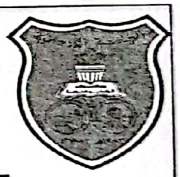
Parallel Processing
0907536

Instructor: Dr. Fahed Jubair
Computer Engineering Department
University of Jordan

Parallel Processing



- Parallel processing is using *multiple processors in parallel* to solve a computation problem **more quickly** than a single processor
- Parallel computing requires parallel machines + parallel programs
  - Parallel machines (aka multiprocessors) have hardware organizations such that multiple processors can perform multiple jobs in parallel
  - Parallel programs are programs that explicitly specify how computation and data are divided among the multiple processors of a parallel machine

© All Right Reserved

CamScanner

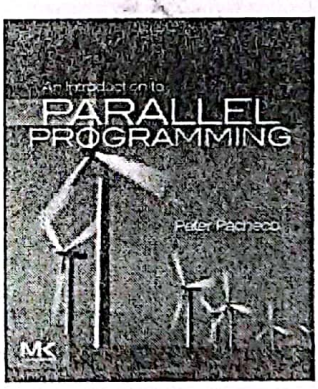
Class Objective

1. Study the most common parallel machine architectures
2. Shared-memory programming with OpenMP
3. Distributed-Memory programming with MPI
4. GPU programming with CUDA
5. Study some commonly-used parallel algorithms
   - Examples: parallel sorting and parallel matrix multiplication

© All Right Reserved

Textbook

Peter Pacheco, *Introduction to Parallel Programming*, 1st edition, 2011



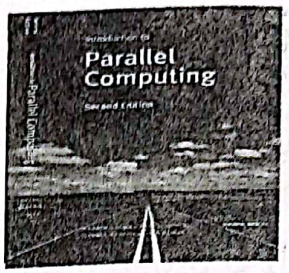
http://www-users.cs.umn.edu/~karypis/parbook/

© All Right Reserved

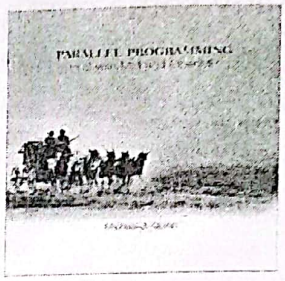
CamScanner

Additional References

Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar., Introduction to Parallel Computing, 2nd edition, 2010



Michael J. Quinn, Parallel programming in C with MPI and OpenMP, 2003



© All Right Reserved

Prerequisites

All students are assumed to be familiar with

- Basic data structures
- Execution analysis of algorithms
- Basic design principles of uniprocessor architecture
- Writing C and C++ programs

© All Right Reserved

CamScanner

Instructor Information

- Fahed Jubair
- B.Sc., University of Jordan
- Ph.D., Purdue University

© All Right Reserved

Grading Policy

• Programming Assignments	20%
• Midterm Exam	30%
• Final Exam	50%

© All Right Reserved

CamScanner

Class Policy

- Attendance is important
- Do not come late
- Cheating will **NOT** be tolerated
- No makeup exams
- Course website: Microsoft teams
- Contact: fjubair@ju.edu.jo

© All Right Reserved

Questions?

© All Right Reserved

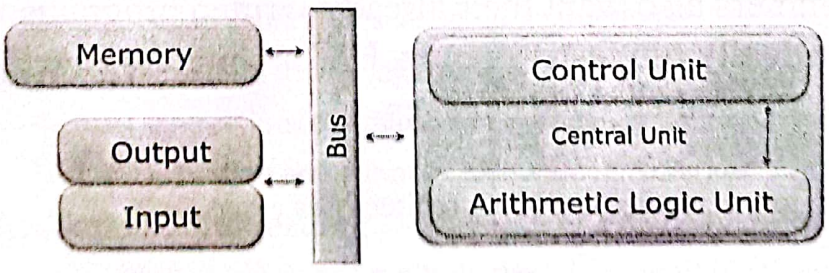
CamScanner

From Sequential to Parallel Computing

Instructor: Dr. Fahed Jubair
Computer Engineering Department
University of Jordan

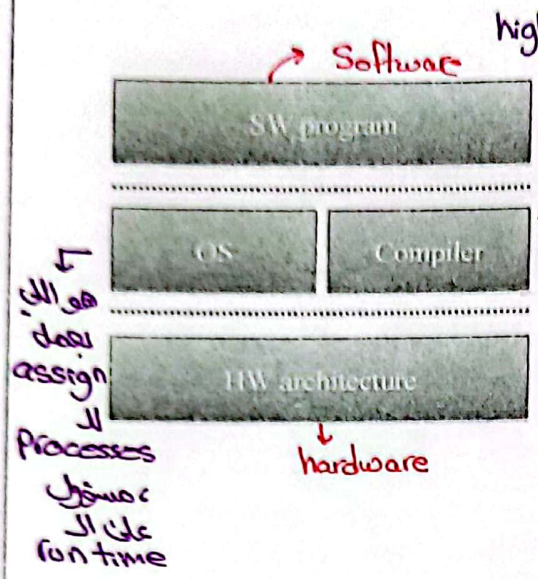
Serial Computers

- Defined by the **von Nuemann** architecture
- Consists of a central processing unit (CPU), main memory and an interconnection between the CPU and memory
- Executes (more or less) **a single job at a time**



© All Right Reserved

HW-SW Abstraction in Serial Computers



Special SW for maintaining the abstraction between HW and SW

© All Right Reserved.

* Programmers Want Simplicity

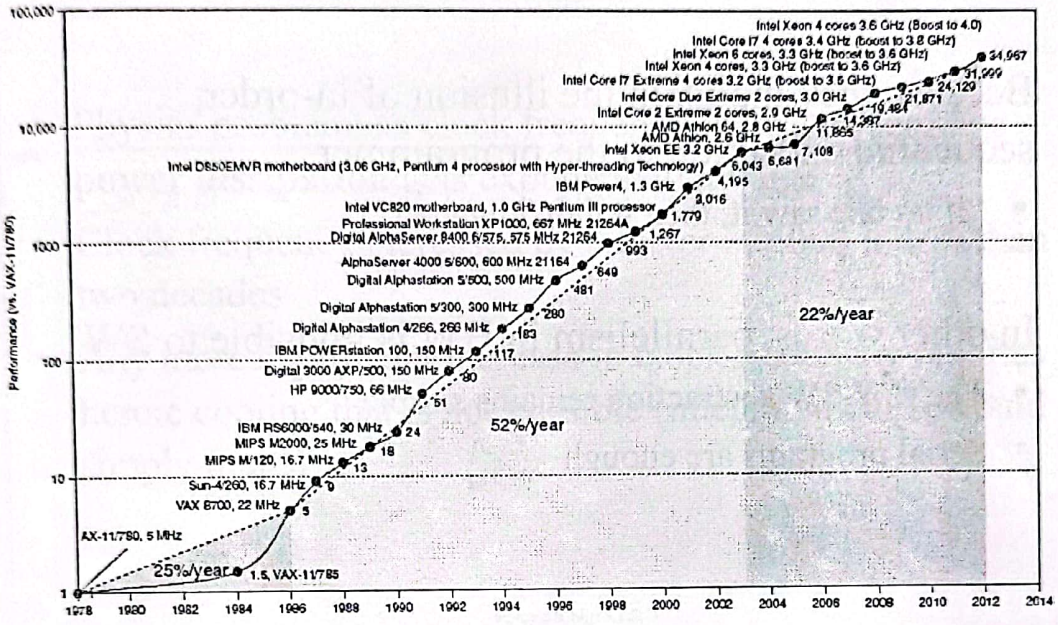
- Programmers want perfect HW-SW abstraction by writing programs that are
  - Unaware of any HW details
  - Serial, i.e., assume in-order, sequential execution
- Programmers also want their already-written programs to **automatically** run faster on **new** HW architectures without making any source-code changes
  - But how often do new HW architectures come out?

© All Right Reserved

* Moore's Law عمل توقع إنه عدد الـ transistor حيتضاعف كل 18 شهر تقريبا بنفس الـ chip area فوبك بقدر أعمل تقنيات أكبر وأفضل.

* الـ HW بتطور بسرعة الـ SW ماهم بيتطور قده فحاولوا يوازنوا بين الـ SW والـ HW.

HW Technology Trend in Serial Computers



© All Right Reserved.

*hyperthreading 8 2 threads parallel at the same time بنفيذ end user لـ متفيذين.

Serial Processors Are Secretly Not Serial

Due to Moore's law, HW architects were able to use more gate logic for exploiting **instruction level parallelism (ILP)** in CPUs

- Superscalar execution
  - Multiple functional units that execute instructions in parallel
- Instruction Pipelining
  - Overlapped execution of sequential instructions
- Speculative, out-of-order execution
  - Reorder instructions that have no dependency
  - Branch prediction and memory load value prediction
- Hardware multithreading
  - Interleaved execution of multiple threads on multiple functional units

© All Right Reserved

CamScanner

Why Superscalars Are Still Serial Processors?

- Because they maintain the illusion of in-order, sequential execution to the programmer
  - "if no one saw it, then it didn't happen"
- In other words, parallelism in HW is invisible to SW
  - The HW-SW abstraction remains simple
  - Serial programs are enough

© All Right Reserved.

The Free Lunch is Over

As of 2003, the trend in doubling the clock speed per year has stopped due to the following three problems:

1. The power wall (the main problem)
2. The memory wall
3. The ILP wall

© All Right Reserved.

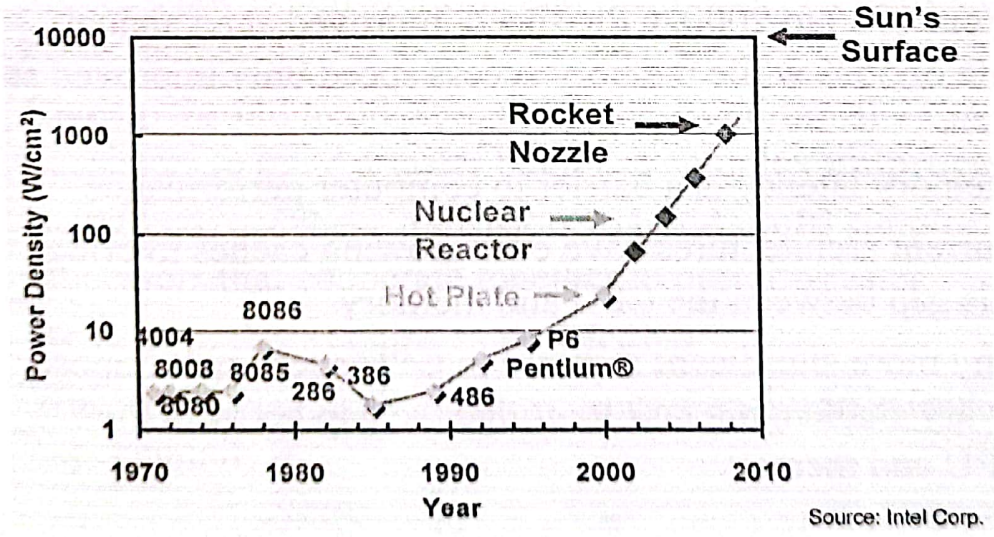
CamScanner

The Power Wall

- Physics problem: as clock frequency increases, **leakage power dissipation** gets exponentially worse
- Clock frequency increased by a factor of 4000 in less than two decades
- Any more significant increase in clock frequency requires heroic cooling that is not possible (integrated chips would simply melt!)

© All Right Reserved.

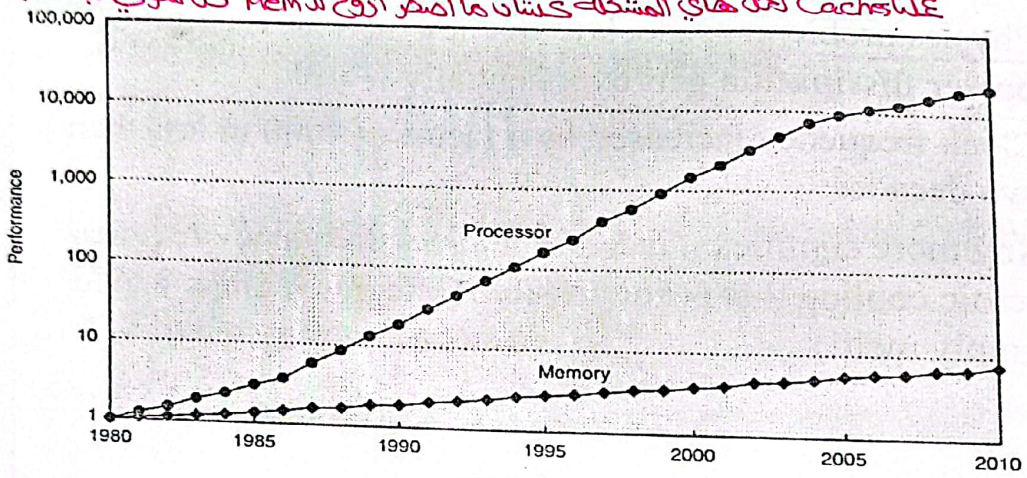
The Kind of Power Dissipation needed To Keep Doubling The Clock Speed



© All Right Reserved.

The Memory Wall

- Refers to the growing disparity of speed between CPU and memory outside the CPU chip



© All Right Reserved.

The Processor-DRAM Performance Gap

- From 1986 to 2000, CPU speed improved at an annual rate of 55% while memory access speed only improved at 10%
- Memory latency is a barrier to computer performance
- Current architectures have ever growing caches to bridge this gap between the CPU and memory
- Any more significant increase in CPU speed will impose a vary challenging task to bridge the gap between the CPU and memory

© All Right Reserved.

The ILP Wall

- To keep doubling the clock frequency, more aggressive ILP mechanisms are needed
- However, more aggressive ILP means deeper pipelines and more complex functional units, which makes the power leakage problem worse!

بلشوا يطلعوا أفكار كثيرة لحد ما بطلت في أفكار وتطورات .

© All Right Reserved.

New Trend: Multicores

- Moore's law is still working and the number of transistors keeps increasing
- Give up the old trend: stop using these additional transistors to build a faster but more complex **uniprocessor**
- New trend: use these transistors to build **multi** simpler processors that cooperate together to perform faster
- We refer to theses chips as **multicores**
- * With multicores, leakage power increases linearly (not exponentially)

ما بضاعف ال Clock speed بضل زي ما أنا بس بحط واحد جمبي بيساعدني

* ال OS بعمل Schedule بين ال Processes وزيادة ال Cores بحسن ال throughput فبستفيد ال OS

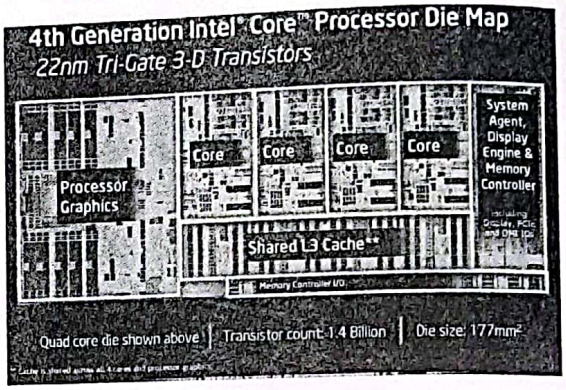
كدد الوظائف اللي بعملها خلال وحدة الزمن .

© All Right Reserved.

CamScanner

Multicore Processor Example: Intel Haswell

- June 2013
- 22 nm technology
- 1.4 billion transistors
- 4 cores
- 3.1 GHz
- 6 MB L3 Cache
- Hyperthreading



© All Right Reserved

* Process : active entity
* Thread : is a child of process.

Sequential Programming No longer works

- All modern processors are multicores
- With multicores, we can simultaneously execute more than one job a at a time (thus, improving throughput)
- However, multicores cannot automatically make the execution time (latency) of a serial program faster
- To make programs run faster, we need to **rewrite** them into an *explicitly parallel programs*
  - **The era of sequential programming is over!**

© All Right Reserved.

The Rise of Parallel Programming

- Explicit parallel programs **explicitly** specify how concurrent computation is performed by the multiple processors of a parallel machine
- The merit: parallel programs achieve faster computation
- The problem: parallel programs are *significantly* more difficult to write than sequential programs

© All Right Reserved.

Example: Parallel Sum

- Let us develop a parallel algorithm (written in pseudo-code) to compute the sum of 1000 integers stored in an array
  - Assume the algorithm will run on a multicore machine with 8 cores: core 0, core 1, …, core 7
  - Let us use the term "thread" to refer to each sequence of instructions executed on a core
  - Therefore, we have 8 threads: thread 0, thread 1, …, thread 7 such that thread *i* executes on core *i*
  - Let the variable *thread_id* refer to the thread number

Note that the sequential version is trivial

```
sum = 0
for ( i = 0; i < 1000; i++)
    sum = sum + A[ i ]
```

© All Right Reserved

Example (cont.)
Parallel Sum Algorithm

- We have 1000 integers and 8 threads ⇒ make each thread computes the sum of 125 integers

local Sum

- Our parallel sum algorithm works as follows:
  i. First, each thread computes its partial sum independently
  ii. Then, combine partial sums from each thread into a global sum

بعمل Partition . كلهم بعملوا نفس الـ Code اللي بختلف الـ data لكل thread .

© All Right Reserved

Example (cont.)
First Step: Partition The Work

- Let variables *lb* and *ub* determine the lower and upper array bounds for each thread
- Let variable *my_sum* determines the partial sum computed by each thread
- Note that each thread has its own private copy of *lb*, *ub* and *my_sum*

```
lb = 125 * thread_id
ub = lb + 124
my_sum = 0
for ( i = lb ; i < ub ; i++)
    my_sum = my_sum + A[ i ]
```

Wait! how does this code get executed by the 8-core parallel machine?

© All Right Reserved

Example (cont.)
Execution Model

- The same program is duplicated among threads
- This paradigm is referred to as the *single-program-multiple-data* (*SPMD*)

```
lb = 125 * thread_id
ub = lb + 124
my_sum = 0
for ( i = lb ; i ≤ ub ; i++)
    my_sum = my_sum + A[ i ]
```

At runtime →

lower		upper
0 * 125 ←	Thread 0	$lb = 0$, $ub = 124$ → 0 + 124
1 * 125 ←	Thread 1	$lb = 125$, $ub = 249$ → 125 + 124
2 * 125 ←	Thread 2	$lb = 250$, $ub = 374$ → 250 + 124
	Thread 3	$lb = 375$, $ub = 499$
	Thread 4	$lb = 500$, $ub = 624$
	Thread 5	$lb = 625$, $ub = 749$
	Thread 6	$lb = 750$, $ub = 874$
	Thread 7	$lb = 875$, $ub = 999$

* كل واحد إله Copy من ال Code .
* ال data بتكون Private بين ال thread .
* ممكن يكون في Shared data
وهي بهاي المثال Array A
والباقي كله Private .

© All Right Reserved.

Example (cont.)
Second Step: Combine Partial Sums

- Let variable *global_sum* be a global variable that each threads updates with its partial sum
- We will let thread 0 be responsible for initializing *global_sum*
- Note that there is a single copy of *global_sum* that is shared between all threads

* هاد ال Code منطقي بس في مشكلة إنه ال global sum لو كان عندي

```
if ( thread_id = 0 )
    global_sum = 0
lb = 125 * thread_id
ub = lb + 124
my_sum = 0
for ( i = lb ; i <= ub ; i++)
    my_sum = my_sum + A[ i ]
global_sum = global_sum + my_sum
```

رح يصير + incorrect out إله حيكون Slow initialization

Race condition عنا ممكن ، إنه واحد ياخذ بيانات نفس اللي أخذها اللي قبله (بيانات خاطئة حتكون)

- Note that the order at which threads update *global_sum* is **random**
- Here, the order is not important because **addition is associative**

© All Right Reserved.

CamScanner

Example (cont.)
Race Condition Scenario

```
if ( thread_id = 0 )
    global_sum = 0
lb = 125 * thread_id
ub = lb + 124
my_sum = 0
for ( i = lb ; i <= ub ; i++)
    my_sum = my_sum + A[ i ]
global_sum = global_sum + my_sum
```

بنفس الوقت

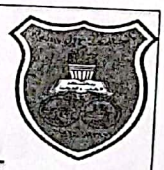
Assume threads 1 & 3 arrived here simultaneously

Also assume the current status of global sum and partial sums is:

my_sum (thread1) = 20
my_sum (thread3) = 33
global_sum = 60

© All Right Reserved

Example (cont.)
Race Condition Scenario



1	Thread 1 loads *global_sum* from memory intro register (i.e., R) R = 60	Thread 3 loads *global_sum* from memory intro register (i.e., R) R = 60
2	Thread 1 adds *my_sum* to R R = 60 + 20 = 80	Thread 3 adds *my_sum* to R R = 60 + 33 = 93
3	Thread 1 stores R into *global_sum*	Thread 3 stores R into *global_sum*

- Depending on which threads stores its value last, *global_sum* could be either 80 or 93
- This is called a race condition, which occurs when at least two threads simultaneously access a shared variable and at least on of these threads is performing a write operation

© All Right Reserved.

CamScanner

Example (cont.) Synchronization

- To eliminate race conditions, programmers must **synchronize** between threads by forcing them to **sequentially** update the global sum by their partial sums
- Therefore, we will use a *critical section* as our form of synchronization
- A critical section is executed by a thread **atomically**, i.e., uninterrupted by any other thread

© All Right Reserved.

Example (cont.) ← Critical Section

الحل إنه أحطوا جوا Critical Section

```
if ( thread_id = 0 )
    global_sum = 0
lb = 125 * thread_id
ub = lb + 124
my_sum = 0
for ( i = lb ; i <= ub ; i++)
      my_sum = my_sum + A[ i ]

critical section {
      global_sum = global_sum + my_sum
}
```

© All Right Reserved.

CamScanner

Example (cont.)
Another Race Condition Scenario

```
if ( thread_id = 0 )
    global_sum = 0
lb = 125 * thread_id
ub = lb + 124
my_sum = 0
for ( i = lb ; i <= ub ; i++)
    my_sum = my_sum + A[ i ]
critical section {
    global_sum = global_sum + my_sum
}
```

Thread 0 has the responsibility of initializing *global_sum* to 0

A race condition could occur if thread 0 is too slow that some other threads reached the critical section before thread 0 initializes *global_sum* to 0

© All Right Reserved.

Example (cont.)
Barriers

- One way to eliminate the data race shown in the previous slide is to force all threads to wait for thread 0 to initialize *global_sum* to 0
- To do so, we will use a *barrier*, which is a form of synchronization where all threads must wait untill all other threads reach the same point in execution
- The hardware architecture must support the thread-waiting mechanisms of barriers

© All Right Reserved.

CamScanner

Barrier -1
Signal wait -2
Critical Section -3
ممكن نستخدم 3 حلول

Example (cont.)
Final Parallel Code

```
if ( thread_id = 0 )
    global_sum = 0
threads_barrier
lb = 125 * thread_id
ub = lb + 124
my_sum = 0
for ( i = lb ; i <= ub ; i++)
    my_sum = my_sum + A[ i ]
critical section {
    global_sum = global_sum + my_sum
}
```

كل الـ thread بيشتوا ليوصلوا كلهم ويطلعوا مع بعض ويبلش الـ Execution.

Very different from the trivial sequential code!

لو حطينا هون threads_Barrier ما بتفرق بس أسوأ لإنه الـ waiting رح يصير أكثر وأسوأ.

Mutal exclusion

© All Right Reserved.

برجعه لسا الـ code فيه مشاكل لسا ما بينت

Example (cont.)
Performance Analysis

- Ideally, our parallel algorithm aims to achieve 8-times faster execution than sequential version (i.e., achieves *linear speedup*)

معناه إذا انقسم الشغل لـ 8 بتحسن وبتضاعف 8 مرات

ما بتقدر يكون Parallel بنسبة 100%

- But parallelism has overheads:

لـ مشاكل بنقدر نحصل عليها.

المشاكل لـ

  1. Work partitioning overhead: extra computation needed to partition computation among threads

تقسيم الشغل

  2. Synchronization overhead:
     - Barriers force fast threads to wait for slow threads
     - Critical sections are executed serially
  2. The creation and termination of threads also have an overhead

- To write efficient parallel programs, programmers need to think about how to minimize parallelism overheads in their codes

© All Right Reserved.

CamScanner

Example (cont.) – Scalability

يعني كلما زدنا عدد الـ cores كلما تحسن الـ Speedup

بس هنا مو شرط يصير لإنه بالغالب كلما زدنا الـ cores كلما زاد الـ Parallism – overhead.

- Ideally, we want our algorithm to perform faster when executed on a parallel machine with higher number of cores
  - i.e., our algorithm needs to be *scalable*
- The problem is parallelism overhead, which increases as the number of cores (or threads) increases
- This imposes yet another challenge to programmers: how to write *scalable* parallel programs!

يفرض..

© All Right Reserved.

Challenges in Writing Parallel Programs

لإنه مو دايمًا بنفكر Parallel بالعادة بنفكر Serial

1. Programmers need to think parallel
   - Rewrite the sequential algorithm into a parallel algorithm
   - In some cases, the parallel algorithm is a totally new algorithm

2. Programmers need to understand the architecture of the underlying parallel machine (HW-SW abstraction is gone!)
   - HW details affect the programmer's decision on how to find efficient partitioning of computation between threads
   - HW details affect the programmer's decision on how to choose efficient synchronization methods

© All Right Reserved.

CamScanner

Challenges in Writing Parallel Programs

لإنه بال Parallel لو عملت test لل code وكان في Race condition
ممكن ما يبين إلا لأعمل test لكثير شغلات ولأكثر من مرة.

3. Testing and debugging parallel programs is hard
   - For example, some race conditions are very subtle that may not occur for the first 999999 runs, and occurs on the 1000000th run

4. Parallelism has an overhead
   - If not properly handled, this overhead can kill the scalability of a parallel program

© All Right Reserved.

How To Write A Parallel Program?

- In general, there are three approaches for writing an explicit parallel program:
  1. Write from scratch
  2. Parallelize an existing sequential program
  3. Parallelize an existing sequential program by a compiler

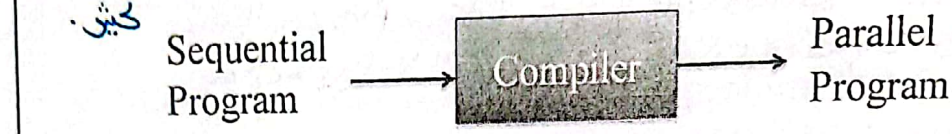
Done manually (by hand)

Parallel code لـ Serial code يحول من

© All Right Reserved.

CamScanner

Wait! Why Not Always Use A Parallelizing Compiler?



- A parallelizing compiler converts a sequential program into a parallel program
- The merit: it is automatic!
- The problem: so far, parallelizing compilers have shown limited success due to the significant complexity of creating parallel programs
- Examples on existing parallelizing compilers: SUIF, PLUTO, Rose, Cetus, Polaris, and Intel C++ compiler

© All Right Reserved.

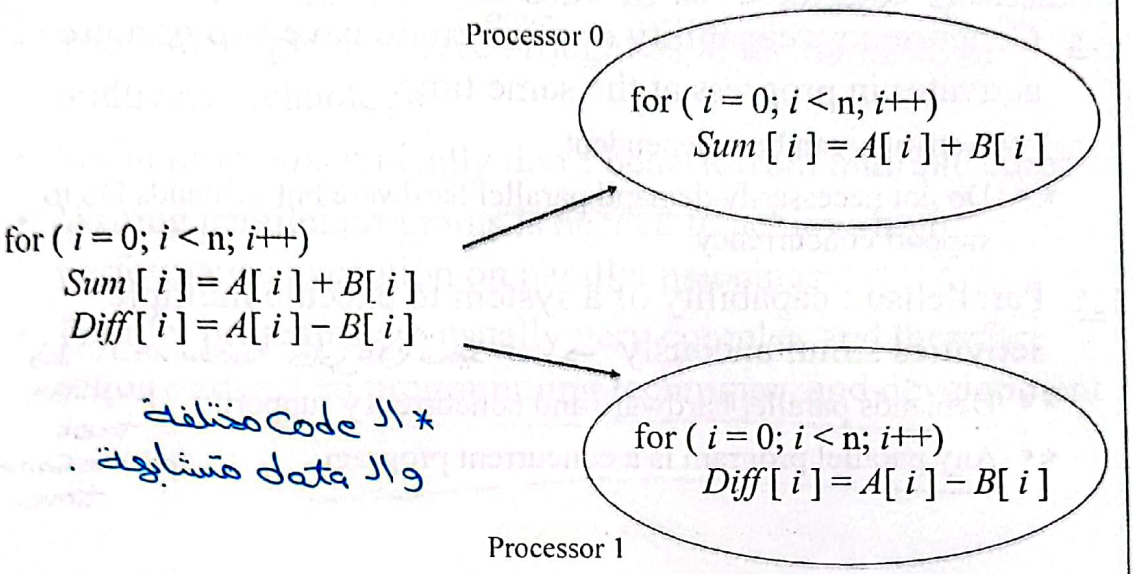
Parallel Program Types

Parallel applications can be classified depending on how problem partitioning is performed, as follows:

1. Task-parallel applications
   - Partition various tasks carried out solving the problem among the processors
2. Data-parallel applications
   - Partition the data used in solving the problem among the processors
   - Each processor carries out similar operations on it's part of the data

© All Right Reserved.

Example
Work Partitioning – Task Parallelism

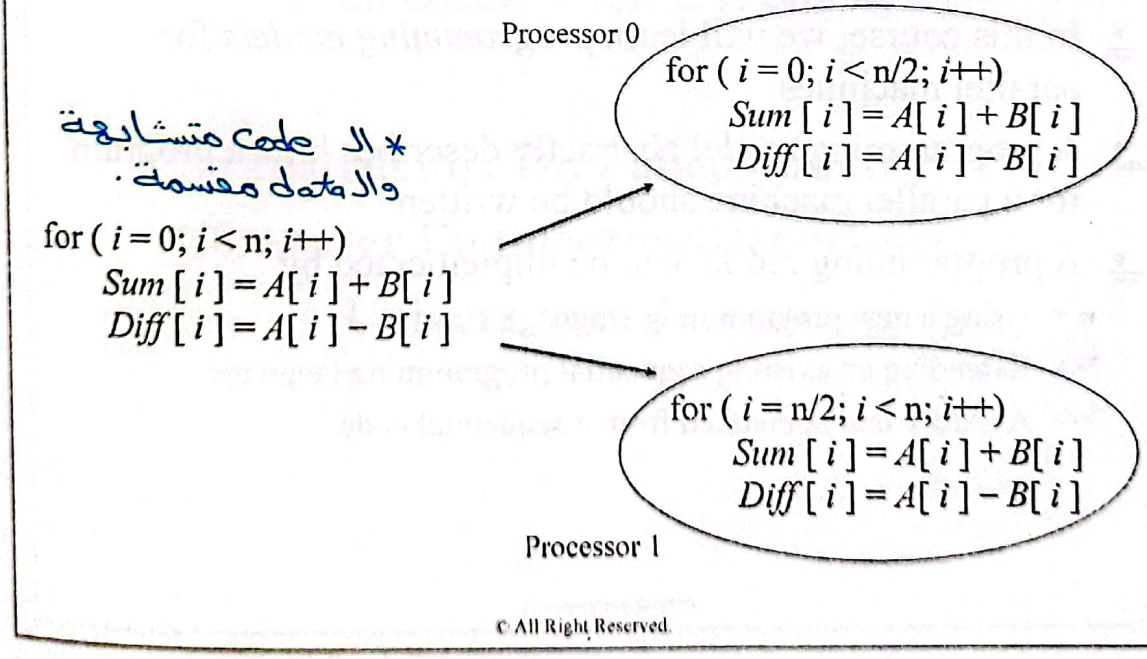


* الـ Code مختلفة
والـ data متشابهة.

© All Right Reserved.

* أي برنامج data parallism فهو task parallism لكن العكس غير صحيح.

Example
Work Partitioning – Data Parallelism



* الـ Code متشابهة
والـ data مقسمة.

© All Right Reserved.

CamScanner

Terminology
Concurrency vs Parallelism

or more

Switching الـ من سرعة 2 task at the same time بتعيألك إنه بتنفذ

- Concurrency: capability of a system to have two or more activates in progress at the same time
  - Activates can be independent
  - Do not necessarily demand parallel hardware but demands OS to support concurrency
- Parallelism: capability of a system to execute multiple activities simultaneously → فعلاً الـ hardware قادر على تنفيذ multiple task at the same time.
  - Demands parallel hardware and concurrency support
  - Any parallel program is a concurrent program

© All Right Reserved.

Terminology
Programming Model

ما بتعلم لغة برمجة جديدة
بتعلم قواعد وlibrarys جديدة في اللغة تعرفني كيف أصمم.

- In this course, we will learn *programming models* for parallel machines
- A programming model abstractly describes how a program for a parallel machine should be written
- A programming model can be implemented by
  - Using a new programming language
  - Extending an existing sequential programming language
  - A library that is invoked from a sequential code

© All Right Reserved.

CamScanner

Summary

- The laws of physics have brought us to the doorstep of multicore technology
- Serial programs typically don't benefit from multiple cores
- Writing parallel programs is needed to achieve high-performance execution on parallel machines
- Parallel programs are usually very complex and therefore require advanced programming techniques and development

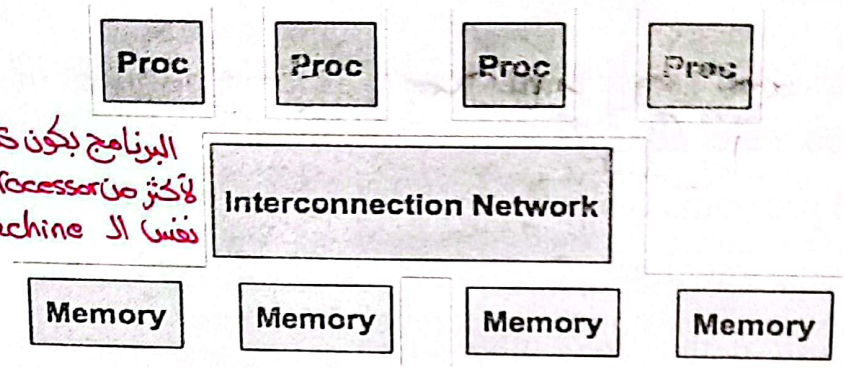
© All Right Reserved.

Parallel Hardware

Instructor: Dr. Fahed Jubair
Computer Engineering Department
University of Jordan

CamScanner

An Abstract Parallel Machine



البرنامج بكون هنده access
لأكثر من Processor على
نفس ال Machine .

In general, parallel machines differ in their

- Concurrency model, which affects how instruction stream and data stream are partitioned among processors
- Communication model, which affects how processors and memory units are connected by the interconnection network

© All Rights Reserved.

Concurrency Model

Flynn's Taxonomy

Processor architectures, in general, are classified based on instruction stream and data stream, as follows:

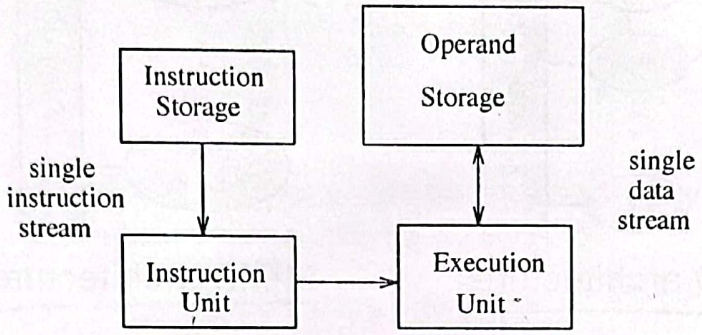
serial ① SISD Single instruction stream Single data stream	parallel ② like GPU's (SIMD) Single instruction stream Multiple data stream
mostly theoretical ③ MISD Multiple instruction stream Single data stream	parallel ④ (MIMD) Multiple instruction stream Multiple data stream

← ما في عارض
الواقع أجهزة
بتستعمله

© All Rights Reserved.

Serial Computing Platforms

- SISD processors
  - Von Neumann uniprocessor architecture



© All Rights Reserved.

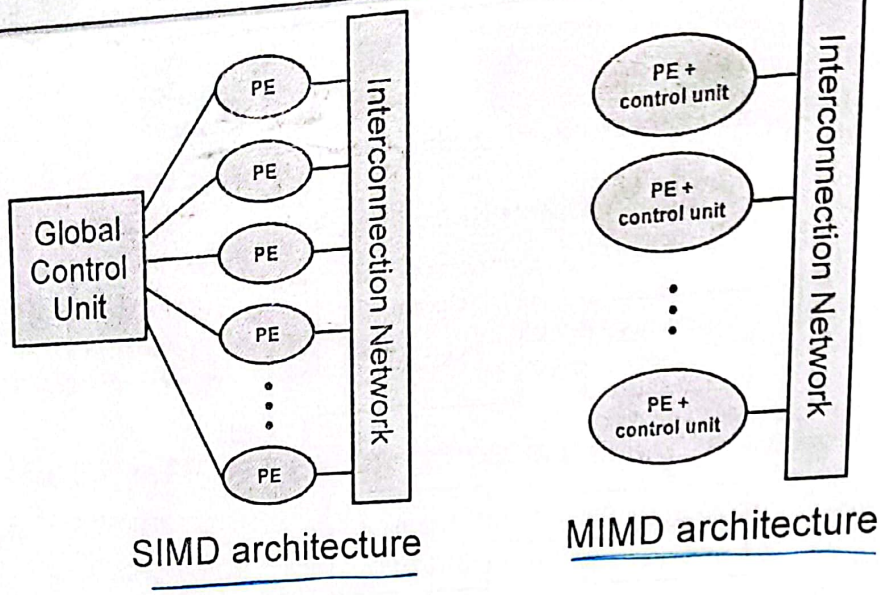
Parallel Computing Platforms

- SIMD processors → كل ال Processors عندهم نفس البرنامج Physically فيه Shared copy بينهم كلهم.
  - All processors execute the same instruction simultaneously, but while operating on different pieces of data
  - Work best with data-parallel applications
- MIMD processors → مو شرط ال Processors يكون عندهم نفس البرنامج ، كل واحد اله copy خاص فيه حتى لو تشابهت البرامج
  - All processors execute in parallel but on different instructions and different pieces of data
  - Can handle both data-parallel and task-parallel applications

© All Rights Reserved

CamScanner

SIMD and MIMD Processors



PE = Processing Element

© All Rights Reserved.

SIMD vs. MIMD

- SIMD platforms
  - Single control unit
  - Only one copy of program
  - Special purpose: not suited for all applications

مشاكل البرامج بتزبطعليه

- MIMD platforms
  - Multiple control units
  - Multiple programs
  - Suitable for broad range of applications

task parallism أو data parallism بتقدر تشغلهم

. more flexible و

© All Rights Reserved.

CamScanner

SIMD Example

Conventional SISD loop that sums two arrays

```
for ( i=1; i<=N; i++ )
    sum[ i ] = A[ i ] + B[ i ]
```

SIMD code has a single instruction that executes simultaneously on N ALUs

```
sum[ 1 : N ] = A[ 1 : N ] + B[ 1 : N ]
```

اللي بكونوا data parallel هم اللي بزبطوا ينفعلوا SIMDS . أما الـ task parallel ما بزبط

SIMD Processor Example: Vector Processor

أسرع كتير من الـ MIMD

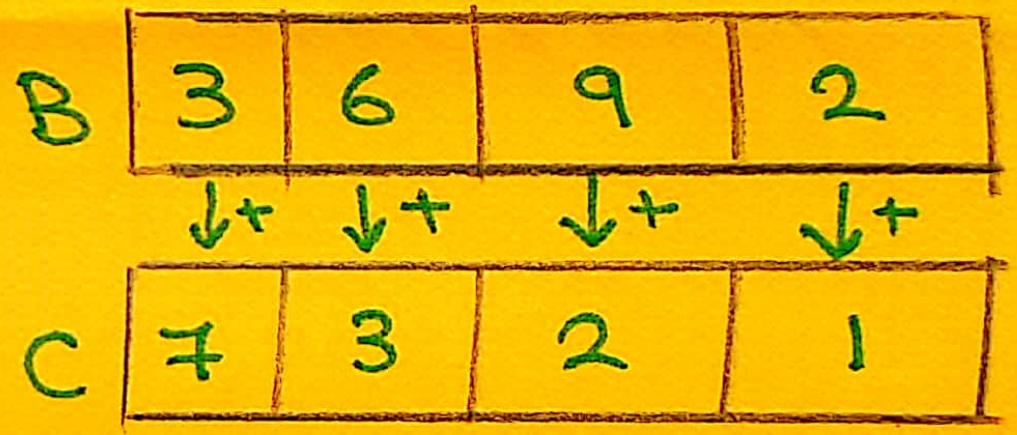
- A vector processor executes *vector instructions*, which are instructions that operate on one-dimensional arrays of data called *vectors*
  - In contrast, conventional processors execute instructions that operate on individual data elements (called *scalars*)
- Vector processors have
  - Vectorized and pipelined functional unit: the same operation is applied to each element in the vector (or pairs of elements)
  - Vector registers: capable of storing a vector of data elements
- The merit: fast
- The problem: limited to data-parallel applications

© All Rights Reserved

CamScanner

SIMD :

$A = B + C \Longrightarrow$



in Parallel

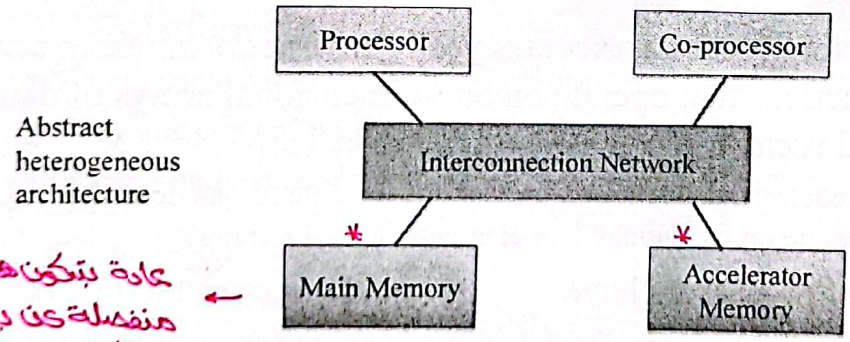
يجمعوا B و C و بطلعولك A مرة وحدة

Vector Processors in Modern Architectures

- Today's processors usually have SIMD units or vector processors as co-processors (aka *accelerators*)
- On-chip SIMD units:
  - For example, modern Intel processors have *on-chip graphical processing units (GPUs)* for fast image processing
    - Intel *instruction set architecture* has special instructions for for multimedia support (e.g., referred to as multimedia extensions or MMX)
- Off-chip co-processors:
  - Typically sold as PCIe cards
  - Architectures that have both conventional processors and co-processors are referred to as *heterogeneous architectures*

© All Rights Reserved.

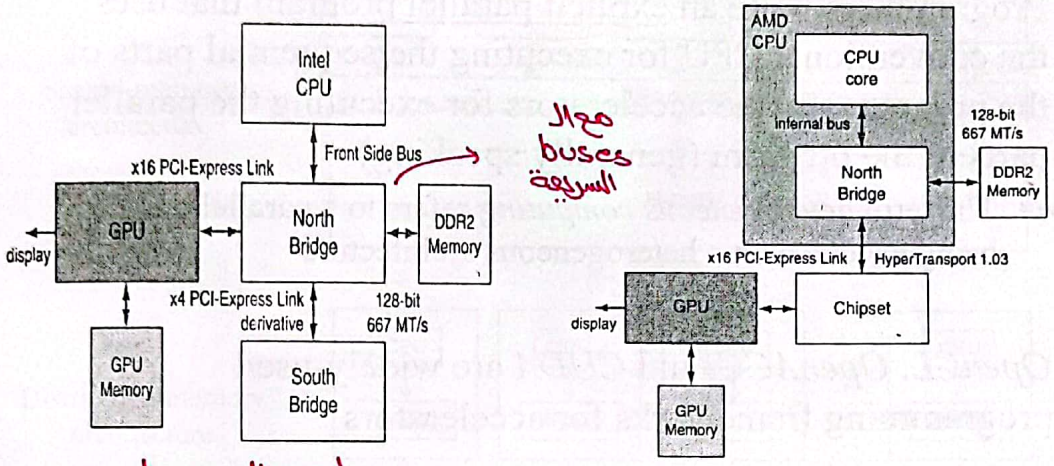
Heterogeneous Architecture



- Heterogeneous platform: Processor (on-chip) + Co-Processor (on-chip or off-chip)
- In contrast, a homogeneous platform has only on-chip processors

© All Rights Reserved

Heterogeneous Architecture Example: Off-Chip GPUs



مع ال buses السريعة

مع ال buses البطيئة

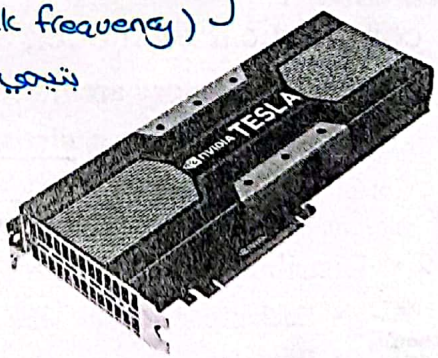
« بدون الدخول بالتفاصيل »

© All Rights Reserved.

Accelerator Example: "كرت شاشة" NVIDIA Tesla K40 (July 2013)

cores مبسطة ، بتختلف عن ال cores

- 2880 streaming cores → عدد ال cores كتير كبير
- 732 MHz → بطيئة السرعة (clock frequency) → بعوضه بال Parallism
- 12GB memory → بتيجي ال Mem جواته
- Memory B/W 288 GB/s
- 4.29 TFLOP for single-precision
- 1.43 TFLOPS for double-precision
- 235 Watts
- Kepler GK110 architecture



* حنحكي بالتفاصيل لقدام.

© All Rights Reserved.

Achieving High-Performance Computing On Accelerators

or co-Processor ↓ like GPU's

- Programmers write an explicit parallel program that uses the conventional CPU for executing the sequential parts of the program and the accelerators for executing the parallel parts of the program (generally speaking)
  - The term *heterogeneous computing* refers to a parallel program being executed on a heterogeneous architecture

- *OpenCL*, *OpenACC*, and *CUDA* are widely-used programming frameworks for accelerators

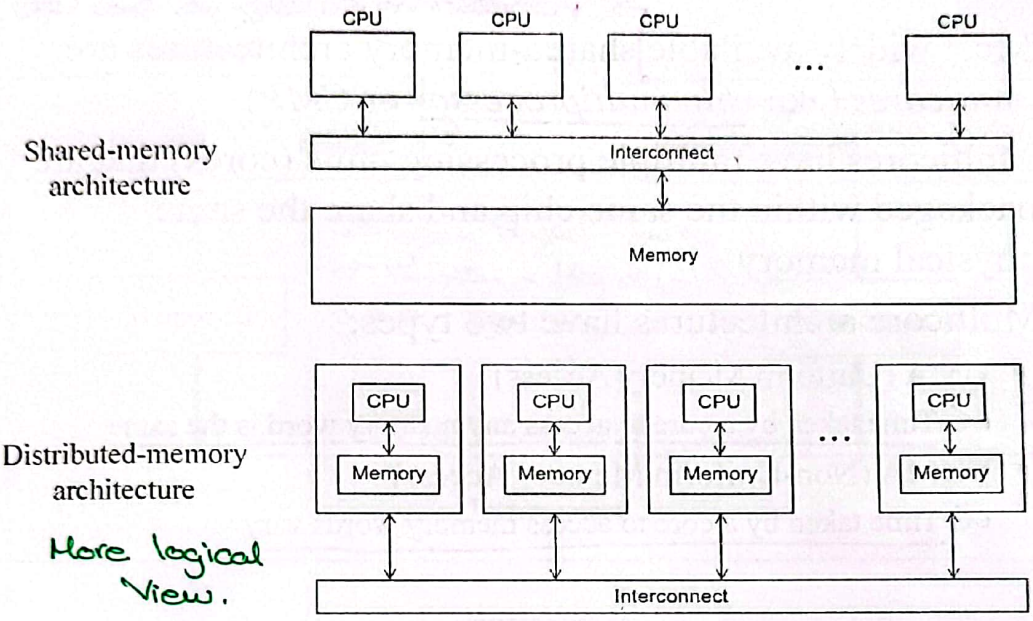
© All Rights Reserved.

MIMD Processors

- Execute different programs on different processors
- MIMD processors are classified based on their communication model, as follows:
  1. Shared-memory architectures
     - All processors can **directly** access a common memory
     - Processors **communicate implicitly** by reading or writing shared data structures in memory (for example, via load and store instructions)
  2. Distributed-memory architectures:
     - Each processor can only access **local** memory
     - Processors **communicate explicitly by** exchanging messages (for example, through a LAN network)

© All Rights Reserved.

Abstract View of MIMD Architectures



Shared-memory architecture

Distributed-memory architecture

More logical view.

© All Rights Reserved.

Shared-Memory vs Distributed-Memory

	Shared-memory	Distributed-memory
Execution units	Threads	Processes → لانه شغال كل واحد لحاله
Memory sharing	All threads share the same data space	Each process has its own data space
Operating system	Typically, all threads run within the same OS	Typically, each process runs within a separate OS
Communication method	Threads communicate implicitly by accessing shared variables in memory	Processes communicate explicitly by exchanging messages
Communication speed	Communication between processors in shared-memory is typically faster than distributed-memory	
Scalability	Distributed-memory platforms are typically more scalable (have higher number of processors) than shared-memory platforms → لانه لو حبيت عدد هائل من ال Processors ما حتقدر أحطهم كلهم بجهاز بس	

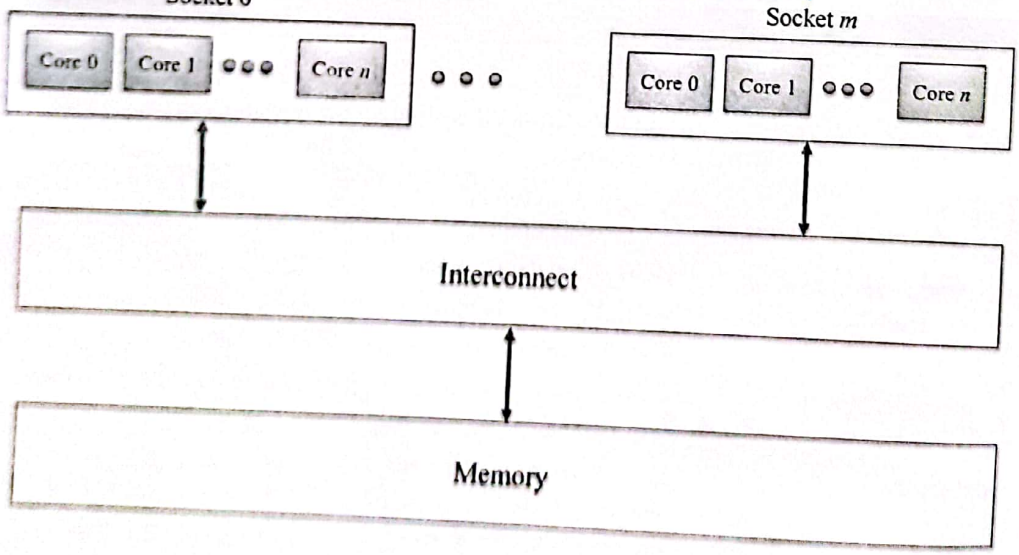
© All Rights Reserved

Shared-Memory Platforms

- Most widely available shared-memory architectures are *multicores* (aka *chip multiprocessor* or *CMP*)
- Multicores have multiple processing units (cores) that are packaged within the same chip and share the same physical memory
- Multicore architectures have two types:
  1. UMA (Uniform Memory Access)
     - Time taken by a core to access any memory word is the same
  2. NUMA (Non-Uniform Memory Access)
     - Time taken by a core to access memory words vary

© All Rights Reserved.

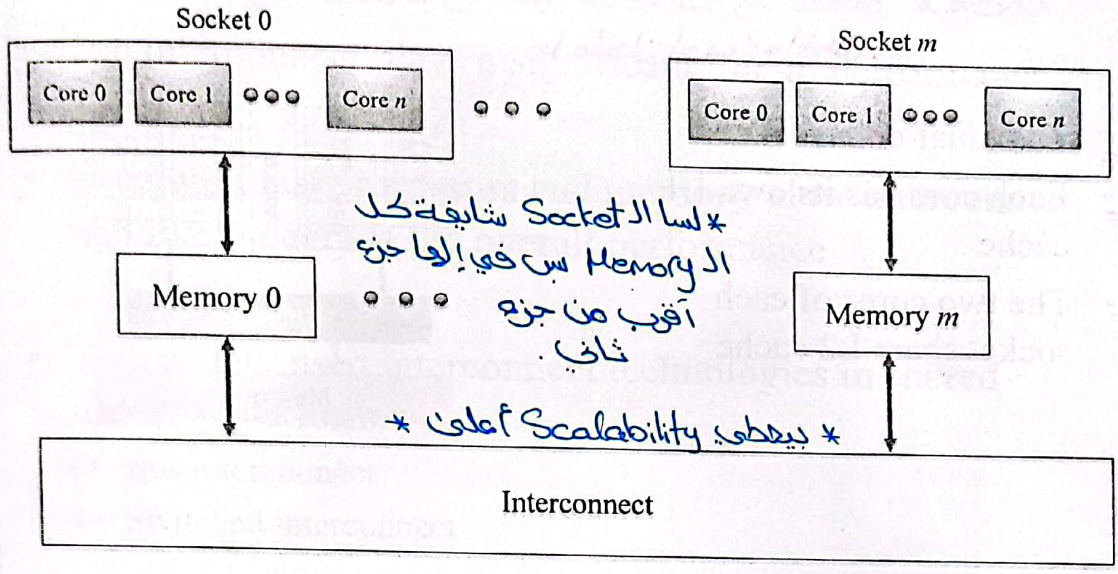
Abstract View of UMA



© All Rights Reserved

Abstract View of NUMA

موجود عشان الـ Scalability



* لسا الـ Socket شايفة كل الـ Memory بس في إلها جزء أقرب من جزء تاني.

* بيعطي أعلى Scalability

* بخلال الـ cluster أفضل منهم.

© All Rights Reserved.

UMA vs NUMA

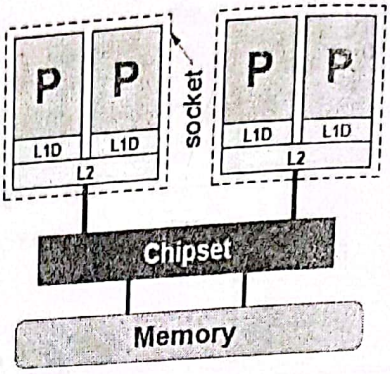
UMA	NUMA
• Data placement is unimportant ⇒ simpler HW and SW	• Data placement affects performance ⇒ more complex HW and SW
• Less scalable	• More scalable

© All Rights Reserved.

UMA Example

* كل ما زاد حجم الـ cache والـ cache levels
كلما كان أحسن وأفضل.

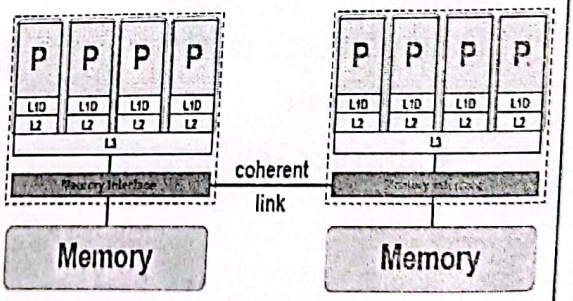
- Two dual-core sockets
- Each core has its own L1 cache
- The two cores of each socket share L2 cache



© All Rights Reserved.

NUMA Example

- Two quad-core sockets
- Each core has its own L1 and L2 caches
- The four cores of each socket share L3 cache



© All Rights Reserved.

Interconnection Networks In Shared-Memory Platforms

- The interconnect plays a significant role in performance
- Even with fast processors and memory, slow interconnect will likely degrade the overall performance
- Two widely used interconnect technologies in shared memory platforms:
  - Bus interconnect
  - Switched interconnect

© All Rights Reserved.

Bus Interconnect

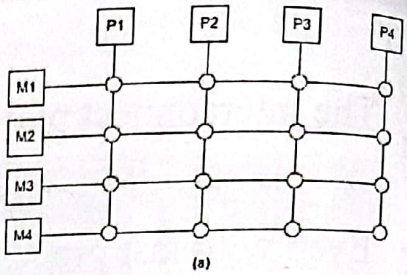
- A collection of parallel communication wires together with some hardware that controls access to the bus
- Communication wires are shared by the devices that are connected to it
- The merit: low cost
- The problem: limited scalability
  - As the number of devices connected to the bus increases, contention for use of the bus increases, and performance decreases

© All Rights Reserved

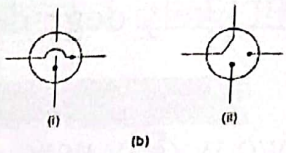
CamScanner

Switched Interconnect

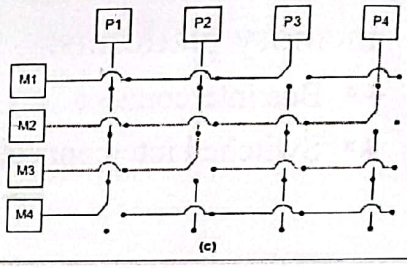
- Uses switches to control the routing of data among the connected devices



- Crossbar interconnect
  - Allows simultaneous communication among different devices
  - Faster than buses
  - But the cost of the switches and links is relatively high



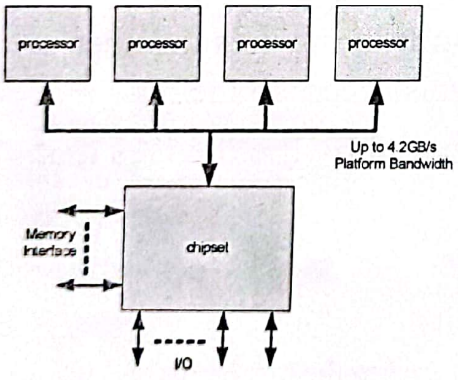
بصير افتح وأسكر الـ Switches حسب مين بده يحكي معه مين .



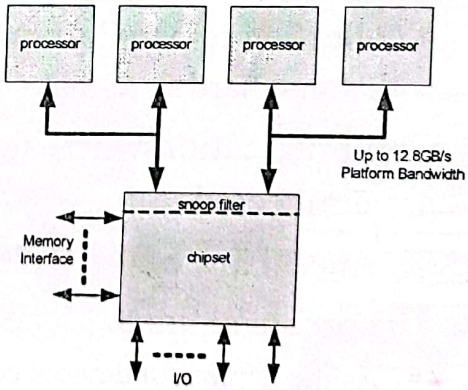
© All Rights Reserved.

Example: Intel Interconnect Evolution

* صار يتنافس 2 بدل 4 خففنا *



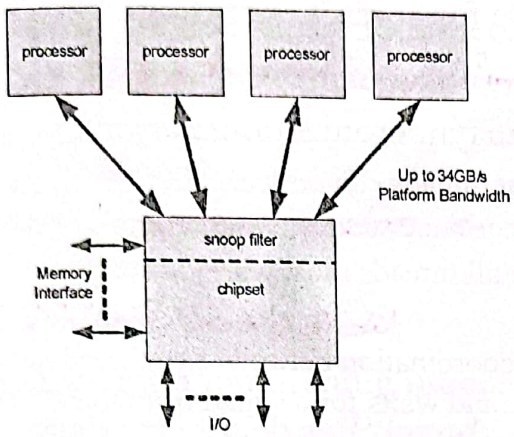
Traditional Shared Frontside Bus
(until 2004)

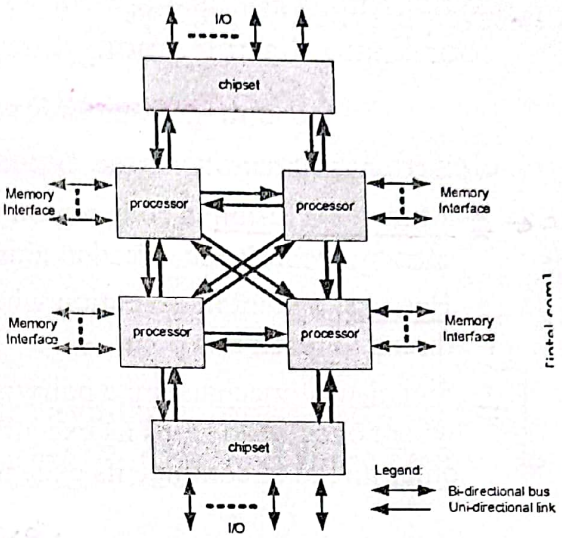


Dual Independent Buses
(until 2005)

© All Rights Reserved.

Example:
Intel Interconnect Evolution





Dedicated Interconnects
(until 2007)

هل
هيك

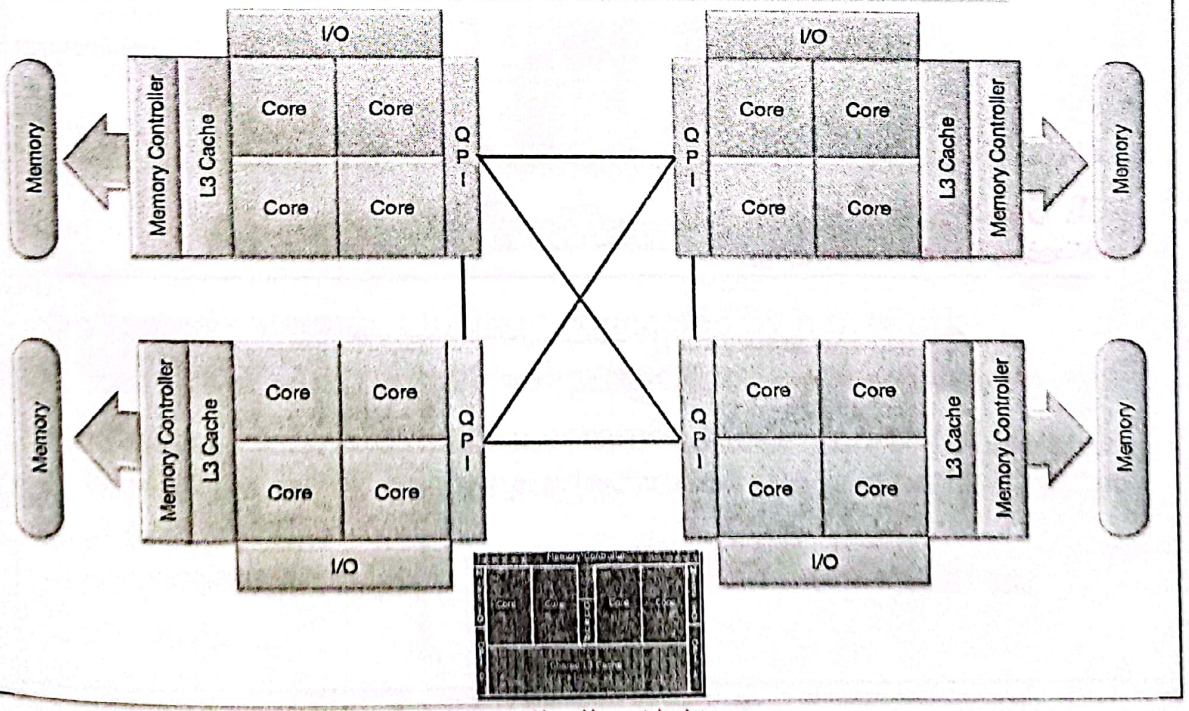
Quick Path Interconnect

غالبي و مشاهيك بستخدموا

© All Rights Reserved.

* الـ HW هو سهل بس هو الي فتح المجال لزيادة الـ cores .
* حتى الـ SW بالـ NUMA هو سهل

Shared-Memory Processor Example:
Intel Nehalem (Nov 2008)



* انتقل إلى السلكيات الخارجية *

الأساسية ← مسائل ←

HW ≤ ← Fundamental Issues In Shared-Memory Architectures

1. **Cache Coherence Protocols** → بحلها ال HW
   - Ensure that a thread accesses the most recent value of a shared variable

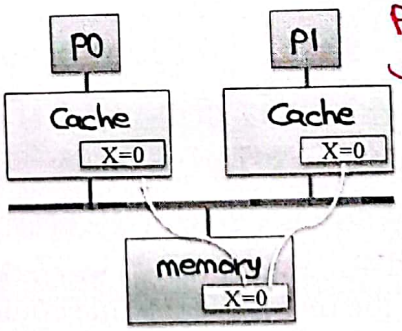
2. **Memory Consistency Models**
   - Explain how memory operations for each thread can be ordered

3. **Synchronization Primitives**
   - Describe the supported atomic operations by the instruction set architecture

Caches in Shared-Memory Can Cause Problems

Assume write-update caches

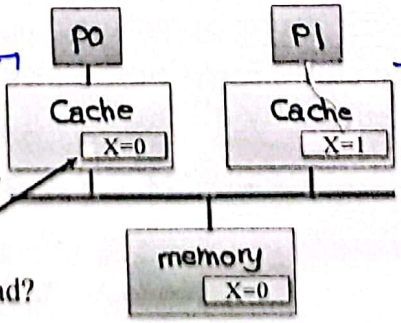
1 P0 and P1 read X



ال Cache ال Programer مابعرف عنه اشي ولا كيف بتصرف كله بكون HW ، مع ذلك ممكن يسببلي issues.

2 P1 overrides x

← بس صار P1 عبارة عن writer و كتب عال Copy اللي عنده لازم تكون القيمة الجديدة موجودة عند الكل وإلا بصير ناتج خاطئ.



→ عمل write (cache hit) لإنه X موجودة أصلاً.

What occurs if P0 performs read?

Cache Incoherence

- Both UMA and NUMA architectures often allow caches
- Caching shared-data can cause an *incoherency* problem:
  - A global state defined by the main memory
  - A local state defined by individual caches
  - Incoherent values in both states could lead to a correctness problem
- The incoherency problem can occur with write-update or write-through caches
- Remember that caches are managed by hardware (programmers cannot help)
- Solution: introduce hardware-managed *coherence protocols* in shared-memory architectures that maintain shared data coherent

Sufficient Conditions For Cache Coherence

- A memory system is coherent if the following conditions are satisfied:
  1. All reads by any processor P return **the most recently written value**
     - Note that the most recent value could have been written by the same processor P or any other processor
     - This property preserves program order
  2. Writes to the same location by any two processors are seen in the same order by all processors
     - Write serialization property

Cache Coherence Protocol

الحل:

- Two coherence protocols:
  - Snooping cache coherence
  - Directory-based coherence

« حنشرحهم بشكل مبسط »

Snooping Caches

- Bus-based coherence protocol
- ALL caches monitor ALL traffic on the bus
  - i.e., cache controller makes coherence updates upon processor access and bus request
- Example: when a processor writes shared data → لو حد من الـ Processors حاول به يكتب
  - Invalidates copies of all other processors بصيروا الباقي invalid لحد ما يخلص
    - Ensure exclusive access on writes مشان لو واحد به يقرأ ، يقرأ القيمة الجديدة مش القديمة بعدين بصير update للقيمة الجديدة لكل الـ Processors
    - Called write-invalidation protocol
    - Most commonly used protocol
  - Alternatively, updates copies of all other processors
    - Called write-update protocol

* أما لحد ما احنا كلنا readers ما بصير في مشاكل *

CamScanner

← الـ System البسيط بكفيه Snooping .

← يستخدم لما يكون عندك System huge (multicores) .

← يصير عندك زي table موجود بالـ Main Mem أو عند كل الـ Processors .

Directory-Based Caches

بكون فيه كل المعلومات ، أي حد بده يعمل read أو write بعمل Consultation مع الـ Directory - Based ليتأكد إنه العملية اللي حيعملها صحيحة او مش صحيحة .

- Uses a data structure called a directory that stores the status of each cache line
  - The directory can be in one centralized place or distributed
- When a variable is updated, the directory is consulted, and the cache controllers of the cores that have that variable's cache line in their caches are invalidated

تشاور .

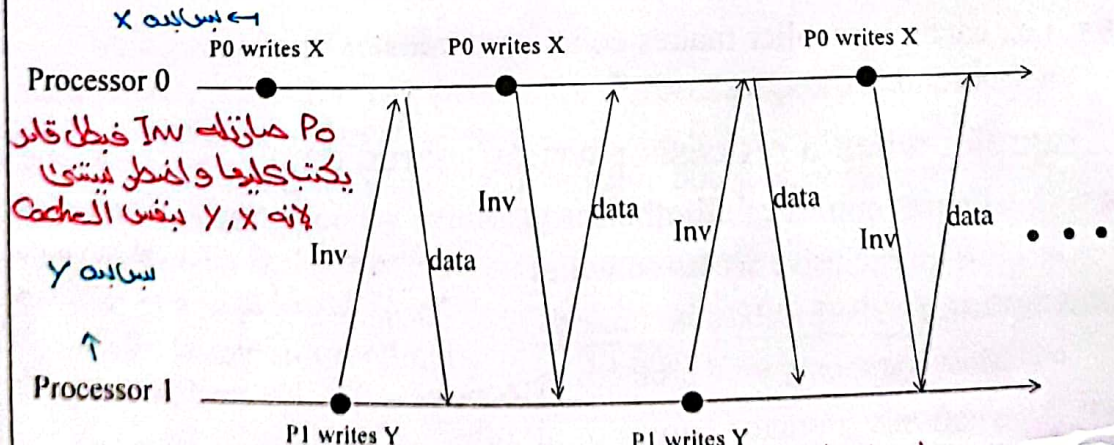
- Works with any interconnect network

← مشكلة موجودة ولابد منها وحلها موسهل .

False Sharing

← انحطوا جمب بعض وبنفس الـ Cache

- Assume that variables X and Y reside in the same cache block



P0 صارتله Inv فبطل قادر يكتب عليها واضطر يستنى لأنه Y,X بنفس الـ Cache

← وفي كمان Spatial locality ← لما بدك تجيب data من الـ Mem جيبها هي وجيرانها لأنه ممكن نطلبها لقدام .

Cache block is constantly invalidated (remember temporal locality) and coherences misses are caused **despite NO real data sharing**

Fundamental Issues In Shared-Memory Architectures

1. **Cache Coherence Protocols**
   - Ensure that a thread accesses the most recent value of a shared variable

2. **Memory Consistency Models** → مشكلة ما إلها علاقة بالـ Cache
   - Explain how memory operations for each thread can be ordered

3. **Synchronization Primitives**
   - Describe the supported atomic operations by the instruction set architecture

Memory Ordering

Initially, A=B=0 in both processors

P0	P1
A = 1	B = 1
print B	Print A

Is it possible P0 and P1 print B=0, A=0 ? What if the compiler or the processors P0 and P1 reordered the instructions?

هون الكتابة لازم تسبق القراءة في P0,P1 وهاي علاقة ممكن ماشين كـ logical

Initially, flag=0 in both processors

P0	P1
A = 1	While (flag==0) { }
flag = 1	print A

عشان أضمن A تنكتب قبل ما يجين قراءة

Is it possible P1 prints A=0 ? What if P0 issues the stores of A and flag in order but were reordered in interconnect?

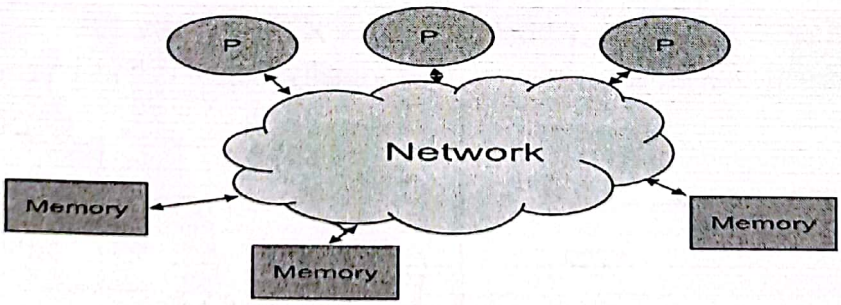
CamScanner

Memory Ordering (cont.)

- Coherency specifies rules for accessing ONE memory location across threads
- Coherency does not say anything about the order of MULTIPLE memory operations issued by a thread
- This leads us to the next fundamental issue in shared-memory architectures: *memory consistency models*

لـ اتفاق على ورق بين الـ HW والـ SW

Memory Consistency Model



- A memory consistency model is a contract between the hardware and the software that defines a set of rules governing how the memory system will process memory operations issued by a processor

CamScanner

Importance of Memory Consistency Models

- Memory consistency models establish the requirements of correct execution to the programmer, which affects his algorithm design

- Memory consistency models inform the compiler or the HW about which memory operations can be reordered without violation of correctness, which affects performance

Memory Consistency Types

- Many memory consistency types have been introduced in the literature that vary in their "strictness" in enforcing memory ordering

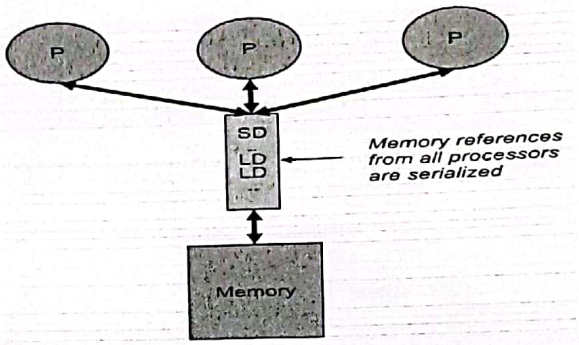
1. • Example of strict models is the *Sequential Consistency* model → بيكون الصعوبة عال HW أكثر، بحرمك من ال Performance شوي بس بالمقابل بسهلك البرمجة كثير

2. • Example of loose models is the *Relaxed Consistency* model → ال HW بعمل زي ما بده وال SW بصير يصلح ويحلل الأخطاء

المستخدم بالغالب لإنه بهمنا ال Performance يضل موجود ما نخسره.

CamScanner

Sequential Consistency



[Lamport] "A multiprocessor system is sequentially consistent if the result of any execution is the same as if 1) the operations of all processors were executed in some sequential order, and 2) the operations of each individual processor appear in this sequence in the order specified by the program"

Sequential Consistency Example

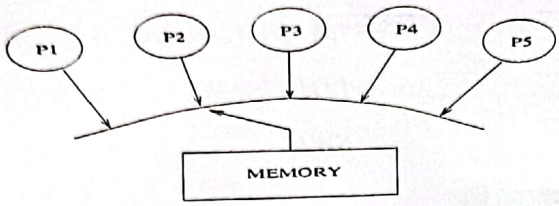
Assume processors P0 and P1 execute the following operations:

P0	P1
LD0	*LD1*
SW0	*SW1*

Which of the following is a possible execution scenario under the sequential consistency model?

LD0	*LD1*	*LD1*	*SW0*	*LD0*
SW0	*LD0*	*LD0*	*LD0*	*SW1*
LD1	*SW0*	*SW1*	*LD1*	*SW0*
SW1	*SW1*	*SW0*	*SW1*	*LD1*
✓	✓	✓	✗	✗

Implications of Sequential Consistency



Memory is a "single-time shared block"

- Every processor issues memory operations in program order
- Total memory operations order corresponds to any possible "interleaving" uniprocessor order
- All write operations are atomic, i.e., an operation by a processor "appears" instantaneous to other processors

Sequential consistency ما بسمحلك بالي كان بعملك ياه حل الـ

Relaxed Memory Model

- Sequential consistency is too strict → حازم اصلاً
  - Disallowing memory operations to be reordered hurts performance
  - Forcing all memory operations to be atomic is too expensive
- Solution: use *relaxed memory consistency model*
  - Allow loads and stores to complete out-of-order and relax their atomicity requirement
  - Only enforce sequential consistency for *synchronization operations* → بعطلك شرط إنك مثلاً تصنف الـ LD العادية و Special ، ومابصير العادية تسبق الـ Special بالتنفيذ

Ex:

PO

lw
sw
lw
sw

→ بتعملهم reorder زي ما بدك.

lw special → لازم اللي قبلها يشتغلوا قبلها واللي بعدها بعدها.

lw
sw
lw
sw

→ بتعملهم reorder زي ما بدك.

Relaxed Consistency Example

Assume processors P0 and P1 execute the following operations:

P0	P1
LD0	*LD1*
SW0	*SW1*

Which of the following is a possible execution scenario under the relaxed consistency model?

LD0	*LD1*	*LD1*	*SW0*	*LD0*
SW0	*LD0*	*LD0*	*LD0*	*SW1*
LD1	*SW0*	*SW1*	*LD1*	*SW0*
SW1	*SW1*	*SW0*	*SW1*	*LD1*
✓	✓	✓	✓	✓

Sequential Consistency vs Relaxed Consistency

- Two intuitive differences:
  - Writing parallel programs with sequential consistency is easier than writing programs with relaxed consistency
  - But parallel programs with relaxed consistency perform faster than parallel programs with sequential consistency

- Keep in mind that it is the programmer job to enforce a particular memory order (when needed) by using synchronization operations
  - This holds for both consistency models

CamScanner

Fundamental Issues In Shared-Memory Architectures

1. Cache Coherency Protocols
   - Ensure that a thread accesses the most recent value of a shared variable

2. Memory Consistency Models
   - Explain how memory operations for each thread can be ordered

3. **Synchronization Primitives**
   - Describe the supported atomic operations by the instruction set architecture

← الحل باستخدام استراتيجية الـ Penalty وإعطاء أوقات معينة لكل Processes .

* الـ Code اللي تحت فيه مشكلة إنه ممكن يصير في Infinite loop لو عندي كثير Processes بتنافسوا إنه يعملوا الـ Store .

Synchronization Primitives

- Shared memory architectures support special atomic operations that enable the aforementioned three synchronization methods
- Those atomic operations are defined by the instruction set architecture
- Example: atomic exchange with load-link/conditional-store instructions

← عشان أعمله Atomic

الـ Value اللي بالـ Mem أحطها بالـ reg والعكس (Swap)

```
try: mov r3, r4   # move exchange value
     ll r2, 0(r1)  # load linked
     sc r3, 0(r1)  # store conditional
     beqz r3, try  # if store fails
     mov r4, r2    # load value to r4
```

reg ←

Mem ←

← عملت نسخة لـ Value بـ reg 3

← حطيت الـ Value اللي بالـ reg في الـ Mem

Succeeds only of no updates occurred to the memory location since the load-link

When stores fails, it sets r3 to zero

← لو 0.r3 معناها فشلت الـ Store فبرجع بعيد الـ code أما لو نجح فخلص بتنجح عملية الـ Swap

Shared Memory Synchronization

- Synchronization is a special code that describes how to coordinate sharing among threads for correct execution
- Generally, there are three types of synchronizations:
  - Mutual exclusion: a code region that threads can only execute atomically, i.e., one thread at a time
  - Barriers: a point in execution where all threads must wait for every thread to reach this point
  - Signal-wait mechanism: a pairwise coordination between two threads where one thread stops its execution and waits for a signal from the other thread to continue its execution

© All Rights Reserved.

Synchronization Examples

Signal-wait mechanism

T0	T1
A = 1	While (flag==0) { }
flag = 1	print A

T1 repeatedly tests a condition till it becomes true

mutual exclusion

```
while (true){
  if (LOCK(&t) == true) break ;
}
....   // critical code
UNLOCK(&t) ;
```

Each thread repeatedly tries to **acquire** the lock till its successful

Only one thread enters the critical section

When finishing the critical section, **release** the lock

© All Rights Reserved.

Achieving High-Performance Computing On Multicores

- Programmers write a *multithreading program*, which is a parallel program that runs on multicores
  - Programmer partitions the computation among threads
  - Programmer insert synchronization operations to coordinate data sharing between threads
  - Communication between threads is done implicitly through the memory

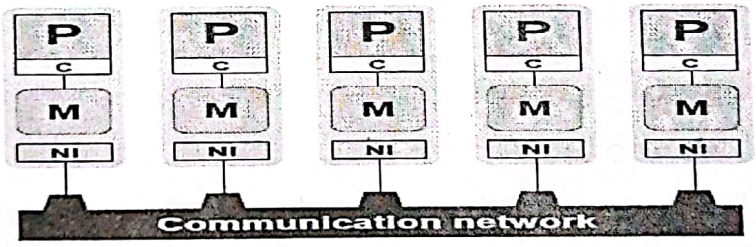
- Several programming models are available for shared-memory platforms, such as *Pthreads*, *OpenMP*, *Cilk*, and *Intel TBB*

* في لغات بتدعم الـ Multithreads أو بتستخدم الـ (Program models) و بتعملهم Support بلغات البرمجة.

© All Rights Reserved.

* مابدك فيه أي إضافة هالـ hardware كل اللي بنحتاجه Network نشبكهم ببعض.

Distributed-Memory Platforms



- A set of computers that are connected by a network
- Each computer runs a local program on a local memory
- Processors **do not** share a common memory, instead they communicate by exchanging send/receive messages through the network
- Network Interfaces (NI) mediate the connections to the network

© All Rights Reserved.

CamScanner

Properties of Distributed-Memory Platforms

Merits: من أهم فوائده: Scalability

- Allows an easy scale-out: simply add new machines to the network
- Allows simpler hardware: the complications of sharing a physical memory (such as implementing synchronization operations) do not exist
- Inexpensive: you can connect simple uniprocessors using a cheap network to build a parallel machine
- Dependable: Operational nodes can replace faulting nodes because all nodes are independent

Problems: المشاكل — مشكلة ثانية: Communication latency

- Messaging has longer latency than memory access
- Programmers need to insert communication messages in their codes "Programing" المشكلة بال

© All Rights Reserved.

Interconnection Networks In Distributed-Memory Platforms

- Divided into two groups
  - Direct interconnect
    - Each switch is directly connected to a processor memory pair, and the switches are connected to each other
  - Indirect interconnect
    - Switches may not be directly connected to a processor

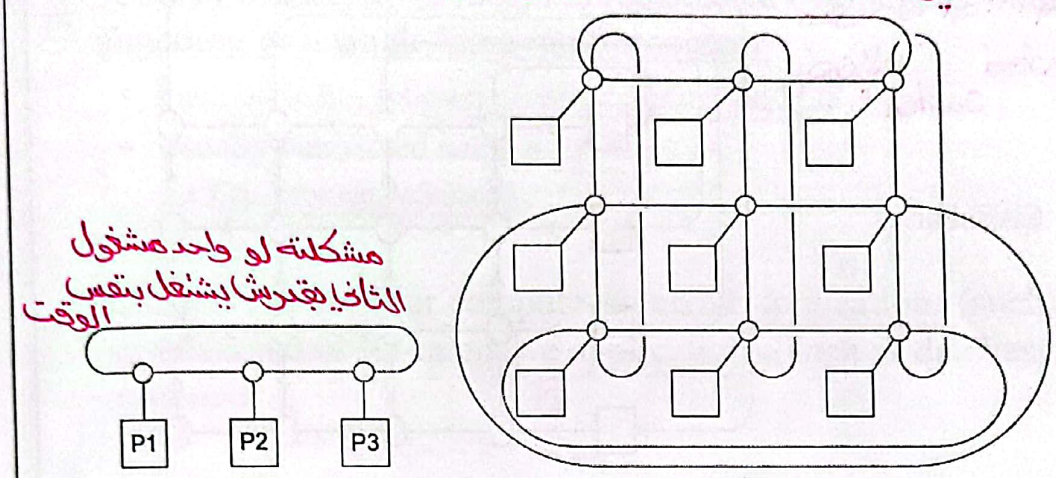
© All Rights Reserved.

CamScanner

Direct Interconnect

" فيها Parallism أعلى حتى لو أقلى "

مشكلة لو واحد مشغول
التاني بقدرش يشتغل بنفس
الوقت

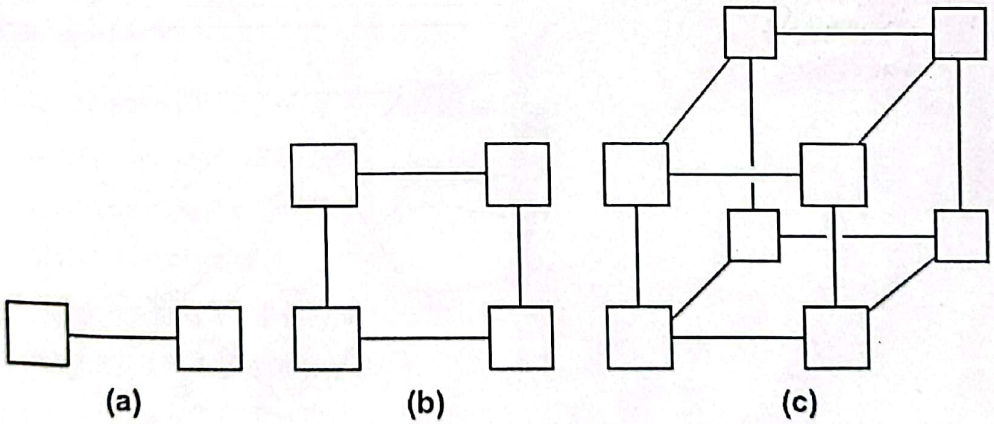


(a) ring

(b) toroidal mesh

© All Rights Reserved.

Direct Interconnect (cont.)



(a) (b) (c)

(a) One-, (b) two-, and (c) three-dimensional hypercubes

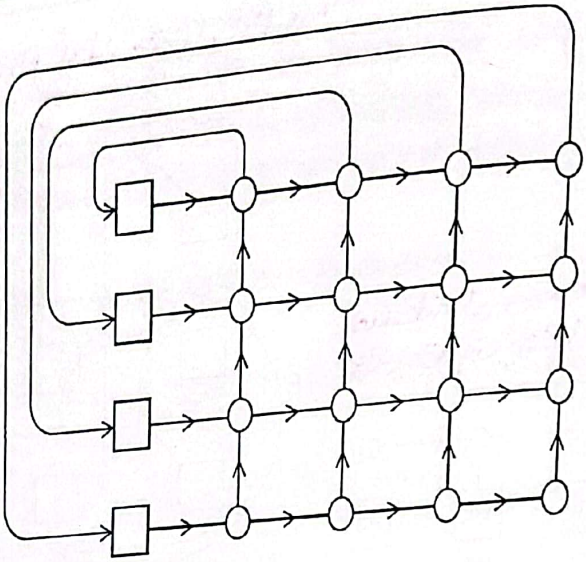
© All Rights Reserved.

CamScanner

Indirect Interconnect

مش كل Computer بكون اله "dedicated Switch"

Crossbar

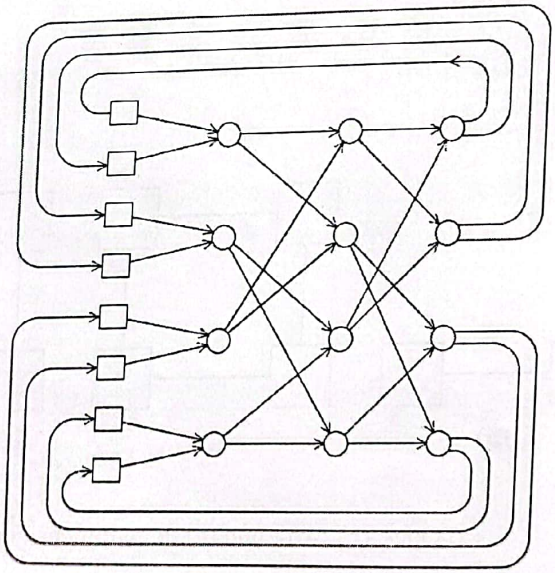


© All Rights Reserved.

Indirect Interconnect (cont.)

أغلى ولكنه أكثر Parallism

Omega



© All Rights Reserved.

CamScanner

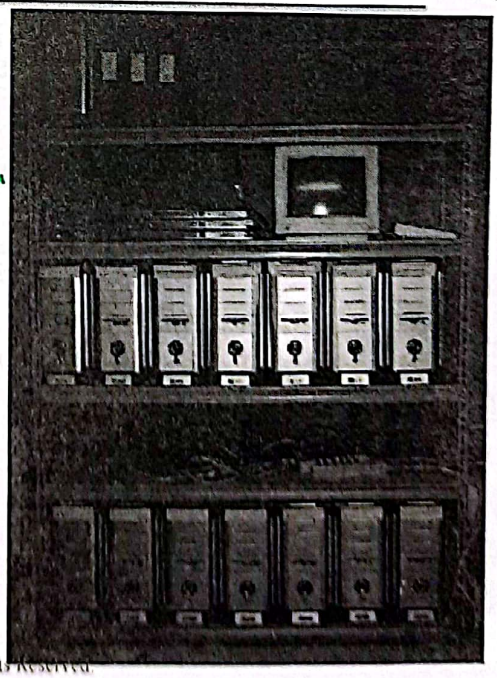
Computer Cluster

- A set of computers (nodes) interconnected over a network that functions as a single large multiprocessor
  - Each node has its own private memory and OS
  - Usually connected using a LAN
    - E.g., Ethernet, Infiniband
- Clusters are used for compute-intensive applications (such as simulations) or IO-intensive applications (such as database systems)
- Clusters are, by far, the most common type of distributed-memory platforms

© All Rights Reserved.

Beowulf Cluster

- A simple, low-cost home-built cluster
- Commercial off-the-shelf computers
- Ethernet network (or any other network type)



© All Rights Reserved.

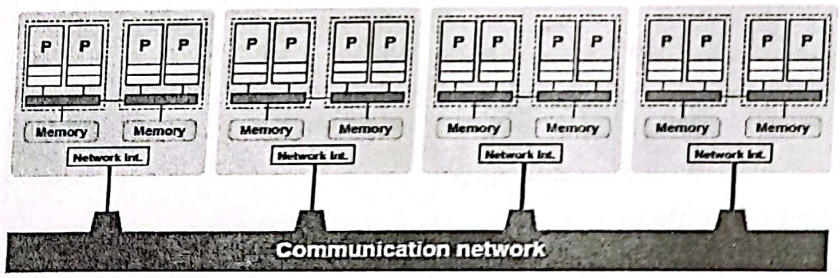
Achieving High-Performance Computing On Clusters

- Programmers write a *message-passing* program, which is a parallel program that runs on a cluster
  - Programmer partitions the work among cluster nodes
  - Programmer inserts explicit send/receive messages to exchange shared data between cluster nodes
- *MPI* is currently the dominant programming model for clusters

© All Rights Reserved

يعني صرت تقدر تكتب برنامج
. OpenMP مع MPI

Hybrid Clusters



- Each node consists of a multicore chip (in this case NUMA)
- The best of both worlds: scalability + faster performance per node
- Drawback: more costly nodes + might complicate programmability in come cases
- Today, most HPC clusters have multicore nodes

© All Rights Reserved.

high performance Computing

Clusters Are Ubiquitous

- The high scalability, immunity to failure, cost effectiveness and computational power made clusters the backbone of most (if not all) HPC platforms on the planet
- Nowadays, millions of clusters are scattered around the world to make the current internet era we live in possible
- Let us consider three examples where you can find clusters:
  - Supercomputers
  - Distributed systems
  - Cloud computing

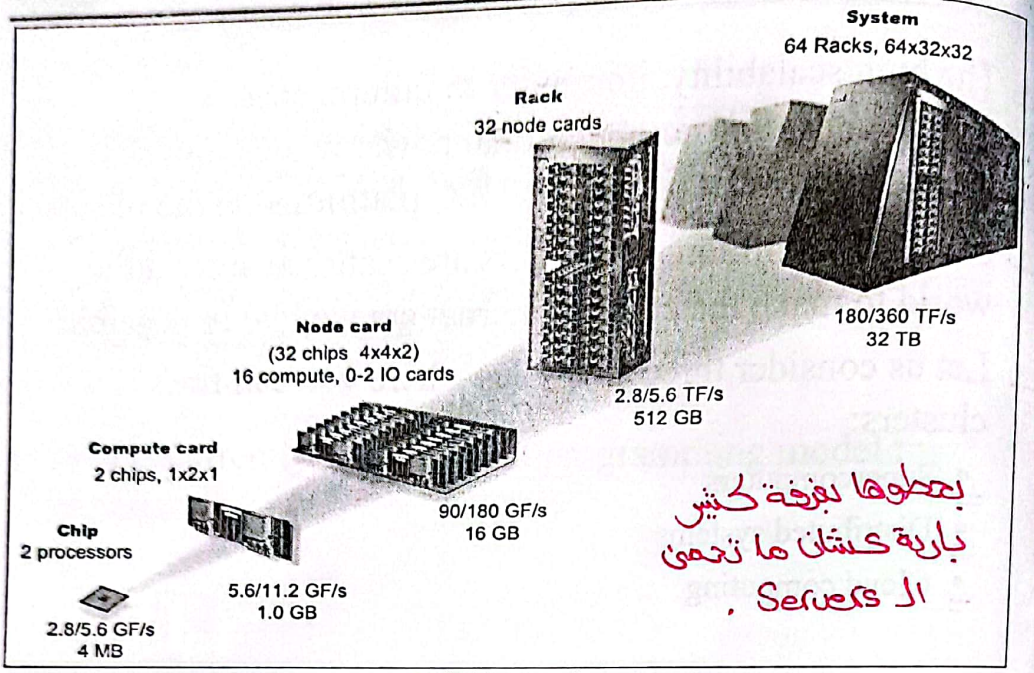
© All Rights Reserved.

Supercomputers

- A supercomputer is basically a *HPC* cluster with a large number of nodes and a high-performance interconnection network
- A node in a supercomputer typically has a hierarchical SIMD / MIMD architecture with a lot of cores
- Performance is measured in floating-points operations per second
- Supercomputers are used for scientific computing: quantum mechanics, weather forecasting, climate research, oil and gas exploration, molecular modeling, …etc
- Supercomputer example:
  - BlueGene/L (2007), 106.496 nodes x 2 PowerPC (700MHz)
  - IBM Sequoia (2012), 16,3 PFlops, 1.6 PB memory, 98304 compute nodes, 1.6 Million cores, 7890 kW power

© All Rights Reserved.

BlueGene/L



بحطوها بغرفة كتير
باردة عشان ما تحمى
ال Servers .

© All Rights Reserved.

Top500

بصنف ال Supercomputers ←

- Annual ranking of the most powerful 500 supercomputers in the world (https://www.top500.org/)
- LINPACK benchmarks are used to measure performance
- Current #1 is *Fugaku*, a supercomputer located in RIKEN Center for Computational Science, Japan
  - A64FX 48C 2.2GHz
  - 7,630,848 cores
  - 5,087,232 GB RAM → وحدة بنقيس فيها سرعة
  - 442,010 TFlop/s (Linpack performance) ال Supercomputer
  - https://www.r-ccs.riken.jp/en/fugaku/project

© All Rights Reserved.

Distributed Systems

- [Tanenbaum] "A distributed system is a collection of independent computers that appear to the users of the system as a single computer"
- [Coulouris et al.] "system in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages"
- My definition: A distributed system is a collection of tightly-coupled or loosely-coupled clusters that cooperate together to make a particular functionality continuously available to the user → cluster of clusters

مثلاً لما تفتح Facebook بكل مكان نفس الخدمات والـ clusters تاعته موزعة بين آسيا وأوروبا وهكذا.

Why Distributed Systems?

- Scalability → Computers أبني ديتابيز أضل أقدر أزيد فيه
  - Many applications nowadays (e.g., web services, online games, peer-to-peer applications, social networks) have enormous number of users with dispersed geographical locations
- Parallelism → أخدم Multiple Users بنفس الوقت
  - Users can access the resources simultaneously without interfering with each other (i.e., do independent tasks)
- Highly Reliability → إذا Machine وحدة فشلت بقى الباقي يحل محلها
  - The failure of one machine will not affect other machines
- Transparency → مش مطلوب من الـ User يحدد موقعه ليفتح الـ App أوعيك
  - Users have little or no knowledge of where distributed systems' resources are physically located

CamScanner

Distributed Systems Course

- Many universities dedicate a full course to teaching distributed systems (typically offered to graduate students)
- Discussed topics in this course include:
  - Design approaches for distributed systems
  - Networking and internet protocols
  - Distributed operating system
  - Processes communication in distributed operating systems and remote procedure call protocols
  - Distributed file systems
  - Secure Networks
  - Distributed system applications such as peer-to-peer systems and web services

Suggested textbook

المواضيع اللي طرحت في هذا الكتاب

Making data available as a serves

extention to the Internet

Cloud Computing

مثل جوجل درايف احنا ما بنعرف مكانه ال Physical بس بنخزن عليه فيعتبر Cloud.

- A relatively new computing paradigm where users have on demand access to an "infinite" pool of virtual resources

ما بعطي access عال Physical بعطي access عال Virtual.

- The main principle behind cloud computing model is to offer *computing*, *storage*, and *software* as a service or as a utility *via the internet*

- My personal view: cloud computing is a fancy term for a *virtual distributed system* that offers internet services on infrastructure level, platform level and software level

CamScanner

Why Cloud Computing?

- On-demand pay-as-you-go services
  - You pay for what you use

+ بطل يهمني الـ VM بأي ما استخدمه وما دخلني فيه.

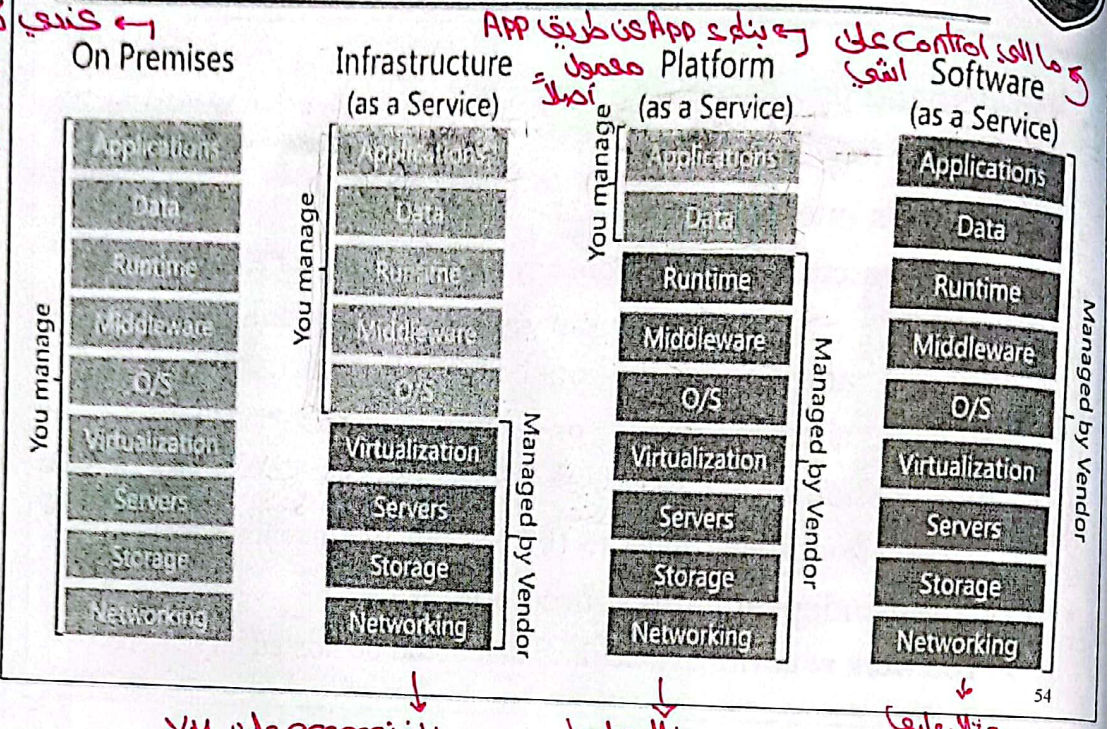
- Ubiquitous internet access
  - You can access the cloud from anywhere as long as you have internet → مادام معك internet بتوصل لأي اشي بأي وقت
- Cut capital and operational cost of buying hardware
  - Skip the pain of maintaining on-premises clusters
- Rapid elasticity → يشغل الـ VM's تلقائي حسب كمية الـ users اللي بيشغلوا، لو users قليل بيشغل VM's قليل ولو كثار بيشغل كتار.
  - E.g., go from 10 machines to 100 or from 100 machines to 10
- Location-independent resource pooling
  - You work with virtual machines that could be hosted anywhere

Cloud Computing Services

- *Software as a service* (*SaaS*)
  - Users access applications that are hosted on the cloud
- *Platform as a service* (*PaaS*)
  - Users build and deploy their own software applications on the cloud
- *Infrastructure as a service* (*IaaS*)
  - Users remotely control a collection of "virtual" machines in terms of operating system, storage, network connectivity and applications

CamScanner

IaaS, PaaS and SaaS



54

Cloud Platform Vendors

- Amazon Web Services (AWS)
- Windows Azure
- Google App Engine
- Rackspace

55

Last Remarks

- Parallel machines can be classified based on their *concurrency* and *communication* models
- Writing efficient parallel programs require exposing the programmer to the architectural details of parallel machines
- There is no standard parallel machine, and therefore there is no standard parallel programming environment
- In this course, we will learn how to write parallel programs for shared-memory and distributed-memory platforms

© All Rights Reserved.

Parallel Software

Instructor: Dr. Fahed Jubair

Computer Engineering Department

University of Jordan

CamScanner

Parallel Computing Quick Recap

- Parallel computing achieved by running a parallel program on a parallel machine
- Different parallel machines have different architectural organizations and therefore they require different types of parallel programs

© All Rights Reserved.

Single program Multiple Data ← SPMD

- The programmer only needs to write a single program that is parameterized by the thread/process number
- At runtime, each thread/process executes the pieces of code that correspond to their thread/process number
  - E.g., using conditional branches
- SPMD is the most common style for parallel programing with MIMD architectures
  - The programmer writes a single program that gets replicated on all processors

© All Rights Reserved.

CamScanner

مشكلته انه ممكن حجم الـ Code يكبر لو كان كل Process/thread بنفذوا Code مختلف تمامًا عن بعض.

SPMD (cont.)

- Assume the number of threads/processes is n
- Below is a high-level picture of SPMD-style coding

بنقدر نعمل الـ Code بطريقة أنكفى عشان نصغر حجم الـ Code.

```
// multithreading code
if (thread_id == 0)
    do task 0
else if (thread_id == 1)
    do task 1
else if (thread_id == 2)
    do task 2
...
else  // thread_id = n − 1
    do task n − 1
```

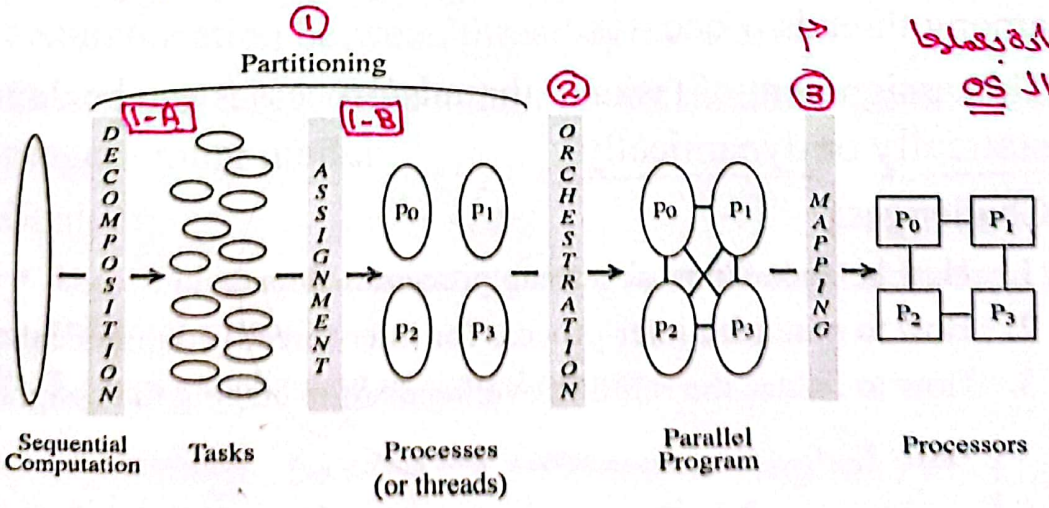
```
// message-passing code
if (process_id == 0)
    do task 0
else if (process_id == 1)
    do task 1
else if (process_id == 2)
    do task 2
...
else  // process_id = n − 1
    do task n − 1
```

© All Rights Reserved.

* لما نصمم Parallel program بنمر بمراحل معينة :

General Steps For Writing A Parallel Program

- [Culler et al.] described four general steps to parallelize a given sequential program:



علشان يعملها الـ OS

© All Rights Reserved

CamScanner

1st Step: Decomposition

أهم خطوة

بدي أخلق منه Parallel tasks قدما بقدر. Serial computation

- Break up computation into tasks with as much concurrency as possible
- Tasks can be created statically or dynamically

من أطها بقرر كم أقسم الـ task (أسهل)

- The challenge is how to identify enough concurrency to keep the processors busy all the time, yet not so much that the overhead of managing the tasks becomes substantial compared to the useful work done
- In some cases, programmers need to perform algorithmic changes to increase parallelism

بدك تخلق كثير tasks بس ما بدي يكون في overhead عالي

أنا بخلق task وبس تشتغل هي بتخلق task ثانية واللي بعدها ممكن تخلق ثالثة وهكذا ما بخلقهم من أول مرة مثل اللي بعمل Merge Sort

© All Rights Reserved

2nd Step: Assignment

صح إني خلقت tasks كثيرة بس ما عندي كثير threads فوين بدي أوزع الـ tasks اللي قسمتها على الـ threads أو الـ Processes المتاحة عندي.

- Specify the mechanism by which tasks will be distributed among threads/processes
- This assignment of tasks to threads/processes can be done statically or dynamically
- Challenges:
  1. How balance the work among processes/threads
  2. How to minimize inter-process (or inter-thread) communication
  3. How to reduce the runtime overheads of managing the assignment

© All Rights Reserved.

CamScanner

Putting The Two Together

- The two steps together (decomposition and assignment) are called *work partitioning* → سواء بشتغل Multi threading أو Message passing رح أعملهم.
- Logically, work partitioning is algorithm-dependent and machine-independent
- However, the architectural details of parallel machines often impacts the programmer decisions on how to perform work partitioning ↓

* بفصل الخطوتين عن بعض لو لقيت مجموعة من الـ tasks بينهم علاقة فالأفضل أحطهم بنفس المكان بدل ما يروحوا أماكن مختلفة.

© All Rights Reserved

3rd Step: Orchestration

"Synchronization" = ↺

كشان ما يكون في Race condition ←

- Specify mechanisms of synchronization and communication between threads/processes
- This step is largely dependent on the architecture and the programming model
- Goal:
  - * Use efficient synchronization and communication mechanisms to reduce their cost
  - * Reduce serialization of shared resources

* Message Passing → Messages on physical Mem
* Multithreading → Load/store

من ناحية التواصل

© All Rights Reserved

CamScanner

بتدخل فيها خاصة بال Message passing لإنها ممكن تأثر على كيش.
بعملها الـ os ، بتأثر على Performance والها علاقة فيها.

4th Step: Mapping

- Determine which threads/processes run on which processors
- Most programming models have runtime support to perform this step automatically to the programmer
- Mapping can have significant performance impact
  - For example, in NUMA architecture, mapping threads that access overlapped shared data regions to the same socket reduces remote memory accesses
  - Network topology can play a role in reducing communication latency for NUMA and distributed-memory architectures

© All Rights Reserved.

Example – Histogram

بدي أعدكم رقم وقع بين 0 و 1 وكم رقم بين 1 و 2 وهكذا.

- Given an array of float numbers chosen from 0.0 – 5.0, consider a program that builds a histogram by distributing all float numbers into five "bins" where: → Counters
  - Bin[0] has the count of all floats in 0 <= *num* < 1
  - Bin[1] has the count of all floats in 1 <= *num* < 2
  - Bin[2] has the count of all floats in 2 <= *num* < 3
  - Bin[3] has the count of all floats in 3 <= *num* < 4
  - Bin[4] has the count of all floats in 4 <= *num* <= 5

© All Rights Reserved.

CamScanner

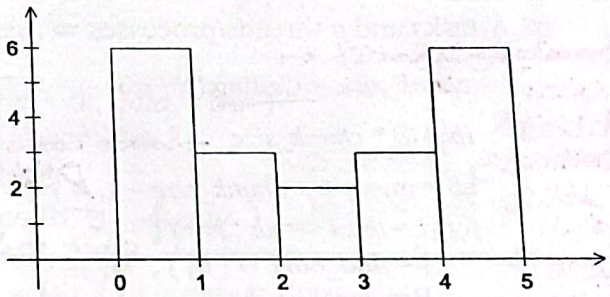
Histogram (cont.)

For example, if we have the following 20 floats:

1.3,2.9,0.4,0.3,1.3,4.4,1.7,0.4,3.2,0.3,4.9,
2.4,3.1,4.4,3.9,0.4,4.2,4.5,4.9,0.9

Then the histogram is determined as follows:

Bin[0] = 6
Bin[1] = 3
Bin[2] = 2
Bin[3] = 3
Bin[4] = 6



© All Rights Reserved.

Histogram – Sequential Code

- Let $D[N]$ be the array of floats, where N is the number of floats
- Let Bin[5] be the array of bins
- Let *find_bin*(*float d*) be a function that determines which Bin float *d* belongs to

```
for ( i = 0; i < N ; i++) {
        j = find_bin ( D [ i ] ) ;
        Bin [ j ]++ ;
}
```

© All Rights Reserved.

بقسم الـ Code لـ chunks كل واحد بعمل جزء معين بعدين بجمع النواتج.

Histogram – Partitioning

- Let p be the total number of threads/processes and id be the thread/process number
- Decomposition
  - Each iteration in the loop represents a single task
  - We have N tasks
- Assignment
  - N tasks and p threads/processes ⇒ assign N/p tasks to each process/thread

« Sudo Code »

لإنه مو شرط يقسموا بعض فعملنا Ceiling لناخذ الرقم الأكبر من الرقم الكسري

```
chunk_size = Ceiling (N / p)
lb = id * chunk_size
ub = min ( lb + chunk_size – 1, N )
for (i = lb ; i <= ub ; i++) {
    j = find_bin ( D [ i ] ) ;
    Bin_local [ j ]++ ;
}
```

→ lower bound

N-1

upper bound

Race condition ما في

Is this partitioning really even?
What if N=124 and p=16?
Can you create a better math for more balanced partitioning?

بهاي الطريقة ممكن الـ threads الأخيرات ما ياخدوا اشي.

© All Rights Reserved.

EX: 30 task ، 4 thread ← 30/4=8.5 ← 9 لكل thread ، الأول بياخذ 9 والثاني 9 والثالث 9 والرابع بس 3 فهذا مش منطقي ولا عادل فهيك ممكن تطول لتخلص المهمة عكس لو كانوا موزعين ب منطق.

Histogram – Orchestration with Multithreading

- Each threads computes a local Bin array
- All threads update the global Bin array
- Use **mutual exclusion** to avoid race conditions

```
for (i = lb ; i <= ub ; i++)
    j = find_bin ( D [ i ] ) ;
    Bin_local [ j ]++ ;
}

critical section {
    for ( i = 0 ; i < 5 ; i++)
        Bin [ i ] = Bin [ i ] + Bin_local [ i ] ;
}
```

عملت update لـ Bin الخارجية.

© All Rights Reserved.

Histogram – Orchestration with Message–Passing

- Each process computes a local Bin array
- All processes send their local Bin arrays to process 0
- Process 0 combines all local Bin arrays to compute overall Bin array

```
for (i = lb ; i <= ub ; i++)
    j = find_bin ( D [ i ] ) ;
    Bin [ j ]++ ;
}
if ( process_id != 0 )
    send_msg ( receiver = 0 , data = Bin ) ;
else
    for (i = 1 ; i < T ; i++) {
        recv_msg ( sender = i , & data ) ;
        for (j = 0 ; j < 5 ; j++)
            Bin [ j ] = Bin [ j ] + data [ j ] ;
    }
```

© All Rights Reserved.

« لقدام بنشوف وبندرس طرق أفضل »

Input and Output

- How do you think threads/processes in parallel programs should
  - Read inputs from user?
  - Print errors?
  - Dump text to output files?
- Input/output functionalities can potentially complicate parallel programming
- For simplicity, in this course, we will let only a single thread or process responsible for input/output operations

© All Rights Reserved

Measuring Performance

- Recall the standard definition of performance
- For a program x, performance is measured as follows:

  $\text{Performance}_x = 1 / \text{Execution time}_x$

- Program x is n times faster than program y if:

  $\text{Performance}_x / \text{Performance}_y = \text{Execution time}_y / \text{Execution time}_x = n$

© All Rights Reserved.

Speedup

- Number of the threads/processes = p
- Execution time of the serial version = T_{serial}
- Execution time of the parallel version = $T_{parallel}$

$$\text{Speedup} = \frac{T_{serial}}{T_{parallel}}$$

→ before Improument (الكبير)
→ After Improument (الصغير)

- It is called "linear speedup" when **speedup = p** → Ideal Speedup صعب نحصلها لإنه مستحيل نكون Parllel 100% وعندي overhead.
- Achieving linear speedups is difficult due to
  - Some parts of the algorithm are purely sequential
  - Runtime overhead of parallelism

© All Rights Reserved.

CamScanner

Execution Time Breakdown

The sequential part of the code that can be parallelized

The sequential part of the code that is "purely" sequential

Ex: I/O code

$$T_{serial} = T_{parallelizable} + T_{non\text{-}parallelizable}$$

$$T_{parallel} = \frac{T_{parallelizable}}{P} + T_{non\text{-}parallelizable} + T_{overhead}$$

كلما زاد بنخر ؟

Assuming linear speedup

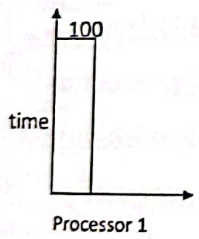
كلما قلت P بنخر

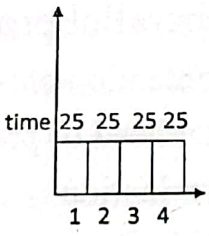
كلما زاد بنخر

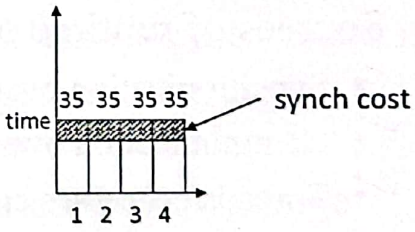
Parallelism overhead

© All Rights Reserved.

Example







$$S_p = \frac{100}{25} = 4.0,$$

Perfect parallelization!
Does it ever occur?

$$S_p = \frac{100}{35} = 2.85,$$

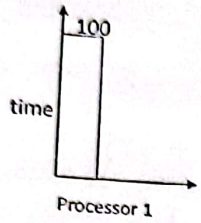
perfect load balancing

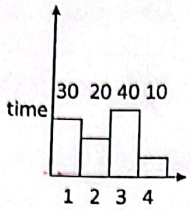
أقرب للأرض الواقع تقريبًا

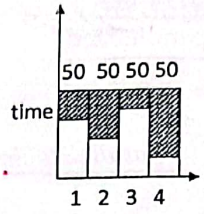
© All Rights Reserved.

CamScanner

Example (cont.)







closest to real life parallel programs

الواقعي أكثر

$$S_p = \frac{100}{40} = 2.5,$$

load imbalance

$$S_p = \frac{100}{50} = 2.0,$$

load imbalance and sync cost

© All Rights Reserved.

Sources of Overhead

- Sources of runtime overhead in parallel programs: Main Source
  - Synchronization overhead (e.g., contention on shared resources)
  - Communication overhead (e.g., overhead of preparing messages)
  - Threads/processes creation and termination
  - Extra computation to perform partitioning
  - Dynamic tasks creation
  - Load imbalance

© All Rights Reserved.

CamScanner

Amdahl's Law

هيفيد كدراسة مبدئية

- Let f be the fraction that is "parallelizable" of the program

$$\text{Speedup} = \frac{T_{serial}}{T_{parallel}} = \frac{1}{1-f+\frac{f}{p}}$$

كل ما كبرت من مصلحتي

كل ما كبرت من مصلحتي ← speedup

$$\text{Speedup} \leq \frac{1}{1-f}$$

- Amdahl's law tell us that, even if p=∞, the serial part of a parallel program limits the performance to an upper bound of 1/(1-f)
- For example, if f = 95%, then the maximum theoretical speedup we can get is 20! ← $\frac{1}{1-95\%}$

© All Rights Reserved.

No Need To Despair

ما تخليه داعي للتشاؤم من Amdahl's بس

1. Amdahl's law does not take the problem size into account
   - For many problems, as we increase the problem size, the "inherently serial" fraction of the program decreases in size
2. Thousands of scientific applications have shown huge speedups on large distributed-memory systems
3. Many applications benefit from "small" speedups

© All Rights Reserved.

CamScanner

Scalability

- A parallel program is ***scalable*** if it can handle ever increasing problem sizes
- Let us define efficiency as follows:

$$\text{Efficiency} = \frac{T_{serial}}{p * Tparallel}$$

- A parallel program is ***strongly scalable*** if the efficiency, for a fixed problem size, remains fixed while increasing the number of threads/processes
- A parallel program is ***weakly scalable*** if the efficiency, for a fixed number of threads/processes, remains fixed while increasing the problem size

© All Rights Reserved

Benchmarks

- A set of standard programs that measure performance
- Reasonable representation of real-world applications
- Role of benchmarks:
  - Compare different machines
  - Help exploring architectural designs
  - Identify bottlenecks
- Parallel benchmarks are benchmarks that are designed for measuring the performance of parallel machines
- Examples of parallel benchmark suites: NPB, PARSEC, Rodinia and SPLASH-2

© All Rights Reserved.

« للمطالعة وزيارة المعرفة »

PARSEC

- Princeton Application Repository for Shared-Memory Computers
- Benchmark Suite for multicores
- Available at http://parsec.cs.princeton.edu/
- Objectives:
  - Multithreaded Applications: Future programs must run on multicores
  - Emerging Workloads: Increasing CPU performance enables new applications
  - Diverse: Multicores are being used for more and more tasks
  - State-of-Art Techniques: Algorithms and programming techniques evolve rapidly

© All Rights Reserved.

PARSEC (cont.)

Program	Application Domain	Parallelization
Blackscholes	Financial Analysis	Data-parallel
Bodytrack	Computer Vision	Data-parallel
Canneal	Engineering	Unstructured
Dedup	Enterprise Storage	Pipeline
Facesim	Animation	Data-parallel
Ferret	Similarity Search	Pipeline
Fluidanimate	Animation	Data-parallel
Freqmine	Data Mining	Data-parallel
Streamcluster	Data Mining	Data-parallel
Swaptions	Financial Analysis	Data-parallel
Vips	Media Processing	Data-parallel
X264	Media Processing	Pipeline

© All Rights Reserved.

CamScanner

« مشكلة ال Parallel program إنه ما إله Standard »

Concluding Remarks

- In this course, we will focus on writing programs for homogeneous MIMD systems using the SPMD-style
- Parallel program design
  - Decompose computation into tasks
  - Assign tasks to threads/processes
  - Coordinate data sharing between threads/processes
  - Map threads/processes to processors
- Performance evaluation is very important to assess programming quality as well as the underlying architecture and how they interact
- Scalability and efficiency measure the quality of parallel programs

© All Rights Reserved.

* C++ أسرع من ال Java لإنه طريقة ال translation تختلف.
* C++ مش Portable عكس ال Java والبايثون لإنه بتقدر تطبقه على أماكن مختلفة.

Introduction to OpenMP

Program model ← بقدر أشغلها
not language فقط Multicore

Instructor: Dr. Fahed Jubair

Computer Engineering Department

University of Jordan

CamScanner

Definition

- [Wikipedia] OpenMP (**Open** **M**ulti-**P**rocessing) is an *application programming interface* (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran
- In other words, OpenMP is a programming model that specifies how to **extend** a sequential programming language in order to support multithreading programming
- Major compiler vendors for OpenMP: PGI, Cray, Intel, Oracle, HP, Fujitsu, Microsoft, AMD, IBM, NEC, ... Etc
- Current standard: version 5.1, Available at https://www.openmp.org/spec-html/5.1/openmp.html

© All Rights Reserved.

OpenMP Components

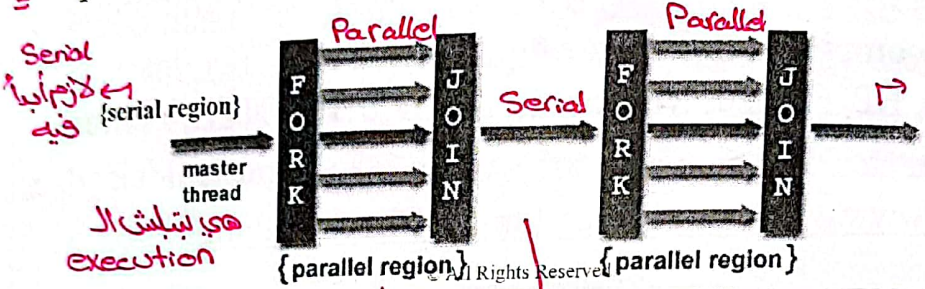
- Compiler directives and clauses
  - Interpreted when OpenMP compiler option is turned on
  - Each directive applies to the succeeding structured block
- Runtime function calls
  - Supported functionalities by an OpenMP runtime library
- Environment variables
  - User-controlled variables that affect the execution

© All Rights Reserved.

CamScanner

Fork-Join-Parallelism

- OpenMP uses the fork-join model for its parallel execution
  - Only **master thread** executes serial regions
  - Master thread **forks** new threads at the beginning of parallel regions
  - Multiple threads share work in parallel
  - Threads **join** at the end of the parallel regions
- Each thread works on global shared and its own private variables
- Threads communicate implicitly by reading and writing shared variables
- OpenMP uses a relaxed memory model



© All Rights Reserved

A Starting Example

```
// sequential code
void main ( )
{
    double x[512] ;
    int i ;

    for ( i = 0; i < 512; i++) {
        do_some_work( x[ i ] ) ;
    }
}
```

© All Rights Reserved

* وظيفة الـ Programer يعرف ويتأكد إنه هذا الجزء من الـ Code بزبط
ينكتب كـ Parallel وما في dependency.

A Starting Example

```
// OpenMP code
#include <omp.h>
void main ( )
{
    double x[512] ;
    int i ;

#pragma omp parallel num_thread( 8 )
{
    #pragma omp for
    for ( i = 0; i < 512; i++) {
        do_some_work( x[ i ] ) ;
    }
}
}
```

OpenMP runtime library

This directive means a parallel region is started

This clause means to fork eight threads

اختيارية مش إجبارية لو ما حطيتها في default values

- This directive means the for loop is a *work-sharing* construct
- By default, the iteration space is **evenly** partitioned between the eight threads

بلش الـ Parallel "fork"

→ سكر الـ Parallel "join"

* أنا بس بحكي لـ Compiler عن طريق directives وين في Parallel ووين في Serial ومين Private var. ومين Shared var. وهو لحاله الـ Compiler تبع الـ OpenMP بحول لـ P-thread وبتصرف بالـ low level وهو بقسم الـ chuncks و الـ ط ا و الـ ub وكل هذا.

© All Rights Reserved.

Why OpenMP?

- It is simple!
  - Programmers incrementally add compiler directives to sequential programs
  - Compiler takes care of the ugly details: threads creation and termination, synchronization, data partitioning, … etc
- Widely applicable → « قابل للتطبيق على نطاق واسع »
  - Work with both data-parallel and task-parallel applications
  - OpenMP has compiler directives/clauses for almost everything
- The same source code can be used for both sequential and OpenMP codes
  - Simply ignore directives when compiling as a sequential program

© All Rights Reserved.

CamScanner

Compiler Directives

- Assuming C/C++, all directives must start with *#pragma omp*
- Directives types:
  - #pragma omp parallel → Parallel region directive
  - #pragma omp for
  - #pragma omp section
  - #pragma omp single

    } Work-sharing constructs directives
  - #pragma omp critical
  - #pragma omp atomic

    } Synchronization directives
  - #pragma omp barrier
  - #pragma omp master
  - #pragma omp threadprivate → THREADPRIVATE directive
- Each directive has a set of clauses that define additional attributes (see next slides)

© All Rights Reserved.

Clauses

- Data attribute clauses
  - shared
  - private, lastprivate, firstprivate
  - copyin, copyprivate
- Synchronization clauses
  - nowait
- Work-sharing clauses
  - schedule
  - reduction
  - collapse

© All Rights Reserved.

Runtime Library Routines

- Number of threads: omp_{set,get}_num_threads
  - أعرف رقم الـ thread
  - أعرف أو اتحكم بعدد الـ threads كامل
- Thread ID: omp_get_thread_num
- Wall clock time: omp_get_wtime → execution time مفيد عشان احسب الـ
- Acquire/release lock: omp_{set,unset}_lock
- And more …

© All Rights Reserved.

Environment Variables

- OMP_NUM_THREADS → اتحكم بعدد الـ threads
- OMP_THREAD_LIMIT → Maximum أحط
- OMP_SCHEDULE
- OMP_PROC_BIND → اتحكم واتدخل بين thread و Processor أي يروح على
- And more …

© All Rights Reserved.

CamScanner

* الـ default إنه كل اشي داخل الـ parallel region بكون Sharing بين الكل.

The Parallel Region Directive

- Programmers use the "#pragma omp parallel" directive to mark a block of code as a *parallel region*
- At runtime, when the master thread reaches a parallel region, it creates a team of threads that execute this region in parallel
  - i.e., the code of the parallel region is duplicated and all threads will execute this code simultaneously
- The master thread will also be a part of this team (as thread number 0)
- The clauses "shared" and "private" are used to determine if each variable is private or shared inside a parallel region
  - Variables declared inside a parallel region are always *private*
  - Variables that are not specified a "private" or a "shared" clause are, by default, shared

© All Rights Reserved

How Many Threads?

- The number of threads in a parallel region can be specified using any of the following:
  - "num_threads" clause
  - "omp_set_num_threads" library function
  - "OMP_NUM_THREADS" variable
  - Implementation default, which is usually the same number as the number of cores in the system

© All Rights Reserved

CamScanner

* gcc → اذا الـ code بالـ C → this command for c programs with extension .c
* g++ → اذا الـ code بالـ C++ → this command for c++ programs with extension .cpp

The Traditional Hello World Example

```
#include <omp.h>
#include <stdio.h>

void main () {

  int nthreads, tid;

  /* Fork a team of 8 threads with each thread having a private tid variable */
#pragma omp parallel num_threads(8) private(tid)
  {

    /* Obtain and print thread id */
    tid = omp_get_thread_num();
    printf("Hello World from thread = %d\n", tid);

    /* Only master thread does this */
    if (tid == 0)
      {
        nthreads = omp_get_num_threads();
        printf("Number of threads = %d\n", nthreads);
      }

  }  /* All threads join master thread and terminate */
```

بدل lostream اللي كانت بالـ C++

ممكن يرجّع وممكن ما يرجّع

*tid 8 بخزن رقم الـ thread

*nthreads 8 بتخزن عدد الـ threads

بنخلق 8 copies غير اللي معرفة فوق، وما دخلهم فيها.

لو ما حطيتها بخليها نفس عدد الـ cores

كل واحد منهم بعملها

Format for Num

*by default بكونوا shared

بنرجع عدد الـ threads كامل.

"حيطبع 9 جمل"

②

Compiling and Running Hello World Example

- Assuming we have GNU GCC compiler, we can compile the Hello World OpenMP program on a linux machine using the following command

  gcc –fopenmp hello_world.c –o hello.out

- To run the program, we use the command

  ./hello.out

- You can set the number of threads using the OMP_NUM_THREADS environment variable, as follows:

عشان ما أضل أعمل recompile لعدد الـ threads.

bash shell: export OMP_NUM_THREADS=8

ما تحط spaces

csh/tsch shell: setenv OMP_NUM_THREADS "8"

© All Rights Reserved.

→ default.bash (ubuntu)

①

commands

- sudo apt-get update → بتتأكد إنه gcc و openMP نازل
- gcc -fopenmp hello-word.c → a.out بخلق file باسم
output executable file will be stored in a file with ل
the name a.out ← أنا سميته .
- gcc -fopen mp hello-word.c -o hello.out
- ./hello.out → run لنعمل

* output of code :

بطبع 8 جمل ← ترتيبها غير مضمون لكن لازم إنه
عبارة thread 0 تطلع قبل الـ Number of threads .

→ إحدى الحلول :

```
hello world from thread=5
hello world from thread=0
hello world from thread=1
hello world from thread=2
Number of threads = 8
hello word from thread=7
hello word from thread=6
hello word from thread=4
" " " " =3
```

Work-Sharing Constructs

- Sub-regions where the work is **partitioned** among the participating threads **inside** a parallel region
- OpenMP has three types of directives for work sharing constructs:
  1. #pragma omp for
     - Used for paralell *for* loops
     - Work-sharing happens by partitioning the iteration space of the *for* loop
     - Loop-level parallelism works well with data parallelism
  2. #pragma omp section
     - Work-sharing happens by breaking work into separate, discrete sections
     - Each section is executed by a thread
     - OpenMP sections work well with function parallelism
  3. #pragma omp single
     - Only one thread (can be any thread in the team) will execute the code

© All Rights Reserved.

Parallel *for* Loops

(data Parallel Program)

```
#pragma omp parallel
  {
#pragma omp for private(i) schedule(static)
  for ( i=1; i<=198; i++ )
    C[i] = (A[i] + B[i])/2.0 ;

#pragma omp for private(i) schedule(static)
  for( i=0; i<=199; i++ )
    A[i] = sqrt(C[i]) ;
  }
```

- By default, arrays A, B, and C are shared
- By default, there is an **implicit barrier** at the end of each parallel *for* loop

This means use *static* scheduling when dividing iterations among threads (see next slide)

© All Rights Reserved.

The Schedule Clause

- Specify how iterations of a parallel *for* loop is partitioned
- schedule (*static*, [chunk])
  - Loop iterations are divided into pieces of size chunk and then **statically** assigned to threads ***in round robin fashion***
  - If chunk size is not specified, the iterations are evenly (if possible) divided contiguously among the threads (called *block partitioning*)
  - Special case: if chunk size is 1, then the partitioning is called cyclic
- schedule (*dynamic*, [chunk])
  - Loop iterations are divided into pieces of size chunk, and dynamically scheduled among the threads
  - When a thread finishes one chunk, it grabs another
  - If chunk size is not specified, then by default, it is equal to 1

© All Rights Reserved

Static vs Dynamic Scheduling

- Static scheduling has lower overhead but is more prone to load imbalance
- Dynamic scheduling has higher overhead but less prone to load imbalance
- So how do we decide which scheduling scheme is better?

© All Rights Reserved.

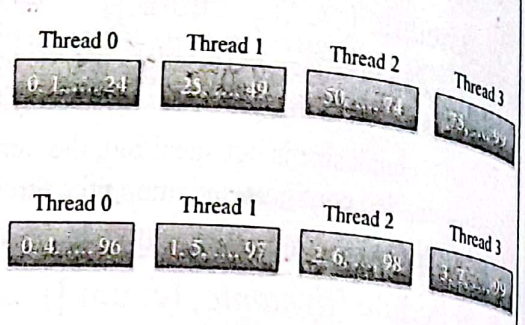
Iteration Scheduling Affects Thread-Data Distribution

- Assuming the number of threads is 4, let us derive the distribution of *shared array A* on these four threads

①
```
#pragma omp for schedule(static)
for ( i = 0; i < 100; i++ )
      A[ i ] = ...
```

②
```
#pragma omp for schedule(static, 1)
for ( i = 0; i < 100; i++ )
      A[ i ] = ...
```

③
```
#pragma omp for schedule(dynamic)
for ( i = 0; i < 100; i++ )
      A[ i ] = ...
```



Only known at runtime

ال default للchunk بكون 1

غير ال Static

In general, data-thread distribution for a particular array is a function of iteration scheduling type + the array subscript function

© All Rights Reserved.

* ال Parallisation يفضل ينعمل لل outer مش ال inner loop عشان نخفف barrier ، بنقدر نعمل Paralize الهم الاثنين (نحكي عنها لقدام) حاليا سب بنعمل لوحدة منهم .

Exercise

- Assuming the number of threads is 4, describe the thread-data distribution of arrays A and B in the following parallel loops

1
```
#pragma omp for schedule(static)
for ( i = 0; i < 100; i++ )
      A[ i ] = B[ i + 1 ];
```

2
```
#pragma omp for schedule(static)
for ( i = 0; i < 20; i++ )
      A[ 2 * i ] = B[ 2 * i + 1 ];
```

3
```
#pragma omp for schedule(static)
for ( i = 99; i >= 0; i-- )
      A[ i ] = B[ i ];
```

Backword loop

بمشي عليه كما هو ما بتحول.

1 barrier

4
```
#pragma omp for schedule(static)
for ( i = 0; i < 100; i++ )
  for ( j = 0; j < 100; j++ )
    A[ i ][ j ] = B [ j ][ i ];
```

بقرأ ك column

بقرأ ك rows

أفضل عشان ال locality عشان ال Mem هيك بتخزن

5
```
for ( i = 0; i < 100; i++ ) {
#pragma omp for schedule(static)
  for ( j = 0; j < 100; j++ )
    A[ i ][ j ] = B [ j ][ i ];
}
```

serial

Parallel

2 barrier

© All Rights Reserved.

The Nowait Clause

- Remove implicit barrier that is present at the end of parallel *for* loops
- Improve performance because it removes unnecessary synchronization
- Can cause race conditions when used incorrectly

```
#pragma omp parallel
  {
#pragma omp for private(i) schedule(static) nowait
    for ( i=0; i<200; i++ )
      C[i] = (A[i] + B[i])/2.0 ;

#pragma omp for private(i) schedule(static)
    for( i=0; i<200; i++ )
      A[i] = sqrt(C[i]) ;

  }
```

← بقلل من الـ wait time لأنه بنقل الـ barrier

→ كشان أشيل الـ barrier بشيله لما أكون متأكدة

We can eliminate implicit barrier here because the same threads in both parallel loops produce and consume the same data (i.e., there is no cross-thread dependencies)

© All Rights Reserved.

Which Nowait Can Cause Race Conditions?

```
#pragma omp parallel
  {
#pragma omp for private(i) schedule(static, 1) nowait
    for ( i=0; i<200; i++ )
      C[i] = (A[i] + B[i])/2.0 ;

#pragma omp for private(i) schedule(static, 1)
    for( i=0; i<200; i++ )
      A[i] = sqrt(C[i]) ;

  }
```

✓ No race conditions

```
#pragma omp parallel
  {
#pragma omp for private(i) schedule(static) nowait
    for ( i=0; i<200; i++ )
      C[i] = (A[i] + B[i])/2.0 ;

#pragma omp for private(i) schedule(static)
    for( i=0; i<199; i++ )
      A[i] = sqrt(C[i]) ;

  }
```

✗ A race condition might occur

لأنه الـ range اختلفوا.

```
#pragma omp parallel
  {
#pragma omp for private(i) schedule(dynamic) nowait
    for ( i=0; i<200; i++ )
      C[i] = (A[i] + B[i])/2.0 ;

#pragma omp for private(i) schedule(dynamic)
    for( i=0; i<200; i++ )
      A[i] = sqrt(C[i]) ;

  }
```

✗ A race condition might occur

لأنه dynamic ما بعرف كل حد شو أخذ فما عندي ضمانة إنه ما في barrier.

© All Rights Reserved.

CamScanner

Final global answer ← يعني يحسب الـ

The Reduction Clause

- The REDUCTION clause performs a reduction on the variables that appear in its list
  - A private copy for each list variable is created for each thread
  - At the end of the reduction, the reduction variable is applied to all private copies of the shared variable
  - The final result is written to the global shared variable
- Reduction operations include +, -, *, /, &, |, ^
- For example, the below parallel loop computes the summation in parallel and puts the final result in the shared variable "*sum*"

```
#pragma omp for private(i) reduction(+:sum)
  for(i=0;i<SIZE;i++)
    sum = sum + A[i];
```

© All Rights Reserved.

The Reduction Clause (cont.)

1. Reduction clause only supports scalar reduction
   - Variables in the reduction list cannot be arrays or structure data types
2. Reduction clause works for both integer and real numbers
   - However, reduction operations may not be associative for real numbers
3. Naturally, reduction clauses are only legal for shared variables
4. A reduction variable must be used only in the statement where the reduction is applied

© All Rights Reserved.

The SINGLE Directive

- The SINGLE directive specifies that the enclosed code is to be executed by only one thread in the team
- Any thread in the team can execute the code enclosed by the SINGLE directive
- By default, there is an implicit barrier at the end of the enclosed code by the SINGLE directive
  - The programmer may remove this barrier using the NOWAIT clause
- Single directives are useful when initializing global variables or dealing with I/O operations

© All Rights Reserved.

Synchronization Directives

- OpenMP provides a set of directives to perform synchronization operations
- Examples:
  - #pragma omp master
    - Specifies a region that is to be executed only by the master thread
    - All other threads on the team skip this section of code
    - There is no implied barrier associated with this directive
  - #pragma omp critical
    - Specifies a region of code that must be executed by only one thread at a time
    - There is no implied barrier associated with this directive
  - #pragma omp atomic
    - Similar to CRITICAL directive but can only applies to one-statement regions
  - #pragma omp barrier
    - All threads must wait till all other threads reach this barrier
    - All threads then resume executing in parallel the code that follows the barrier

data level parallism

Example: Parallel Sum in OpenMP

```
#include <omp.h>
#include <stdio.h>
#define SIZE 1000

void main( ){

  int i, nthreads;
  int sum;
  int A[SIZE];
  double exec_time;

  for(i=0; i<SIZE; i++)
      A[i] = i%100;

  exec_time = omp_get_wtime();

#pragma omp parallel
  {

#pragma omp single
    sum = 0;

#pragma omp for private(i) reduction(+:sum)
  for(i=0;i<SIZE;i++)
    sum = sum + A[i];

#pragma omp master
  printf("number of threads=%d\n",omp_get_num_threads());

  }

  exec_time = omp_get_wtime() - exec_time;
  printf("sum=%d\n", sum);
  printf("execution time=%f\n",exec_time);

}
```

by default « Shared »

بحسب الوقت الحالي كـ double NUM (قبل)

* single, for, sections بيجي barrier وراهم

* critical, Master ما بيجي وراهم barrier

ما بنشيل فيها الـ barrier

Exercise: Can a nowait clause be added here?

لنتأكد إنه بالبداية Sum = 0

بكون Parallel

ما بيجي وراها barrier by default

الوقت (بعد) وطرحه من اللي قبل.

بطلع بالـ Seconds

© All Rights Reserved.

Example: Dot Product in OpenMP

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 256

void main( ){

  int i;
  float x[N],y[N];
  float dot_product;
  double exec_time;

  for(i=0; i<N; i++){
    x[i] = ((float)rand()/(float)(RAND_MAX));
    y[i] = ((float)rand()/(float)(RAND_MAX));
  }

  exec_time = omp_get_wtime();

#pragma omp parallel shared(dot_product) private(i)
  {

#pragma omp single
    dot_product= 0.0;

#pragma omp for reduction(+:dot_product)
    for(i=0;i<N;i++)
      dot_product = dot_product + x[i] * y[i];

#pragma omp master
  printf("number of threads=%d\n",omp_get_num_threads());

  }

  exec_time = omp_get_wtime() - exec_time;
  printf("dot product is %f\n", dot_product);
  printf("execution time=%f\n",exec_time);

}
```

* أي Variable بتعرف جوا الـ Parallel region بكون by default = Private

→ Parallel بكون

© All Rights Reserved.

Example: Histogram in OpenMP

global Reduction في Scalar مش فما بنكم الـ OpenMP

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define SIZE 20
int find_bin( int a);

void main( ){

  int i,j;
  double A[SIZE];
  int bin[5], my_bin[5];
  double exec_time;

  for(i=0; i<SIZE; i++)
    A[i] = ((float)rand()/(float)(RAND_MAX)) * 5;

  exec_time = omp_get_wtime();

#pragma omp parallel private(my_bin, i, j)
  {

#pragma omp single
    for(i=0;i<5;i++)
      bin[i]=0;

    for(i=0;i<5;i++)
      my_bin[i]=0;

#pragma omp for
    for(i=0;i<SIZE;i++){
      j = find_bin(A[i]);
      my_bin[j]++ ;
    }

#pragma omp critical
    for(i=0;i<5;i++)
      bin[i] = bin[i] + my_bin[i];

#pragma omp master
  printf("number of threads=%d\n",omp_get_num_threads());

  }

  exec_time = omp_get_wtime() - exec_time;
  for(i=0;i<5;i++)
    printf("bin[%d] = %d\n", i, bin[i]);
  printf("execution time=%f\n",exec_time);
}
```

Initialization for global variable
race condition كلهم يدخلوا في نفس الوقت → ناقص

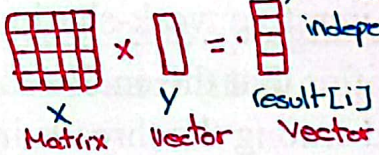
Parallel بيشتغل

Critical ولكن جواه فقط كباقي وحدة وهون هنكرر كباقي وحدة فيه .

*Exercise: Can we use atomic directive instead of critical directive?

© All Rights Reserved.

Example: Matrix-Vector Product in OpenMP



بحسب كل وحدة لحالها independently

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 128
#define M 256

void main( ){

  int i,j;
  float x[N][M], y[M];
  float result[N];
  double exec_time;

  for(i=0; i<N; i++)
    for(j=0; j<M; j++)
      x[i][j] = ((float)rand()/(float)(RAND_MAX));

  for(i=0; i<M; i++)
    y[i] = ((float)rand()/(float)(RAND_MAX));


  exec_time = omp_get_wtime();

#pragma omp parallel private(i,j)
  {

#pragma omp single
    for(i=0; i<N; i++)
      result[i] = 0.0;

#pragma omp for
    for(i=0;i<N;i++)
      for(j=0;j<M; j++)
        result[i] = result[i] + x[i][j] * y[j];

#pragma omp master
  printf("number of threads=%d\n",omp_get_num_threads());

  }

  exec_time = omp_get_wtime() - exec_time;
  printf("execution time=%f\n",exec_time);
}
```

بدل for بقدر أخليه Single + nowait لو كان for بقدر أخليه كمان الـ ranges نفسها بس لو خلي Single ما بزبط لإني لازم استنناها لتخلص كمان اللي تحت بيستخدموها

Parallel → Social → تأكدي

© All Rights Reserved.

CamScanner

A Nice Shortcut

- The special directive "#pragma omp parallel loop" can be used to start a parallel region that consists only of a parallel loop

```
#pragma omp parallel
  {
#pragma omp for private(i) schedule(static,4)
    for ( i=0; i<200; i++ )
      C[i] = (A[i] + B[i])/2.0 ;
  }
```

لما يكون عندي Parallel region وجواها بس وحدة for فقط بكتبها هيك

Can be rewritten

```
#pragma omp parallel for private(i) schedule(static,4)
  for ( i=0; i<200; i++ )
    C[i] = (A[i] + B[i])/2.0 ;
```

© All Rights Reserved.

The SECTIONS Directive

- A non-iterative work-sharing construct
- It specifies that the enclosed section(s) of code are to be divided among the threads in the team
- Each SECTION is executed once by a thread in the team
- Different sections may be executed by different threads
- It is possible for a thread to execute more than one section if it is quick enough and the implementation permits such
- There is an implied barrier at the end of a SECTIONS directive, unless the *nowait* clause is used

© All Rights Reserved.

CamScanner

The SECTION Directive (cont.)

- Consider the following example

```
#pragma omp parallel shared(A, B, C, D) private(i)
  {

#pragma omp sections
    {
      #pragma omp section
      for(i=0; i<N; i++)
        c[i] = a[i] + b[i];

      #pragma omp section
      for(i=0; i<N; i++)
        d[i] = a[i] - b[i];

    } /* end of sections */

  } /* end of parallel region */
```

هون بستعمل Task level parallism

لو حطيت nowait هون بتبقى الـ barrier

بيفهم انه هنزول الـ 2 codes بزبط بنعملهم execute in parallel

هون بكون barrier

* عدد الـ Section ما دخله بعدد الـ thread.

* أي thread ما بياخذ أكثر من Section ، بس واحد بياخذ ولما يصير idel بكون جاهز ياخذ Section ثانية وهكذا. ما بياخذ أكثر من Section بنفس الوقت.

© All Rights Reserved.

* كل Section لأي thread بروح ما بنعرف بصير dynamically.

Summary

- OpenMP is a directive-based, easy-to-use programming model for shared-memory architectures (including both UMA and NUMA)
- OpenMP relies on both a compiler (to interpret directives and clauses) and a runtime library (to invoke special routines)
- OpenMP uses the fork-join model of parallel execution
- OpenMP is implemented in Fortran, C and C++
- Using work-sharing constructs, OpenMP can exploit both types: data-level and task-level parallelism in parallel regions

© All Rights Reserved.

CamScanner

Exercises

1. Write a parallel OpenMP program that counts how many times a negative number is found in a list of 512 randomly generated floating numbers picked from the range [-1:+1]

Serial
لحلها

2. Write a parallel OpenMP program that computes $C = A \times B$, where A, B, C are 2D matrices of size $100x100$. You may initialize arrays A and B to random floating-point numbers in the range [1:100]

3 loops (nested)

3. Write a parallel OpenMP program that computes the XOR between two n-bit binary numbers A and B, where n, A, and B are read from an input text file.

© All Rights Reserved.

* الـ XOR بنعمل in parallel * ← independent

* الـ addition هي اللي بتعتمد عبعض فصعب فيها الـ Parallel *

Loop-Level Parallelism in OpenMP

Instructor: Dr. Fahed Jubair

Computer Engineering Department

University of Jordan

CamScanner

like Matrix Multiplication with Vector

Loop-Level Parallelism

- Tremendous computational algorithms nowadays spend most of their computation in loops
- Therefore, when parallelizing a sequential algorithm, programmers often find themselves need to parallelize loops
- Fundamental questions:
  1. How to determine if a loop is serial or parallel?
  2. How to convert a serial loop into a parallel loop without affecting correctness?
- In this lecture, we will learn how to utilizes loop-level parallelism using OpenMP when parallelizing a sequential program

© All Rights Reserved

Motivating Question

*Parallel loops & carry

ال loops اللي مافي عنده أي loop Independence

- Can the loops on the right be run in parallel?
  - i.e., can different threads run different iterations in parallel?

هل بقدر أعمل outoforder exec. ؟

```
for (i = 1; i < N; i++) {
  a[i] = b[i];
  c[i] = a[i - 1];
}
```

→ Serial

```
for (i = 1; i < N; i++) {
  a[i] = b[i];
  c[i] = a[i] + b[i - 1];
}
```

- What needs to be true for a loop to be parallelizable?
  1. Iterations cannot interfere with each other
  2. i.e., there is no dependences between iterations

Parallel ليكون

© All Rights Reserved

CamScanner

عشان أعرف لو هو Parallel ولا Serial
بصير أفرض وأجرب وأحلل النتائج لأوصل الحل.

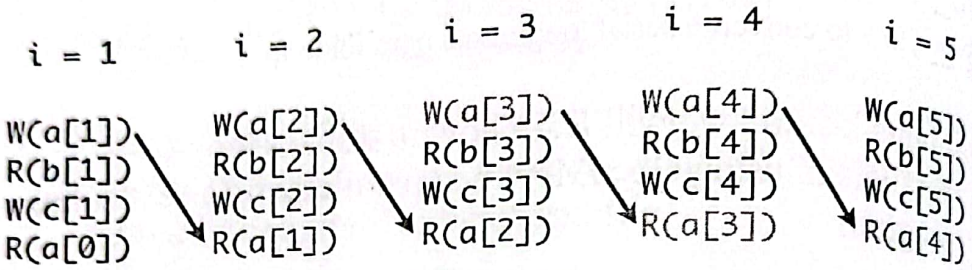
Flow Dependences

- Definition: A **flow dependence** occurs when one iteration writes a memory location that a later iteration reads

```
for (i = 1; i < N; i++) {
  a[i] = b[i];
  c[i] = a[i - 1];
}
```

* لأنه فيه Serial dependence

i = 1	i = 2	i = 3	i = 4	i = 5
W(a[1])	W(a[2])	W(a[3])	W(a[4])	W(a[5])
R(b[1])	R(b[2])	R(b[3])	R(b[4])	R(b[5])
W(c[1])	W(c[2])	W(c[3])	W(c[4])	W(c[5])
R(a[0])	R(a[1])	R(a[2])	R(a[3])	R(a[4])



© All Rights Reserved

Read after write بينهم
= flow dependence

Other Kind of Dependences

- *Anti dependence* – When an iteration *reads* a location that a later iteration *writes* (why is this a problem?) → بزي زي فوق

write after Read

```
for (i = 1; i < N; i++) {
  a[i - 1] = b[i];
  c[i] = a[i];
}
```

- *Output dependence* – When an iteration *writes* a location that a later iteration *writes* (why is this a problem?) → بزي زي فوق

Write After write

```
for (i = 1; i < N; i++) {
  a[i] = b[i];
  a[i + 1] = c[i];
}
```

© All Rights Reserved

* ال RAW ، WAR ، WAW ممنوع يكونوا إذا ال Code كان Parallel.

* الـ dependence بنفس الـ iteration عادي ما بأثر علي ، احنا بنحكي بين iterations مختلفة .

Loop-Carried Dependence

- We can run a loop in parallel if and only if there is no *loop-carried dependences* inside the loop
- **Definition**: A loop has a *loop-carried dependence* if a dependency (whether it is flow, anti- or output dependence) across two iterations is present
- **Definition**: A loop is *parallel* if it has no loop-carried dependence (otherwise, the loop is *serial*)
- To determine if a loop is parallel or serial, we will learn:
  1. How to find and represent loop-carried dependences in loops
  2. How to use this representation to determine if a loop is parallel or serial

* للتحديد: طريقة التجربة زي ما أخذنا بالسلايدز اللي ورا أو Methods *

© All Rights Reserved

Terminology

- Dependences can only go forward in time: always from an earlier iteration to a later iteration.
- Dependence *source* is the earlier statement (the statement at the tail of the dependence arrow)
- Dependence *sink* is the later statement (the statement at the head of the dependence arrow)

```
for (i = 1; i < N; i++) {
  a[i] = b[i];
  c[i] = a[i - 1];
}
```

source

sink → داخل عليها السهم

طالع منه السهم

i = 1	i = 2
W(a[1])	W(a[2])
R(b[1])	R(b[2])
W(c[1])	W(c[2])
R(a[0])	R(a[1])

Source

Sink

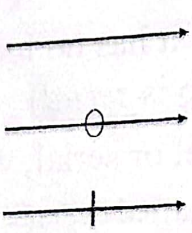
© All Rights Reserved

CamScanner

Representing Dependences

- We use Loop-carried Dependence Graph to abstractly represent dependencies inside loops
  - Represent iterations as nodes in the graph
  - Draw arrows from sources to sinks to represent loop-carried dependencies

إشارات
كشان التميز



Flow dependence

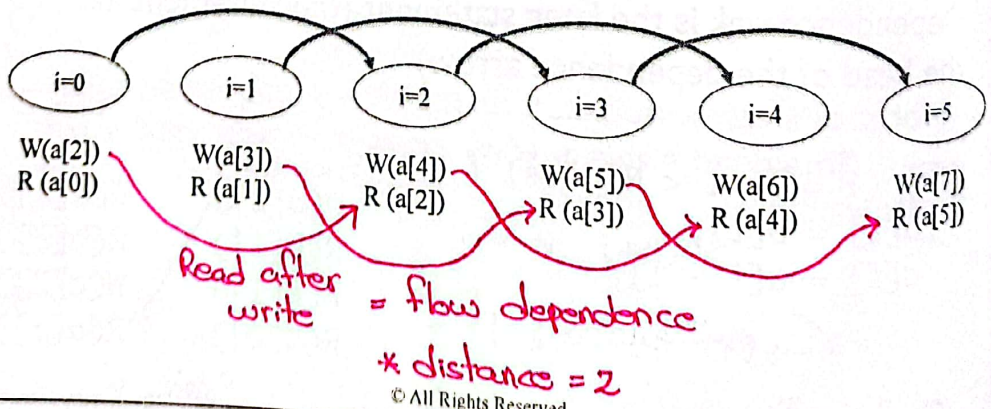
Output dependence

Anti dependence

© All Rights Reserved

1D Loop-Carried Dependence Graph Example 1

```
for (i = 0; i < N; i++) {
  a[i + 2] = a[i]
}
```



Read after write = flow dependence

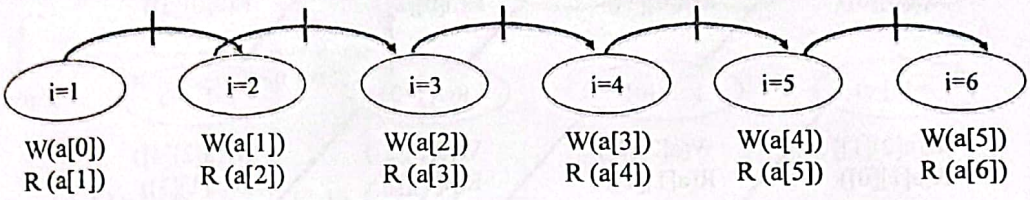
* distance = 2

© All Rights Reserved

1D Loop-Carried Dependence Graph Example 2

```
for (i = 1; i < N; i++) {
  a[i - 1] = b[i];
  c[i] = a[i];
}
```

b,c مابعملوا مشاكل
بنركز a



" write after Read "

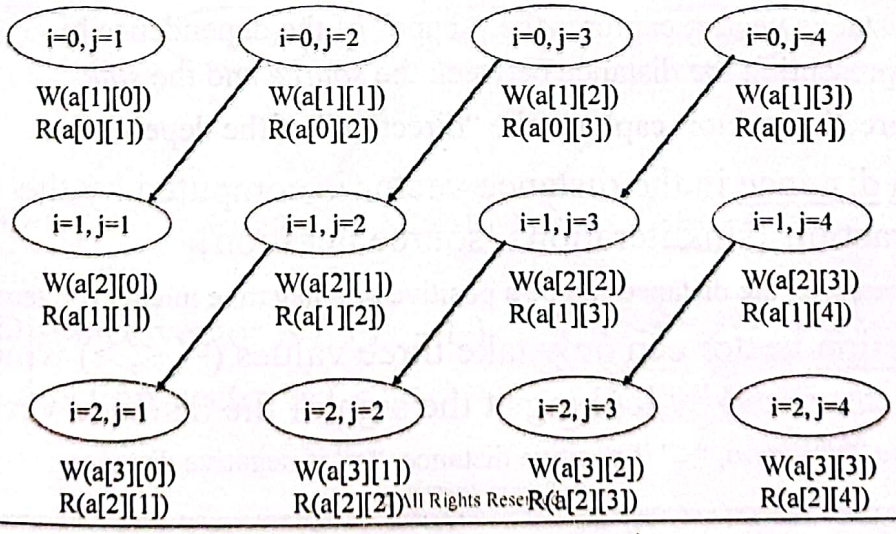
© All Rights Reserved

2D Loop-Carried Dependence Graph Example 1

Serial
لإنه فيه
dependance

```
for (i=0; i<N; i++)
  for(j=1; j<N; j++)
    a [i + 1][j – 1] = a [ i ][ j ] + 1
```

ال outer بتنفذ أول بعدين
ال inner

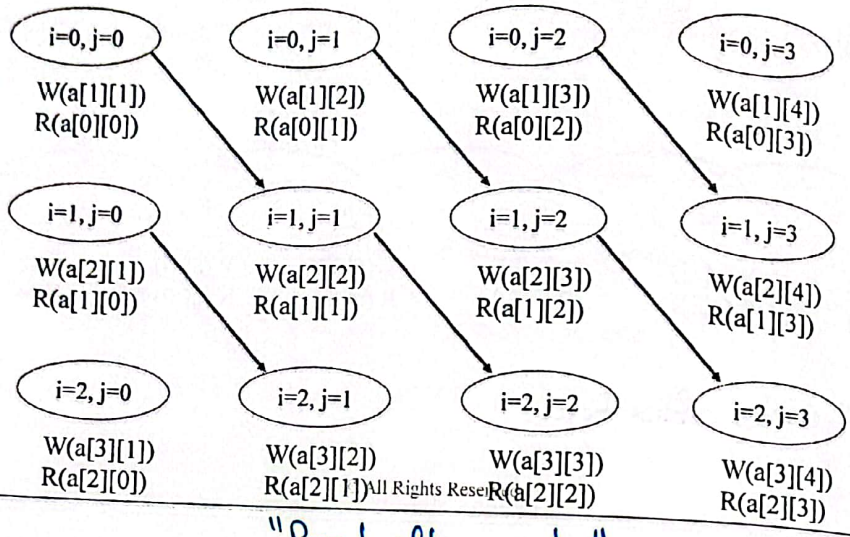


All Rights Reserved

" read after write "
" Same distance "

2D Loop-Carried Dependence Graph Example 2

```
for (i=0; i<N; i++)
  for(j=0; j<N; j++)
    a [i + 1][j + 1] = a [ i ][ j ] + 1
```



All Rights Reserved

"Read after write"

Distance and Direction Vectors

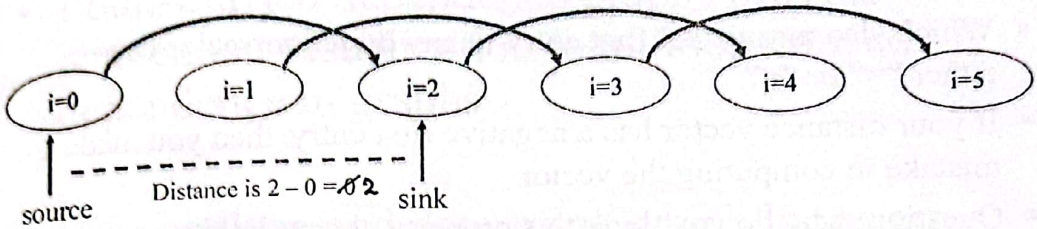
- We can use a more compact from to capture the dependence information in loop-carried dependence graphs using the following two vectors:
  - Distance vector: captures the "shape" of the dependence by representing the distance between the *source* and the *sink*
  - Direction vector: captures the "direction" of the dependence
- Each distance in the distance vector is computed by the subtraction {sink iteration – source iteration}
  - Therefore, the distance can be a positive or a negative integer (or zero)
- Direction vector can only take three values (=, <, >) which are determined by looking at the sign of the distance vector
  - Use "=" if zero, "<" if positive distance, ">" if negative distance

© All Rights Reserved

CamScanner

1D Distance and Direction Vectors

```
for (i = 0; i < N; i++) {
  a[i + 2] = a[i]
}
```



- Distance vector is (+2)
- Direction vector is (<)

Forward
بتروح من صغير لكبير

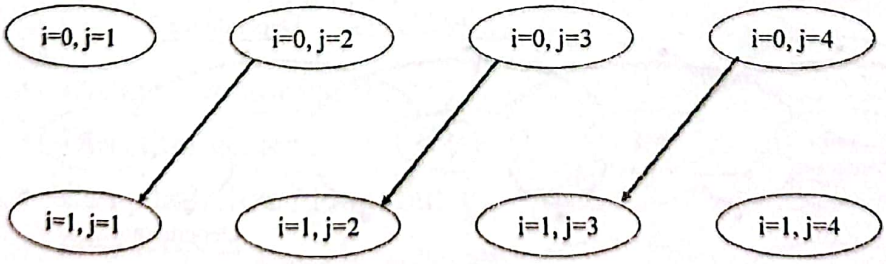
Note that "<" is used to indicate that i1<i2, where i1 is the source iteration and i2 is the sink iteration

© All Rights Reserved

2D Distance and Direction Vectors

```
for (i=0; i<N; i++)
  for(j=1; j<N; j++)
    a [i + 1][j – 1] = a [ i ][ j ] + 1
```

* دائما أول واحد بكون أصغر (رقم موجب موسالب) ممكن يكون صفر بنشوف لقدام.



- Distance vector is (+1 , –1)
- Direction vector is (< , >)

الـ i دائما أصغر بتكون لإنه دائما بتنقل من i صغيرة لكبيرة لكن الـ j ممكن تصدف تروح من أكبر لأصغر.

© All Rights Reserved

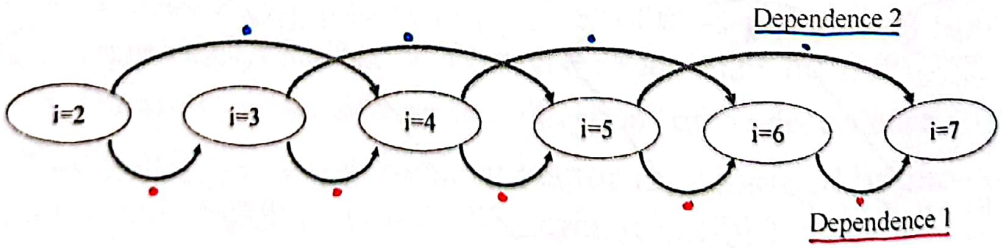
Important Property

- First entry in any distance vector will always be positive (or zero)
  - Which also means that first entry in any direction vector can be either "=" or "<"
  - If your distance vector has a negative first entry, then you made a mistake in computing the vector
  - Question: why do you think this property always hold?

© All Rights Reserved

Multiple Dependencies Represented Separately

```
fibo[ 0 ] = fibo[ 1 ] = 1;
for (i = 2; i < n; i++)
   fibo[ i ] = fibo[ i – 1 ] + fibo[ i – 2 ];
```



- Distance vector1 is (+1) , Distance vector2 is (+2)
- Direction vector1 is (<) , Direction vector2 is (<)

© All Rights Reserved

CamScanner

Finding Loop-Carried Dependences

- Loop-Carried dependence graphs provide convenient visualization to all loop-carried dependence in graphs
- Help analyze and determine which loops are parallel. (Sometimes)

© All Rights Reserved

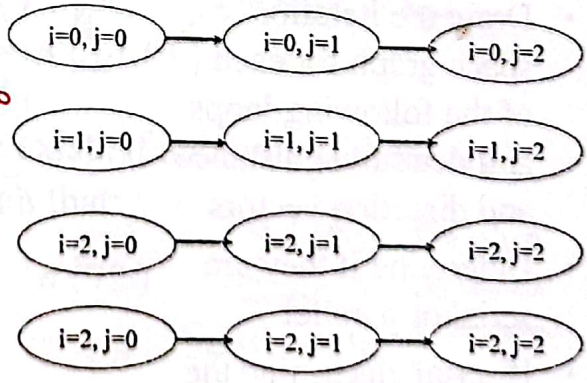
Example 1

```
for (i=0; i<N; i++)
  for(j=0; j<N; j++)
    a [ i ][j + 1] = a [ i ][ j ] + 1
```

مشكلتنا فيه

- Distance vector is (0 , +1)
- Direction vector is (= , <)
- Can parallelize *i* loop, but not *j* loop

يعني الـ dependance بنفس المكان (Parallel)



لو كان = ، ٥ دائما بكون هيك Parallel

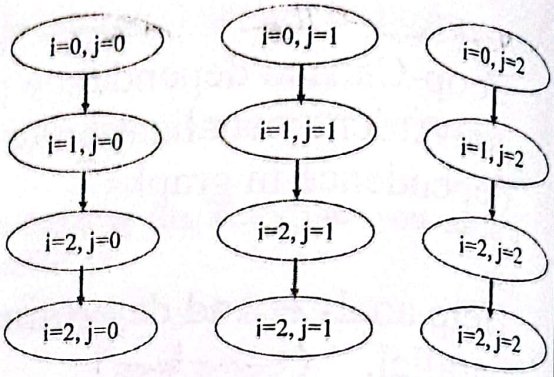
```
#pragma omp parallel for private (i,j)
for (i=0; i<N; i++)
  for(j=0; j<N; j++)
    a [ i ][j + 1] = a [ i ][ j ] + 1
```

© All Rights Reserved

Example 2

```
for (i=0; i<N; i++)
   for(j=0; j<N; j++)
      a [i + 1 ][ j ] = a [ i ][ j ] + 1
```

- Distance vector is (+1 , 0)
- Direction vector is (< , =)
- Can parallelize *j* loop, but not *i* loop




```
for (i=0; i<N; i++)
   #pragma omp parallel for private (j)
   for(j=0; j<N; j++)
      a [ i + 1][ j ] = a [ i ][ j ] + 1
```

هون لل j مش لل i
عملنا Parallel region

© All Rights Reserved

Exercise

- Draw the iteration space graph for each of the following loops and determine distance and direction vectors
- Determine if they are serial or parallel
- If serial, determine the type of loop-carried dependencies

1

```
for ( i=1; i < n; i++) {
      x = x + 1;
      b [ i ] = a [ i ] * x ;
}
```

→ Read after write + WAW + WAR
بس خذ ال RAW لأنه هي الأهم

2

```
for ( i=1; i < n; i++)
  for ( j=1; j < n; j++)
     a [ i ][ j ] = a [ i + 1 ][ j ];
```

3

```
for ( i=1; i < n; i++)
  for ( j=1; j < n; j++)
     a [ i ][ j +1 ] = a [ i ][ j - 1 ]
                       * a [ i ][ j ];
```

© All Rights Reserved

CamScanner

Let us Make Things More Interesting

- Determine if the following loop is parallel or serial:

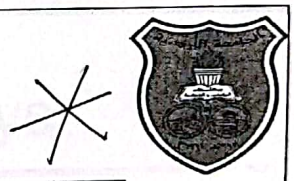
```
for ( i=0; i <= n; i++)
    A [ 2 * i ] = A [ 3 * i + 1 ] ;
```

- More complex array subscripts make using the loop-carried dependence graph method to find dependencies more difficult!

© All Rights Reserved

int solution → complex
floating point Solution → easy

Problem Formulation



```
for ( i=0; i <= n; i++)
    A [ f( i ) ] = ...
    ...   = A [ g( i ) ]
```

function of write

function of read

- A loop-carried dependence exists if and only if there are two iterations $i1$ and $i2$ such that:

$$f(i1) = g(i2)$$

$$0 \le i1, i2 \le n$$

$i1 \neq i2$

* إذا ما في Solution معناها هذا ال Code Parallel أما لو في Solution معناها Serial

- If $i1 < i2$, then it is a flow dependence → Read after write
- If $i2 < i1$, then it is an anti dependence → write after read

© All Rights Reserved

CamScanner

The Data Dependence Test

- The previous slide show that we can find loop-carried dependencies by solving the below system of equations and inequalities
- Hence, to show that the below loop is parallel, we need to show that this system of equations and inequalities **has no integer solutions**
  - i.e., there is no $i1$ and $i2$ within $[0:n]$ such that $f(i1) = g(i2)$

```
for ( i=0; i <= n; i++)
    A [ f( i ) ] = ...
    ...   = A [ g( i ) ]
```

Data dependence test ⟹

$$f(i1) = g(i2)$$
$$0 \le i1, i2 \le n$$

© All Rights Reserved

Example

```
for ( i=0; i <= n; i++)
    A [ 2 * i ] = A [ 3 * i + 1 ] ;
```

- The loop is parallel if there is no solution for the following system of equations and inequalities:

$$2 * i1 = 3 * i2 + 1$$
$$0 \le i1, i2 \le n$$
$$i1 \neq i2$$

© All Rights Reserved

CamScanner

Loop Normalization

- Loop normalization allows us to generalize the aforementioned data dependence test for any loop

i:0 = lower = 0
i++ = Stride = 1

لازم أتأكد من ال Normalize ال loop

```
for (i = L; i < U; i += S)
  ... a[i] ...
```

Normalize

```
for (i = 0; i < (U - L)/S; i += 1)
  ... a[S*i + L] ...
```

© All Rights Reserved

Multi-Dimensional Arrays

```
for ( i=0; i <= n; i++)
  for ( j=0; j <= m; j++)
    A [ f1( i ) ] [ f2( j ) ] = ...
      ...      = A [ g1( i ) ] [ g2( j ) ]
```

- The loop is parallel if there is no solution for the following system of equations and inequalities:

المعادلات اللي بينن وجود dependence

$$f_1(i1) = g_1(i2)$$

$$f_2(j1) = g_2(j2)$$

$$0 \le i1, i2 \le n$$

$$0 \le j1, j2 \le m$$

© All Rights Reserved

CamScanner

Exact Solution is NP-Complete

- It has been shown that providing an exact solution, in general, to the data dependence test is NP-complete
- Therefore, we rely on approximate solutions that are accurate for the common case but may have *false positives*
- If a test concludes that a dependence exists, while in fact it does not, then we call this result is a false positive
  - It does not affect correctness to claim a loop is serial while it is actually parallel
- However, a data dependence test should never produce a *false negative*
  - Claiming a loop is parallel while it is in fact serial is incorrect

© All Rights Reserved

The GCD Test

- One simple way to perform a data dependence test is to rewrite the it as *Diophantine equations* and perform the *GCD test*
- Our question: $f(i) = a * i + b, g(i) = c * i + d$

  Does $f(i1) = g(i2)$?
- Rewrite $f(i1) = g(i2) \Rightarrow a * i1 + b = c * i2 + d \Rightarrow a * i1 - c * i2 = d - b \Rightarrow a_1 * i1 + a_2 * i2 = a_3$
- The GCD test:

  A *Diophantine equation* $a_1 * i1 + a_2 * i2 = a_3$ has a solution if and only if $\gcd(a_1, a_2)$ evenly divides a_3

© All Rights Reserved

Generalizing The GCD Test

- The Diophantine equation

  $a_1 * i_1 + a_2 * i_2 + \cdots + an * in = c$ has a solution iff $\gcd(a_1, a_2, \ldots, an)$ evenly divides c

Examples:

15*i +6*j -9*k = 12	has a solution	gcd=3
2*i + 7*j = 3	has a solution	gcd=1
9*i + 3*j + 6*k = 5	has no solution	gcd=3

© All Rights Reserved

Exercise

- Use the GCD test to decide whether each loop in the following is parallel or serial:

1
```
for ( i=0; i < 20; i++)
    A [ 2 * i ] = A [ 4 * i + 1 ] ;
```

2
```
for ( i=0; i < 20; i++)
    A [ 2 * i + 3] = A [ 2 * i ] ;
```

3
```
for ( i=0; i < 20; i++)
    A [ 3 * i ] = A [ 7 * i ] ;
```

© All Rights Reserved

Other Data Dependence Tests

1. GCD test: simple but often inaccurate because it does not account for loop bounds information
2. Banerjee test (Utpal Banerjee): more accurate test that takes loop bounds into consideration
3. Omega test (William Pugh): most accurate test for linear subscripts (uses the "fancy" idea of integer programming)
4. Range test (Blume and Eigenmann): unlike the above tests, this test works with non-linear subscripts

© All Rights Reserved

Quick Recap

- As stated at the beginning of this lecture, there are two fundamental issues programmers deal with when parallelizing a serial program:
  - How to determine if a loop is serial or parallel? (we already covered this question)
  - How to convert a serial loop into a parallel loop without affecting correctness? (our next topic)

© All Rights Reserved

CamScanner

Eliminating Loop-Carried Dependencies

- The best way is to rewrite the algorithm!
  - There are multiple ways to achieve the same outcome
  - Programmers try to rewrite the computation code in the loop so that the outcome is the same but without loop-carried dependencies

© All Rights Reserved

Parallelism Enabling Techniques

- Many techniques have been proposed in the literature that help programmers to perform appropriate algorithmic changes to eliminate loop-carried dependencies
- We cannot cover all techniques, but we will study the following five popular techniques:
  1. Scalar Privatization
  2. Array Privatization
  3. Scalar Reduction
  4. Array Reduction
  5. Induction Variable Substitution
  6. Loop Fission

© All Rights Reserved

CamScanner

*Temporary Variable: * متغير بكون مهم بس داخل نطاق ال iteration اللي فيه

Private Scalar Variables

- **Definition**: a scalar variable inside a loop is *private* if it is written before it is read in every iteration
- For example, x and y are private in the below loop but w and z are not

Temporary

Not Temporary

```
for ( i = 1; i < n; i++) {
    x = i * 2;
    y = w * x;
    z = z + w;
}
```

* كل Temporary بكون فيها WAW لذلك بنعمل Pragma مشان كل iteration تاخذ ال Variable بشكل خاص إلها

© All Rights Reserved

Applying Scalar Privatization

- The idea is simple: take advantage of *private* clause in OpenMP to declare private scalars as private
- By doing so, each thread has a private copy of these variables
  - Loop-carried **anti- and output dependences** will be broken
  - The penalty is that more storage is needed

```
for ( i = 1; i < n; i++){
    x = i * 2;
    A[x] = A[x] * 3;
}
```

privatization ⇒

```
#pragma omp parallel for private(x)
for ( i = 1; i < n; i++){
    x = i * 2;
    A[x] = A[x] * 3;
}
```

© All Rights Reserved

CamScanner

Complications

- Consider the **correctness** of applying privatization to the below loop

```
#pragma omp parallel for private(x)
for ( i = 1; i < n; i++){
   x = f(i);
   A[x] = A[x] * 3;
}

y = x * x;
```

This x is equal to the x computed by the last iteration

هيك الـ code خاطئ لإنه آخر iteration ماحتاخذ قيمة X لإنه X=f(i) ← Private فلما تكون هيك قيمتها بتكون داخل نطاق الـ Parallel فقط.

- Privatizing scalar x will **cause an error** because the last value of x will not be seen by the code following the parallel loop
- To solve this problem, OpenMP provides **LASTPRIVATE** clause

© All Rights Reserved

"الحل"

LASTPRIVATE Clause

- Behaves exactly like PRIVATE clause, except that the copy computed by the last iteration is copied to the original variable upon exiting the loop

```
#pragma omp parallel for lastprivate(x)
for ( i = 1; i < n; i++){
   x = f(i);
   A[x] = A[x] * 3;
}

y = x * x;
```

هيك الـ OpenMP بتقوم إنه يحط بالـ X قيمة آخر iteration

bringing in values from the outside context into the parallel region.

"بيجي بالامتحان ابحثي كنه"

* **Exercise**: there is also **FIRSTPRIVATE** clause by OpenMP. Google it and find out what it does.

© All Rights Reserved

CamScanner

Privatization Exercises

- Parallelize the following loops using proper OpenMP pragmas

1

```
for ( i=0; i < n; i++) {
    x = i + 2 ;
    y = x * 2;
    A [ y ] = A [ y ] / 2 ;
}
```

→ ال Private بكونوا : i, x, y

2

```
for ( i=0; i < n; i++)
  for ( j=0; j < n; j++) {
      x = A[ i ][ j ] + y ;
      B[ i ][ j ] = y + x;
  }
```

→ ال Private بكونوا i, x

Read only variable

فبخليه shared دائما

© All Rights Reserved

Array Privatization

- The idea of privatization can be extended to array storages
- The merit: Enables parallelization
- The problem: Increases memory footprint + Not always feasible

```
for ( i=1; i < n; i++) {
  for ( j=0; j < 5; j++) {
    a [ j ] = b [ i ][ j ] * 2 ;
  }
  for ( j=0; j < 5; j++) {
    c [ i ][ j ] = a [ j ] * 2 ;
  }
}
```

بكبرها

بقزنها

لازم دائما تفحصي ال loop وبتتأكدي انك بتعملي صح.

privatization

```
#pragma omp for private(a, i, j)
for ( i=1; i < n; i++) {
  for ( j=0; j < 5; j++) {
    a [ j ] = b [ i ][ j ] * 2 ;
  }
  for ( j=0; j < 5; j++) {
    c [ i ][ j ] = a [ j ] * 2 ;
  }
}
```

Exercise: write a proper definition for a private array

© All Rights Reserved

Scalar Reduction

- A reduction operation inside a loop refers to a variable being iteratively updated by the iterations of the loop
- For example, the loop below has a *scalar reduction* operation because the scalar "*sum*" is being iteratively updated by the reduction statement "$sum = sum + A[i]$"
- The bad news: reduction operations cause loop-carried dependencies
- The good news: OpenMP provides the REDUCTION clause to enable parallelism (we already studied this clause in lecture 4)

```
for ( i = 1; i < n; i++){
    sum = sum + A[ i ];
```

© All Rights Reserved

Reduction Eligibility

- A variable is eligible for reduction inside a loop if and only if it is read and written by the statement where the reduction is performed

```
for ( i=1; i < n; i++){
    A[ i ] = A[ i ] * A[ i ];
    x = x * A[ i ];
}
```

x is eligible for reduction because it is read and written by a single statement (we call it the reduction statement)

```
for ( i=1; i < n; i++){
    y= y / A[ i ];
    B[ i ] = y ;
}
```

y is NOT eligible for reduction because it is being used by two statements

© All Rights Reserved

Array Reduction

- Similar to scalars, reduction can also target arrays
- OpenMP does **not** allow using arrays with the REDUCTION clause
- Therefore, programmers are responsible for writing appropriate code to perform array reductions
- Recall the histogram example from lecture 4

```
#pragma omp parallel private(my_bin, i, j)
  {

#pragma omp single
    for(i=0;i<5;i++)
      bin[i]=0;

    for(i=0;i<5;i++)
      my_bin[i]=0;

#pragma omp for
    for(i=0;i<SIZE;i++){
      j = find_bin(A[i]);
      my_bin[j]++ ;
    }

#pragma omp critical
    for(i=0;i<5;i++)
      bin[i] = bin[i] + my_bin[i];

  }
```

© All Rights Reserved

Induction Variables

- **Definition**: a variable inside a loop is called an *induction variable* if it is incremented or decremented by a fixed amount in each iteration of the loop
- For example, x and y are induction variables in the below loop but w and z are not

```
for ( i = 1; i < n; i++) {
    x = x + 1;
    y = y - 5;
    w = w * 2;
    z = z + i;
}
```

Induction variables cause loop-carried dependencies

© All Rights Reserved

Induction Variable Substitution

- A techniques that rewrites induction variables inside a loop so that they are a *recurrence function* of the loop index
- By doing so, loop-carried dependencies will be eliminated

Has a loop-carried dependency

```
x = x0;
for (i = 1; i < n; i++) {
    x = x + a;
    ...
}
```

Has no loop-carried dependency

```
for (i = 1; i < n; i++) {
    x = a * i + x0;
    ...
}
```

Induction loop substitution also results in making variable x private

© All Rights Reserved

Example

- Parallelize the following loop

```
x = 0;
for (i = 1; i < 100; i++) {
    x = x + 1;
    A[x] = A[x] * A[x];
}
```

Induction variable substitution → 1

```
for (i = 1; i < 100; i++) {
    x = i;
    A[x] = A[x] * A[x];
}
```

Privatization → 2

```
#pragma omp parallel for private (x)
for (i = 1; i < 100; i++) {
    x = i;
    A[x] = A[x] * A[x];
}
```

© All Rights Reserved

Induction Variable Substitution Exercises

- Parallelize the following loops using proper OpenMP pragmas

"جربيها يوم"

1

```
x = -1 ;
for ( i=1; i < n; i++) {
      x = x + 2 ;
      y = y + x * 2 ;
}
```

2

```
z = 1 ;
for ( i=0; i < n; i++) {
      x = A[ z ] ;
      z = z + 1 ;
}
```

© All Rights Reserved

2 loops ← إلى آخذ loop وأقسمها

Loop Fission

- Splits a loop into two or more loops (see the below example)
- Also called loop distribution

ال execution هو اللي اختلف
مرتا نفصل كل الكتابة فوق بعض
كل القراءة تحت.

```
for ( i=1; i < n; i++) {
      a [ i ] = b [ i ] + 1;
      c [ i ] = a [ i ] ;
}
```

```
for ( i=1; i < n; i++)
      a [ i ] = b [ i ] + 1;

for ( i=1; i < n; i++)
      c [ i ] = a [ i ] ;
```

But how loop fission can be useful for parallelizing loops?

© All Rights Reserved

CamScanner

i=1
a[0]=b[1]
c[1]=a[1]

i=2
a[1]=b[2]
c[2]=a[2]

i=3
a[2]=b[3]
c[3]=a[3]

⟹ Write after Read

Splitting Sequential Loops Into Parallel Loops

Serial

```
for ( i=1; i < n; i++){
    a [ i - 1 ] = b [ i ] ;
    c [ i ] = a [ i ] ;
}
```

Cannot be parallelized due to loop-carried dependence

Loop fission ⟹

وصرت أقدر أنفذه Parallel

```
#pragma omp for
for ( i=1; i < n; i++)
    c [ i ] = a [ i ] ;
```

barrier بينهم

```
#pragma omp for
for ( i=1; i < n; i++)
    a [ i - 1 ] = b [ i ] ;
```

After applying loop fission, we get two parallel loops

* سلبيته إني زدت عدد ال branches وعدد ال Code + بزيد ال Cashe Miss

* لما يكون عندك cycle dependence (يعني كل وحدة بتعتمد عبعض) ما بتقدر تعمل loop Fission

Question: can you see any drawbacks of loop fission in the above example?

© All Rights Reserved

Legality of Loop Fission

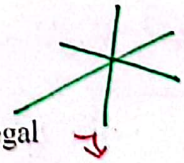
- Loop fission is legal only when dependencies do NOT form a cycle inside the loop body

```
for ( i=1; i < n; i++){
    a [ i ] = b [ i ] ;
    a [ i + 1] = a [ i ] + c [ i ];
}
```

Cycle exits
⇒ Loop fission is illegal

```
for ( i=1; i < n; i++){
    b [ i ] = a [ i - 1] ;
    a [ i ] = c [ i ] ;
}
```

Cycle does not exit
⇒ Loop fission is legal



هون بتقدر تقسمها بنحط اللي تحت فوق واللي فوق تحت ، مش دايما طبعًا حسب فهمك للـ Code بتقرر .

© All Rights Reserved

CamScanner

Loop Fission Exercises

- Parallelize the following loops using proper OpenMP pragmas

1

```
for ( i=1; i < n; i++) {
    c [ i - 1 ] = b [ i ] ;
    c [ i ] = a [ i ] ;
}
```

2

```
for ( i=1; i < n; i++)
  for ( j=1; j < n; j++) {
      B[ i ][ j ] = A[ i+1 ][ j+1 ] + y ;
      A[ i ][ j ] = C[ i ][ j ];
  }
```

© All Rights Reserved

Using Temporaries With Loop Fission

```
for ( i=0; i < n; i++)
    b [ i ] = b [ i + 1] ;
```

Has loop-carried anti-dependence

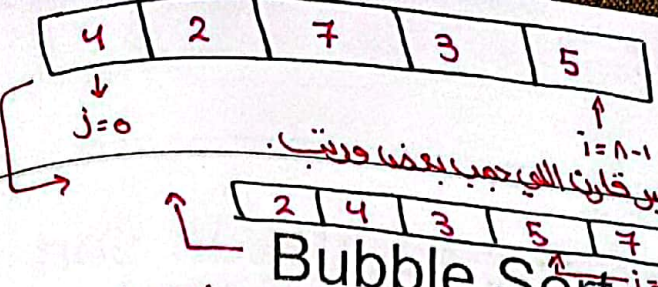
Introduce temporary array

```
for ( i=0; i < n; i++) {
    tmp [ i ] = b [ i + 1 ] ;
    b [ i ] = tmp [ i ] ;
}
```

Apply loop fission

```
#pragma omp parallel for
for ( i=0; i < n; i++)
    tmp [ i ] = b [ i + 1 ] ;
#pragma omp parallel for
for ( i=0; i < n; i++)
    b [ i ] = tmp [ i ] ;
```

© All Rights Reserved



Bubble Sort

- Sorts N integers in ascending order
- The outer loop first finds the largest element in the list and stores it in $a[N-1]$; it then finds the next-to-the-largest element and stores it in $a[N-2]$, and so on

```
for(i=N-1; i>=1; i--){
   for(j=0; j<i; j++){
      if(a[j] > a[j+1]){
         tmp = a[j];
         a[j] = a[j+1];
         a[j+1] = tmp;
      }
   }
}
```

Both outer and inner loops have loop-carried dependencies

* $O[N^2]$

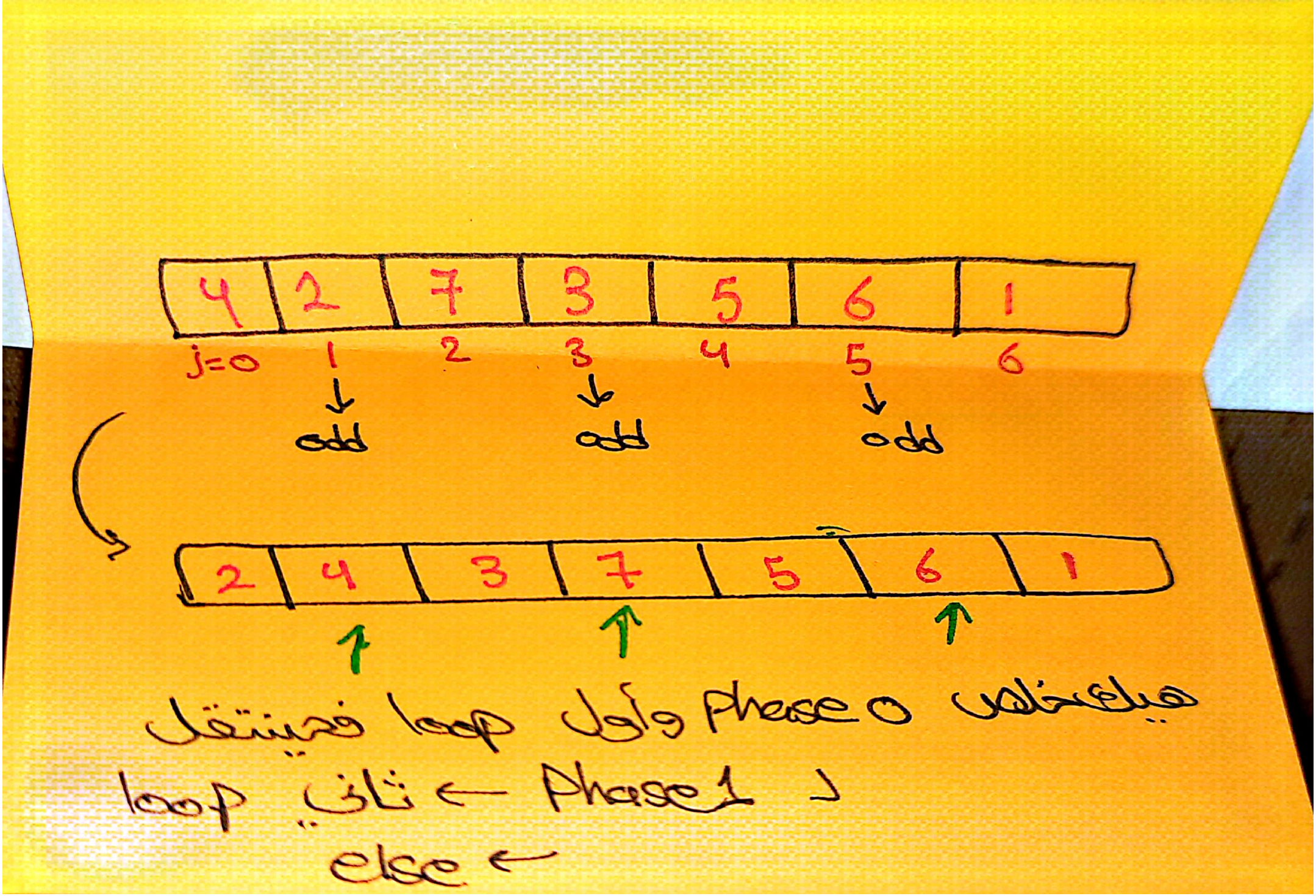
© All Rights Reserved

Odd-Even Transportation Sort

- Similar to bubble sort but with more opportunities for parallelism
- During "even" phases, each odd-subscripted element, $a[i]$, is compared to the element to its left, $a[i-1]$, and if they're out of order, they're swapped
- During "odd" phases, each odd-subscripted element , $a[i]$, is compared to the element to its right, $a[i+1]$, and if they're out of order, they're swapped
- A theorem guarantees that after N phases, the list will be sorted

```
for(phase=0; phase<N; phase++){
  if(phase%2 == 0){
    for(j=1; j<N; j+=2)
      if(a[j-1] > a[j]){
        tmp = a[j-1];
        a[j-1] = a[j];
        a[j] = tmp;
      }
  }
  else {
    for(j=1; j<N-1; j+=2)
      if(a[j] > a[j+1]){
        tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
      }
  }
}
```

© All Rights Reserved



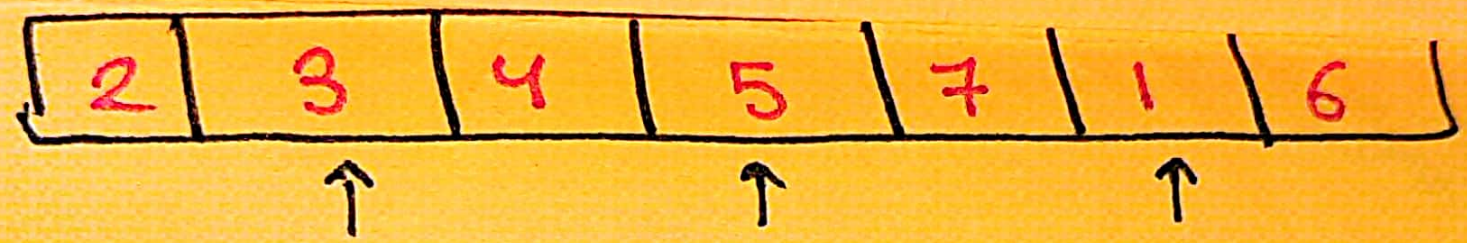
4	2	7	3	5	6	1
j=0	1	2	3	4	5	6
	odd		odd		odd	

2	4	3	7	5	6	1

هيك خلص Phase 0 وأول loop فحينتقل

لـ Phase 1 ← ثاني loop

else ←



2	3	4	5	7	1	6

وبتظل تعيد لحد ما تترتب مئة بالمئة.

ال outer loop ← serial لإنو ال Phases

بعتمدوا عبعض لكن بقدر أعمل ال inners يكونوا Parallel

Odd-Even Transportation Sort

- Suppose a = {9, 7, 8, 6}
- The algorithm phases is shown in below

Phase	Subscript in Array							
	0		1		2		3	
0	9	↔	7		8	↔	6	
	7		9		6		8	
1	7		9	↔	6		8	
	7		6		9		8	
2	7	↔	6		9	↔	8	
	6		7		8		9	
3	6		7	↔	8		9	
	6		7		8		9	

© All Rights Reserved

Odd-Even Transportation Sort OpenMP Code (Version 1)

- Outer loop is serial
- Inner loops are parallel
- The problem with this version is that the overhead of creating and terminating threads is proportional to *N*

سلبيته إنه بعمل fork, join بعقدار n.

```
for(phase=0; phase<N; phase++){
  if(phase%2 == 0){
    #pragma omp parallel for private(j,tmp)
    for(j=1; j<N; j+=2)
      if(a[j-1] > a[j]){
        tmp = a[j-1];
        a[j-1] = a[j];
        a[j] = tmp;
      }
  }
  else {
    #pragma omp parallel for private(j,tmp)
    for(j=1; j<N-1; j+=2)
      if(a[j] > a[j+1]){
        tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
      }
  }
}
```

© All Rights Reserved

CamScanner

Odd-Even Transportation Sort OpenMP Code (Version 2)

```
#pragma omp parallel private(phase)
{
  for(phase=0; phase<N; phase++){
    if(phase%2 == 0) {
      #pragma omp for private(j,tmp)
      for(j=1; j<N; j+=2)
        if(a[j-1] > a[j]){
          tmp = a[j-1];
          a[j-1] = a[j];
          a[j] = tmp;
        }
    }
    else {
      #pragma omp for private(j,tmp)
      for(j=1; j<N-1; j+=2)
        if(a[j] > a[j+1]){
          tmp = a[j];
          a[j] = a[j+1];
          a[j+1] = tmp;
        }
    }
  }
}
```

بنعمل كل اللي فوق وبوقف بشوف كل اللي تحت بنعملهم join

هيك صار أفضل.

- Another OpenMP version will less parallelism overhead

لإنه قللت ال fork, join صاروا ينعملوا مرة بس.

© All Rights Reserved

Odd-Even Transportation Sort Performance Comparison

Odd-even sort with two parallel for directives and two for directives.
(Times are in seconds.)

thread_count	1	2	3	4
Two parallel **for** directives	0.770	0.453	0.358	0.305
Two **for** directives	0.732	0.376	0.294	0.239

© All Rights Reserved

CamScanner

Summary

- We determine if a loop is serial or parallel by analyzing loop-carried dependencies in the loop
- To convert a serial loop into a parallel loop, programmers need to rewrite the algorithm
- We studied six techniques that help programmers in parallelizing sequential loops

© All Rights Reserved

Exercise 1

```
for ( i=0; i < n; i++)
   for ( j=0; j < m; j++)
      A[ i ][ j + 2] = A[ i + 2][ j + 1] ;
```

- Draw the iteration space graph for the above loop
- Determine the distance and direction vectors

© All Rights Reserved

CamScanner

Exercise 2

- Normalize the following loops
- Write the dependency tests for the normalized loops
- Use the gcd test to determine if normalized loops are parallel or serial

```
for ( i = 1; i < n; i += 2) {
    A[ i ] = A[i + 1] ;
}
```

```
for ( i = 0; i < n; i += 2) {
    A[ 2 * i ] = A[ 3 * i ] ;
}
```

© All Rights Reserved

Exercise 3

- Parallelize the following loops using OpenMP

```
j = 1 ;
for ( i = 1; i < n; i + +)
{
    j = 2 + j ;
    A[ j ] = A[j + 1] / 2 ;
}
```

```
b = 0 ;
c = 0 ;
for ( i = 1; i < n; i + +)
{
    b = b + (i % 2) ;
    c = c + b ;
}
```

© All Rights Reserved

CamScanner

Exercise 4

Question 5.8 in textbook

Consider the following loop:

```
a[0] = 0;
for (i = 1; i < n; i++)
    a[i] = a[i-1] + i;
```

There's clearly a loop-carried dependence, as the value of a[i] can't be computed without the value of a[i - 1]. Can you see a way to eliminate this dependence and parallelize the loop?

© All Rights Reserved

Function-Level Parallelism in OpenMP

Instructor: Dr. Fahed Jubair
Computer Engineering Department
University of Jordan

CamScanner

Function-Level Parallelism

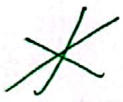
- Programs with *function-level* (aka *task-level*) parallelism focuses on distributing parallel functions (i.e., tasks) on threads

- OpenMP provides two directives that are commonly used for writing task-parallel programs: بساعدني أوزع الشغل على أكثر من Task
  1. The SECTIONS directive } We will focus on this directive
  2. The TASK directive } Beyond the scope of this class

حنركز عليه

© All Rights Reserved.

Quick Review
The SECTIONS Directive



- A SECTIONS directive specifies that all enclosed section(s) of code are to be divided between threads such that each section is executed once by a thread

رمجتهم لو ال region كلها section

```
#pragma omp parllel sections
{
  #pragma omp section
  function1();

  #pragma omp section
  function2();

  #pragma omp section
  function3();
}
```

ال Procedure كلها dynamic فما بنتحكم بين ال functions وال threads مين ياخذ مين.

© All Rights Reserved.

CamScanner

Challenges

- There are two primarily challenges that programmers need to tackle when converting a sequential algorithm into a task-parallel algorithm:
  1. How to maximize the number of tasks that can run in parallel?
  2. How to assign these parallel tasks to threads such that parallelism overhead is minimized?

© All Rights Reserved.

Algorithms With Function-Level Parallelism

We will study two sorting algorithms that take advantage of function-level parallelism:

- Merge Sort
- Quick Sort

} أشهر الموجودين

© All Rights Reserved.

CamScanner

Merge Sort

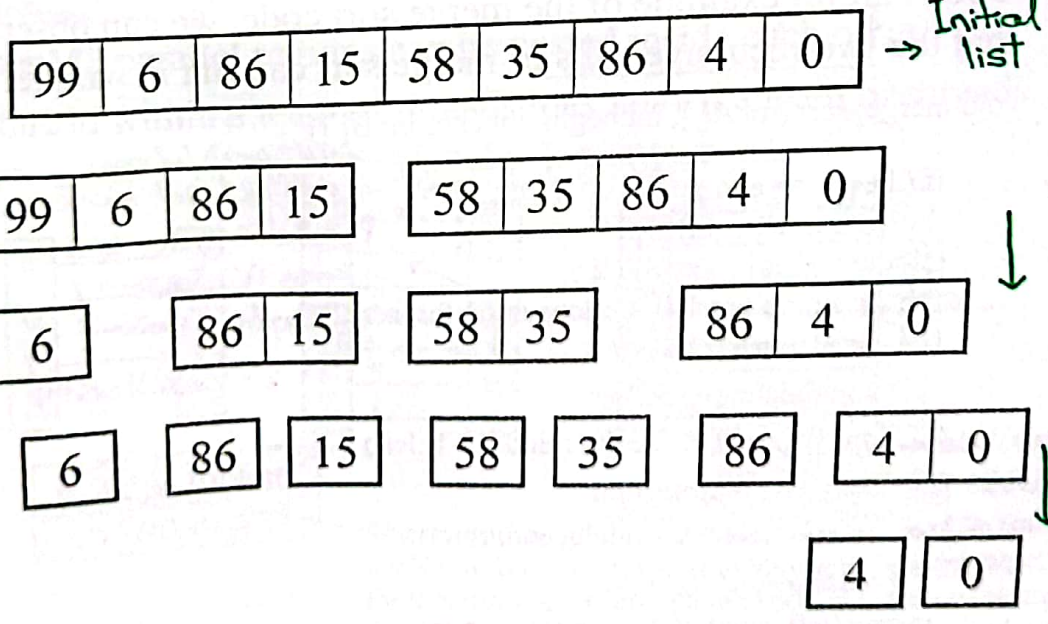
- **Split** the unsorted list into n sublists, each containing one element
- Repeatedly **merge** sublists to produce new sorted sublists until there is only 1 sublist remaining, which will be the sorted list

"Serial code"

```
// A is the input unsorted list and B is the output sorted list
void merge_sort (int A[ ], int begin, int end, int B[ ])
{
    if ( begin == end )    // if A has only one integer
        return A[begin] ;
    middle = (begin + end) / 2 ;
    merge_sort ( A, begin, middle – 1, left) ;  // sort left half [begin : middle - 1]
    merge_sort ( A, middle, end, right);        // sort right half [middle : end]
    merge (left, right, B);        // merge left and right sublists into B
}
```

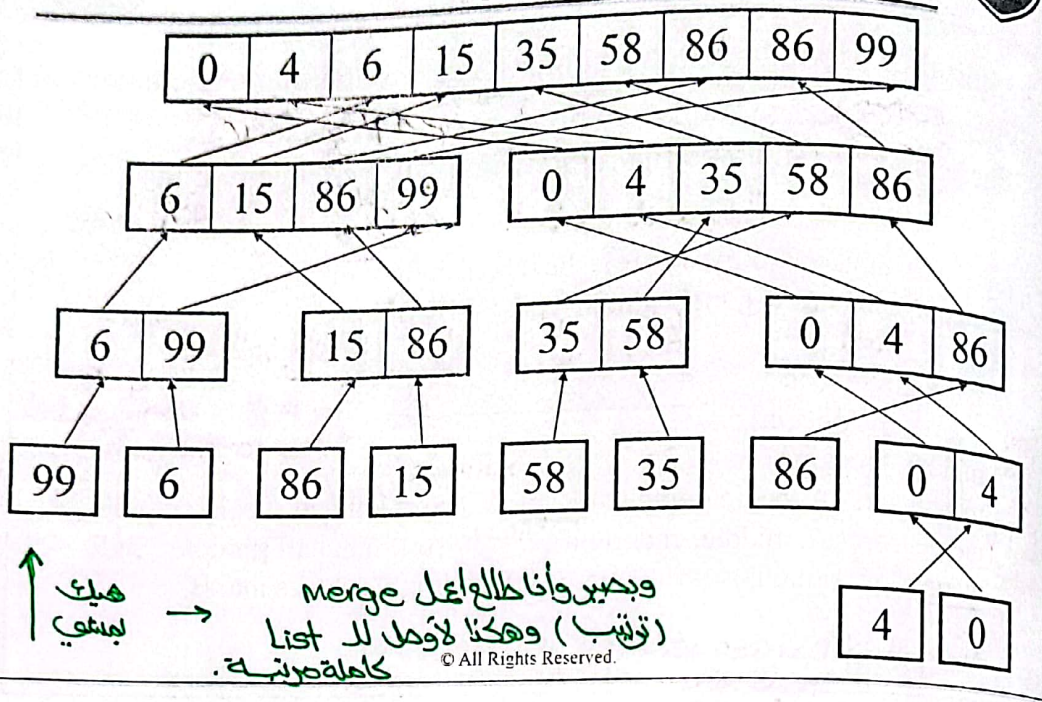
© All Rights Reserved.

Merge Sort Example
Split Phase



© All Rights Reserved

Merge Sort Example
Merge Phase



هيك بمشي → وبصير وأنا طالع اعمل merge (ترتيب) وهكذا لأوصل لل List كاملة مرتبة.

© All Rights Reserved.

Parallelism In Merge Sort

- Upon careful examine of the merge sort code, we can observe that the two recursive calls of merge sort can run in parallel

```
void merge_sort (int A[ ], int begin, int end, int B[ ])
{
    if ( begin == end )
        return A[begin] ;
    middle = (begin + end) / 2 ;
    #pragma omp parallel sections num_threads(2)
    { /* start of parallel region */
        #pragma omp section
        merge_sort ( A, begin, middle – 1, left) ;
        #pragma omp section
        merge_sort ( A, middle, end, right);
    } /* end of parallel region */
    merge (left, right, B);
}
```

Parallel region جولها شغلة وحدة فقط: Section

لأنه عدد ال Task = 2

كلما أناديها ← merge_sort (A, begin, middle – 1, left) ; → Run in parallel

يرجع لفوق تعيد فصار عنا Parallel region جوا Parallel region

Run in parallel

merge_sort (A, middle, end, right); → Run in parallel

© All Rights Reserved.

Hold On!

- The recursive call of merge_sort in the previous slide will cause parallel regions to be created within each other!
- Does OpenMP even allow this? If so, then how does the parallel execution occur?

* صار عندي nested parallism

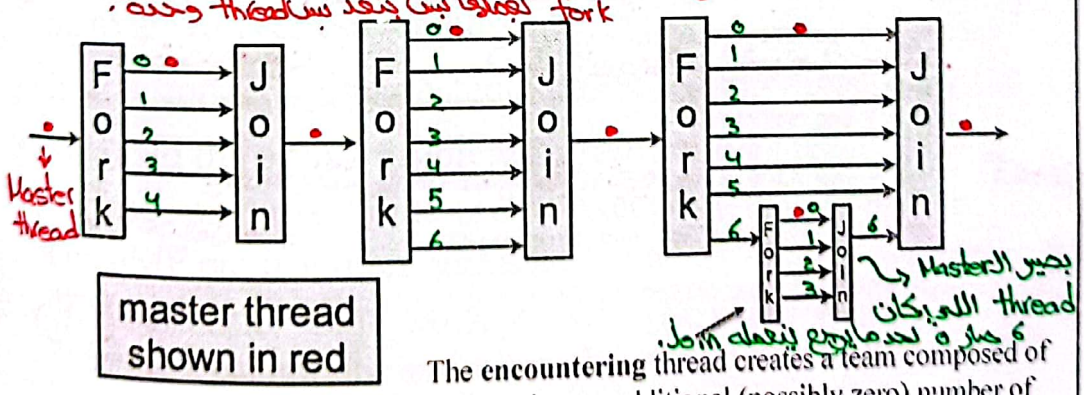
© All Rights Reserved.

Nested Parallelism

هاد بصير لو صار عندي limit
على عدد الـ thread

- OpenMP parallel regions can be nested inside each other
- A thread within a team of threads may spawn a team of threads

running thread ← لو كان مسموحلك بس 6 لما يجي الـ inner يعمل fork نعملها بس ينفذ بس thread وحدة.



master thread shown in red

بصير الـ Master thread اللي كان 6 صار 0 لحد ما يرجع ينعمله join.

The **encountering** thread creates a team composed of itself and some additional (possibly zero) number of threads (the encountering thread becomes the master thread)

© All Rights Reserved.

More about Nested Parallelism

- It is possible to enable/disable nested parallelism in OpenMP using the environment variable OMP_NESTED or the library routine *omp_set_nested( )*
- If nested parallelism is enabled, when a thread on a team within a parallel region encounters a new parallel construct, an additional team of threads is forked off of it, and it becomes their master
- If nested parallelism is disabled, when a thread on a team within a parallel region encounters a new parallel construct, execution continues on this additional single thread only

← وبصير رقمه ٥ جوا هاي ال Parallel region

© All Rights Reserved.

Example on Nested Parallel Regions

```
#pragma omp parallel num_threads(4)  region ①
{
  printf("I am thread %d from region 1\n",omp_get_thread_num());
  #pragma omp barrier  → عشان نتأكد ال 4 انطبعوا قبل ما نكمل
region ② #pragma omp single  →  thread واحد حينفذ هاي
  {
    #pragma omp parallel num_threads(2)
    {  ← غير ال single صار في كمان وحدة.
      printf("I am thread %d from region 2\n",omp_get_thread_num());  ← حتنطبع مرتين
      #pragma omp barrier
region ③ #pragma omp single  → واحد من الاثنين اللي انعملوا قبل شوي
      {
        #pragma omp parallel num_threads(8)  حيعمل هاي
        {
          printf("I am thread %d from region 3\n",omp_get_thread_num());
        }
      }
    }
  }
}
```

← بتنطبع 8 مرات.

© All Rights Reserved.

Example on Nested Parallel Regions (cont.)

```
$ gcc –fopenmp nested.c –o a.out
$ export OMP_NESTED = TRUE
$ ./a.out
```

I am thread 3 from region 1
I am thread 0 from region 1
I am thread 2 from region 1
I am thread 1 from region 1
I am thread 0 from region 2
I am thread 1 from region 2
I am thread 7 from region 3
I am thread 4 from region 3
I am thread 0 from region 3
I am thread 3 from region 3
I am thread 1 from region 3
I am thread 2 from region 3
I am thread 6 from region 3
I am thread 5 from region 3

* لو ما حطينا Single ساعتها كلهم
بنعملهم fork فبتصير ال
Print = 8

© All Rights Reserved.

Remarks on Nested Parallelism

- Nesting parallel regions provides an immediate way to allow more threads to participate in the computation
- However, nesting parallel regions can easily create too many threads and oversubscribe the system
- In addition, creating nested parallel regions adds overhead
- To avoid the above problems, programmers usually try to impose some discipline on the execution using appropriate OpenMP environment variables
  - For example, "OMP_MAX_ACTIVE_LEVELS" environment variable controls the maximum allowed number of active nested parallel regions

لو حطيته 3 مثلا فبكون معك
إمكانية
تخلق
regions
جوا بعض بعدد 3

© All Rights Reserved.

CamScanner

Quick Sort

بتاخذ رقم معين وبتحط كل الأصغر منه عالشمال وكل الأكبر منه عاليمين وبتحطه هو بالنص وهذا الرقم بيسمى "Pivot" وموشرط تحطوم عالترتيب، بس تقسوم بكل ترتيب اللي عاليمين لحال واللي عالشمال لحال.

Sorts an array of integers as follows:

- Pick an element, called a pivot, from the array
- Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way)
- Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values

Source: Wikipedia

© All Rights Reserved.

Quick Sort (cont.)

```
void quick_sort (int A[ ], int begin, int end) {
    if ( begin < end ) {
        p = partition (A, begin, end ) ;
        quick_sort ( A, begin, p – 1 ) ;
        quick_sort ( A, p + 1, end ) ;
    }
}

int partition (int A[ ], int begin, int end) {
    pivot = A[end] ;    // let us pick the last element as our pivot
    i = begin ;
    for ( j = begin; j < end; j++ ) {
        if ( A[ j ] <= pivot ) {
            swap ( A[ j ] , A[ i ] );  // swap the content of A[ j ] and A[ i ]
            i++ ;
        }
    }
    swap ( A[ end ] , A[ i ] );
    return i ;
}
```

Pointer باشر للرقم اللي بالنص.

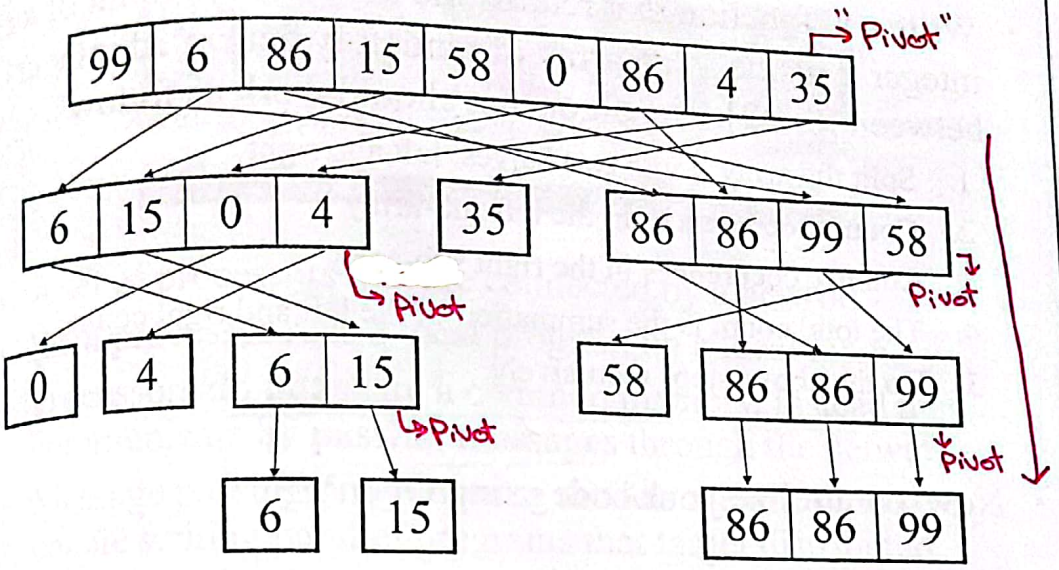
Parallel بقدر أخليهم

الـ Pivot بطلوعن الحسبة لأنه مرتب أساسا

© All Rights Reserved.

CamScanner

Quick Sort Example



© All Rights Reserved.

Parallelism In Quick Sort

- Upon careful examine of the quick sort code, we can observe that the two recursive calls of quick sort can run in parallel

```
void quick_sort (int A[ ], int begin, int end) {
    if ( begin < end ) {
        p = partition (A, begin, end ) ;
        #pragma omp parallel sections num_threads(2)
        {
            #pragma omp section
            quick_sort ( A, begin, p – 1 ) ;
            #pragma omp section
            quick_sort ( A, p + 1, end ) ;
        }
    }
}
```

Run in parallel

© All Rights Reserved.

Exercise

- Write a C function that returns the occurrences count of an integer x inside A, an array of randomly generated integers between [0: 99]. The algorithm should work as follows:
  1. Split the array A into two halves: left and right
  2. Count x occurences in the left sub-array
  3. Count x occurences in the right sub-array
  4. The total count is the summation of the left and right counts
  5. Do the above steps recursively

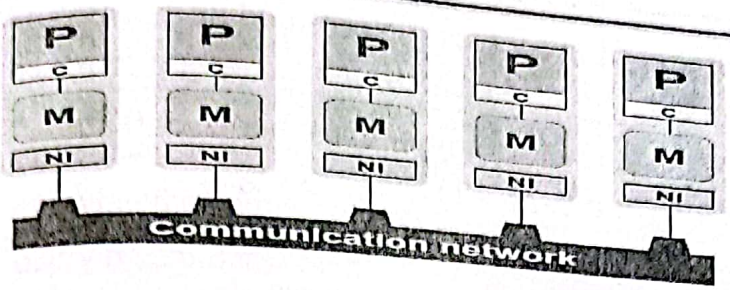
- Now, parallelize your code using OpenMP

© All Rights Reserved.

Introduction to MPI

Instructor: Dr. Fahed Jubair
Computer Engineering Department
University of Jordan

Distributed-Memory Architectures



- A set of processors that are connected by a network
- Each processor runs a local program on a local memory
- Processors **do not** share a common memory, instead they communicate by **passing messages** through the network
- Message passing programming models are introduced to enable writing parallel programs that target distributed memory architectures

© All Rights Reserved.

هي فقط library مافيها
Compiler زي ال openMP
مافيها directory.

MPI

- *Message Passing Interface* (MPI) is the most commonly-used standard for message passing programs
- MPI is a standard developed by MPI Forum (http://www.mpi-forum.org) that specifies how to implement a library that can be used with conventional sequential languages such as Fortran, C and C++ for writing SPMD-style parallel programs for distributed-memory architectures
- MPI standard versions: **1.0** (1994), **2.0** (1997), **3.0** (2012)
- Several implementations for MPI are available
  - E.g., MPICH, OpenMPI, Intel, Microsoft

free — commercial

© All Rights Reserved

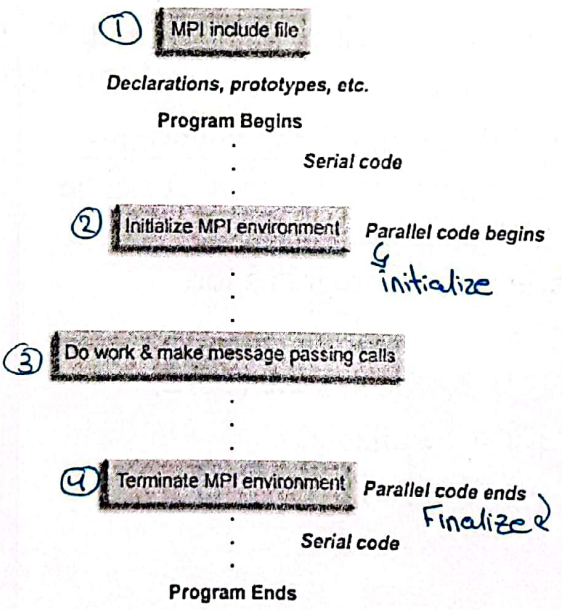
لازم نزلها عال Linux
بـ Command

Execution Model of MPI

- Fixed number of processes (determined on startup) that starts the execution together and lasts till the execution terminates
- Each process is given a unique rank between 0 and (*number_of_processes* – 1)
- Each process has its instruction and data address spaces
  - No physical sharing between processes
  - All data must be explicitly partitioned and placed
  - No data races due to hardware artifacts
- To exchange data, processes use library routines to communicate with each other
- Essentially, each process runs like a sequential process except that it makes calls to the MPI library

© All Rights Reserved.

A Generic MPI Program Structure



Source:
https://computing.llnl.gov/tutorials/mpi

© All Rights Reserved.

MPI Library Routines

- Environment management routines: used for setting and interrogating the MPI execution environment
  - E.g., initializing and terminating the MPI environment
  - E.g., querying the number of processes
  - E.g., querying a process rank
- Communication routines: generally, divided into two types of routines:
  - Point-to-Point: a message is sent from a specific sending process to a specific receiving process
  - Collective: **all** processes participate in the communication

© All Rights Reserved.

MPI Communicator

- MPI uses objects called *communicators*, which are a collection of processes that can send messages to each other
- MPI execution, as startup, defines a default communicator that encapsulates all the processes in the system, called MPI_COMM_WORLD
- Most MPI routines require you to specify a communicator as an argument

© All Rights Reserved.

Environment Routines

- MPI_Init (&arg1, &arg2)
  - Initializes the MPI execution environment
  - This function must be called in every MPI program, must be called before any other MPI functions and must be called only once in an MPI program
- MPI_Comm_size (comm, &size) → أعرف عدد الـ processes
  - Returns the total number of MPI processes in the specified communicator, such as MPI_COMM_WORLD
- MPI_Comm_rank (comm, &rank) → أعرف شو الـ Rank تاعي
  - Returns the rank of the calling MPI process within the specified communicator
- MPI_Finilize ( )
  - Terminates the MPI execution environment
  - This function should be the last MPI routine called in every MPI program - no other MPI routines may be called after it

© All Rights Reserved.

The Traditional Hello World Example

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]){
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  printf("Hello from process %d\n", rank);
  if(rank == 0)
    printf("number of processes is %d\n", size);
  MPI_Finalize();

  return 0;
}
```

© All Rights Reserved.

Compiling Hello World Example

- We can compile the Hello World OpenMP program on a linux machine using the following command

wrapper script to compile

source file

mpicc mpi_hello.c -o mpi_hello

لو بدي أعطي الـ output اسم

لإنه عندي c كتبت mpicc ، لو عندي c++ بكتب mpic++

create this executable file name (as opposed to default a.out)

© All Rights Reserved.

Running Hello World Example

mpiexec -n <number of processes> <executable>

mpiexec -n 4 ./mpi_hello

ممكن تنكتب mpirun

run with 4 processes

كل copy بتشتغل على Independent Machine

*ممكن تعملها عنفس الجهاز بصيرها بيشتغلوا Independentally لإنه ما عنا clusters.

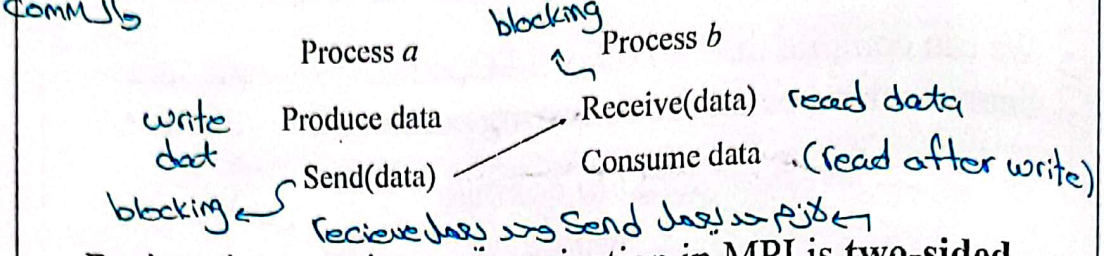
One possible output

Hello from process 2
Hello from process 0
number of processes is 4
Hello from process 1
Hello from process 3

© All Rights Reserved

CamScanner

MPI Point-to-Point Communication



- Basic point-to-point communication in MPI is **two-sided**
  - Process *a* must make a call to a send routine and specify process *b* as the receiver in the send message arguments
  - Process *b* must make a call to a receive routine and specify process *a* as the sender in the receive message arguments
  - Both, send and receive routines must have matching communicator object and similar description (size and type) of the data being sent/received

12

Routines

- Generally, most of the MPI point-to-point routines can be used in either *blocking* or *non-blocking* mode
- Let us first consider the basic type: blocking communication
- Blocking send/receive routines have the following properties:
  - A blocking send routine will only "return" after it is safe to modify the application buffer (your send data) for reuse
  - Safe means that modifications will not affect the data intended for the receive task
  - Safe does not imply that the data was actually received - it may very well be sitting in a system buffer
  - A blocking receive only "returns" after the data has arrived and is ready for use by the program

© All Rights Reserved

13

Process a

Produce data

Send (data, rec=b)

Process b

consume data

recieve(data, sen=a)

*App buffer → جزء من Mem → Ex: a b

X=10 recieve(

Send(&X, b) &y, a)

↓

عادي غيرت اسم الـ x لـ y هاي الـ Mem تاكتي

*System buffer → buffer os

Ex: Process a

$x=10$

blocking till copy/paste is complete ← send(8x, b)

$x=5$

بتتخزن بال App buffer ما بتروح لل os buffer

Process b

recieve(8y, a) ↓ blocking

Print y

لحد ما توصل ال data على ال App buffer تاعه

① App buffer $\boxed{x=10}$

② system buffer $\boxed{x=10}$

Copy

Paste

④ ← App buffer $\boxed{x=10}$

system buffer $\boxed{x=10}$

↳ ③

* ال Comm بصير بين ال System buffers .

Blocking Send/Receive Routines

elements عدد ال ← → element نوع ال → label for Msg ↓ للتنظيم (اختياري)

Mem Var ال ←

MPI_Send(buffer, count, type, dest, tag, comm)

↳ Partner

MPI_Recv(buffer, count, type, source, tag, comm, status)

دائمًا MPI_COMM_WORLD

- Buffer: the variable name that is being sent/received
- Count: number of data elements to be sent
- Type: type of data to be sent
- Source: rank of the sender process (needed for receive messages)
- Dest: rank of the receiver process (needed for send messages)
- Tag: arbitrary non-negative integer assigned by the programmer to uniquely identify a message (send and receive operations should match message tags) → Tag (Send) مادخلها بـ Tag(recieve)
- Comm: communicator object
- Status: contains further information about received messages

object بخزن ← * 0 → Succefull * any num → error

© All Rights Reserved.

MPI Elementary Data Types

For reasons of portability, MPI predefines its elementary data types

لازم هيك نكتبهم

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG	signed long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

- Programmers may also create their own data types (called **Derived Data Types**)

© All Rights Reserved

CamScanner

Error Handling

- Most MPI routines include a return/error code parameter that can be tested to check if a routine call succeeds/fails
- However, the default behavior of failed routines is to abort the execution
- Users can override this behavior (if necessary) to capture this error
- The code MPI_SUCCESS (zero) is returned when a routine succeeds

© All Rights Reserved.

Send بعمل ← 0
recieve بعمل ← 1
2 processes

MPI Program Example 1

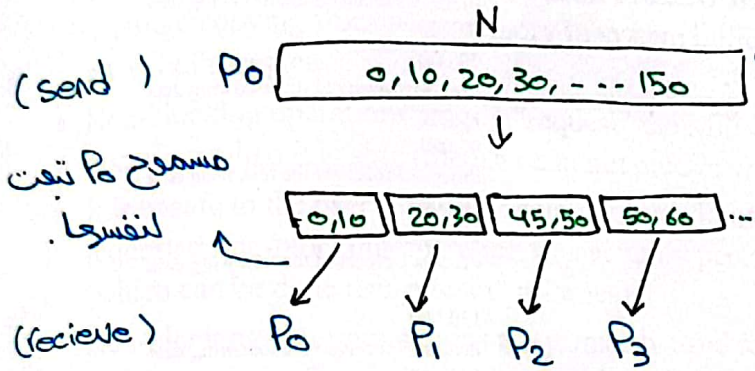
```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]){
  int rank, size;
  int sbuff, rbuff;
  MPI_Status stat;
  int err_code;
  int tag = 1;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  MPI_Comm_size(MPI_COMM_WORLD, &size);

  if(rank == 0){
    sbuff = 100;
    err_code = MPI_Send(&sbuff, 1, MPI_INT, 1, tag, MPI_COMM_WORLD);
  }
  else{
    err_code = MPI_Recv(&rbuff, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &stat);
    printf("data=%d is received\n", rbuff);
  }

  MPI_Finalize();

  return 0;
}
```

© All Rights Reserved.

Compiling and Running Example 1

```
$ mpicc example1.c –o a.out
$ mpiexec –n 2 ./a.out
```

data=100 is received



هاد الـ Process بده يبعت لأكثر من Processes (chunks)

مسموح Po تبعت لنفسها.

* 4 send, 4 recieve, Po just sends, each P recieve once.

© All Rights Reserved.

طرق تعريف الـ Array:

1- int A[100];

2- int* A; → A=malloc (100 * sizeof(int));

∴ Pointer → Array بأشر

(int*) int ← حجم الـ

ما بكون إله type فلازم اعمله casting زي ما موضح:

MPI Program Example 2

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define N 16
int main(int argc, char *argv[]){
  int rank, size, chunk, i, tag=1;
  float *sbuff, *rbuff;
  MPI_Status stat;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  MPI_Comm_size(MPI_COMM_WORLD, &size);

  chunk = N / size;

  if(rank == 0){
    sbuff = (float*) malloc(N * sizeof(float));
    for(i=0; i<N; i++)
      sbuff[i] = 10.0 * i;

    for(i=0; i<size; i++)
      MPI_Send(&sbuff[chunk*i], chunk, MPI_FLOAT, i, tag, MPI_COMM_WORLD);
  }

  rbuff = (float*) malloc(chunk * sizeof(float));
  MPI_Recv(rbuff, chunk, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &stat);
  printf("Process %d received the following data\n", rank);
  for(i=0; i<chunk; i++)
      printf("%.3f\n", rbuff[i]);

  MPI_Finalize();

  return 0;
}
```

عدد الـ elements

حجم التقسيم

عدد الـ Processes

الفرق بينهم الأولى Static لازم تكون عارف عدد الـ element أمّا الثانية بتقدر تستخدمها هيك: A= (int*) malloc (N* sizeof(int));

بعت chunk لكل واحد

lower bound

نوع الـ data الحجم

Sender

* للتسهيل حتكون N دائما Power of 2. وكذلك عدد الـ Processes

دائما

* مسموح تبعت بنوع وتستقبل بنوع ما دام نفس عدد الـ bytes

© All Rights Reserved

* لازم عدد الـ Send والـ recieve يكونوا Match.

* عدد الـ bytes اللي انبعثوا لازم بيساووا عدد الـ bytes اللي بستقبلهم.

Compiling and Running Example 2

* مو شرط ينطبعوا بالترتيب .

```
$ mpicc example2.c –o a.out
$ mpiexec –n 8 ./a.out
```

```
Process 0 received the following data
0.000
10.000
Process 1 received the following data
20.000
30.000
Process 2 received the following data
40.000
50.000
Process 3 received the following data
60.000
70.000
Process 4 received the following data
80.000
90.000
Process 5 received the following data
100.000
110.000
Process 6 received the following data
120.000
130.000
Process 7 received the following data
140.000
150.000
```

© All Rights Reserved

global sum بعمل Process لـ local sum بعدين بتجمع chunks الـ بعمل

MPI Program Example 3
Parallel Sum

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define N 512

int main(int argc, char *argv[]){
  int rank, size, chunk, i, tag=1;
  int A[N];
  int *buff;
  MPI_Status stat;
  int local_sum, global_sum;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  MPI_Comm_size(MPI_COMM_WORLD, &size);

  chunk = N / size;

  if(rank == 0){
    for(i=0; i<N; i++)
      A[i] = rand() % 100; // each integer is randmly picked from 0-99
    for(i=0; i<size; i++)
      MPI_Send(&A[chunk*i], chunk, MPI_INT, i, tag, MPI_COMM_WORLD);
  }

  buff = (int*) malloc(chunk * sizeof(int));
  MPI_Recv(buff, chunk, MPI_INT, 0, tag, MPI_COMM_WORLD, &stat);
  local_sum = 0;
  for(i=0; i<chunk; i++)
    local_sum += buff[i];
  MPI_Send(&local_sum, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);

  if(rank==0){
    global_sum = 0;
    for(i=0; i<size; i++){
      int tmp;
      MPI_Recv(&tmp, 1, MPI_INT, i, tag, MPI_COMM_WORLD, &stat);
      global_sum += tmp;
    }
    printf("global sum is %d\n", global_sum);
  }
  MPI_Finalize();
  return 0;
}
```

Process 0 distributes data

Each process computes local sum

Process 0 collects local sums to compute global sum

لـ عشان يبعتوهم address

لازم بعت single int

© All Rights Reserved

* هون الـ Code الـ Performance بأثر كتير بحيث لازم ينتبه حتى لو حطوا Recieve bloking الـ بتوزع chunk بدل ما تنزل كبيرة
حتى الـ Send blocking أخف من الـ Recieve .

بتناوي Send وبتعمل return مباشرة
وكذلك الـ recieve فما بعلق هيك. لو كان في أو ممكن
حصول خطأ بحط MPI wait.

Non-Blocking Communication

حل مشكلة الـ blocking

- The other type of point-to-point communication in MPI
- Non-blocking send/receive routines have the following properties:
  - Non-blocking send and receive routines return almost immediately, i.e., they do not wait for any communication events to complete, such as message copying from user memory to system buffer space or the actual arrival of message
  - Non-blocking operations simply "request" the MPI library to perform the operation when it is able (the user can not predict when that will happen)
  - It is unsafe to the user to modify data until you know for a fact that the requested non-blocking operation was actually performed by the library (which can be done using "wait" routines)
  - Non-blocking communications are primarily used to overlap computation with communication and exploit possible performance gains

© All Rights Reserved.

Non-Blocking Send/Receive Routines

non blocking

MPI_Isend(buffer, count, type, dest, tag, comm, request)

MPI_Irecv(buffer, count, type, source, tag, comm, request)

non blocking

بدل Status

بخزن رقم الـ request

- The only new argument we did not see before is "request"
- Since non-blocking operations may return before the requested system buffer space is obtained, the system issues a unique "request number"
- The programmer uses this system assigned "handle" later (in a WAIT type routine) to determine completion of the non-blocking operation
- In C, the "request" argument is a pointer to a predefined structure MPI_Request

© All Rights Reserved

CamScanner

بتنادي Send وبتعمل return مباشرة
وكذلك ال recieve فما بعلق هيك. لو كان في أو ممكن
حصول خطأ بحط MPI wait.

Non-Blocking Communication

حل مشكلة ال blocking

- The other type of point-to-point communication in MPI
- Non-blocking send/receive routines have the following properties:
  - Non-blocking send and receive routines return almost immediately, i.e., they do not wait for any communication events to complete, such as message copying from user memory to system buffer space or the actual arrival of message
  - Non-blocking operations simply "request" the MPI library to perform the operation when it is able (the user can not predict when that will happen)
  - It is unsafe to the user to modify data until you know for a fact that the requested non-blocking operation was actually performed by the library (which can be done using "wait" routines)
  - Non-blocking communications are primarily used to overlap computation with communication and exploit possible performance gains

© All Rights Reserved.

Non-Blocking Send/Receive Routines

non blocking

MPI_Isend(buffer, count, type, dest, tag, comm, request)

MPI_Irecv(buffer, count, type, source, tag, comm, request)

non blocking

بدل Status

بخزن رقم ال request

- The only new argument we did not see before is "request"
- Since non-blocking operations may return before the requested system buffer space is obtained, the system issues a unique "request number"
- The programmer uses this system assigned "handle" later (in a WAIT type routine) to determine completion of the non-blocking operation
- In C, the "request" argument is a pointer to a predefined structure MPI_Request

© All Rights Reserved.

CamScanner

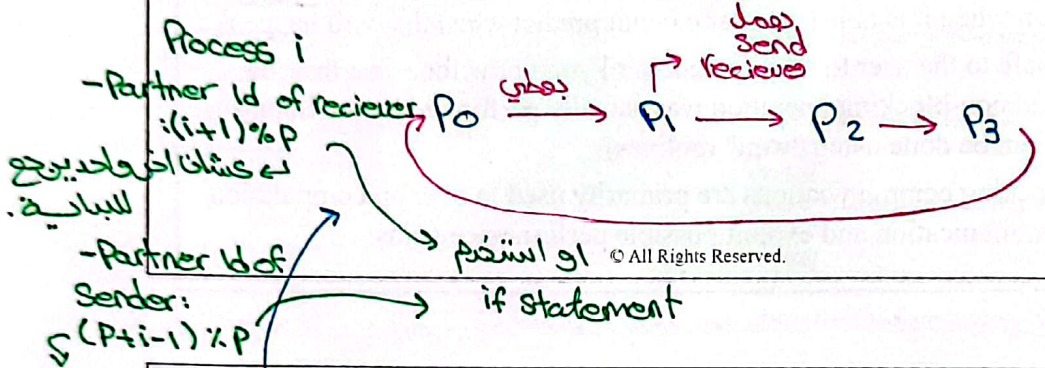
Wait Routines

MPI_Wait (request, status) → معناها استنى هذا الـrequest ليخلص وخزن الـstatus

MPI_Waitall (count, array_of_requests, array_of_statuses)

بتقدر تستنى أكثر من حد بدل ما تستنى واحد واحد

- MPI_Wait blocks the calling process until a specified non-blocking send or receive operation has completed
- For multiple non-blocking operations, the programmer can specify any, all or some completions



Process i

-Partner Id of reciever :(i+1)%p

← عشان اخر واحد يرجع للبداية.

-Partner Id of Sender: (P+i-1)%p

عشان أول وحدة تترسلها من 3 مش من -1

$P_0 \xrightarrow{\text{بعطي}} P_1 \longrightarrow P_2 \longrightarrow P_3$

بعمل send recieves

او استخدم if statement

© All Rights Reserved.


```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]){
  int rank, size, tag=1;
  int dst, sndr;
  int sbuff, rbuff;
  MPI_Status stats[2];
  MPI_Request reqs[2];

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  MPI_Comm_size(MPI_COMM_WORLD, &size);

  dst = (rank + 1) % size ;
  sndr = (rank + size - 1) % size ;

  MPI_Irecv(&rbuff, 1, MPI_INT, sndr, tag, MPI_COMM_WORLD, &reqs[0]);

  sbuff = rank * 10 ;
  MPI_Isend(&sbuff, 1, MPI_INT, dst, tag, MPI_COMM_WORLD, &reqs[1]);

  MPI_Waitall(2, reqs, stats);

  printf("process %d rbuff = %d\n", rank, rbuff);

  MPI_Finalize();

  return 0;
}
```

MPI Program Example 4

بدي اعمل Send recieves at the same time.

بيعت الـ address

عملت هاي أو الـ Send أول ما بنفرق لأنه non blocking.

1 → بيعت 10

2 → 20 " ---

هون انا عملت block

© All Rights Reserved.

Compiling and Running Example 4

```
$ mpicc example4.c –o a.out
$ mpiexec –n 16 ./a.out
```

process 1 rbuff = 0 ← 0 يستقبل من
process 2 rbuff = 10 ← 1 " "
process 3 rbuff = 20 ← 2 " "
process 4 rbuff = 30
process 5 rbuff = 40
process 8 rbuff = 70
process 9 rbuff = 80
process 10 rbuff = 90
process 11 rbuff = 100
process 12 rbuff = 110
process 14 rbuff = 130 ← 13 " "
process 15 rbuff = 140 ← 14 " "
process 0 rbuff = 150 ← 15 " "
process 13 rbuff = 120
process 6 rbuff = 50
process 7 rbuff = 60

© All Rights Reserved

Example 5

```
for ( i = 0; i < N; i++)
    c[i] = a[i] * b[i];
```

Product يعمل

- Let us try to parallelize the above loop using MPI as follows:
  - Process 0 initializes arrays *a* and *b*, partitions them into equal chunks and distributes these chunks onto all processes 2*P = Msg عدد الـ
  - Each process locally multiplies its chunks to obtain array *c*
  - Process 0 collects array *c* from all processes

Msg=P يستقبل P0
منافسه ومن ٩ ومن b وهكذا

© All Rights Reserved.

CamScanner

MPI Program Example 5

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define N 16
int main(int argc, char *argv[]){
  int rank, size, chunk, i, m, tag1=1, tag2=2;
  double a[N], b[N], c[N];
  double *buff1, *buff2;
  MPI_Status *stats, *stats0, *stats1;
  MPI_Request *reqs, *reqs0, *reqs1;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  MPI_Comm_size(MPI_COMM_WORLD, &size);

  chunk = N / size;

  if(rank == 0){
    for(i=0; i<N; i++){
      a[i] = 12.4 * (i % 100);
      b[i] = -3.33 * (i % 100);
    }
    reqs0 = (MPI_Request*) malloc(2* size * sizeof(MPI_Request));
    stats0 = (MPI_Status*) malloc(2* size * sizeof(MPI_Status));
    m = 0;
    for(i=0; i<size; i++){
      MPI_Isend(&a[chunk*i], chunk, MPI_DOUBLE, i, tag1, MPI_COMM_WORLD, &reqs0[m++]);
      MPI_Isend(&b[chunk*i], chunk, MPI_DOUBLE, i, tag2, MPI_COMM_WORLD, &reqs0[m++]);
    }
    MPI_Waitall(m, reqs0, stats0);
  }
```

Array size

# of processes

random initialization (Double Num)

Using non-blocking sends allows the waiting times of send messages to be overlapped

Counter

كل مرة بعت chunk من a و chunk من b

© All Rights Reserved.

MPI Program Example 5 (cont.)

ال receive اللي بنفذها.

```
reqs = (MPI_Request*) malloc(2 * sizeof(MPI_Request));
stats = (MPI_Status*) malloc(2 *sizeof(MPI_Status));
buff1 = (double*) malloc(chunk * sizeof(double));
MPI_Irecv(buff1, chunk, MPI_DOUBLE, 0, tag1, MPI_COMM_WORLD, &reqs[0]);
buff2 = (double*) malloc(chunk * sizeof(double));
MPI_Irecv(buff2, chunk, MPI_DOUBLE, 0, tag2, MPI_COMM_WORLD, &reqs[1]);
MPI_Waitall(2, reqs, stats);

for(i=0; i<chunk; i++)
  buff1[i] = buff1[i] * buff2[i];

MPI_Send(buff1, chunk, MPI_DOUBLE, 0, tag1, MPI_COMM_WORLD);
```

the waiting times of the two receive messages are overlapped

بدل ما بزيد buffer ليحسب

الجواب استخدم buff1

Each process sends its result back to process 0

© All Rights Reserved.

CamScanner

MPI Program Example 5 (cont.)

```
if(rank == 0){
  reqs1 = (MPI_Request*) malloc(size * sizeof(MPI_Request));
  stats1 = (MPI_Status*) malloc(size * sizeof(MPI_Status));
  m = 0;
  for(i=0; i<size; i++)
    MPI_Irecv(&c[chunk*i], chunk, MPI_DOUBLE, i, tag1, MPI_COMM_WORLD, &reqs1[m++]);

  MPI_Waitall(m, reqs1, stats1);

  printf("array a: ");
  for(i=0; i<N; i++)
    printf("%0.4f, ", a[i]);
  printf("\narray b: ");
  for(i=0; i<N; i++)
    printf("%0.4f, ", b[i]);
  printf("\narray c: ");
  for(i=0; i<N; i++)
    printf("%0.4f, ", c[i]);
  printf("\n");
}

MPI_Finalize();
return 0;
}
```

the waiting times of receive messages are overlapped

© All Rights Reserved.

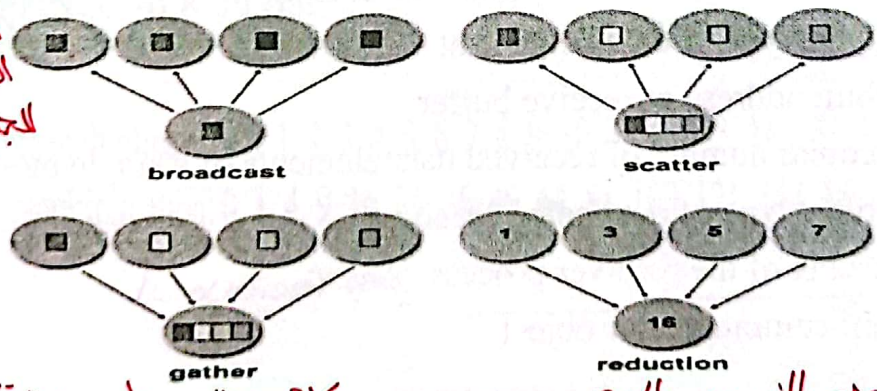
النوع الثاني غير ال Point-to-Point
هو Special case منه.

Collective Communication Routines

- Unlike point-to-point communication, collective communication involves all processes within the scope of a communicator
- We will consider the following collective communication types:

واحد Sender والكل reciever، نفس ال data للجميع

بيعت Chunks شفناه قبل



اخترعوهم للتسهيل على المبرمج ومن أجل ال optimization using some advanced algorithms

كلهم بيعتو وواحد بيستقبل وبجمعهم

© All Rights Reserved.

gather with operation

ال root ممكن يكون 0 أو أي Process

* كلهم

"1 to all all to 1" في all to all لقدام

بغنينا عن كتابة الـ codes اللي كتبناهم ورا .

MPI Scatter Routine

في Sender
وفي reciever

MPI_Scatter (sendbuf, sendcnt, sendtype,
recvbuf, recvcnt, recvtype, root, comm)

- Sendbuf: address of send buffer
- Sendcount: number of sent data elements for a single message
- Sendtype: type of data being sent
- Recvbuf: address of receive buffer
- Recvcount: number of received data elements
- Recvtype: type of data being received
- Root: rank of the sender process الـ Sender اللي بهمني
- Comm: communicator object

© All Rights Reserved.

MPI Gather Routine

MPI_Gather (sendbuf, sendcnt, sendtype,
recvbuf, recvcnt, recvtype, root, comm)

- Sendbuf: address of send buffer
- Sendcount: number of sent data elements
- Sendtype: type of data being sent
- Recvbuf: address of receive buffer
- Recvcount: number of received data elements of a single message
- Recvtype: type of data being received
- Root: rank of the receiver process الـ recieve بهمني
- Comm: communicator object

© All Rights Reserved

CamScanner

MPI Program Example 6

A * A اضرب

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define N 16

int main(int argc, char *argv[]){
  int rank, size, chunk, i;
  int *A, *tmp;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  chunk = N / size;
  tmp = (int*) malloc(chunk * sizeof(int));
  if(rank == 0){
    A = (int*) malloc(N * sizeof(int));
    printf("A before multiplication: ");
    for(i=0; i<N; i++){
      A[i] = i;
      printf("%d, ",A[i]);
    }
  }

  MPI_Scatter(A, chunk, MPI_INT, tmp, chunk, MPI_INT, 0, MPI_COMM_WORLD);

  for(i=0; i<chunk; i++)
    tmp[i] = tmp[i] * tmp[i] ;

  MPI_Gather(tmp, chunk, MPI_INT, A, chunk, MPI_INT, 0, MPI_COMM_WORLD);

  if(rank == 0){
    printf("\nA after multiplication: ");
    for(i=0; i<N; i++)
      printf("%d, ",A[i]);
    printf("\n");
  }
  MPI_Finalize();
  return 0;
}
```

بس عند الـ Sender علشان

طيا حجم الـ chunk مو الأول

root

blocking foreach chunk

Send (فوق) (Recieve

الـ behaviour متفقين عليه ما بدهيس تخالفه .

ممنوع يتناديوا جوا if Statment لازم قدام الكل يتناديوا

© All Rights Reserved.

Compiling and Running Example 6

$ mpicc example6.c –o a.out
$ mpiexec –n 8 ./a.out

A before multiplication: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
A after multiplication: 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225,

© All Rights Reserved.

CamScanner

MPI Reduce Routine

MPI_Reduce (sendbuf, recvbuf, count, datatype, op, root, comm)

نوع الoperation ←

هون الفرق انهم تاعون الSend والrecieve ممنوع يختلفوا من ناحية الtype والCount

- Sendbuf: address of send buffer
- Recvbuf: address of receive buffer
- Count: number of sent data elements
- Datatype: type of data being sent
- Op: reduce operation
- Root: rank of the process where the final reduced value will be available
- Comm: communicator object

© All Rights Reserved.

Reduction Operation in MPI

أهم ثلاثة ←

Operation	Meaning	Datatypes
MPI_MAX	Maximum	C integers and floating point
MPI_MIN	Minimum	C integers and floating point
MPI_SUM	Sum	C integers and floating point
MPI_PROD	Product	C integers and floating point
MPI_LAND	Logical AND	C integers
MPI_BAND	Bit-wise AND	C integers and byte
MPI_LOR	Logical OR	C integers
MPI_BOR	Bit-wise OR	C integers and byte
MPI_LXOR	Logical XOR	C integers
MPI_BXOR	Bit-wise XOR	C integers and byte
MPI_MAXLOC	max-min value-location	Data-pairs
MPI_MINLOC	min-min value-location	Data-pairs

© All Rights Reserved.

CamScanner

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define N 512

int main(int argc, char *argv[]){
  int rank, size, chunk, i;
  int A[N];
  int *buff;
  int local_sum, global_sum;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  chunk = N / size;
  buff = (int*) malloc(chunk * sizeof(int));
  if(rank == 0){
    for(i=0; i<N; i++)
      A[i] = rand() % 100; // each integer is randmly picked from 0-99
  }

  MPI_Scatter(A, chunk, MPI_INT, buff, chunk, MPI_INT, 0, MPI_COMM_WORLD);

  local_sum = 0;
  for(i=0; i<chunk; i++)
    local_sum += buff[i];

  MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

  if(rank==0)
    printf("global sum is %d\n", global_sum);

  MPI_Finalize();
  return 0;
}
```

MPI Program Example 7
Parallel Sum

استقبل فيه الجواب الزماني

اعمل malloc عند ال root
بس لو كان ال code كبير جداً

© All Rights Reserved.

MPI Broadcast Routine

MPI_Bcast (buffer, count, datatype, root, comm)

نفسه تبع ال Sender

- Buffer: address of send/receive buffer
- Count: number of sent data elements
- Datatype: type of data being sent
- Root: rank of the process doing the broadcasting
- Comm: communicator object

وال recieve
لأنه نفس
ال data
بتكون. فهيك
وفرّنا
Memory.

© All Rights Reserved.

MPI Program Example 8

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define N 512

int main(int argc, char *argv[]){
  int rank, size, chunk, i;
  int A[N];
  int *buff;
  int local_sum, global_sum;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  chunk = N / size;
  buff = (int*) malloc(chunk * sizeof(int));
  if(rank == 0){
    for(i=0; i<N; i++)
      A[i] = rand() % 100; // each integer is randmly picked from 0-99
  }

  MPI_Scatter(A, chunk, MPI_INT, buff, chunk, MPI_INT, 0, MPI_COMM_WORLD);

  local_sum = 0;
  for(i=0; i<chunk; i++)
    local_sum += buff[i];

  MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

  MPI_Bcast(&global_sum, 1, MPI_INT, 0, MPI_COMM_WORLD);

  printf("process %d, global sum is %d\n", rank, global_sum);

  MPI_Finalize();
  return 0;
}
```

Global sum is broadcasted to every process

© All Rights Reserved.

Wait A Minute!

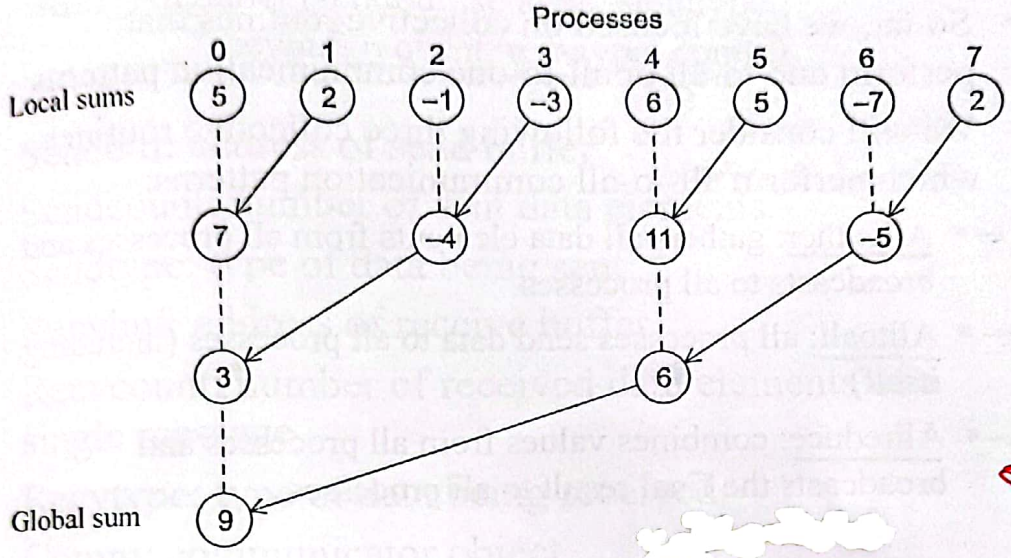
- Do we really need collective communication routines? Can't we use send and receive routines for everything?
- Collective operations are introduced to give the MPI runtime environment opportunities to **optimize common communication patterns**
- For example, MPI runtime environments use advanced algorithms (such as *tree-structured* communication algorithms) to perform collective communication routines faster
- MPI runtime environments also often try to utilize the underlying network topology in reducing communication latency

© All Rights Reserved.

CamScanner

أمثلة كيف إنهم بعملو optimize .

Tree-Structured MPI_Reduce

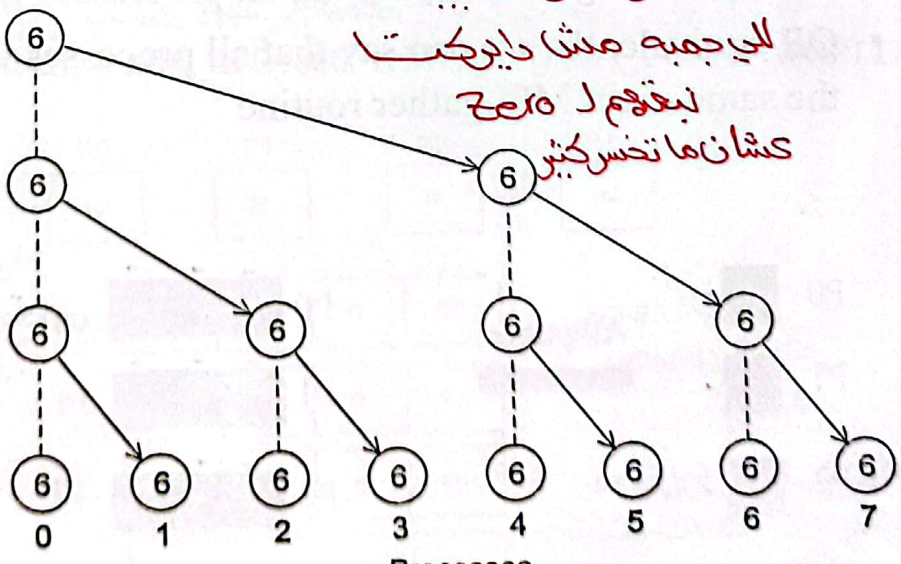


© All Rights Reserved.

Tree-Structured MPI_Bcast

P0 broadcasts 6 to P0-P7

هون صار كل واحد بيعت
للي جمبه مش رايركت
نبعتهم لـ zero
عشان ما تخسر كثير



© All Rights Reserved

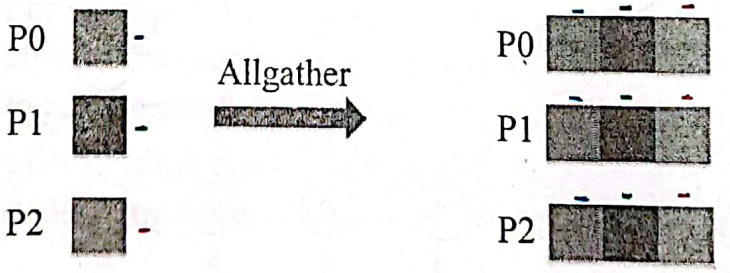
More Collective Communication Routines

- So far, we have focused on collective routines that perform one-to-all or all-to-one communication patterns
- We will consider the following three collective routines, which perform all-to-all communication patterns:
  - Allgather: gathers all data elements from all processes and broadcasts to all processes
  - Alltoall: all processes send data to all processes (including itself)
  - Allreduce: combines values from all processes and broadcasts the final result to all processes

© All Rights Reserved.

MPI Allgather Operation

- Each process gathers data from all processes
- OR equivalently, we can say that all processes will perform the same exact MPI gather routine



© All Rights Reserved.

MPI Allgather Routine

MPI_Allgather (sendbuf, sendcnt, sendtype,
recvbuf, recvcnt, recvtype, comm)

- Sendbuf: address of send buffer
- Sendcount: number of sent data elements
- Sendtype: type of data being sent
- Recvbuf: address of receive buffer
- Recvcount: number of received data elements of a single message
- Recvtype: type of data being received
- Comm: communicator object

© All Rights Reserved.

MPI Allgather Example

Assume number of processes is 4

MPI_Allgather (sbuff, 1, MPI_INT, rbuff, 1, MPI_INT, MPI_COMM_WORLD)

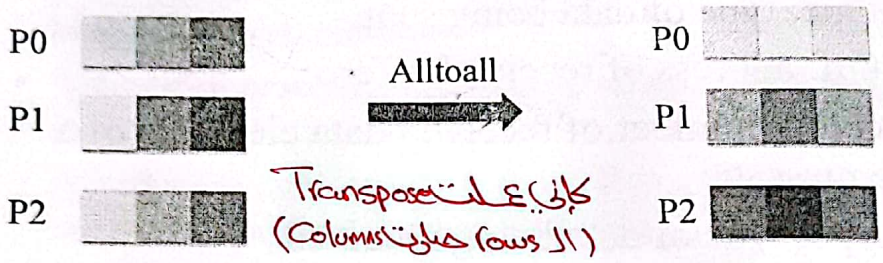
	P0	P1	P2	P3
sbuff (before the call)	10	11	0	99

rbuff (after the call)				
P0	10	11	0	99
P1	10	11	0	99
P2	10	11	0	99
P3	10	11	0	99

© All Rights Reserved.

MPI Alltoall Operation

- Each process sends a chunk of data to every other process (including itself)
- Basically, alltoall operation means every process will perform a scatter operation



© All Rights Reserved.

MPI Alltoall Routine

MPI_Alltoall (sendbuf, sendcnt, sendtype,
recvbuf, recvcnt, recvtype, comm)

- Sendbuf: address of send buffer
- Sendcount: number of sent data elements
- Sendtype: type of data being sent
- Recvbuf: address of receive buffer
- Recvcount: number of received data elements
- Recvtype: type of data being received
- Comm: communicator object

© All Rights Reserved.

MPI Alltoall Example

Assume number of processes is 4

MPI_Alltoall (sbuff, 1, MPI_INT, rbuff, 1, MPI_INT, MPI_COMM_WORLD)

sbuff (before the call)

P0	1	2	3	4
P1	11	12	13	14
P2	21	22	23	24
P3	31	32	33	34

© All Rights Reserved.

MPI Alltoall Example (cont.)

Assume number of processes is 4 → int = 4 bytes

MPI_Alltoall (sbuff, 1, MPI_INT, rbuff, 1, MPI_INT, MPI_COMM_WORLD)

rbuff (after the call)

P0	1	11	21	31
P1	2	12	22	32
P2	3	13	23	33
P3	4	14	24	34

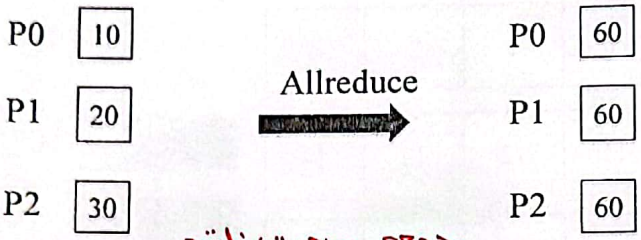
© All Rights Reserved

CamScanner

MPI Allreduce Operation

- All processes will perform the same exact MPI reduction routine

P0	10		P0	60
P1	20	Allreduce →	P1	60
P2	30		P2	60



جمعهم وبعت ناتج
الجمع للكل (موشرط جمع
بس هون كمثال)

© All Rights Reserved.

MPI Allreduce Routine

MPI_Allreduce (sendbuf, recvbuf, count, datatype, op, comm)

(مافي root) + بمجموا ال count وال datatype

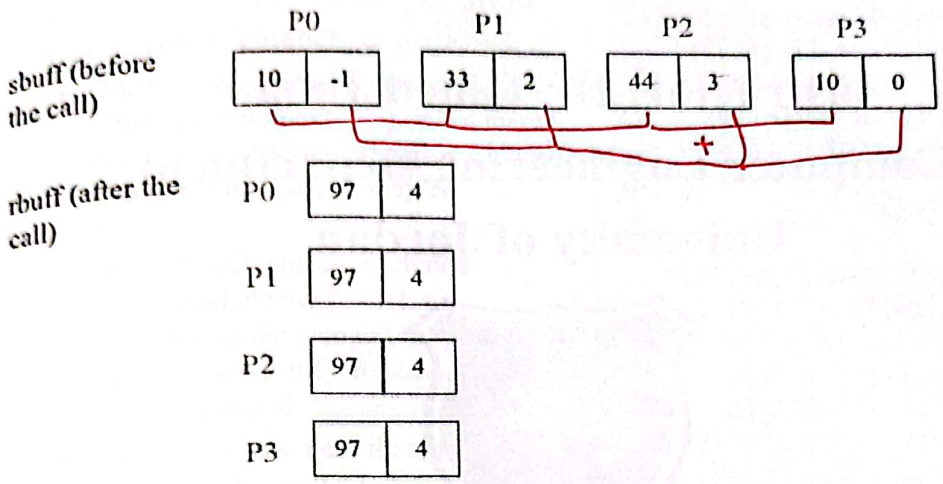
- Sendbuf: address of send buffer
- Recvbuf: address of receive buffer
- Count: number of sent data elements
- Datatype: type of data being sent
- Op: reduce operation
- Comm: communicator object

© All Rights Reserved.

CamScanner

MPI Allreduce Example

MPI_Allreduce (sbuff, rbuff, 2, MPI_INT, MPI_SUM, MPI_COMM_WORLD)



© All Rights Reserved.

Conclusions

- MPI is the dominant API for writing message-passing programs
- Programmers need to explicitly partition data and generate communication messages
- The main strategies for obtaining good performance with MPI: minimizing communication, maximizing concurrency, and ensuring load-balance
- When applicable, collective routines can generally obtain better performance than basic send/receive routines

© All Rights Reserved

CamScanner

Parallel Sort in MPI

Instructor: Dr. Fahed Jubair
Computer Engineering Department
University of Jordan

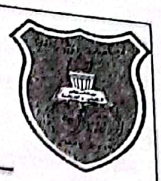
Sort Algorithms

- Sort algorithms are typical targets for parallelization due to their common use in numerical applications
- We have already studied how to write the following three parallel sort algorithms in OpenMP
  - Odd-even transportation sort
  - Merge sort
  - Quick sort
- In this lecture, we will pick two of the above algorithms (odd-even transportation sort and merge sort) and study how to write MPI programs for them

← ندرس وحدة من الطرق ولا هم الهم طرق كثير
ما الهم Standard .

© All Rights Reserved

Odd-Even Transportation Sort Serial Algorithm



As we previously learned in lecture 5, odd-even transportation sort works in **phases**, as follows:

- During "even" phases, each odd-subscripted element, $a[i]$, is compared to the element to its left, $a[i-1]$, and if they're out of order, they're swapped
- During "odd" phases, each odd-subscripted element , $a[i]$, is compared to the element to its right, $a[i+1]$, and if they're out of order, they're swapped
- A theorem guarantees that after N phases, the list will be sorted

```
for(phase=0; phase<N; phase++){
  if(phase%2 == 0){
    for(j=1; j<N; j+=2)
      if(a[j-1] > a[j]){
        tmp = a[j-1];
        a[j-1] = a[j];
        a[j] = tmp;
      }
  }
  else {
    for(j=1; j<N-1; j+=2)
      if(a[j] > a[j+1]){
        tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
      }
  }
}
```

swap بعمل مع اللي شمالي

swap بعمل مع اللي يمينه

© All Rights Reserved.

Odd-Even Sort Example

	0	1	2	3	4	5	6	7
Initial list	40	28	33	17	16	10	55	20
Phase 0	40	28	33	17	16	10	55	20
Phase 1	28	40	17	33	10	16	20	55
Phase 2	28	17	40	10	33	16	20	55
Phase 3	17	28	10	40	16	33	20	55
Phase 4	17	10	28	16	40	20	33	55
Phase 5	10	17	16	28	20	40	33	55
Phase 6	10	16	17	20	28	33	40	55
Phase 7	10	16	17	20	28	33	40	55

* عملية الـ swap بين كل رقمين بقدر أعملها in parallel

* الـ phases لازم تكون serial

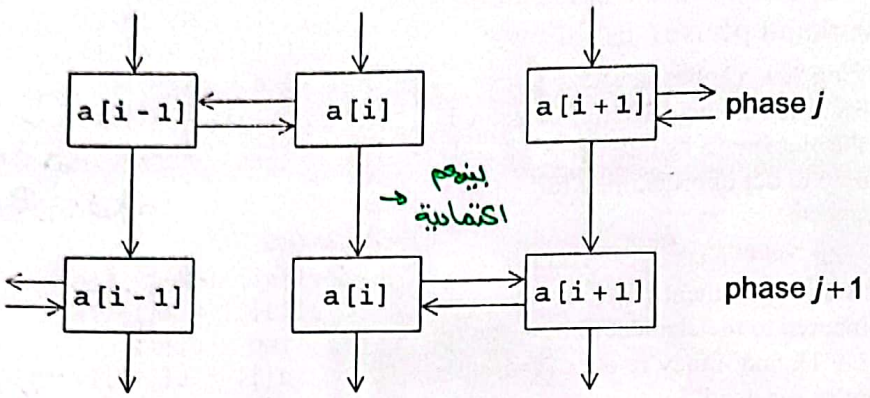
* ممكن أعملها بأقل من 8 phases

© All Rights Reserved.

CamScanner

Task Communication in Odd-Even Sort

In each phase, assuming each compare-swap represents a task



The task that's determining the value of a[i] needs to communicate with either the task determining the value of a[i-1] or a[i+1]

© All Rights Reserved.

Parallelizing Odd-Even Sort

- Assume n is the number of unsorted integers
- Assume p is the number of process
- If $n=p$, then the parallel algorithm is fairly straightforward
  - Depending on the phase, process i sends its current value, a[i], to either process $i-1$ or process $i+1$
  - At the same time, process i should receive the value stored on process $i-1$ or process $i+1$, respectively, and then decide which of the two values it should store as a[i] for the next phase
- However, the more likely scenario is that $n >> p$
  - We need to modify the parallel odd-even sort algorithm

© All Rights Reserved.

Distributed-Memory Odd-Even Sort Assumptions

chunk size

- Let us assume our algorithm will start and finish with n/p keys assigned to each process
  - Also, for simplicity, let us assume n is evenly divisible by p
- At the start, there are no restrictions on which keys are assigned to which processes
- When the algorithm terminates, the keys should be sorted in (say) increasing order such that if q and r are processes such that $q < r$, then every key in q is either less or equal to every key in r

* الفكرة الجديدة : ال Smaller بروحوا شمال و ال bigger keys بروحوا عاليمين
(مثلاً بخلي أصغر 32 عالشمال وأكبر 32 باليمين وبصير أقارن chunks مش أرقام)

© All Rights Reserved

EX : P0 ببعت كل تاعه ل P1 و P1 ببعت ل P0 بس P0 بحتفظ بالنص الأصغر و P1 الأكبر.

الشمال بعمل merge low واليمين merge high

Distributed-Memory Odd-Even Sort Pseudo Code

أي نوع عادي

عدد ال Process

```
Sort local keys;
for (phase = 0; phase < comm_sz; phase++) {
    partner = Compute_partner(phase, my_rank);
    if (I'm not idle) {
        Send my keys to partner;
        Receive keys from partner;
        if (my_rank < partner)
            Keep smaller keys;
        else
            Keep larger keys;
    }
}
```

كان الك Partner

no ل j+1 ل-1

مثلاً لو كان عالأطراف

أنا الصغير

أنا الكبير

لإيجاد ال Partner 8 بحتاج
1- phase
2- rank
3- عددهم

Theorm: If the above parallel odd-even transposition sort is run with p processes, then after p phases, the input list will be sorted

© All Rights Reserved

CamScanner

Parallel Odd-Even Sort Example

Time	Process			
	0	1	2	3
Start	15, 11, 9, 16	3, 14, 8, 7	4, 6, 12, 10	5, 2, 13, 1
After Local Sort	9, 11, 15, 16	3, 7, 8, 14	4, 6, 10, 12	1, 2, 5, 13
After Phase 0	3, 7, 8, 9	11, 14, 15, 16	1, 2, 4, 5	6, 10, 12, 13
After Phase 1	3, 7, 8, 9	1, 2, 4, 5	11, 14, 15, 16	6, 10, 12, 13
After Phase 2	1, 2, 3, 4	5, 7, 8, 9	6, 10, 11, 12	13, 14, 15, 16
After Phase 3	1, 2, 3, 4	5, 6, 7, 8	9, 10, 11, 12	13, 14, 15, 16

© All Rights Reserved.

C Implementation

```
st_time = MPI_Wtime();

qsort(my_keys, chunk, sizeof(int), Compare); // built-in sort function in C

for(phase=0; phase < comm_size; phase++){
  partner = compute_partner(phase, rank, comm_size);
  if(partner != MPI_PROC_NULL) { // not idle
    MPI_Request reqs[2];
    MPI_Status stats[2];
    MPI_Isend(my_keys, chunk, MPI_INT, partner, tag, MPI_COMM_WORLD, &reqs[0]);
    MPI_Irecv(partner_keys, chunk, MPI_INT, partner, tag, MPI_COMM_WORLD, &reqs[1]);
    MPI_Waitall(2, reqs, stats);
    if(rank < partner)
      Merge_low(my_keys, partner_keys, chunk); // keep smaller keys
    else
      Merge_high(my_keys, partner_keys,chunk); // keep larger keys
  }
}

end_time = MPI_Wtime();
sort_time = end_time - st_time;
MPI_Reduce(&sort_time, &max_sort_time, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
```

© All Rights Reserved.

Compute_partner Function

```
int compute_partner ( int phase, int myrank, int comm_size){
  int partner;
  if(phase % 2 == 0) { /* even phase */
    if(myrank % 2 != 0) /* odd rank */
      partner = myrank - 1;
    else                /* even rank */
      partner = myrank + 1;
  }
  else { /* odd phase */ → عكس ال even
    if(myrank % 2 != 0) /* odd rank */
      partner = myrank + 1;
    else                /* even rank */
      partner = myrank - 1;
  }

  if(partner == -1 || partner == comm_size) →
    partner = MPI_PROC_NULL;

  return partner;
}
```

هدول حالة
اني idle

© All Rights Reserved.

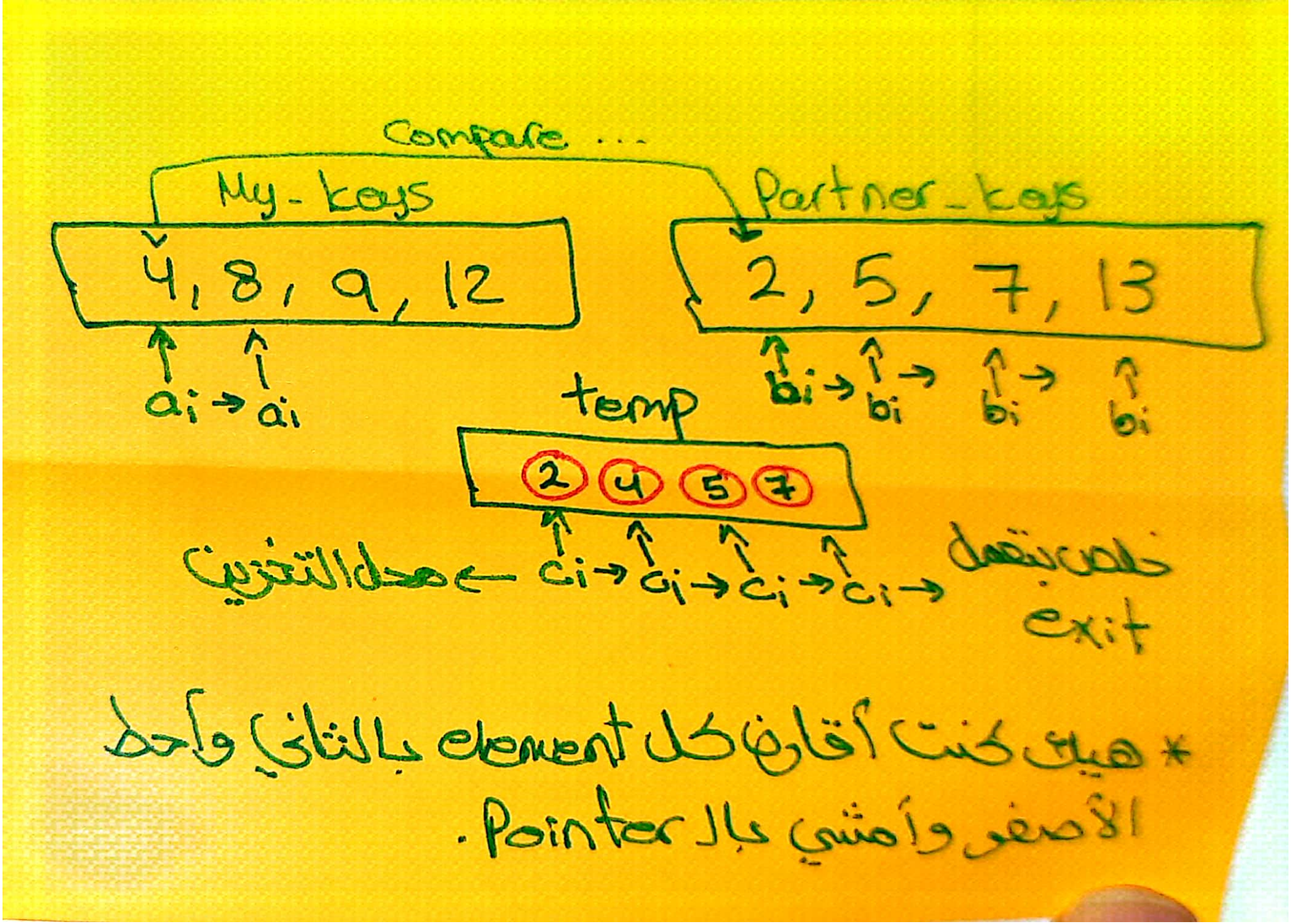
Merge_low Function

```
void Merge_low(int *myKeys, int *partnerKeys, int num_of_keys) {
  int ai, bi, ci;
  int *temp = (int*) malloc(num_of_keys * sizeof(int));
  ai = 0;
  bi = 0;
  ci = 0;
  while (ci < num_of_keys) {
    if (myKeys[ai] <= partnerKeys[bi]) {
      temp[ci] = myKeys[ai];
      ci++; ai++;
    } else {
      temp[ci] = partnerKeys[bi];
      ci++; bi++;
    }
  }
  memcpy(myKeys, temp, num_of_keys
  free(temp);
}
```

→ chunk size

→ في ال C ما في garpage collector
زي ال C++ والـ java فلازم نحطها.

© All Rights Reserved

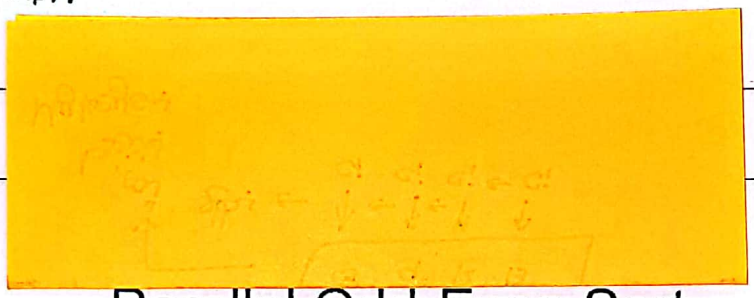


* هيك كنت أقارن كل element بالثاني واحد الأصغر وأمشي بال Pointer.

عكس ال low

Merge_high Function

```
void Merge_high(int *myKeys, int *partnerKeys, int num_of_keys) {
  int ai, bi, ci;
  int *temp = (int*) malloc(num_of_keys * sizeof(int));
  ai = num_of_keys-1;
  bi = num_of_keys-1;
  ci = num_of_keys-1;
  while (ci >= 0) {
    if (myKeys[ai] >= partnerKeys[bi]) {
      temp[ci] = myKeys[ai];
      ci--; ai--;
    } else {
      temp[ci] = partnerKeys[bi];
      ci--; bi--;
    }
  }
  memcpy(myKeys, temp, num_of_keys * sizeof(int));
  free(temp);
}
```



Parallel Odd-Even Sort

* مرات لو حالفت بعدد الـ Processes بتصير أسوأ مش أحسن

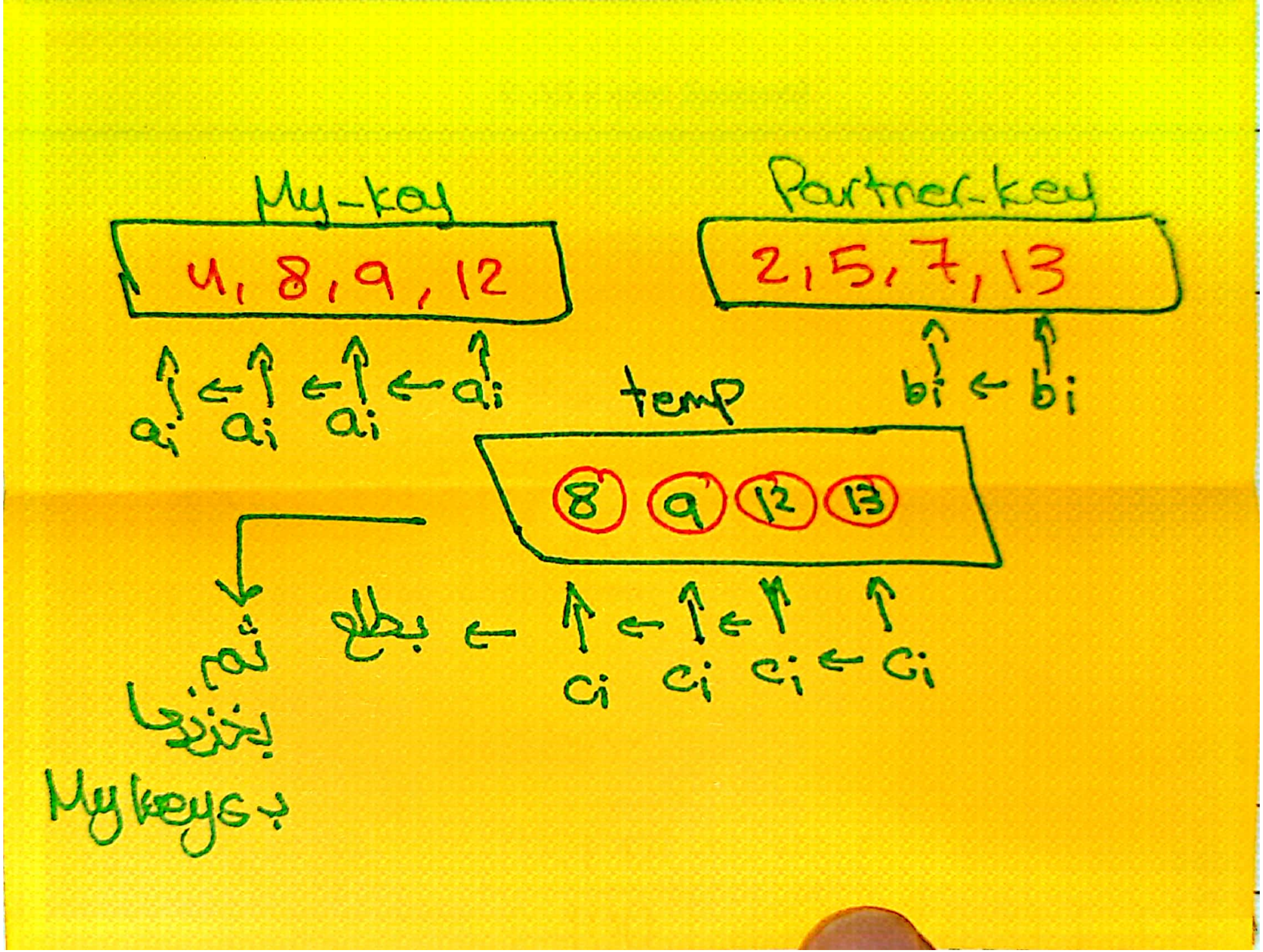
Processes	Number of Keys (in thousands)				
	200	400	800	1600	3200
1	88	190	390	830	1800
2	43	91	190	410	860
4	22	46	96	200	430
8	12	24	51	110	220
16	7.5	14	29	60	130

بطلت انزل كثير لإنه ما في Parallel كفاية فكبرنا ال input size

أفضل وحدة

(times are in milliseconds)

© All Rights Reserved.


My-key
4, 8, 9, 12
Partner-key
2, 5, 7, 13
temp
8
9
12
13
My keys:

Parallel Merge Sort

- Several algorithms have been proposed to perform merge sort on distributed-memory platforms

خسرس وحدة منهم لإنه الها أكثر من تصميم

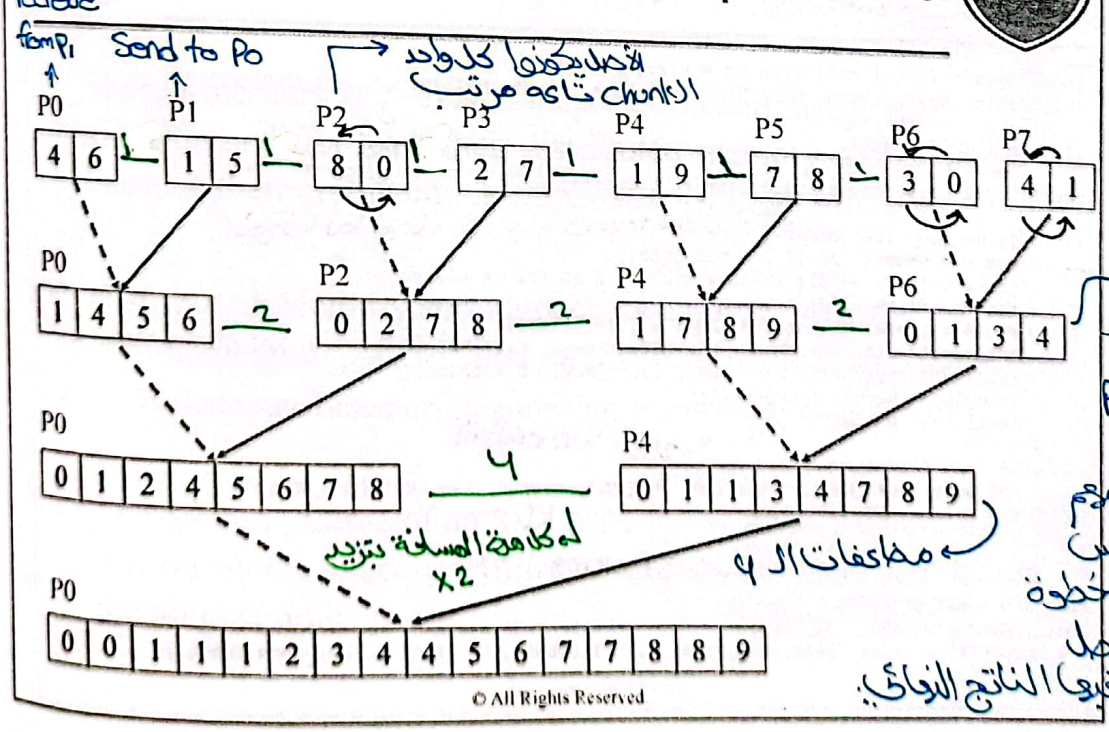
- We will consider the following algorithm:
  - Initially, process 0 has all keys, which it distributes evenly to all processes
  - Each process locally sorts its list of keys
  - Processes communicate their lists of keys using a **tree-structured communication pattern** and merge them
  - Process 0 will have the final sorted list of keys

© All Rights Reserved.

2^n processes → # of steps = n

وهكذا الباقي بأول step

Tree-Structured Merge Operations



نص الـ Processes فضيوا خلص بطلنا نستخدمهم + حنعمل نفس خطوة أول خطوة لحد ما نوصل لـ P0 فيها الناتج النهائي.

© All Rights Reserved

بفيدوني بالـ code
* أي حد بعمل Send المرة الجاي بصير idle طول وببطل موجود.
* لو كنت reciever حكون موجود المرة الجاي.

Distributed-Memory Odd-Even Sort Pseudo Code

```
Process 0 distributes keys evenly among processors ;
Sort local keys ;
steps = log2 (p) ;
for ( i = 1 ; i <= steps ; i++) {
    determine partner ;
    if (I'm not idle) {
      if ( I'am the receiver ) {
          receive partner keys ;
          merge partner keys with my keys ;
      }
      else {  // I'am the sender
          send my keys to partner ;
      }
    }
}
```

مهمة، لإنه عندي tree

هو اللي بعمل merge

© All Rights Reserved.

C Implementation

random generation for keys + calculate chunk size = n/P

في خطوات مسبقة للـ code بالتيمر موجود

تنقلي إذا أنا involved not involved

```
local_keys = (int*) malloc(chunk * sizeof(int));
MPI_Scatter(keys, chunk, MPI_INT, local_keys, chunk, MPI_INT, 0, MPI_COMM_WORLD);

st_time = MPI_Wtime();

qsort(local_keys, chunk, sizeof(int), Compare);

for(d=1; d<comm_size; d=d*2){
  if(rank % d == 0){  // if I'm not idle
    if(rank % (d*2) == 0){ // only true if I am the receiver
      int *rbuff, *mbuff;
      rbuff = (int*) malloc(d * chunk * sizeof(int));
      MPI_Recv(rbuff, d * chunk, MPI_INT, rank+d, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      mbuff = merge(rbuff, d * chunk, local_keys, d * chunk);
      free(local_keys); free(rbuff);
      local_keys = mbuff;
    }
    else { // I am the sender
      MPI_Send(local_keys, d * chunk, MPI_INT, rank-d, tag, MPI_COMM_WORLD);
    }
  }
}

end_time = MPI_Wtime();
sort_time = end_time - st_time;
MPI_Reduce(&sort_time, &max_sort_time, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
```

root

distance → بنوقف لما نوصل n

لإنه الـ Sender كيمينه

لإنه الـ reciever كشماله

كشان نختار أبطأ وحدة

© All Rights Reserved.

Merge Function

```
int* merge (int left[], int l_size, int right[], int r_size){
  int size = l_size + r_size;
  int* tmp = (int*) malloc(size * sizeof(int));
  int i, j, k;
  i = 0;
  j = 0;

  for(k=0; k<size; k++){
    if(i < l_size && j < r_size && left[i] <= right[j])
      tmp[k] = left[i++];
    else if(i < l_size && j < r_size && left[i] > right[j])
      tmp[k] = right[j++];
    else if(i < l_size)
      tmp[k] = left[i++];
    else
      tmp[k] = right[j++];
  }

  return tmp;
}
```

© All Rights Reserved.

Shared Mem ← openMP ll *
Distributed Mem programming ← MPI ll *

Summary

- Shared-memory odd-even transportation and merge sort algorithms differ from their distributed-memory counterparts because their architectures emphasize different priorities
- Distributed-Memory sort algorithms employ "smart" communication patterns that reduce communication latency
  - In distributed-memory, minimizing communication is a priority
- The execution time of an MPI program is usually determined by measuring the execution times of all processes and then taking the longest time

© All Rights Reserved.

CamScanner

GPU Programming using CUDA

Instructor: Dr. Fahed Jubair
Computer Engineering Department
University of Jordan

Review: Parallel Computing Platforms

- SIMD processors
  - All processors execute the same instruction simultaneously, but while operating on different pieces of data
  - Work best with data-parallel applications

- MIMD processors
  - All processors execute in parallel but on different instructions and different pieces of data
  - Can handle both data-parallel and task-parallel applications

© All Rights Reserved

CamScanner

مثال عال SIMD ← GPUs → coprocessor

- In late 1990s, Graphical Processor Units (GPUs) have been developed to respond to the high demand of accelerating computer graphics applications
- However, many programmers have used GPUs for other applications, introducing what is known as General Purpose Computing on GPUs, or GPGPUs
- This lead to developing of several libraries and APIs that target GPGPUs, such as OpenCL and CUDA
- In this lesson, we will study CUDA C/C++ programming

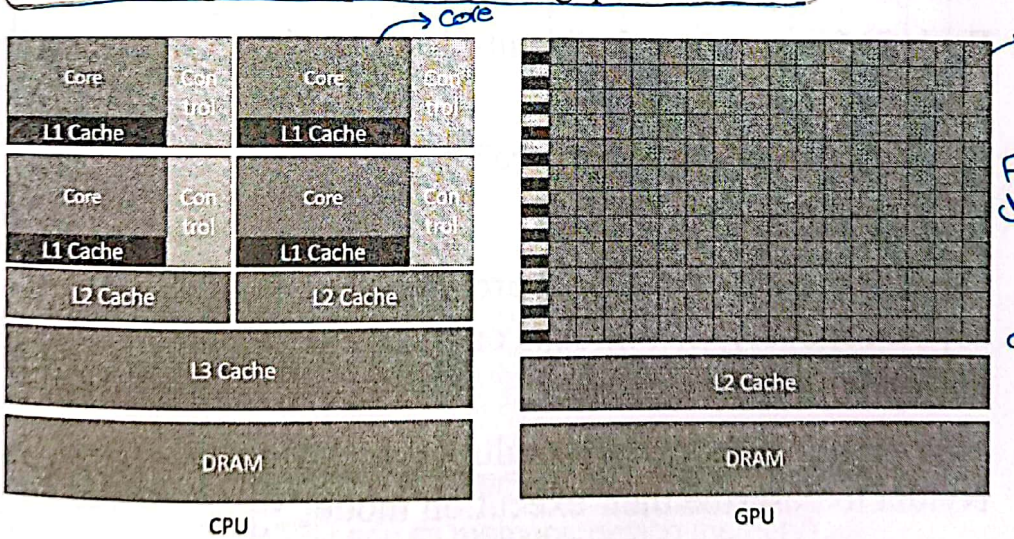
بس بتشتغل عال NVIDIA مع ذلك هي مشهورة كيش

ليستخدم للكل وبصير يحول حسب هو وين استخدم.

© All Rights Reserved.

Why GPU Computing?

* While GPUs have less programming flexibility, GPUs simply offer much higher computation throughput than CPUs



ال Core 8 هو ال Functional unit اللي بيعمل execution.

* ال graphics هي App Parallel Processing (data parallel)

© All Rights Reserved.

* core في ال GPU لا تساوي core في ال CPU.

What is CUDA?

- Introduced by Nvidia in 2006 for general purpose programming on Nvidia GPUs
- Supported by multiple programming languages, including C/C++
- CUDA Toolkit is available for download here:
  https://developer.nvidia.com/cuda-toolkit
- CUDA C/C++ programming guide is available here:
  https://docs.nvidia.com/cuda/cuda-c-programming-guide/

بتشبه openmp بس مش باستخدام Pragma .

© All Rights Reserved.

Terminology

- Nvidia GPUs are composed of multiple streaming multiprocessors, or SMs
- Each SM is composed of multiple streaming processors, or SPs
- An SP is analogous to a "core" in CPUs
- Each SP runs a "thread"
- Unlike conventional SIMD architectures, Nvidia SPs run asynchronously, i.e., threads on different SPs may execute in different speeds
- SIMT (single-instruction-multiple-thread) is a term used by Nvidia to describe their execution model

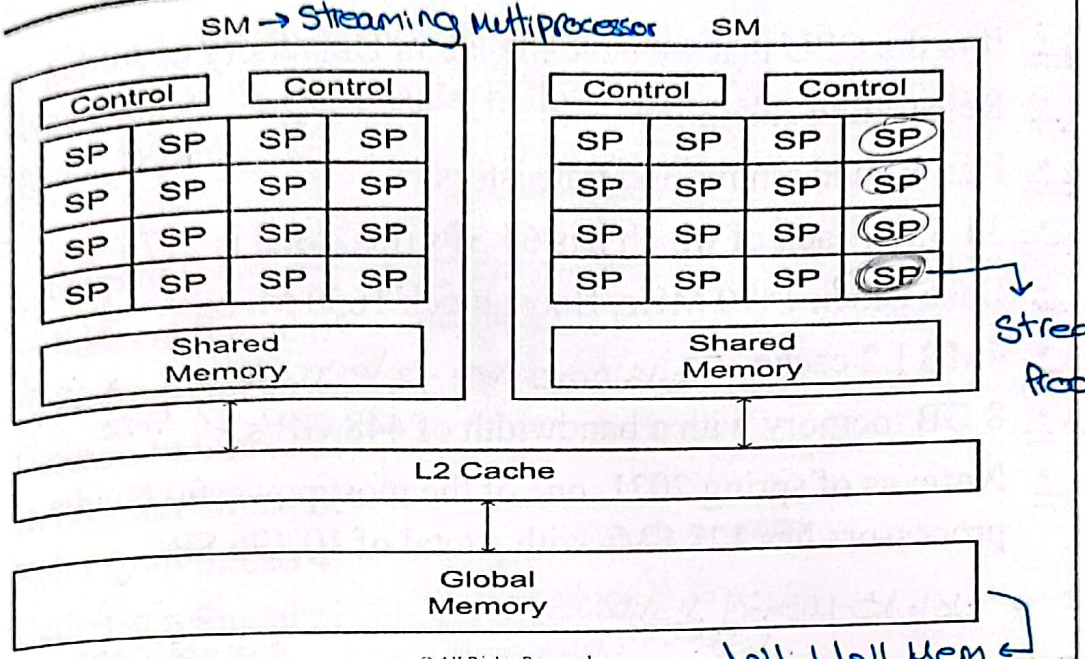
عادي الـ threads يسبقوا بعض مش نفس تصميم الـ SIMD لازم كلهم يبتنوا بعض. لكن ما زال SIMD

بحبوا يستخدموها بدل كلمة SIMD

© All Rights Reserved.

CamScanner

Nvidia GPU Architecture



Streaming Processor = Core

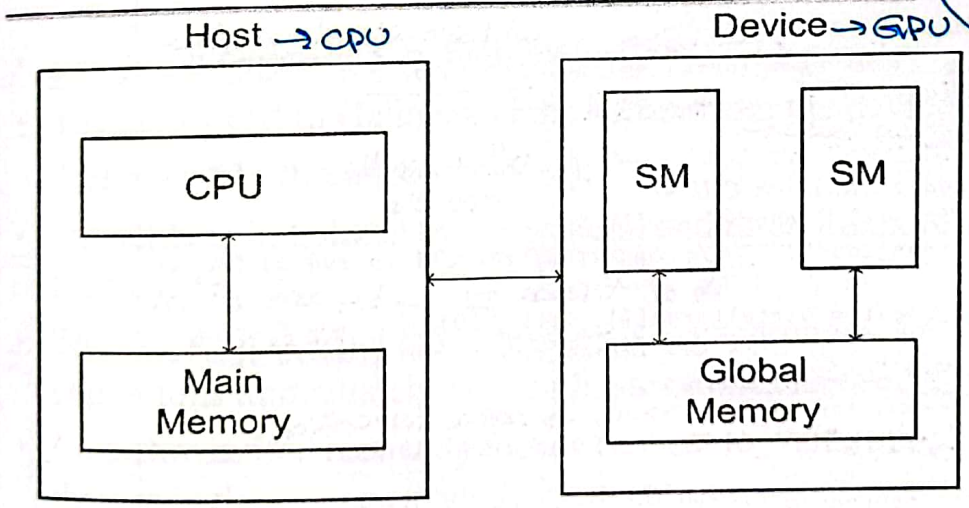
Mem الها لحالها

مش نفس تاعت ال cpu

© All Rights Reserved.

*logically → Mem ال share ممكن في بين cpu و gpu بس physically لأ.

Heterogeneous Architecture



وهاي صعوبة للمبرمج أخل أودي ال data بين Mem ال cpu وال gpu

- Terminology:
  - *Host* The CPU and its memory (host memory)
  - *Device* The GPU and its memory (device memory)

© All Rights Reserved

Example:
NVIDIA GeForce RTX 2060 SUPER

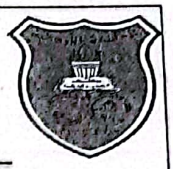
- It is the GPU that we have the lab in University of Jordan
- Released in July 2019
- Has around ten million transistors
- 34 SMs, each of which has 64 SPs (i.e., total is 2176 SPs)
- Base Clock 1470 MHz, Boost clock 1650 MHz
- 4MB L2 cache
- 8 GB memory with a bandwidth of 448 GB/s
- **Note**: as of spring 2021, one of the most powerful Nvidia processors has 128 SMs with a total of 10,496 SPs

© All Rights Reserved.

CUDA Hello Program

```
#include <stdio.h>
#include <cuda.h>   /* Header file for CUDA */

/* Device code:  runs on GPU */
__global__ void Hello(void) {
   printf("Hello from thread %d!\n", threadIdx.x);
} /* Hello */

/* Host code:  Runs on CPU */
int main(int argc, char* argv[]) {
   int thread_count;      /* Number of threads to run on GPU */

   thread_count = strtol(argv[1], NULL, 10);
                          /* Get thread_count from command line */

   Hello <<<1, thread_count>>>();
                          /* Start thread_count threads on GPU, */

   cudaDeviceSynchronize();          /* Wait for GPU to finish */

   return 0;
} /* main */
```

Compilation and Execution



- Compilation and execution commands
  nvcc cuda_hello.cu –o cuda_hello
  ./cuda_hello 6
- Output:
  Hello from thread 0!
  Hello from thread 1!
  Hello from thread 2!
  Hello from thread 3!
  Hello from thread 4!
  Hello from thread 5!

© All Rights Reserved.

A Closer Look at the Hello Program

- Execution begins in the main by the host
- The below call to Hello starts the kernel from the device:
  Hello <<< 1 , thread_count>>>();
- threadIdx is initialized by the system, and threadIdx.x returns the thread rank
- The call to the kernel from the host is asynchronous, i.e., the call returns immediately and the host resumes its execution
- CudaDeviceSynchronize() forces the host to wait until all threads in the device finish their execution

© All Rights Reserved

Threads and Blocks

- CUDA organizes threads into blocks
- A thread block (or a block) is a collection of threads that run on a single SM
- Remember that each thread runs on a single SP
- When the kernel starts, each block is assigned to an SM, and threads inside this block run on that SM
- A grid is a collection of blocks

© All Rights Reserved.

CUDA
Hello Program
Version 2

```
#include <stdio.h>
#include <cuda.h>    /* Header file for CUDA */

/* Device code:  runs on GPU */
__global__ void Hello(void) {

   printf("Hello from thread %d in block %d\n",
         threadIdx.x, blockIdx.x);
}  /* Hello */

/* Host code:  Runs on CPU */
int main(int argc, char* argv[]) {
   int blk_ct;                  /* Number of thread blocks */
   int th_per_blk;      /* Number of threads in each block */

   blk_ct = strtol(argv[1], NULL, 10);
                  /* Get number of blocks from command line */
   th_per_blk = strtol(argv[2], NULL, 10);
      /* Get number of threads per block from command line */

   Hello <<<blk_ct, th_per_blk>>>();
          /* Start blk_ct*th_per_blk threads on GPU, */

   cudaDeviceSynchronize();          /* Wait for GPU to finish */

   return 0;
}  /* main */
```

Compilation and Execution

- Compilation and execution commands

nvcc cuda_hello.cu –o cuda_hello

./cuda_hello 2 3

عدد ال block ← ← عدد ال thread

كل واحد من ال block بياخد SM بطريقة randomly

* عدد ال thread وال core مو ضروري يتشابهوا لكن لو كان ال thread أكثر حيصير في Serialization

- Output:

Hello from thread 0 in block 0
Hello from thread 1 in block 0
Hello from thread 2 in block 0
Hello from thread 0 in block 1
Hello from thread 1 in block 1
Hello from thread 2 in block 1

هون ما بتميز ال thread من رقمه بس ، بتميزه من رقمه ورقم ال block

مش شرط block تطلع قبل block1

Ex: عندي 64 thread و 32 core ← أول 32 thread بشتغلوا بس بخلصوا بشتغلوا ال 32 اللي بعدهم (ما بوديهم تلقائيًا ع SM ثانية)

© All Rights Reserved.

Grids

ما حنستعملها هون بس لازم نعرفها ←

* CUDA كبيرة وفيها كثير تفاصيل.

- In addition to threads and blocks, CUDA also allows grids
- A grid is a collection of blocks, and a block is a collection of threads

thread → block → grid

- Built-in variables:
  - threadIdx: the rank of a thread
  - blockDim: the dimensions of a thread block
  - blockIdx: the rank of a block
  - gridDim: the dimensions of a grid

عدد ال threads جوا ال block

عدد ال blocks جوا ال grid.

في المثال فوق = 3

All of these variables are structs with three integer fields: x, y, z

كل ال block فيهم عدد thread ثابت نفس الكل

لو ما استخدمته by default بحطهم بـ grid واحد

* ال grid حاليًا ما حنشتغل فيهم ومش دايمًا بحتاجها.

© All Rights Reserved.

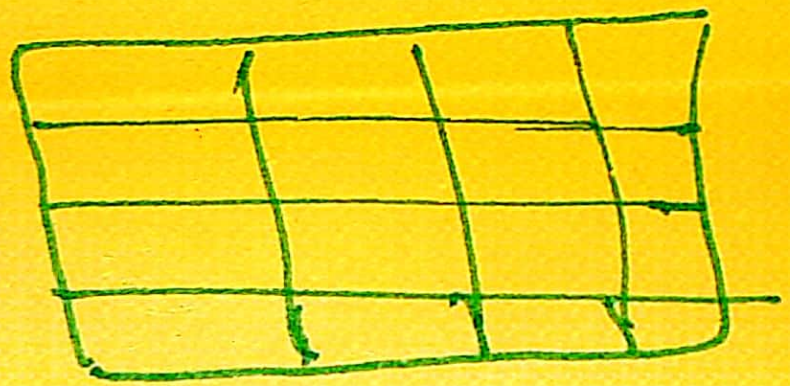
* بتقدر تحول مثلاً 6-D Array لـ 3-D وهكذا. ال CUDA بتتحمل 3-D مش أكثر (1-D/2-D/3-D)

* الـ multidimension Array هو كمنظر الك يالأصل بتخزن 1-D Array ك.

Ex: `int a[1000];`

1-D

`int a[100][100];`



2-D

بس كمنظر الك لكن بوا بتخزن ك 1-D.

فقط من أجل التنظيم

Initializing Grids

كتبتهم

كـ Array 3-D مش 1-D

عدد الـblocks الموجودين

```
dim3 grid_dims, block_dims;
grid_dims.x = 2;
grid_dims.y = 3;
grid_dims.z = 1;
block_dims.x = 4;
block_dims.y = 4;
block_dims.z = 4;
...
Kernel <<<grid_dims, block_dims>>> (...);
```

عندي 3grid حاط 2 blocks بـ x و 3 بـ y و 1 بـ z = 6

لكل block في الـ thread عدد معين

كل شغلنا ما حيكون عهاي حيكون زي Slide (Hello ver.2) لإنه ما بنستخدم الـ grid.

This kernel will have 2 x 3 x 1 = 6 blocks, each of which has 4 x 4 x 4 = 64 threads

بس رتبتهم كـ 3-D

© All Rights Reserved.

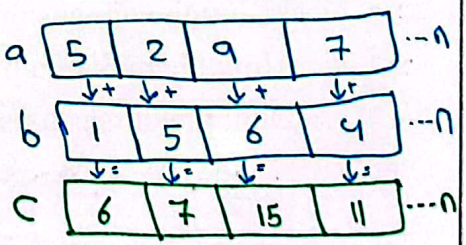
Vector Addition

- GPUs are designed to be effective with data-parallel applications
- Let us consider the CUDA version of performing simple addition of two *n*-integer vectors
- Below is the serial version of the code

*addition between 2 elements → parallel

for ($i = 0$; $i < n$; $i++$) →

$c[i] = a[i] + b[i]$;



* في الـ cpu كنا نحتاج نعمل chunks ، في الـ Gpu ما بنحتاج chunks ، بنعمل عدد الـ thread = n وبنشغلهم Parallel

© All Rights Reserved.

* إلا لو كان عددها تل جدا جدا بصير استخدم chunks.

Vector Addition Kernel

```
__global__ void vec_add(const int x[], const int y[], int sum[], int n)
{
  int my_element = blockDim.x * blockIdx.x + threadIdx.x ;

  if (my_element < n)
     sum[my_element] = x[my_element] + y[my_element] ;

}
```

- Each thread will perform the addition on exactly one element
- The index of the element is the same as the global rank of the thread (see next slide)

* If I have n-elements → create n-threads such that:
thread0 with add elements 0 ⟹ Thread i → element i
thread 1 with add elements 1

© All Rights Reserved.

→ --global-- void vec_add (const int x[], const int y[], int sum[], int n)
{ sum[threadID.x] = x[threadID.x] + y[threadID.x]; }

Element Index

Vec_add<<<1, n >>>(x,y,z,n) ← 0 → n-1

- blockdim.x returns the number of threads

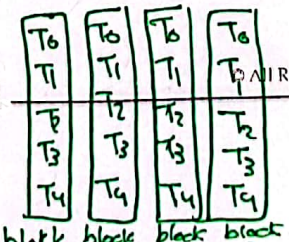
n = 20

* شرح السطر الثاني بال Code

Table 6.4 Global thread ranks or indexes in a grid with 4 blocks and 5 threads per block.

	threadIdx.x				
blockIdx.x	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19

* N = 20 → بدي الthread يكونوا 20 برضه
4 blocks with 5 threads/blk ← فبدي



© All Rights Reserved.

→ Element Index =
5*blk_id + thred_id

بس ناقصني ال i0 element + e0 لـ e0
e4 ← e0 i0 ← block =0 block =1 block =2 block =3 → e5 → e9

SM كل
فيه 64
core فقط
مثلاً لو بدك
thread عدد 1024
ما حيقدر يشغلهم كلهم
in Parallel
حيصير بالتدريج ياخذ 64 ← 64 ← ..
و ال SM الثانيين فاضيين بعملوش اشي
فهذا الحل مو منيح كتير انك تخلي
1-block فقط ، مثلا فيه 32
32 blocks thread بس بتحسين الصعوبة
لمعرفة مكان ال element ، ال code
الي فوق بوضح.

هاي أفضل من السلايد المقابل

لأنه بقلل الصعوبة

The Main Function

```
int main(int argc, char* argv[]){
    int *a, *b, *c;
    int i;

    //allocate vectors on unified memory
    cudaMallocManaged(&a, n*sizeof(int));
    cudaMallocManaged(&b, n*sizeof(int));
    cudaMallocManaged(&c, n*sizeof(int));

    // initlize a and b with random integers
    initialize_arrays (a, b);

    /// perform vec_add on GPU
    vec_add <<<blk_cnt, thrd_per_blk>>> (a, b, c, n);

    cudaDeviceSynchronize();

    //print content of c
    print(c);
    for (i=0; i<n, i++)
        Printf("%d", c[i]);
    //free memory
    cudaFree(a);
    cudaFree(b);
    cudaFree(c);
}
```

unified Mem

Call by refrence

Allocate a, b & c on "unified memory" (see next slide)

عال cpu

عال gpu

حسبتوم بتعتمد زي ماتعلمنا بال Slide قبل

Pintf ("\n");

* ما في Physical Mem Sharing بين cpu و Gpu لا تنسي. فلازم نكتب بال Code Memory transfer ولكن طلعت فكرة ال unified Mem اللي بتخليك تتخيل إنه في هاي المنطقة ال Gpu وال cpu بتشاركوا بال Mem ، ال Compiler تاع cuda هو اللي بعمل لهاي المنطقة ال Code transfer

Unified Memory

- Host and Device have separate physical memories
- Unified memory is a new addition to CUDA that allows programmers to develop programs as if the host and the device share a physical memory
- Underneath, the system takes care of data transfer between the host memory and the decide memory
- Programmers may choose to allocate memory specifically on the host or the device and to do the data transfer between memories by hand
  - Useful for performing optimization

© All Rights Reserved.

CamScanner

* Malloc &

1- int* a

a = malloc(N*sizeof(int));] allocate on GPU Mem only

2- cudaMalloc /// allocate on gpu Mem only

3- cudaMallocManaged /// allocate on unified Memory

CamScanner

Rewriting Vector Addition with Explicit Memory Transfer

- No changes on the kernel

بتضطر
اعمل transfer
(اعملناها كتجربة لو ما استخدمنا ال unified)

```
__global__ void vec_add(const int x[], const int y[], int sum[], int n)
{
  int my_element = blockDim.x * blockIdx.x + threadIdx.x ;

  if (my_element < n)
     sum[my_element] = x[my_element] + y[my_element] ;
}
```

© All Rights Reserved.

The Main Function

```
int main(int argc, char* argv[]){
  int *ha, *hb, *hc;   // host vectors
  int *da, *db, *dc;   // device vectors
  int i;

  //allocate vectors on host memory
  ha = (int*) malloc(n*sizeof(int));
  hb = (int*) malloc(n*sizeof(int));
  hc = (int*) malloc(n*sizeof(int));

  //allocate vectors on device memory
  cudaMalloc(&da, n*sizeof(int));
  cudaMalloc(&db, n*sizeof(int));
  cudaMalloc(&dc, n*sizeof(int));

  // initlize host vectors a and b with random integers
  initlize_arrays(ha, hb);

  // copy content from host to device
  cudaMemcpy(da, ha, n*sizeof(int), cudaMemcpyHostToDevice);
  cudaMemcpy(db, hb, n*sizeof(int), cudaMemcpyHostToDevice);

  /// perform vec_add on GPU
  vec_add <<<blk_cnt, thrd_per_blk>>> (da, db, dc, n);
  // copy content back from device to host
  cudaMemcpy(hc, dc, n*sizeof(int), cudaMemcpyDeviceToHost);

  //print content of host vector c
  print(hc);

  //free memory on host and device
  cudaFree(da);
  cudaFree(db);
  cudaFree(dc);
  free(ha);
  free(hb);
  free(hc);
}
```

cpu

gpu

→ ماحد بقدر يشوف الثاني

destination ← ↳ source

→ cudaDeviceSynchronize();

→ بعمل copy من h وبحطه بال d

→ بعمل copy من D وبحطه بال h

Summary

- CUDA is designed to deliver high performance computing on Nvidia GPUs for data-parallel applications

  it becomes challenging

- We covered basics concepts of CUDA

- We will learn more about CUDA in the lab

*ما بنقدر نحكي ال gpu أفضل دائمًا أو ال cpu أفضل دائمًا
حسب المشكلة بنحدد.

© All Rights Reserved.

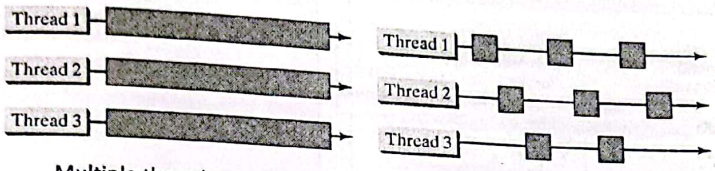
Multithreading Programming in Java

Instructor: Dr. Fahed Jubair
Computer Engineering Department
University of Jordan

CamScanner

Motivation

- Multithreading enables multiple tasks in a program to be executed concurrently
- A thread is the flow of execution, from beginning to end, of a specific task
- The operating system is responsible for scheduling and allocating resources to threads



Multiple threads running on multiple CPUs

Multiple threads share a single CPU (aka. time sharing)

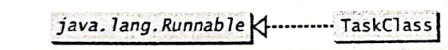
Source for this lecture: Introduction to Java Programming, 10th edition

© All rights reserved.

نفس مفهوم ال openMp ولكن ال execution يختلف.

Multithreading in Java

interface (أي task بتعملها implements بدله)




```
// Custom task class
public class TaskClass implements Runnable {
  ...
  public TaskClass(...) {
    ...
  }

  // Implement the run method in Runnable
  public void run() {
    // Tell system how to run custom thread
    ...
  }
  ...
}
```

```
// Client class
public class Client {
  ...
  public void someMethod() {
    ...
    // Create an instance of TaskClass
    TaskClass task = new TaskClass(...);

    // Create a thread
    Thread thread = new Thread(task);

    // Start a thread
    thread.start();
    ...
  }
  ...
}
```

لازم تكون

© All rights reserved.

Startup Example

```
public class TaskThreadDemo {
  public static void main(String[] args) {
    // Create tasks
    Runnable printA = new PrintChar('a', 100);
    Runnable printB = new PrintChar('b', 100);
    Runnable print100 = new PrintNum(100);

    // Create threads
    Thread thread1 = new Thread(printA);
    Thread thread2 = new Thread(printB);
    Thread thread3 = new Thread(print100);

    // Start threads
    thread1.start();
    thread2.start();
    thread3.start();
  }
}
```

→ مون ببلش

ال thread يشتغل ويكون جاهز لل run لما يجي وقت ينعمله run

* ما ننسى إنه في واحد رابع الأصلي ال main thread غير ال 3 هدول.

```
class PrintChar implements Runnable {
  private char charToPrint; // The character to print
  private int times; // The times to repeat
  public PrintChar(char c, int t) {
    charToPrint = c;
    times = t;
  }

  @Override
  public void run() {
    for (int i = 0; i < times; i++) {
      System.out.print(charToPrint);
    }
  }
}
```

```
class PrintNum implements Runnable {
  private int lastNum;
  public PrintNum(int n) {
    lastNum = n;
  }

  @Override
  public void run() {
    for (int i = 1; i <= lastNum; i++) {
      System.out.print(" " + i);
    }
  }
}
```

بستهلك ال cpu عالفاضي

© All rights reserved.

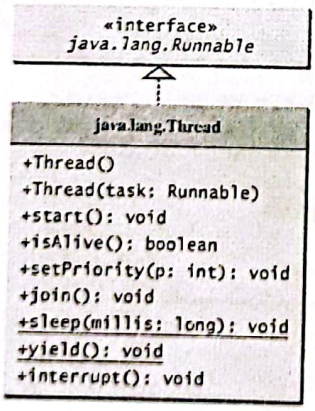
* لو بدي أجبر حد ينتظر حد ليخلص :

1- while (thread.isAlive()) { } // busy wait loop

2- thread.join(); → أقل cpu cycles من ال while

The Thread Class

- The Thread class contains the constructors for creating threads for tasks and methods for controlling threads.



```
«interface»
java.lang.Runnable
```

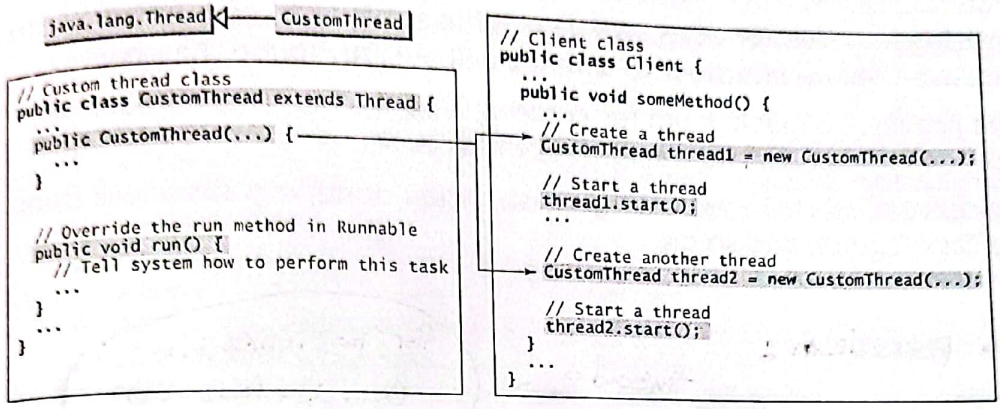
```
java.lang.Thread
+Thread()
+Thread(task: Runnable)
+start(): void
+isAlive(): boolean
+setPriority(p: int): void
+join(): void
+sleep(millis: long): void
+yield(): void
+interrupt(): void
```

ال thread وال Class بنخلقوا يا Separitty يا Couple (باستخدام ال thread مش ال runnable)

Creates an empty thread.
Creates a thread for a specified task.
Starts the thread that causes the run() method to be invoked by the JVM.
Tests whether the thread is currently running.
Sets priority p (ranging from 1 to 10) for this thread.
Waits for this thread to finish.
Puts a thread to sleep for a specified time in milliseconds.
Causes a thread to pause temporarily and allow other threads to execute.
Interrupts this thread.

CamScanner

Alternative Method For Running Threads



© All rights reserved.

Sleep

- Sometimes a thread has nothing to do for some period of time.
- It can relinquish its time on a processor by using sleep(ms).
- This can be useful:
  - to simulate the passage of time.
  - to let other processes work while this process waits.

Example Thread Test

```
class myThread extends Thread {

   private String threadName;
   myThread( String name) {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }

   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep( millis: 50); → Delay
         }
      } catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
      }
      System.out.println("Thread " +  threadName + " exiting.");
   }
}

public class TestThread {
   public static void main(String args[]) {
      myThread R1 = new myThread(  name: "Thread-1");
      R1.start();
      myThread R2 = new myThread(  name: "Thread-2");
      R2.start();
   }
}
```

```
// output
Creating Thread-1
Creating Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

© All rights reserved.

CamScanner

Task Parallism :

```
for(i=0;i<n;i++)
    Sum[i] = A[i] + B[i]
    Diff[i] = A[i] - B[i]
```

Thread 0 → `for(i=0;i<n;i++) Sum[i] = A[i] + B[i]`

Thread 1 → `for(i=0;i<n;i++) Diff[i] = A[i] - B[i]`

Thread Pool

- In many programs, many tasks need to be expected in parallel
- Creating a thread for each task if not efficient for a large number of tasks because the large number of threads will oversubscribe the system
- Java provides a mechanism for creating a pool of specific number of threads for executing large number of tasks
- Each thread starts by executing a task, when done, it grabs a task from the Task Queue, and so on

Data Parallism :

```
for(i=0;i<n;i++)
    Sum[i] = A[i] + B[i]
    Diff[i] = A[i] - B[i]
```

thread 0 → `for(i=0;i<n/2;i++) Sum[i] = A[i] + B[i] Diff[i] = A[i] - B[i]`

thread 1 → `for(i=n/2;i<n;i++) Sum[i] = A[i] + B[i] Diff[i] = A[i] - B[i]`

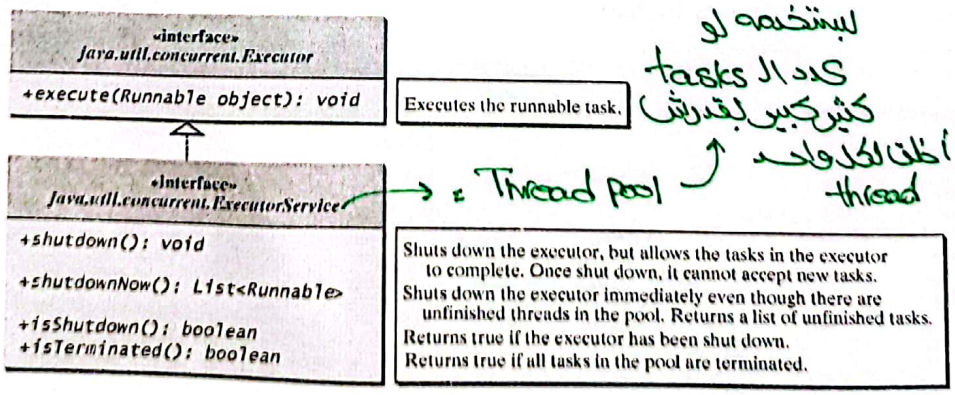
© All rights reserved.

*Anonymous

```
Thread ti = new Thread(new Runnable() {
    ....
});
```

The Executor and ExecutorService Interfaces

- Java provides the Executor interface for executing tasks in a thread pool and the ExecutorService interface for managing and controlling tasks
- ExecutorService is a subinterface of Executor



The Executers Class

- The Executers class has static methods useful for creating an Executor object
- There are two static methods for creating two different types of thread pools:
  - *newFixedThreadPool(int)*
    - Creates a fixed number of threads in a pool
    - If a thread completes executing a task, it can be reused to execute another task
    - If a thread terminates due to a failure prior to shutdown, a new thread will be created to replace it if all the threads in the pool are not idle and there are tasks waiting for execution
  - *newCachedThreadPool()*
    - Creates a new thread if all the threads in the pool are not idle and there are tasks waiting for execution
    - A thread in a cached pool will be terminated if it has not been used for 60 seconds
    - Efficient for many short tasks

هو اللي بخلقلك
مش انت وبحطهم
بعمق وكل اشي -

© All rights reserved.

Example Executor Demo

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class ExecuterDemo
{
    public static void main(String[] args)
    {
        // pool creation
        ExecutorService executor = Executors.newFixedThreadPool( nThreads: 8);

        // tasks' execution
        for(int i=0; i<1000; i++)
            executor.execute(new Task(i));

        // pool shutdown
        executor.shutdown();
    }
}

class Task implements Runnable {
    private int taskNumber;
    public Task(int taskNumber) {
        this.taskNumber = taskNumber;
    }
    @Override
    public void run() {
        System.out.println("task " + taskNumber + " is executed.");
    }
}
```

اختاره
ملاهم
لل
hardware
Machine
تبعتك ولعدد
ال cores .

مايقدر
أخلق
1000
thread
فبستخدم
thread pool

لـ زي ال join

نستنى يخلص كل ال task

الاشي الوحيد المطلوب
منك تتأكد إنه ال task
independent

© All rights reserved.

CamScanner

Race Condition

- During concurrent execution, multiple threads may share data
- Reading and writing to shared data by multiple threads simultaneously may lead to inconsistent update of data (i.e., threads may incorrectly override each other)
- Such a situation when data is corrupted due to concurrent thread execution is called a **race condition**
- Race conditions cause incorrectness in concurrent execution

© All rights reserved.

Example: Race Condition Demo

```
import java.util.concurrent.*;

public class AccountWithoutSync {
  private static Account account = new Account();

  public static void main(String[] args) {
    ExecutorService executor = Executors.newCachedThreadPool();

    // Create and launch 100 threads
    for (int i = 0; i < 100; i++) {
      executor.execute(new AddAPennyTask());
    }

    executor.shutdown();

    // Wait until all tasks are finished
    while (!executor.isTerminated()) { }

    System.out.println("What is balance? " + account.getBalance());
  }

  // A thread for adding a penny to the account
  private static class AddAPennyTask implements Runnable {
    public void run() {
      account.deposit(1);
    }
  }

  // An inner class for account
  private static class Account {
    private int balance = 0;

    public int getBalance() {
      return balance;
    }

    public void deposit(int amount) {
      int newBalance = balance + amount;

      // This delay is deliberately added to magnify the
      // race condition problem and make it easy to see.
      try {
        Thread.sleep(5);
      }
      catch (InterruptedException ex) {
      }

      balance = newBalance;
    }
  }
}
```

© All rights reserved.

Thread Synchronization and Critical Regions

- **Thread synchronization** is a mechanism for controlling access to shared data between multiple threads so that race conditions are avoided
- Example of a thread synchronization mechanism is a **critical region**, which is a region of code that allows one thread to execute this region at a time
- Critical regions are typically used whenever a specific region in the code needs to be executed serially

© All rights reserved.

The *Synchronized* Keyword

Special case
لما أكون الـ method اللي بدي استخدموها هي سبب كل المشكلة وكلها مسؤولة عن سبب المشكلة

- In Java, the synchronized keyword is provided to specify a critical regions
- (1) Using the synchronized keyword with a method will make this entire method executed as a critical region
  - Example: in the previous example, use "public synchronized void deposit(int amount)" to declare the deposit method as a critical region
- (2) Using the synchronized keyword with statements make them executed inside a critical region
  - Example, in the previous example, write the below code to make a critical region

```
synchronized (account) {
    account.deposit(1);
}
```

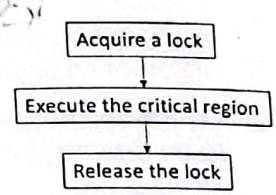
منحدد فيه الـ object المسؤول عن الـ race condition

© All rights reserved.

CamScanner

③ Synchronization Using Locks

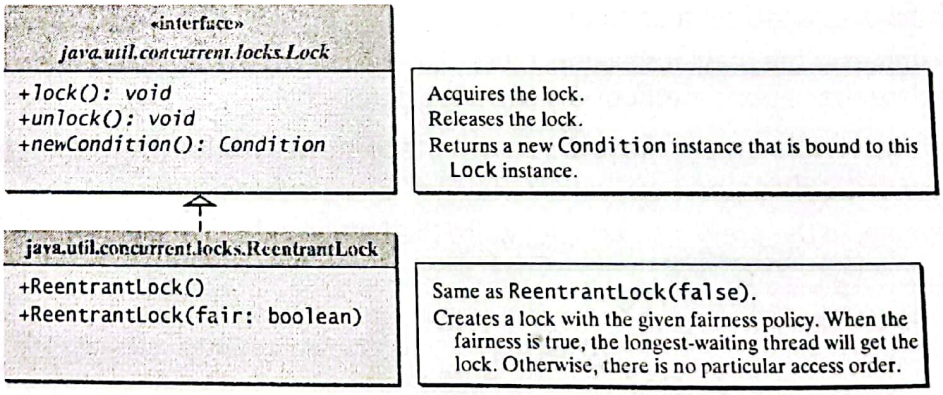
- Locks are mechanisms for implementing critical regions
- Only a thread that has the lock is allowed to enter and execute the critical region
- The *synchronized* keyword uses the lock/unlock mechanism for implementing a critical region
- However, Java also enables acquiring and releasing locks explicitly to allow users more thread control



© All rights reserved.

The Lock Interface and The ReentrantLock Class

- ReentrantLock is a concrete implementation of Lock for creating mutually exclusive locks



© All rights reserved.

CamScanner

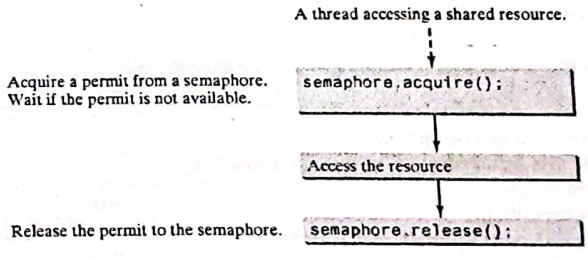
Example: Lock/Unlock Demo

```
import java.util.concurrent.*;
import java.util.concurrent.locks.*;

public class AccountWithSyncUsingLock {
  private static Account account = new Account();

  public static void main(String[] args) {
    ExecutorService executor = Executors.newCachedThreadPool();

    // Create and launch 100 threads
    for (int i = 0; i < 100; i++) {
      executor.execute(new AddAPennyTask());
    }

    executor.shutdown();

    // Wait until all tasks are finished
    while (!executor.isTerminated()) { }

    System.out.println("What is balance ? " + account.getBalance());
  }

  // A thread for adding a penny to the account
  public static class AddAPennyTask implements Runnable {
    public void run() {
      account.deposit(1);
    }
  }

  // An inner class for account
  public static class Account {
    private static Lock lock = new ReentrantLock(); // Create a lock
    private int balance = 0;

    public int getBalance() {
      return balance;
    }

    public void deposit(int amount) {
      lock.lock(); // Acquire the lock

      try {
        int newBalance = balance + amount;
        Thread.sleep(5);
        balance = newBalance;
      }
      catch (InterruptedException ex) {
      }
      finally {
        lock.unlock(); // Release the lock
      }
    }
  }
}
```

© All rights reserved.

Semaphores

- Semaphores are mechanisms that can control the number of threads that can enter a critical region
- Before entering a critical region, a thread must acquire a permit from a semaphore
- After finishing the critical region, a thread returns the permit to the semaphore object



© All rights reserved.

CamScanner

The Semaphore Class

java.util.concurrent.Semaphore	
+Semaphore(numberOfPermits: int)	Creates a semaphore with the specified number of permits. The fairness policy is false.
+Semaphore(numberOfPermits: int, fair: boolean)	Creates a semaphore with the specified number of permits and the fairness policy.
+acquire(): void	Acquires a permit from this semaphore. If no permit is available, the thread is blocked until one is available.
+release(): void	Releases a permit back to the semaphore.

© All rights reserved.

Example: Semaphore Demo

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Semaphore;

public class AccountWithSyncUsingSemaphores {
  private static Account account = new Account();

  public static void main(String[] args) {
    ExecutorService executor = Executors.newCachedThreadPool();

    // Create and launch 100 threads
    for (int i = 0; i < 100; i++) {
      executor.execute(new AddAPennyTask());
    }

    executor.shutdown();

    // Wait until all tasks are finished
    while (!executor.isTerminated()) { }

    System.out.println("What is balance ? " + account.getBalance());
  }

  // A thread for adding a penny to the account
  public static class AddAPennyTask implements Runnable {
    public void run() {
      account.deposit(1);
    }
  }
```

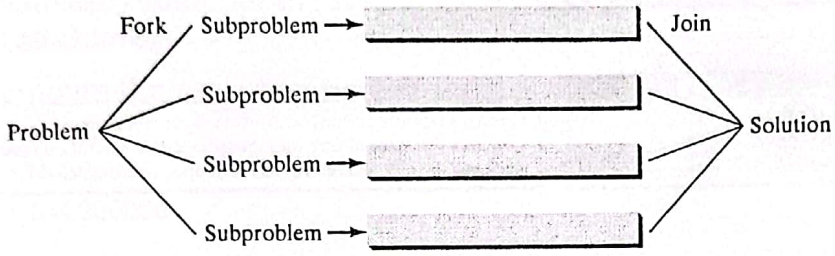
© All rights reserved.

```
  // An inner class for account
  public static class Account {
    private static Semaphore semaphore = new Semaphore(1);
    private int balance = 0;

    public int getBalance() {
      return balance;
    }
    public void deposit(int amount) {

      try {
        semaphore.acquire();
        int newBalance = balance + amount;
        Thread.sleep(5);
        balance = newBalance;
      }
      catch (InterruptedException ex) {
      }
      finally {
        semaphore.release();
      }
    }
  }
}
```

CamScanner

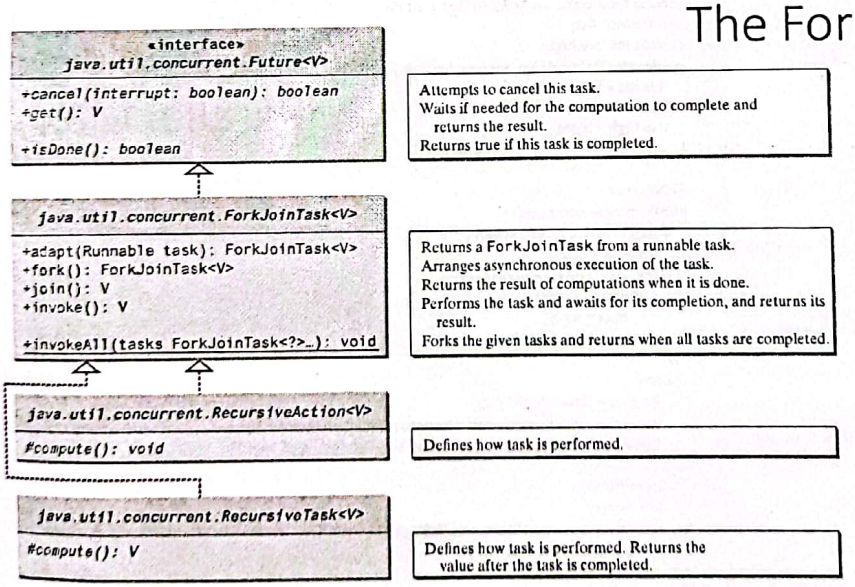
The Fork/Join Model

- In Java, the fork/join model is used for parallel programming
- This is achieved by defining the ForkJoinTask and ForkJoinPool classes



© All rights reserved.

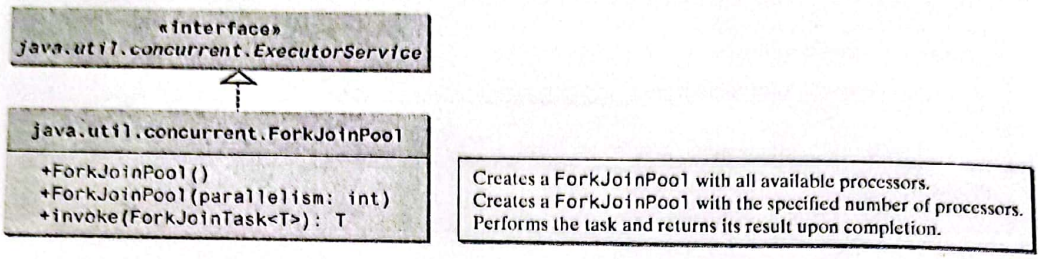
The ForkJoinTask Class



© All rights reserved.

CamScanner

The ForkJoinPool Class



© All rights reserved.

Example Parallel Max

```
import java.util.concurrent.*;

public class ParallelMax {
  public static void main(String[] args) {
    // Create a list
    final int N = 9000000;
    int[] list = new int[N];
    for (int i = 0; i < list.length; i++)
      list[i] = i;

    long startTime = System.currentTimeMillis();
    System.out.println("\nThe maximal number is " + max(list));
    long endTime = System.currentTimeMillis();
    System.out.println("Number of processors is " +
        Runtime.getRuntime().availableProcessors());
    System.out.println("Time with " + (endTime - startTime)
        + " milliseconds");
  }

  public static int max(int[] list) {
    RecursiveTask<Integer> task = new MaxTask(list, 0, list.length);
    ForkJoinPool pool = new ForkJoinPool();
    return pool.invoke(task);
  }
```

```
  private static class MaxTask extends RecursiveTask<Integer> {
    private final static int THRESHOLD = 1000;
    private int[] list;
    private int low,high;
    public MaxTask(int[] list, int low, int high) {
      this.list = list;
      this.low = low;
      this.high = high;
    }

    @Override
    public Integer compute() {
      if (high - low < THRESHOLD) {
        int max = list[0];
        for (int i = low; i < high; i++)
          if (list[i] > max)
            max = list[i];
        return new Integer(max);
      }
      else {
        int mid = (low + high) / 2;
        RecursiveTask<Integer> left = new MaxTask(list, low, mid);
        RecursiveTask<Integer> right = new MaxTask(list, mid, high);

        right.fork();
        left.fork();
        return new Integer(Math.max(left.join().intValue(),
            right.join().intValue()));
      }
    }
  }
```

CamScanner

Exercise
Parallel Sum

- Write a parallel program in Java that computes the sum of all integers between 1 and N
- Parallelize by dividing the list of integers into two chunks, and then recursively compute the sum of each chunk, then merge the summation of all chunks
- Compare the execution time of serial and parallel execution when
  - N=8000
  - N=800000
  - N=8000000

© All rights reserved.

CamScanner