THE UNIVERSITY OF WARWICK

Modern Parallel Algorithms

Artur Czumaj

DIMAP and Department of Computer Science

University of Warwick

Warszawa, November 2022

Modern Parallel Algorithms
Content

What this course is about:

- Some brief (and possibly biased) view of modern design of (theoretical) parallel algorithms, focusing on central models
- Study of algorithms for the central model:

Massive Parallel Computation

- Design of basic algorithms for MPC
- Design of more complex MPC algorithms for fundamental graph problems

Modern Parallel Algorithms
Admin

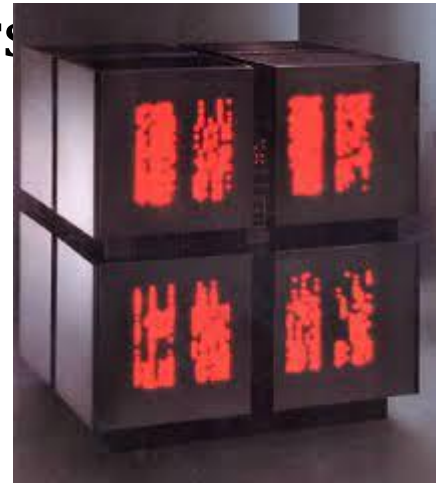
Admin:

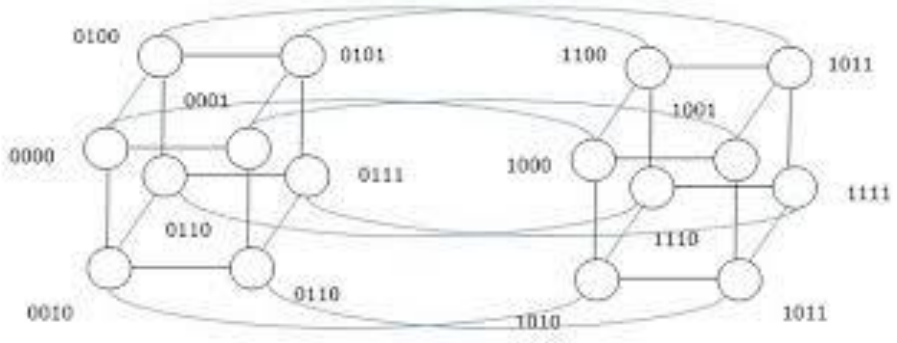
- 3 lectures + 2 exercise sessions
- Slides – used to guide the discussion
- Details typically on the board
- I'm trying to be as interactive as possible

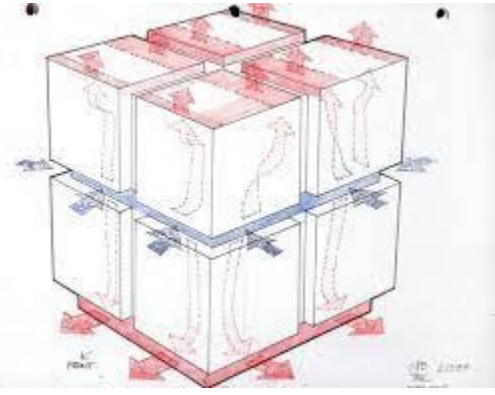
- Exam to be posted next week
- Solutions until late December

Parallel Models and Algorithms
Parallel computing – previous century

- Seriously started in the 80th
- Multiple processors – usually either connected by simple regular networks or with a very small number of processors
- Typically:
  - tightly coupled regular networks,
  - synchronous,
  - with simple frequent communication

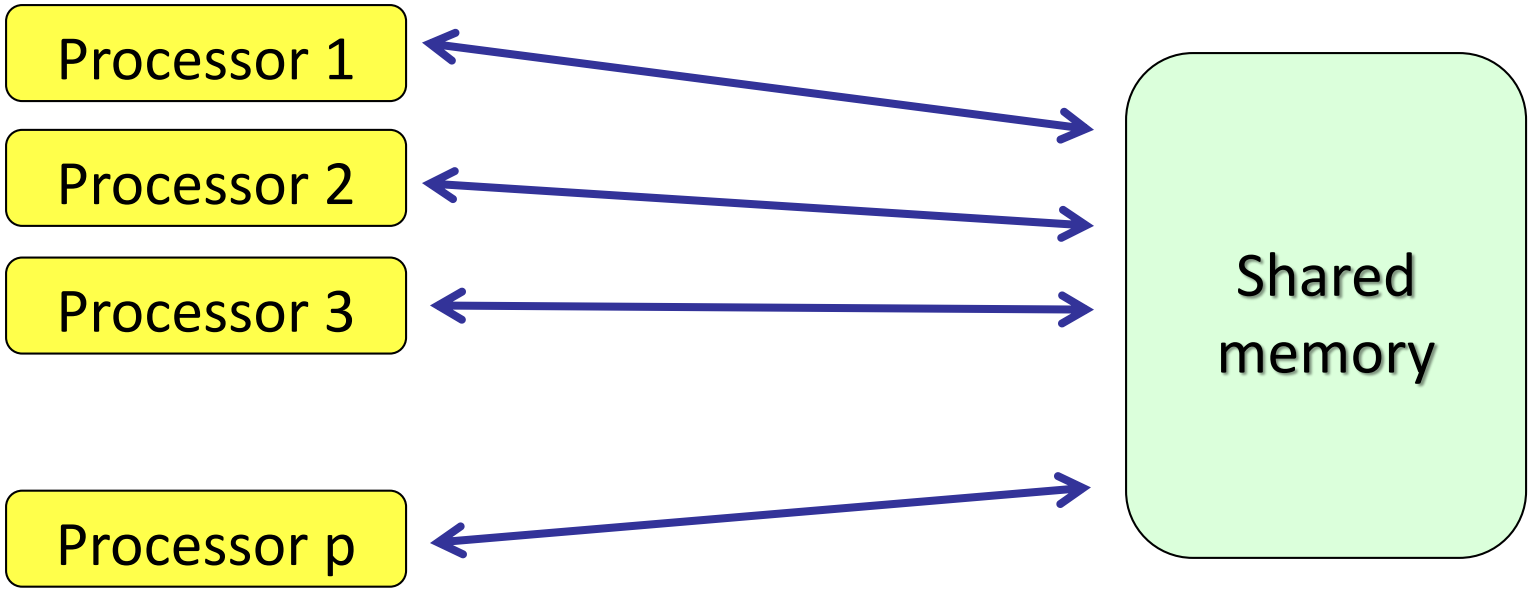






1980 – 95
PRAM

- Theoretical study of algorithms in the 80th mostly for PRAM
- PRAM = Parallel Random Access Machine
  - Parallel version of PRAM



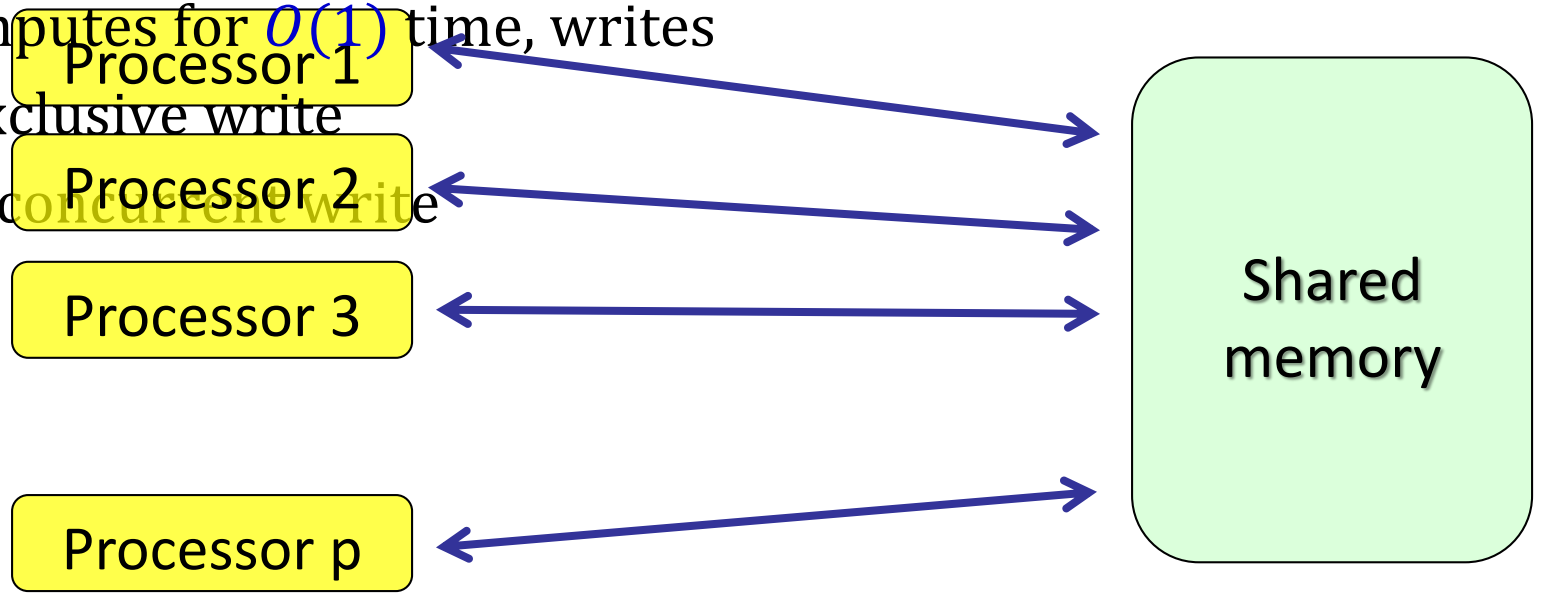
Parallel Random Access Machine (PRAM)

- p processors
- Large shared memory
- In a single step (synchronized)
  - each processor reads, computes for $O(1)$ time, writes
  - EREW – exclusive read, exclusive write
  - CRCW – concurrent read, concurrent write
- Complexity
  - number of processors
  - Time

- Class NC – efficient parallel algorithms (polylog time, polynomial processors)

Parallel Random Access Machine (PRAM)

- p processors
- Large shared memory
- In a single step (synchronized)
  - each processor reads, computes for $O(1)$ time, writes
  - EREW – exclusive read, exclusive write
  - CRCW – concurrent read, concurrent write
- Complexity
  - number of processors
  - Time



- Class NC – efficient parallel algorithms (polylog time, polynomial processors)

Parallel Random Access Machine (PRAM)

- Elegant model (most natural extension of RAM to parallel setting)
- Original hope – like RAM, will be realistic and is simple

- Focuses on inherent parallelism
- Natural complexity classes NC and RNC (randomized NC)

- Problems
  - shared memory is unrealistic
  - need log factors to translate to real machines

Late 90th and thereafter

- Late 90th: PRAM not realistic
- Assumes an arbitrarily large number of processors (like N or N^3)
- Ignores communication costs
  - Attempt to deal with it: Valiant's BSP (bulk synchronous parallel) model

- Programming is hard and machine specific
  - Requires knowing the topology of the machine to get good performance
  - Well beyond the capabilities of most programmers

Modern parallel computing

- Focuses on different paradigms

- Modern real-world parallel system,
  such as MapReduce, Hadoop, Spark, and Dryad

Modern parallel computing

- Focusing on many very powerful computers connected by a network

Modern parallel computing

- Focusing on many very powerful computers connected by a network

Model: Massively Parallel Computation (MPC)

The Massively Parallel Computation (MPC) model:

a theoretical abstraction of real-world parallel system, such as MapReduce, Hadoop, Spark, and Dryad

MPC is the de-facto standard for analyzing (theoretically) algorithms for large-scale parallel computing

Model: Massively Parallel Computation (MPC)

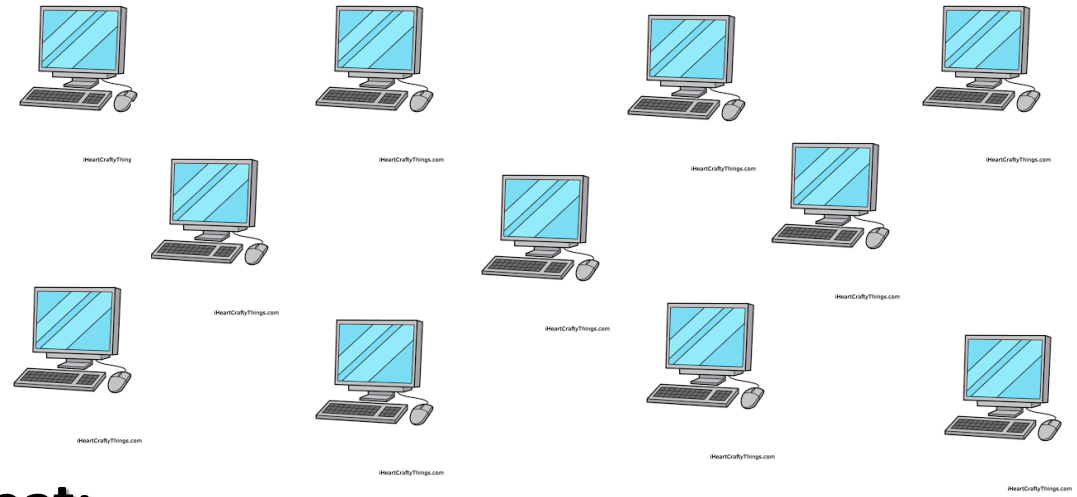
- Natural framework depicting parallel/distributed computation:

- Models natural parallelism via the following framework:

Repeat:

- Split the input into small pieces
- Each machine analyzes/solves one small piece
- Combine (with low congestion) partial solutions

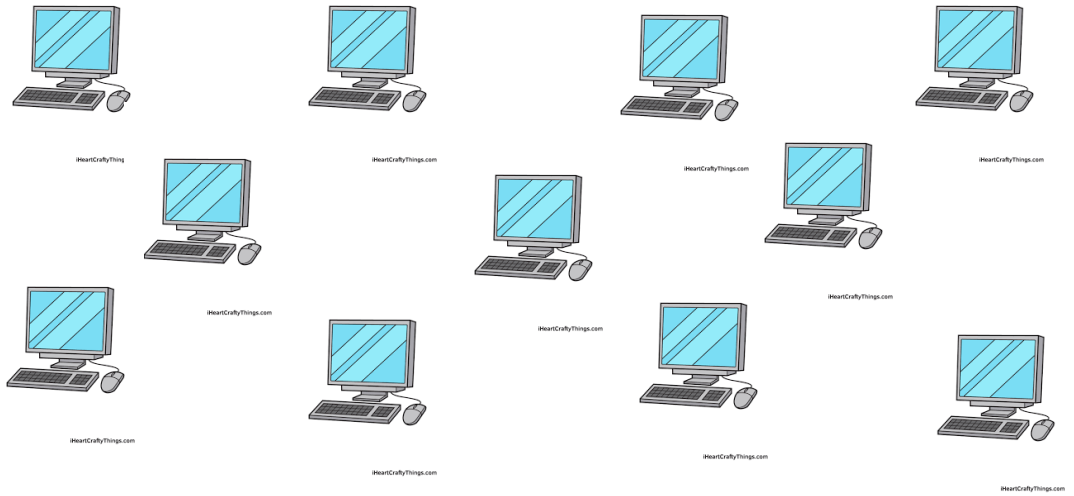
Model: Massively Parallel Computation (MPC)



Repeat:

- Split the input into small pieces
- Each machine analyzes/solves one small piece
- Combine (with low congestion) partial solutions

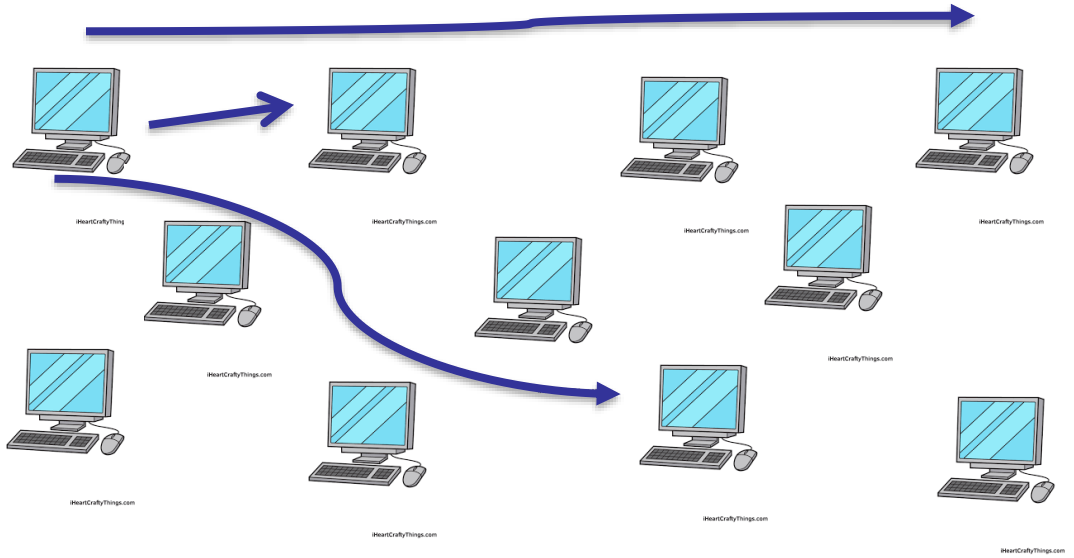
Model: Massively Parallel Computation (MPC)



Synchronized steps:

- Each machine does arbitrary computation
  - local space of $O(s)$ words

Model: Massively Parallel Computation (MPC)



Synchronized steps:

- Each machine does arbitrary computation
  - local space of $O(s)$ words
- Then, machines communicate
  - each machine sends at most s messages
  - each machine receives at most s messages

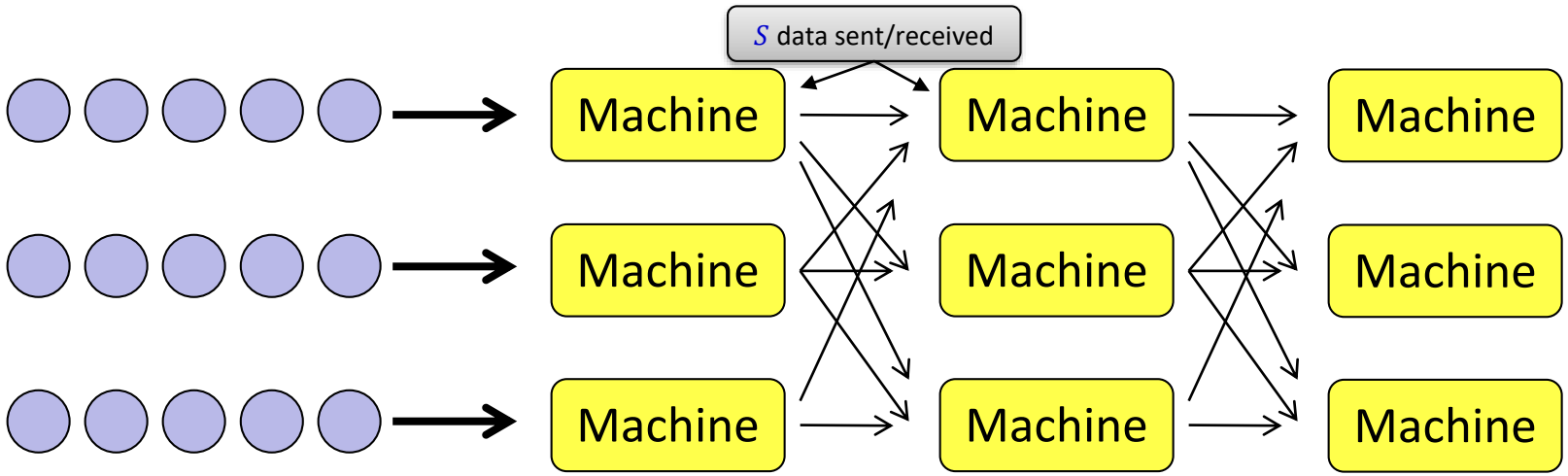
Model: Massively Parallel Computation (MPC)

Introduced by *Karloff, Suri, and Vassilvitskii (2010)*

M machines

s space on each machine

Input: N items



Initially: each machine receives $O(N/M)$ items

Single round:

1. Each machine performs arbitrary computation
2. Each machine sends/receives $O(s)$ data

Model: Massively Parallel Computation (MPC)

- Inspired by MapReduce (*Dean, Ghemawat 2004*)
  - Sleek abstraction that hides details of MapReduce

- Natural framework depicting parallel/distributed computation:

  Repeat:

  - Split the input into small pieces
  - Each machine analyzes/solves one small piece
  - Combine (with low congestion) partial solutions

Model: Massively Parallel Computation (MPC)

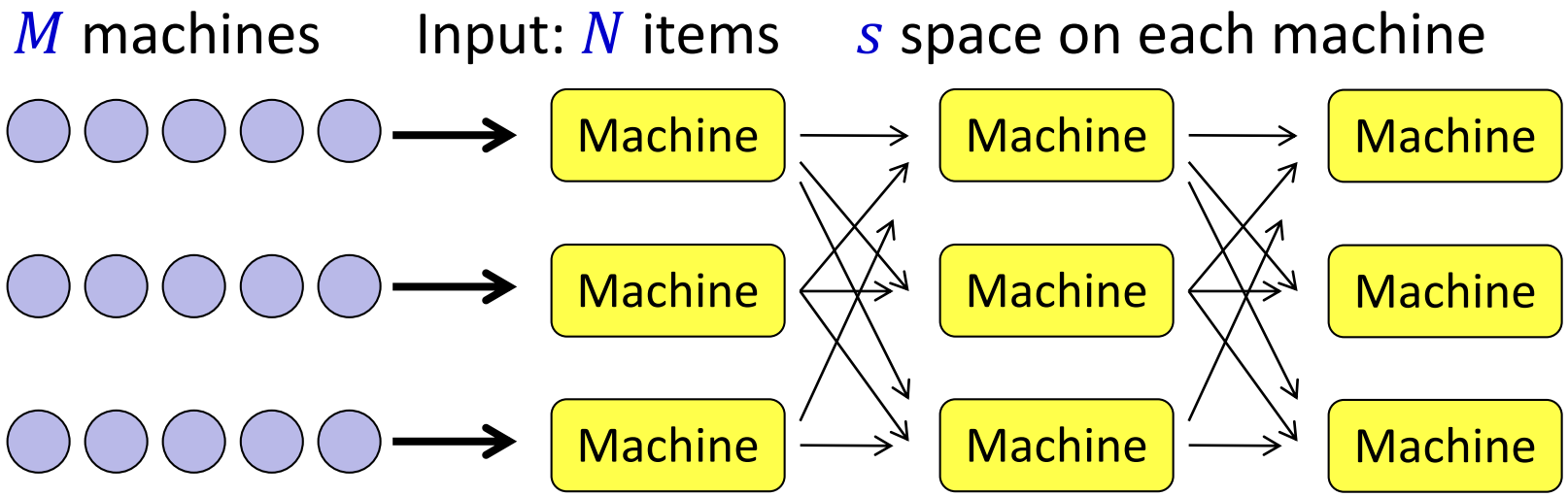
- Inspired by MapReduce (*Dean, Ghemawat 2004*)
  - Sleek abstraction that hides details of MapReduce

- Central parameters: ***local and global space***
  - *Beame, Koutris, Suciu (2013)*
  - *Andoni, Nikolov, Onak, Yaroslavtsev (2014)*
  - *Karloff et al. (2010)*: allow $N^{1-\varepsilon}$ machines with $N^{1-\varepsilon}$ space

    ➔ near quadratic total space $N^{2-2\varepsilon}$
  - A refined version asks for near-linear total space:
    - $M \times s = \tilde{O}(N)$

Model: Massively Parallel Computation (MPC)

Introduced by *Karloff, Suri, and Vassilvitskii (2010)*



- Goals:
  - Small number of rounds
  - Small space per machine
  - Fast local computation

Model: Massively Parallel Computation (MPC)

- Parameters:
  - s
  - M
  - Total space = $s \cdot M$
  - Time

- Ideally:
  - $s \cdot M$ is close to the input size
  - Time should be constant, or as little as possible

Model: Massively Parallel Computation (MPC)

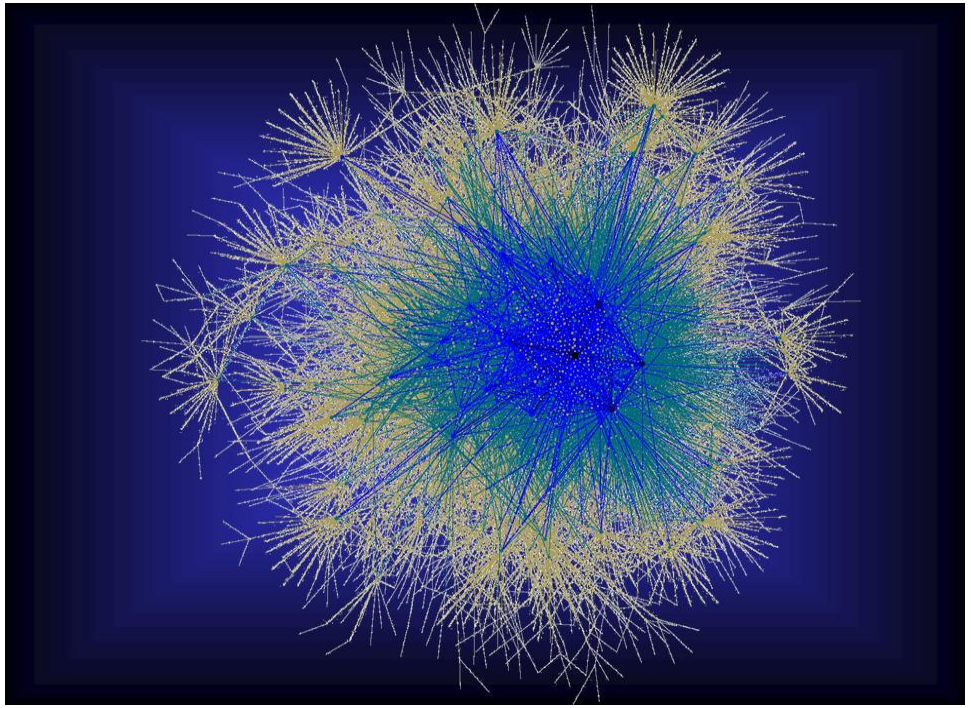
- Parameters:
  - s
  - M
  - Total space = $s \cdot M$
  - Time

How big is s?

- Ideally:
  - $s \cdot M$ is close to the input size
  - Time should be constant, or as little as possible

Case study: MPC for graphs

Input: Edges of an m-edge graph on n vertices

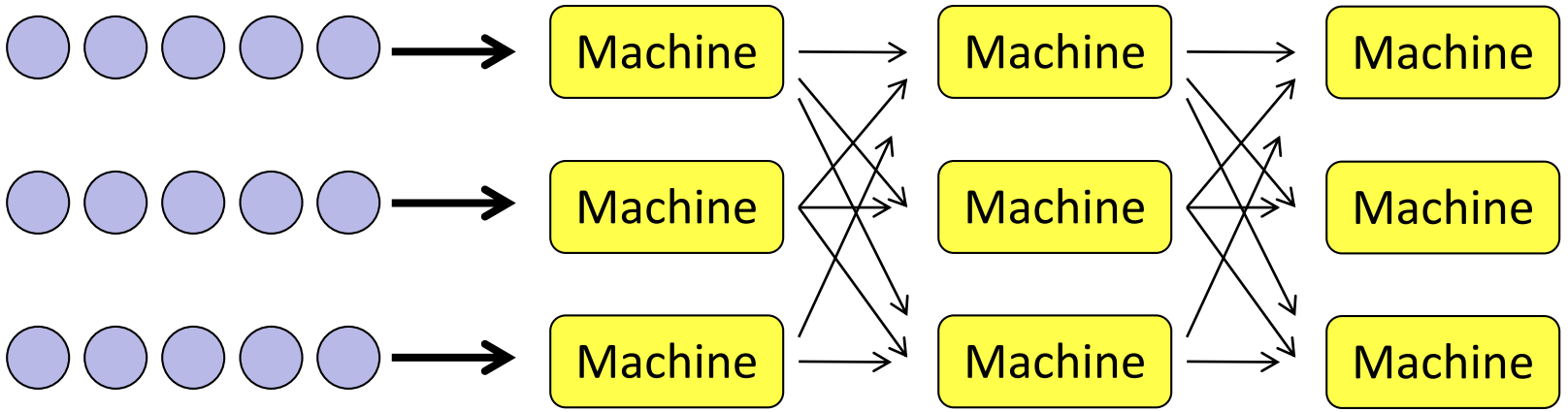


Case study: MPC for graphs

Input: Edges of an m-edge graph on n vertices

$M = O(m/s)$ machines s space on each machine

m-edges and n vertices are distributed among M machines, each machine can accommodate $O(s)$ vertices, $O(s)$ edges



Case study: MPC for graphs

Input: Edges of an m-edge graph on n vertices

$M = O(m/s)$ machines s space on each machine

m-edges and n vertices are distributed among M machines, each machine can accommodate $O(s)$ vertices, $O(s)$ edges

- Natural space regimes:
  - Linear: $s = \Omega(m)$ or $s = \Omega(n^{1+\varepsilon})$ or $s = \Omega(n)$
  - Sublinear: $s = (n^{1-\varepsilon})$ for some (small) constant ε
  - Low-space: $s = O(n^{\delta})$ for any (arbitrarily small) constant δ
  - Polylogarithmic: $s = \log^{O(1)} n$

Case study: MPC for graphs

Input: Edges of an m-edge graph on n vertices

$M = O(m/s)$ machines s space on each machine

m-edges and n vertices are distributed among M machines, each mach (s) vertices, $O(s)$ edges

Many problems are trivial

- Natural space regimes:
  - Linear: $s = \Omega(m)$ or $s = \Omega(n^{1+\varepsilon})$ or $s = \Omega(n)$
  - Sublinear: $s = (n^{1-\varepsilon})$ for some (small) constant ε
  - Low-space: $s = O(n^{\delta})$ for any (arbitrarily small) constant δ
  - Polylogarithmic: $s = \log^{O(1)} n$

Case study: MPC for graphs

Input: Edges of an m-edge graph on n vertices

$M = O(m/s)$ machines s space on each machine

m-edges and n vertices are distributed among M machines, each mach (s) vertices, $O(s)$ edges

Many problems are trivial

Case $s = \Theta(n)$ is the same as the CONGESTED CLIQUE model extensively studied in distributed computing

- Natural
  - Linear: $s = \Omega(m)$ or $s = \Omega(n^{1+\varepsilon})$ or $s = \Omega(n)$
  - Sublinear: $s = (n^{1-\varepsilon})$ for some (small) constant ε
  - Low-space: $s = O(n^{\delta})$ for any (arbitrarily small) constant δ
  - Polylogarithmic: $s = \log^{O(1)} n$

Case study: MPC for graphs

Input: Edges of an m-edge graph on n vertices

$M = O(m/s)$ machines s space on each machine

m-edges and n vertices are distributed among M machines, each machine can accommodate $O(s)$ vertices, $O(s)$ edges

- Natural space regimes:
  - Linear
  - Sublinear: $s = (n^{1-\varepsilon})$ for some (small) constant ε
  - Low-space: $s = O(n^{\delta})$ for any (arbitrarily small) constant δ
  - Polylogarithmic: $s = \log^{O(1)} n$

Many problems are interesting and can be parallelized

Case study: MPC for graphs

Input: Edges of an m-edge graph on n vertices

$M = O(m/s)$ machines s space on each machine

m-edges and n vertices are distributed among M machines, each machine can accommodate $O(s)$ vertices, $O(s)$ edges

- Natural space regimes:
  - Linear: $s = \Omega(m)$ or $s = \Omega(n^{1+\varepsilon})$ or $s = \Omega(n)$
  - Su
  - Low-space: $s = O(n^{\delta})$ for any (arbitrarily small) constant δ
  - Polylogarithmic: $s = \log^{O(1)} n$

Many problems are difficult – here is the focus of modern research

Case study: MPC for graphs

Input: Edges of an m-edge graph on n vertices

$M = O(m/s)$ machines s space on each machine

m-edges and n vertices are distributed among M machines, each machine can accommodate $O(s)$ vertices, $O(s)$ edges

- Natural space regimes:
  - Linear: $s = \Omega(m)$ or $s = \Omega(n^{1+\varepsilon})$ or $s = \Omega(n)$
  - Sublinear: $s = (n^{1-\varepsilon})$ for some (small) constant ε
  - Low-sp
  - Polylogarithmic: $s = \log^{O(1)} n$

Many problems are hard (and typically not better than on PRAM)

Case study: MPC for graphs

Input: Edg

$M = O($ ine

m-edges es,

each ma ges

Low-space $s = O(n^{\delta})$ – most interesting setting:

- allows full **scalability**
- algorithmically challenging
- interesting tasks can be solved efficiently

Here is the focus of modern research

- Natural space regimes:
  - Linear: $s = \Omega(m)$ or $s = \Omega(n^{1+\varepsilon})$ or $s = \Omega(n)$
  - Sublinear: $s = (n^{1-\varepsilon})$ for some (small) constant ε
  - Low-space: $s = O(n^{\delta})$ for any (arbitrarily small) constant δ
  - Polylogarithmic: $s = \log^{O(1)} n$

Basic primitives for MPC algorithms

- Communication primitives
  - broadcasting, communication gathering, …
- Basic data primitives
  - sum computation, prefix-sums, sorting, …
- Simulations
  - PRAM simulations, BSP simulations
  - Distributed CONGESTED CLIQUE and LOCAL simulations

Broadcasting

- There is a single machine (say, M_1) having some N words of data stored in its local memory (and hence $N \leq s$)
- Goal: inform all machines in the system about this data

- We are happy with local space $O(s)$; it's in words

Broadcasting

- There is a single machine (say, M_1) having some N words of data stored in its local memory (and hence $N \leq s$)
- Goal: inform all machines in the system about this data

- If $N \cdot (M-1) \leq s$ then the task is trivial

- Otherwise, if $N \ll s$ then easily $O(\log_{(1+s/N)} M)$ rounds

- Otherwise, if $N \sim s$ or $N \geq s$ then $O(\log_s M)$ rounds

Broadcasting

- There is a single machine (say, M_1) having some N words of data stored in its local memory (and hence $N \leq s$)
- Goal: inform all machines in the system about this data
- If $N \cdot$ (

Broadcasting can be done in $O(\log_s M)$ rounds

- Otherwise, if $N \ll s$ then easily $O(\log_{(1+s/N)} M)$ rounds
- Otherwise, if $N \sim s$ or $N \geq s$ then $O(\log_s M)$ rounds