Introduction to Parallel Programming
Day 1
Session 1: Overview

Ahmed El-Mahdy and Waleed Lotfy

2/2/2016

KFUPM training delivered by Brightskies Technologies

Instructors

- Ahmed El-Mahdy
  - Consultant at Brightskies
  - Associate Prof. at Egypt-Japan University of Science and Technology
  - Director of Parallel Computing Lab
- Waleed Lotfy
  - HPC Systems Eng. at Brightskies
  - Ex-HPC admin at Bibliotheca Alexandrina

Our Approach: Active learning…

- Some little talk from our side
- Some interesting activity for you to think about and do!
- An interactive discussion with introduction of new concepts
- Take home activities!

Course Structure

- Session 1: Overview of Parallel Architecture and Parallel Programming
- Session 2: Shared Memory Programming Model: OpenMP
- Session 3: Vectorisation
- Session 4:
  - Message Passing Programming Model: Intel MPI
  - Intel MKL
  - Running parallel HPC applications on the cluster

Overview of Parallel Architectures and Parallel Programming

A Prologue: Computational Thinking

Let’s start by an activity!

- We want to count the number of attendees
- Steps
  1. A first person writes ‘1’ on a piece of paper
  2. Passes the paper to the next person
  3. The receiving person adds ‘1’ and crosses the current total
  4. Repeat from step 2 until no more persons are left
  5. The last person adds the final ‘1’ announces the total

Let’s count the time!

Let’s analyse what happened

- We’ve a computing system!
- A Person -> **Processing unit!**
- Each person performed an addition -> **Computation**
- Each person passes the paper to next -> **Communication Operation**
- Each person waits for the message to arrive before performing a computation -> **Synchronisation Operation**
- The steps -> **Algorithm**

So let’s estimate the computation time

- If everyone takes T_c to add ‘1’
- And, T_O to pass the message
- Then if there are N persons we need total execution time to be:
  - $T = N \times (T_c + T_O)$

- If T_c = 2 Seconds, and T_O = 2 Seconds and we are 40,
- then we need 160 seconds (2.5min)

Now can we make the count faster?

- How about every row passing working in **parallel**?
- Steps, (or *algorithm*)

1. Every row performs the ‘count’ algorithm on Activity 1
2. The last person in a row add ‘1’ and passes the count to the last person in the next adjacent row
3. The last person announces the result

What would we expect about the time?

- We have four rows
- Each about 10 persons
- Every row will need
  - $10\ (T_c + T_O)$ = 10 × $\ (2 + 2)$=40
- But they run in parallel -> total execution time is a **single** row execution time

What would we expect about the time?

- Then the last person in each row will need
  - $4(T_c + T_O)$ = 4 × 4 = 16
- So total time is
  - $40 + 16 = 56$ sec ~ 1min
- When can compute speedup S by
  - S = 160 / 56 = 2.9x
- So let's try…

Start the stop watch!

Right so?

- What we did is called a **reduction operation**
- This is a very common operation
- It basically reduces a large collection of numbers into just one

Take home puzzle!

- Can you do the computation faster than we did?

Computer Architecture & Parallel Programming

KFUPM training delivered by Brightskies Technologies

Computer Architecture

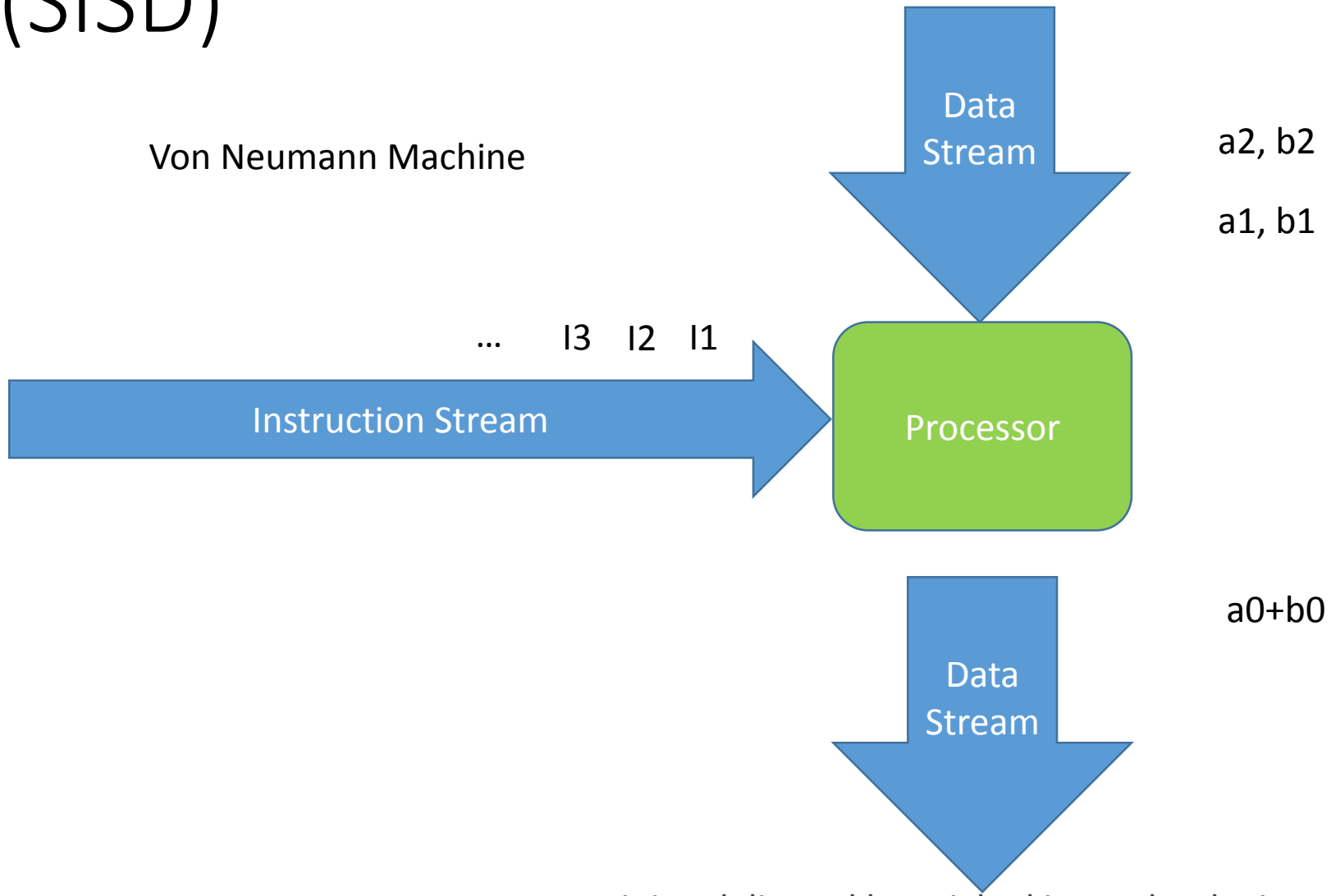
Classification of computing system

- In 1966, Flynn proposed the widely famous taxonomy based on:
  - Instruction Stream
  - Data Stream

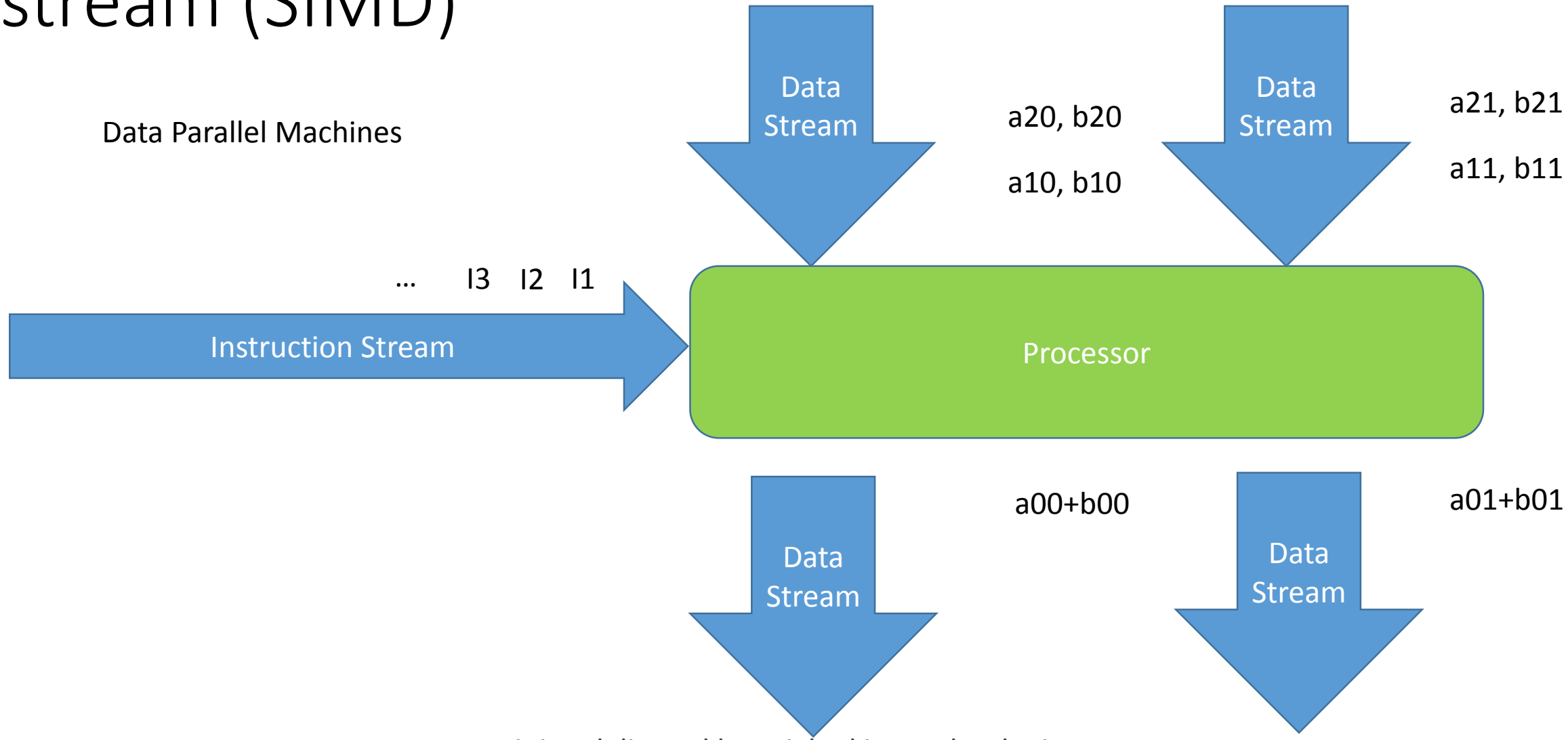
Michael Flynn

Single Instruction stream, Single Data stream (SISD)

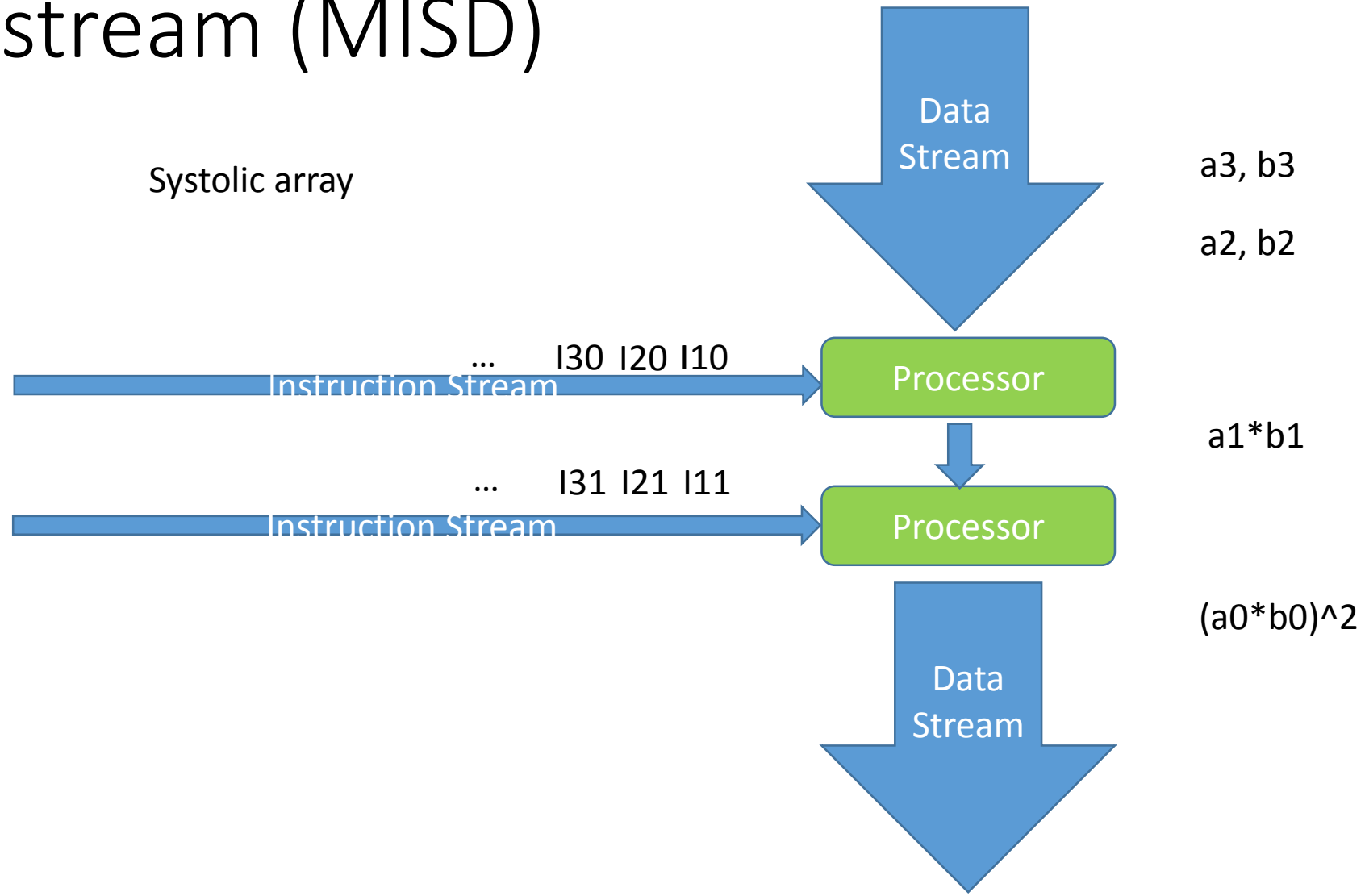
Von Neumann Machine



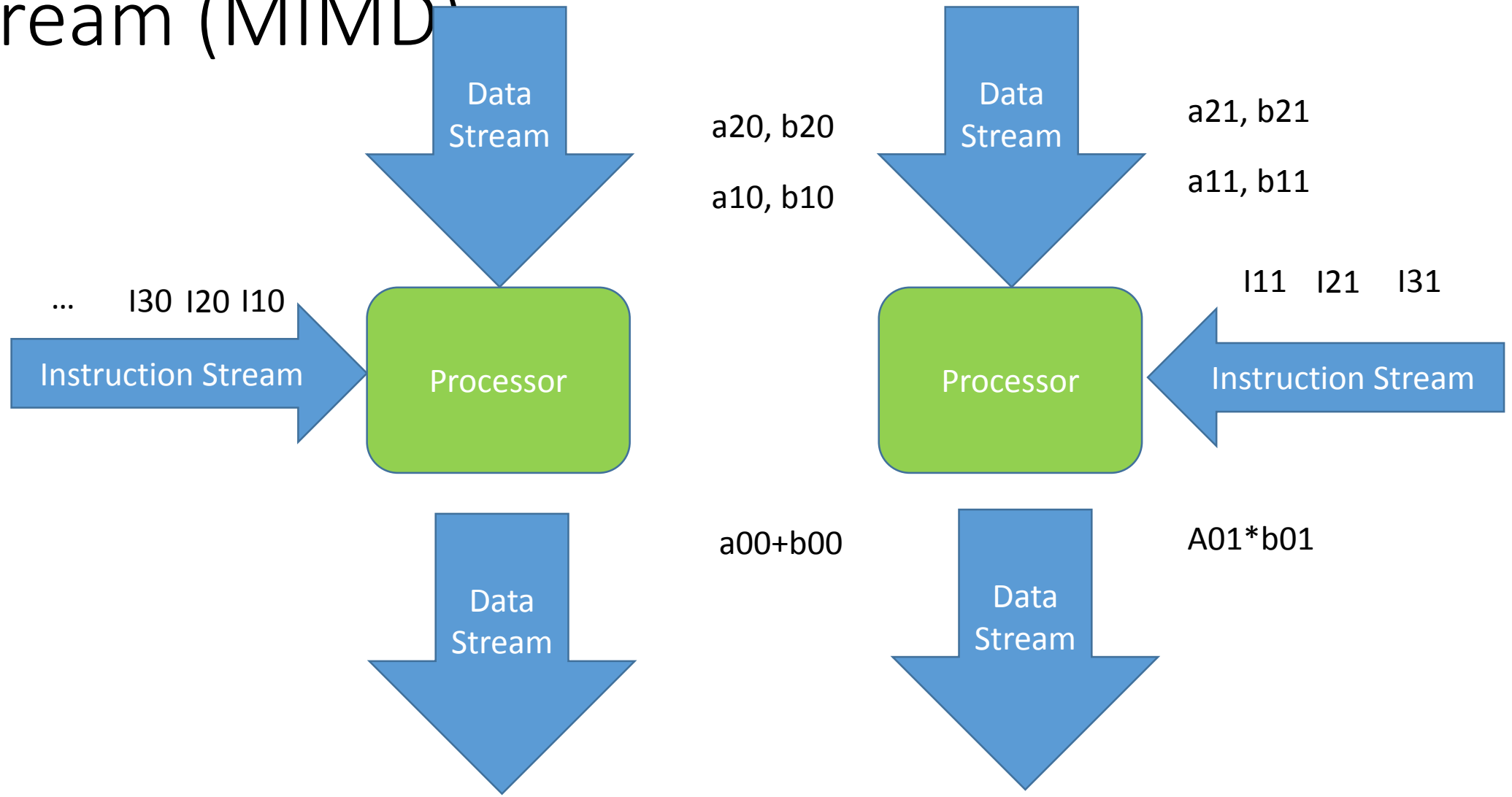
Single Instruction stream, Multiple Data stream (SIMD)



Multiple Instruction stream, Single Data stream (MISD)



Multiple Instruction stream, Multiple Data stream (MIMD)



Communication model

Shared memory

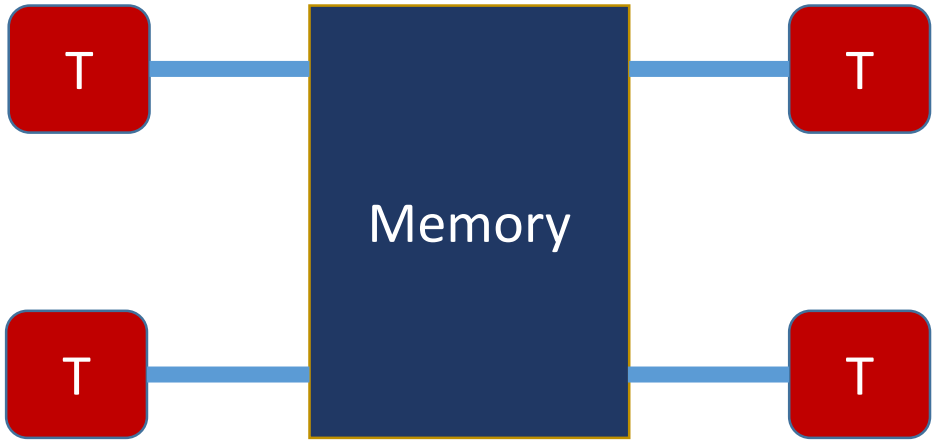
Shared memory

Workers access shared memory regions.

Workers communicate through the shared memory.

Workers in this case are *usually* threads.

Workers *must* rely on the same machine.



Message passing

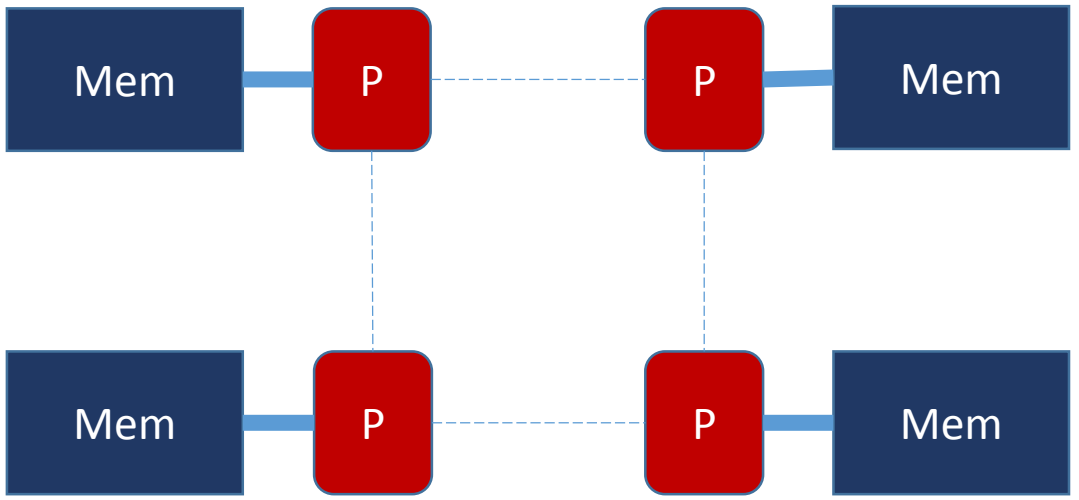
Distributed Memory

Workers have distinct memory footprints.

Workers communicate through Inter-Process Communication (IPC) protocols.

Workers in this case are *usually* processes.

Workers might exist on one or more machines.

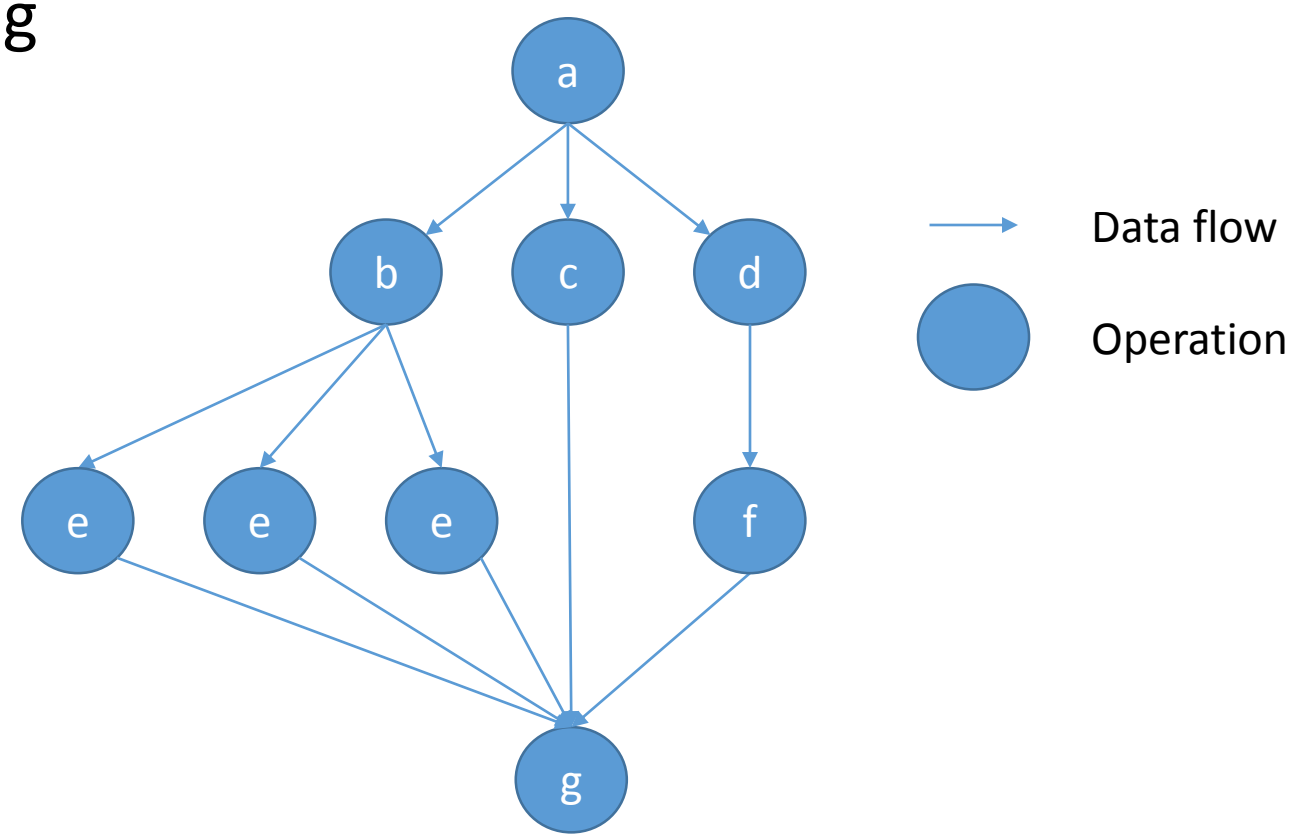


Parallel Programming Principles

Data Flow Graph

Data Flow Graph

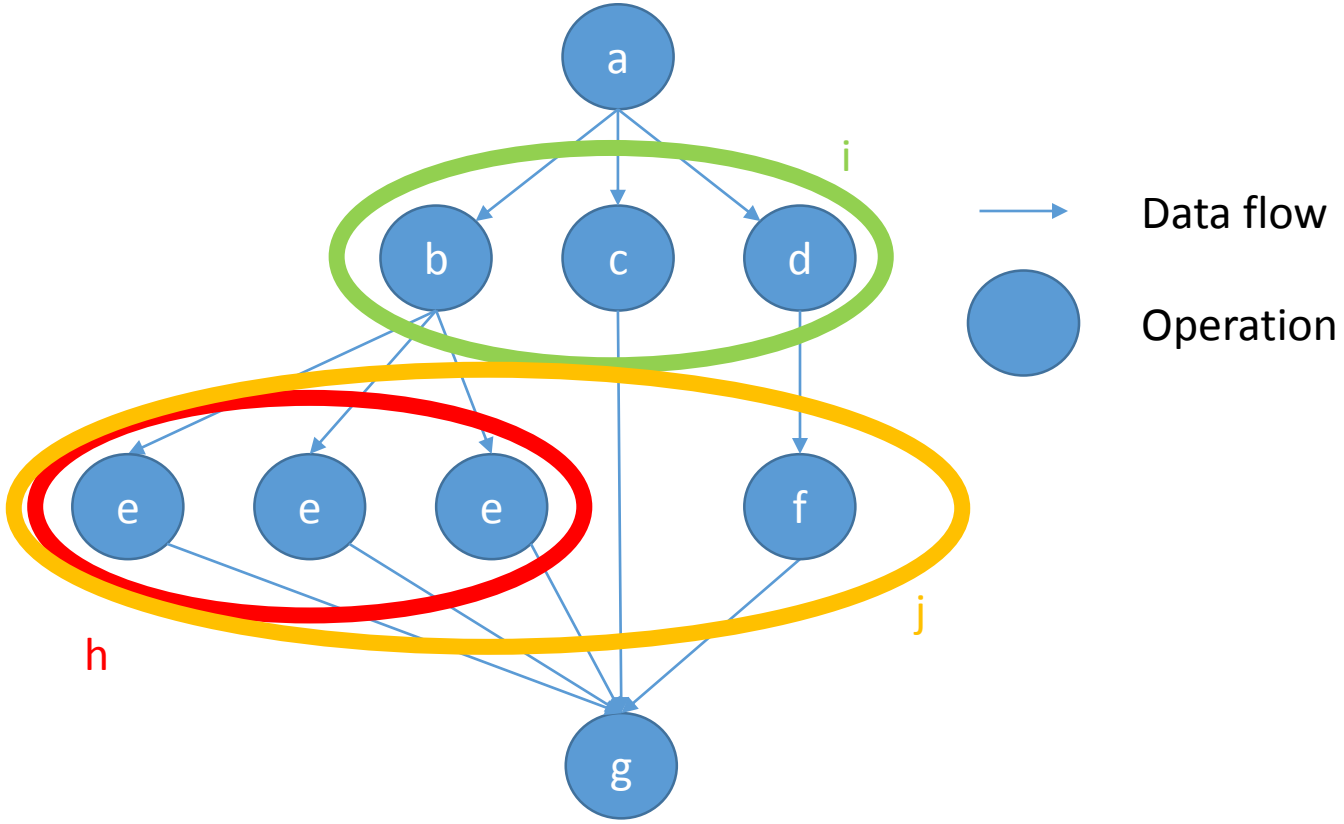
Knowing the parallel programming levels and their data flow graph representation, one could build a data flow graph for the whole algorithm.



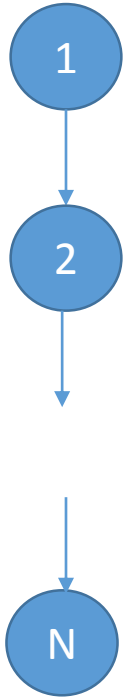
Data Flow Graph

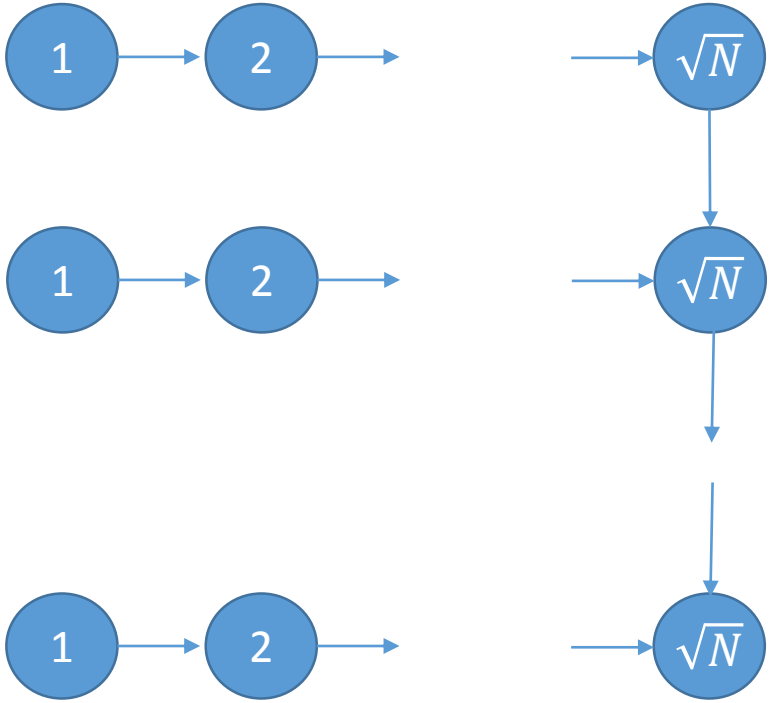
What parallelization techniques can be used with the following graph?

- Data parallelism: (e,e,e)
- Functional/Task parallelism: (b,c,d) & (h,f)
- Pipelining: (a,i,j,g)



In our activity





Parallel Programming Models

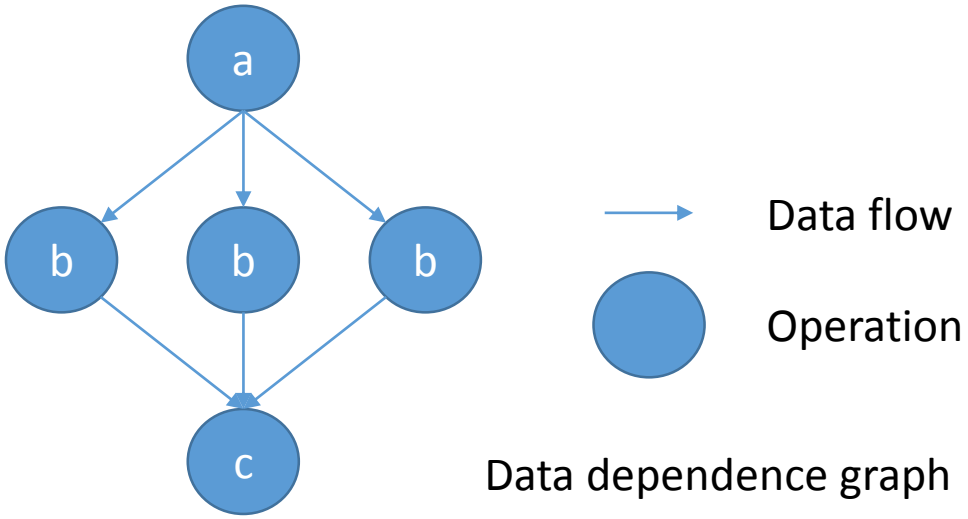
Data Parallelism

Same operation applied on multiple data instances concurrently.

Similar to SIMD architecture.

```
for (int i=0; i < n; i++)
    A[i] = Foo(B[i]);
```

Iterations are divided on workers and processed concurrently.



Data dependence graph

Parallel Programming Models

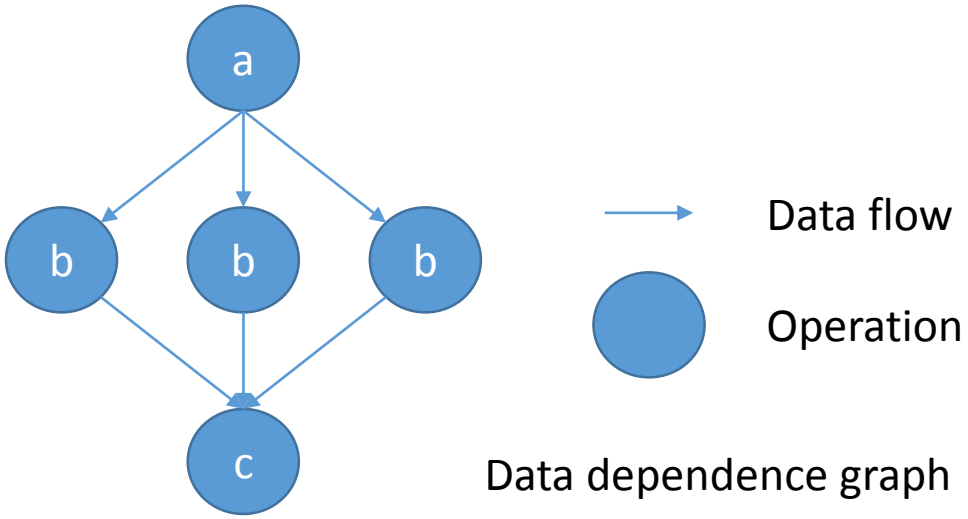
Data Parallelism

Same operation applied on multiple data instances concurrently.

Similar to SIMD architecture.

```
for (int i=0; i < n; i++)
    A[i] = Foo(B[i]);
```

Iterations are divided on workers and processed concurrently.



Data dependence graph

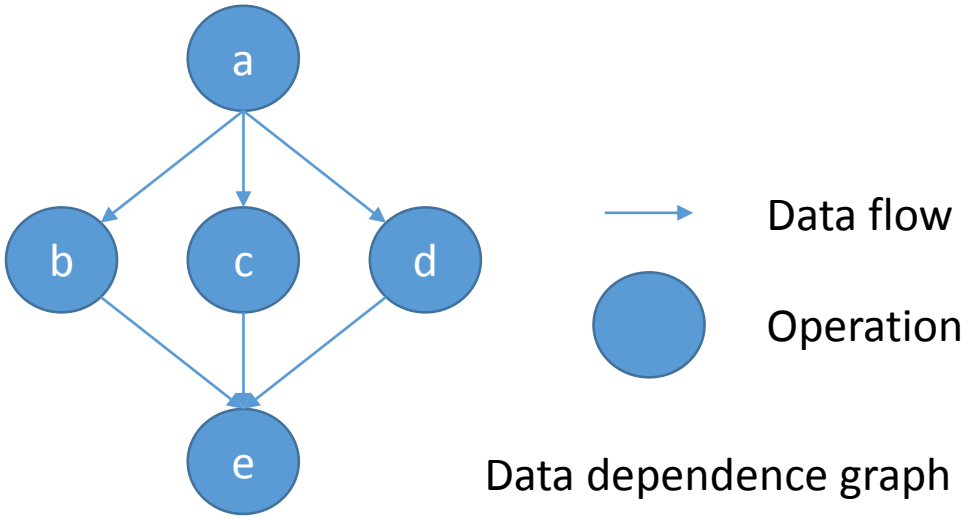
Parallel Programming Models

Task Parallelism

Different operations are applied on same or different data concurrently.

```
A = Foo1(B);
C = Foo2(D);
```

Foo1() and Foo2() run concurrently on different workers and are processed concurrently.



Data dependence graph

Parallel Programming Models

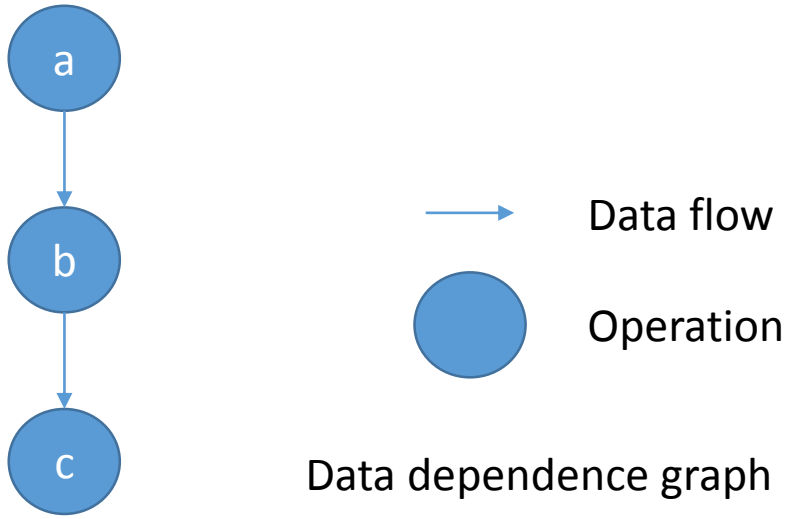
Pipelining

A chain of tasks performed on the same data set.

Can only be parallelized in case of multiple problem instances present.

```
B = Foo1(A);
C = Foo2(B);
```

Foo1() and Foo2() run concurrently on different workers and are processed concurrently.



Data dependence graph

Parallel Programming Models

Pipelining

timestep	operation a	operation b
1	idle	idle
2	processing batch 1	idle
3	processing batch 2	processing batch 1
4	idle	processing batch 2
5	idle	idle

Summary

Before you parallelize your code, you need to know the following:

- Computer architecture:
  - Does it support SIMD and MIMD architectures?
- Communication model
  - Is it shared or message passing
- Build a data flow/dependence graph for your algorithm

Recommended reading list

- Gramma, Gupta, Karypis, Kumar, Introduction to Parallel Computing. Addison-Wesley, 2003.
- D. E. Culler, J. P. Singh, and A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, First Edition. vol. 1. Morgan Kaufmann Publishers, Inc., 1999, pp. 1–1056.
- M. Herlihy and N. Shavit, The Art of Multiprocessor Programming. Elsevier Inc., 2012, pp. 1–537.
- Pacheco, An Introduction to Parallel Programming, 2011
- Timothy G. Mattson, Patterns for parallel programming, 2004