(1)	call funcD	do 10 i=1,N
	load B(1)	x=y*z	alpha=w**3
	C(1)=A(1)*B(1)	sum=x*2	zeta=C(i)
	store C(1)	call sub1(i,j)	10 continue
	next instruct	next instruct	next instruct

Outline

Bit-level parallelism

It has to do with the word-size of the processor.

- Was more important in the past (1970..1986).
- Word-size: 4-bit → 8-bit → 16-bit → 32-bit
- Converged to 64-bit today, rarely need higher sizes (and can emulate with multiple instructions).

Intel C4004, 4-bit (1971) processor

Instruction-level parallelism

Enables to execute independent instructions simultenously in a program.

Non-pipelined and scalar execution
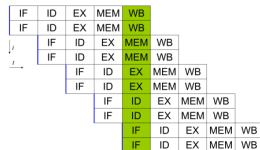
Pipelined and superscalar execution

- **a = b + c; c = d + e;**
  f = a + c; g = c - a;
- Possibility 1: Pipeline. Different stages of two (in)dependent instructions can be executed in the pipeline at the same time.
- Possibility 2: Superscalar execution. Independent instructions can use multiple execution units (ALUs, load/store, vector/floating units, . . .) in a single MIMD processor.
- Most modern processors are superscalar and have very deep pipelines.

Data Parallelism

Carry out same operations on a vector/array of data elements.

Sum of two arrays

- Uses the SIMD-type parallelism
- Most processors have efficient hardware for such computations (vector units in CPU and GPU)
- Very widespread use in scientific computing, AI, computer graphics, signal processing, . . .

Task Parallelism

Execution of a flow of tasks in parallel by respecting the dependencies.

Task parallelism
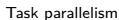
- Examples of independent tasks?
  - Function calls: $a = f(x); b = g(y);$
  - Independent blocks of code in a function/program
  - Execution of a single program with different parameters (e.g., hyperparameter search in simulations, learning, ...)
- Each task executes on a separate compute unit.
- Dependencies must be respected if exist.
  - Load balancing might be a challenge.
  - Effective scheduling of tasks is an entire research domain in itself.

université
PARIS-SACLAY

Outline

CPU

"Central Processing Unit", compute unit consisting of
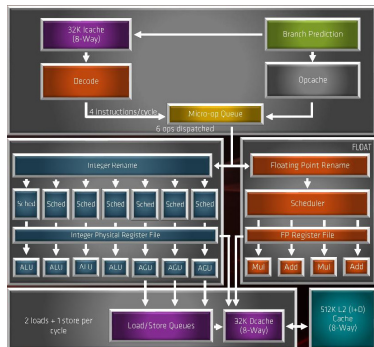
AMD Zen 2 architecture

- Multiple execution units (cores)
- Multiple memory types in a hieararchy (registers, L1, L2, L3 cache, RAM)
- Multiple execution ports in each core (ALUs, vector units, load/store units, . . .)
- SIMD/vector units (AVX2, AVX512, Arm Neon, ...)
- Execution of multiple threads (2-4) simultenously.
- Capable of exploiting the instruction-level parallelism (ILP) through micro-op buffer, instruction reordering, register renaming, . . .
- A big portion of the circuit is dedicated to ILP cache.

GPU

"Graphical Processing Unit", specific vector compute unit consisting of
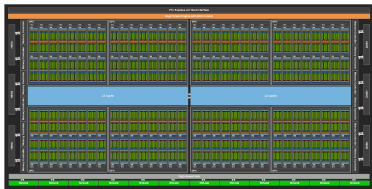
Nvidia Ampere Architecture

- Multiple execution units (streaming multiprocessors (SM))
- Multiple memory levels (registers, shared memory, L1, L2, GPU RAM, CPU RAM)
- Multiple (2-8) and large (16-32 floats) vector units in each SM
- Simultenous execution of thousands of threads with fast context-switching.
- Very big register file (65K)
- Most of the circuit is dedicated to vector computations and thread cohabitation
- Paralllelization takes programming effort

université
PARIS-SACLAY

Supercomputer / Cluster

A group of machines having CPU+GPU connected through a high-speed network

Jolio Curie supercomputer, 300K CPU cores, 1024 GPUs

- Connection through a network with a specific topology (ring, mesh, torus, clique, . . .)
- Efficient communication libraries adapted to the topology
- Capable of treating very large-scale problems
- Today, we exceeded the exaflop scale (10^{18} floating point operations per second)

université
PARIS-SACLAY

Shared memory architecture

- All processors can access to the memory in a global address space
- Multiple processors can operate independently, but share same memory resources.
- Memory modifications made by a processors are visible by all others.

université
PARIS-SACLAY

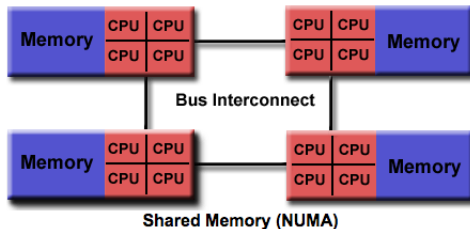# Shared memory architecture: Uniform Memory Access (UMA)

- Symmetric multiprocessor architecture (SMP).
- Identical processors.
- Same memory access time by all processors.
- Each processor will still have its cache, others will notified of a change in the cache. This is called **cache coherency**
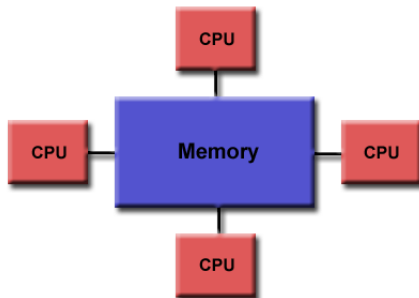
Shared Memory (UMA)

Shared memory: Non-Uniform Memory Access (NUMA)

- Most of the time, formed by connecting multiple UMA domains.
- A processor can directly access the memory situated in another UMA region, passing through a bus interconnect.
- Memory accesses have varying costs in latency and bandwidth.
- Cache across multiple UMA regions might or might not be coherent (mostly is).

Shared Memory (NUMA)

Shared memory architecture

1. Avantages :
   - Global shared address space enables simpler parallel programming and memory management.
   - Data sharing among threads in the same UMA region is simple and fast for.
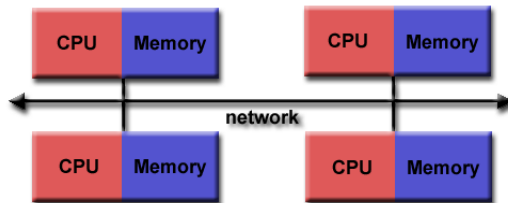
2. Disadvantages:
   - Architecture scalability is difficult when adding more CPUs and NUMA regions. Shared bus and cache coherency becomes difficult to handle when the size of the NUMA region becomes too large.
   - Accessing other NUMA regions can degregade performance, it is the programmer's responsability to optimize the memory footprint of each thread (i.e., stay mostly in UMA zone).

université
PARIS-SACLAY

Distributed memory architecture

- Distributed memory systems require a network to connect all processors.
- Each processor has its own memory address space (UMA or NUMA).
- Direct access to other processors' memory is prohibited; explicit communication, coordination, synchronization is needed when data exchange is necessary.

Distributed memory architecture

- Network can be as simple as Ethernet, or very complicated and high bandwidth (Infiniband, NVLink, . . .).

Introduction
0000

Parallel computing, why?
0000

Basics of parallel computing
00000000000

Types of parallelism
00000

Parallel architecture
000000000000000

## Distributed memory architecture

1. Avantages:
   - Memory and compute power is scalable with the number of processors.
   - Each processor accesses its own memory region rapidly without interference from other processors
   - No cache coherency among distributed machines, easier to handle architecture-wise.

2. Disadvantages:
   - Programmer is responsable for effectively handling the machine (communication, synchronization, . . .)
   - The algorithm, data structures, and data distribution have to be rethought for this type of architecture, which takes effort.

université
PARIS-SACLAY

Hybrid architecture

- Most modern supercomputers are hybrid architectures, having NUMA shared memory and distributed memory architectures simultaneously.
- Each component/machine is a NUMA with multiple CPUs/GPUs, and a high-speed network topology connects all machines.
- Processing units in the same machine share the same memory address space, need no explicit communication.
- Processing units across different machines need explicit communication through the network for data exchange.

universite
PARIS-SACLAY
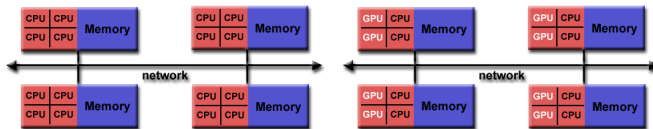
Contact

Oguz Kaya
Université Paris-Saclay and LISN, Paris, France
oguz.kaya@universite-paris-saclay.com
www.oguzkaya.com