Parallel Computing

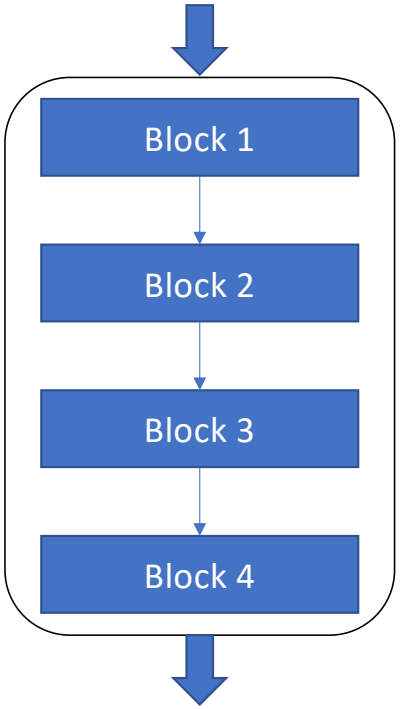
Slides by Dan Fu

Outline

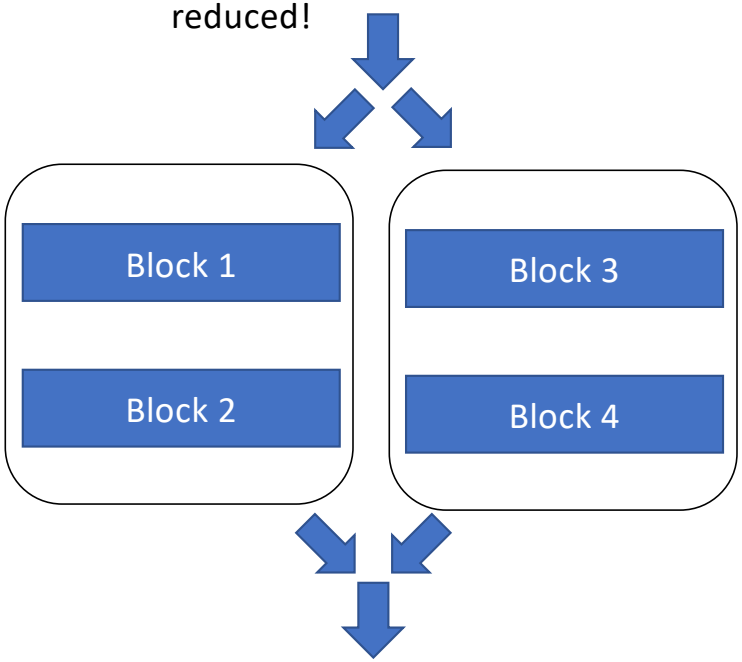
- Why parallelize?
- Lecture review of circuit constructions
- Complexity classes & parallelization
- P-Complete problems and the limits of parallelization in

Motivation

Single processor evaluating blocks of code sequentially

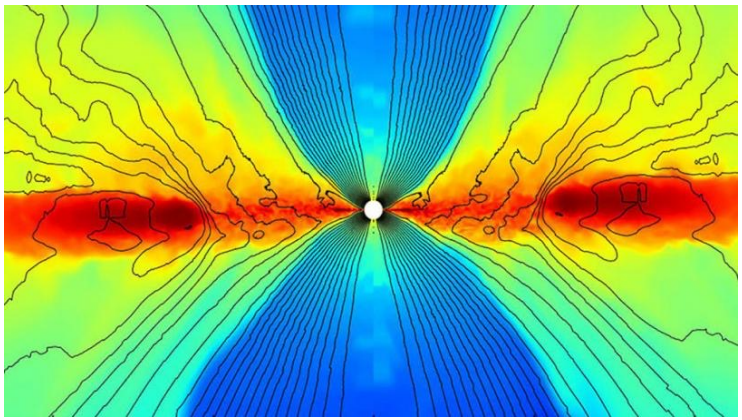


If those code blocks do not have dependencies, then do them elsewhere, then collect their outputs back into the same pipeline. Calculation time has been reduced!



Motivation

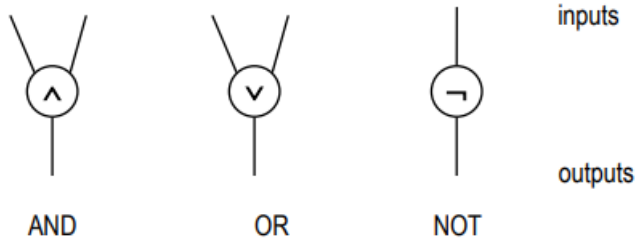
- A lot of math is parallelizable
- Addition, multiplication, division, matrix multiplication, etc.
- This has uses in astronomy, image and signal processing, fluid dynamics, anywhere you are doing a lot of independent computations

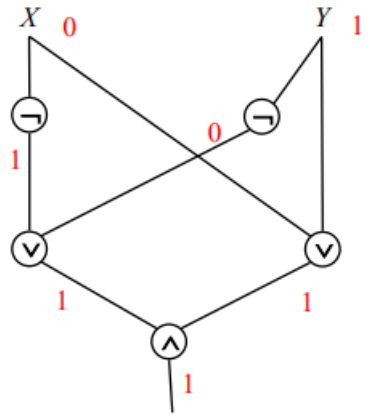


Simulation of matter collapsing into black hole from Northwestern University

Boolean circuits

- Convert languages to circuits
- How can we make this a decision problem?
  - (Accept and reject?)



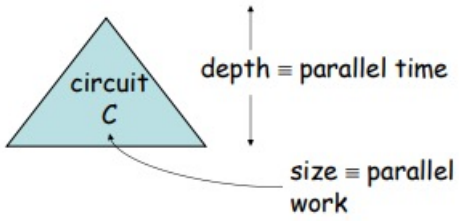


XOR Example

From lectures by Bulatov

Boolean circuits

- Why is this construction important for answering questions about parallelization?
- No cycles
- Finite input
- Logspace, uniform, polynomial size circuits



From lectures by Umans

Boolean circuits

Definition 2 *Circuit family* $\{C_n\}$ *is logspace-uniform if the function mapping* 1^n *to* C_n *can be computed using* $O(\log n)$ *space. Equivalently, each of the following functions can be computed in* $O(\log n)$ *space:*

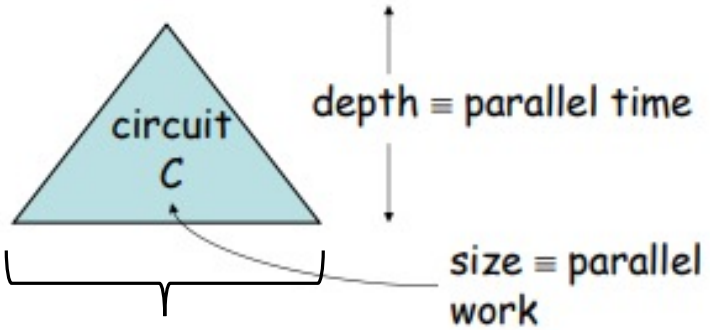
- size(1^n) *returns the number of gates in* C_n *(expressed in binary). By convention the first* n *gates are the input gates and the final gate is the output gate.*
- type$(1^n, i)$ *returns the label (i.e., type of gate) of gate* i *in* C_n*.*
- edge$(1^n, i, j)$ *returns 1 iff there is a (directed) edge from gate* i *to gate* j *in* C_n*.*

With a Turing Machine M using only log-space tape

From Jonathan Katz, UMD

Circuits to Parallelism

- Each gate is a processor
- Finding the lower-bounded depth of this circuit = faster parallelizable algorithm



At most using s processors corresponding to the most gates used at any depth

Class NC

- **N**ick's **C**lass
- Polynomial time and polynomial # of processors
- Captures the class of "efficiently parallelizable problems"

Definition 3.2.6 *The class* $\boldsymbol{NC}$ *is the set of all languages* L *that are decidable in parallel time* $(\log n)^{O(1)}$ *and processors* $n^{O(1)}$.

- NC Hierarchy: $\mathrm{NC} = \cup_k \mathrm{NC}_k$
  - Each level describes a different logarithmic factor in depth

Greenlaw, Hoover, Ruzzo: Limits to Parallel Computation

Relationship with complexity classes

- P/poly
  - Languages accepted by Boolean circuit families of polynomial size
- NP
  - $\mathrm{NP} \not\subset \mathrm{P/poly} \Rightarrow \mathrm{P} \neq \mathrm{NP}$,
  - If any function in NP has exponential-sized circuit (is not “feasibly” parallelizable)
- PSPACE
  - NC (logspace) strictly contained in PSPACE
- LSPACE & NLSPACE ($NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC^k \subseteq NC$)
  - Between $\log n$ and $\log^2 n$
- **NC=?P**
  - The most interesting problems about parallelization

P-Complete

- Every problem in P can be reduce to a P-Complete problem
- Historically, discussed at the same time as NP-Completeness to study sequential time and space
- If a problem is inherently sequential, it is unlikely to parallelize
- It's known that $NC \subseteq P$, but does $NC \neq P$? (Probably, proves the existence of inherently sequential problems)
- Note that class P reductions are log space
- As we discuss the parallelization of P-Complete problems, we are not asking if a problem *can* be parallelized, but whether it can be made much faster by parallelization (think about whether a brute force parallelization is possible, and what it would cost)

Examples of P-Completeness

Some of these should be pretty straightforward. Let’s discuss!

Generic Machine Simulation Problem

Does a Turing machine m accept a string x within t (coded in unary) steps?

GMSP

Does a Turing machine m accept a string x within t (coded in unary) steps?

- What is the operation of a Turing machine?
- Is it solvable *seqentially* in polynomial time?
- Is it parallelizable?
- Is this a useful result?

Circuit Value Problem

Given an encoding $\bar{\alpha}$ of a Boolean circuit α, inputs $x_1, \ldots, x_n$, and a designated output y, is output y of α TRUE on input $x_1, \ldots, x_n$

CVP

Given an encoding $\bar{\alpha}$ of a Boolean circuit α, inputs $x_1, \ldots, x_n$, and a designated output y, is output y of α TRUE on input $x_1, \ldots, x_n$

- Sounds a lot like SAT
- Is it solvable *sequentially* in polynomial time?
- Is it parallelizable?
- Is this a useful result?

Greedy Algorithms?

Picking the best available

Greedy Algorithms

Picking the best available

- Not necessarily P-Complete, but very representative of a P problem
- What's the inherently sequential aspect of greedy algorithms?

$P \neq NC$(Probably)

- If $P = NC$, and we already know that $NC \subseteq P$, then $P \subseteq \mathrm{NC}$
- Implies problems in P also have small space sequential solutions
- Attempts to provide parallel solutions to problems in P are not general
  - General but slow
  - Fast but not general

PRAM Model

- Parallel Random Access Machines

Definition 2.2.1 *Let M be a PRAM. The* **input/output conventions** *for M are as follows. An input $x \in \{0,1\}^n$ is presented to M by placing the integer n in shared memory cell C_0, and the bits $x_1, \ldots, x_n$ of x in shared memory cells $C_1, \ldots, C_n$. M displays its output $y \in \{0,1\}^m$ similarly: integer m in shared memory cell C_0, and the bits $y_1, \ldots, y_m$ of y in shared memory cells $C_1, \ldots, C_m$.*

M computes in **parallel time** $t(n)$ **and processors** $p(n)$ *if for every input $x \in \{0,1\}^n$, machine M halts within at most $t(n)$ time steps, activates at most $p(n)$ processors, and presents some output $y \in \{0,1\}^*$.*

M computes in **sequential time** $t(n)$ *if it computes in parallel time $t(n)$ using 1 processor.*

Greenlaw, Hoover, Ruzzo: Limits to Parallel Computation