Lecture 17: Error Correction; Parallel Algorithms

COMP526: Efficient Algorithms

Will Rosenbaum
University of Liverpool

Updated: November 28, 2024

Announcements

1. Programming Assignment 2 posted
    - Due 29 November
2. No Quiz This Week!
3. Attendance Code:

135170

Meeting Goals

1. Finish discussion error correcting codes
   - Parity checking
   - Hamming Codes
2. Introduce parallel algorithms

Error Correcting Codes

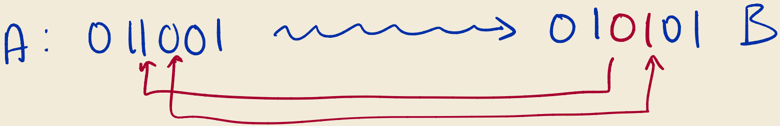
From Last Time

Communcation Model.

- Goal: send a text $S \in \{0,1\}^*$ (bitstream) across a communication channel
- Any bit transmitted through the channel might **flip**
  - $0 \mapsto 1$ or $1 \mapsto 0$
  - *no* erasures or insertions
- To cope with errors:
  - compute and send an encoded bitstream $C(S)$
  - receiver decodes C to get S

Block Codes. Assumptions

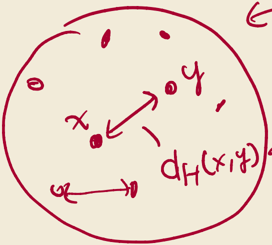
- Messages consists of fixed sized blocks
  - k = **message length**
  - $m \in \{0,1\}^k$
- Encode each message separate as $C(m) \in \{0,1\}^n$
  - $C(m)$ is **codeword** for m
- n is the **block length**



Requirements for Detecting and Correcting

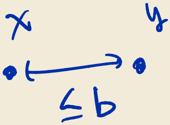
Detecting Requirement. Suppose C can detect errors of flipping up to b bits. Then C has distance $d \geq b + 1$.

← msgs of length n



← # bits on which x y (codewords) differ

Reason:



codewords

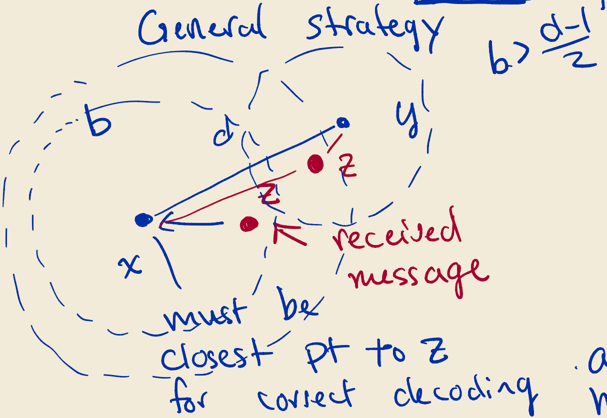
b bits fliped

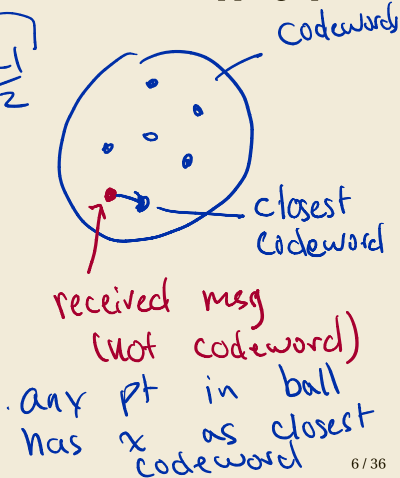
Alice sends x ⟶ Bob receives y

Requirements for Detecting and Correcting

Detecting Requirement. Suppose C can detect errors of flipping up to b bits. Then C has distance $d \geq b + 1$.

Correcting Requirement. Suppose C can correct errors of flipping up to b bits. Then C has distance $d \geq 2b + 1$





Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

min dist. between code words

msg length

block length encoded msg

Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \leq 2^{n-(d-1)}$, hence $n \geq k + d - 1$

detecting ~~correcting~~

eg 2 flipped bits $d \geq 3$

$n \geq k+3-1$
$= k+2$

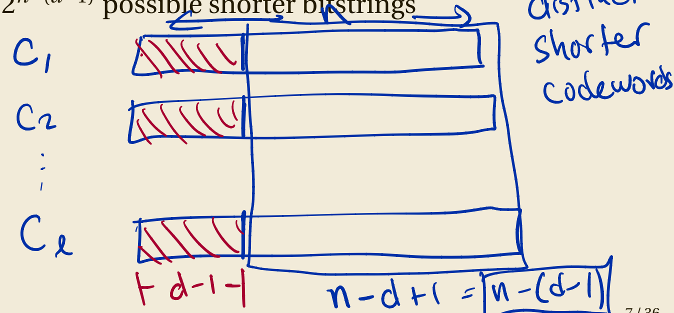
Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \leq 2^{n-(d-1)}$, hence $n \geq k + d - 1$

Proof sketch.

- Consider the deleting the first $d - 1$ bits of each codeword.
- Remaining codewords are still pair-wise distinct
- There are only $2^{n-(d-1)}$ possible shorter bitstrings



Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \leq 2^{n-(d-1)}$, hence $n \geq k + d - 1$

Hamming bound. $2^k \leq 2^n / \sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}$.

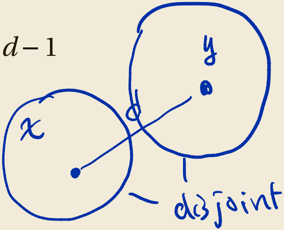
$\binom{n}{f}$ = "n choose f"
= # of subsets of set of n elements of size f

Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \leq 2^{n-(d-1)}$, hence $n \geq k + d - 1$

Hamming bound. $2^k \leq 2^n / \sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}$.



Proof sketch.

- Codewords must be at distance d away
- Thus balls centered at codewords of radius $\lfloor (d-1)/2 \rfloor$ must be disjoint
- Number of balls × *volume* of each ball must be at most 2^n

$2^k \cdot$ Volume of ball $\leq 2^n$

words of length n

Volume of ball of rad. $\lfloor (d-1)/2 \rfloor$ centered at x

Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \leq 2^{n-(d-1)}$, hence $n \geq k + d - 1$

Hamming bound. $2^k \leq 2^n / \sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}$.

Question. These are *im*possibility results. What is possible?

Error Detection & Correction

Error Detection: Parity Bits

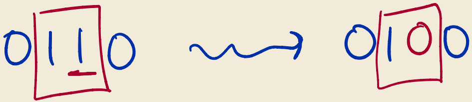
Question. How can we **detect** a single error?

Error Detection: Parity Bits

Question. How can we **detect** a single error?

Obsevation. If a single bit gets flipped, the number of 1s increases or decreases by exactly 1

- the *parity* of the string changes



Error Detection: Parity Bits

Question. How can we **detect** a single error?

Obsevation. If a single bit gets flipped, the number of 1s increases or decreases by exactly 1

- the *parity* of the string changes

Idea. Form C by adding an extra bit to message m that encodes the parity of m

- the extra bit is called a **parity bit**
- which strings are valid codewords?

? Precisely ~~msgs~~ bitstrings w/ even parity (even # of 1s).

01100111 (msg) ⟶ 011001111 (parity)

Error Detection: Parity Bits

Question. How can we **detect** a single error?

Obsevation. If a single bit gets flipped, the number of 1s increases or decreases by exactly 1

- the *parity* of the string changes

Idea. Form C by adding an extra bit to message m that encodes the parity of m

- the extra bit is called a **parity bit**
- which strings are valid codewords?
  - the parity of valid codewords is always 1!

Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$

Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$
- $\mathscr{C} = \{000, 011, 101, 110\}$

Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$
- $\mathscr{C} = \{000, 011, 101, 110\}$

Hamming dist.

min distance between any pair of code words

PollEverywhere Question

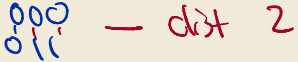
Consider the code C with $k = 2$ bit messages and one parity bit. What is the distance d of C?

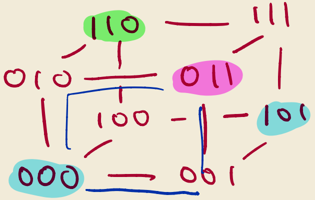
pollev.com/comp526

Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$
- $\mathscr{C} = \{000, 011, 101, 110\}$
- What is the distance of C?





Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$
- $\mathscr{C} = \{000, 011, 101, 110\}$
- What is the distance of C? — 2
- How do we detect errors?

Check parity of received msg

- if even ✓
- if odd know there was an error!

Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?

Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?

Simple Solution. Duplicate each bit 3 times and send the duplicates!

- $k = 1$, $n = 3$
- $C(b) = bbb$
- How do we decode a message?

$1 \mapsto 111$

$0 \mapsto 000$

received: 010 → decode: 000

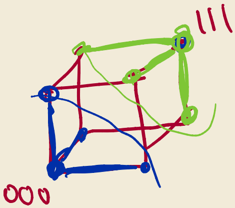
majority vote of bits

Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?

Simple Solution. Duplicate each bit 3 times and send the duplicates!

- $k = 1$, $n = 3$
- $C(b) = bbb$
- How do we decode a message?
- View on Hamming cube!



Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?
Simple Solution. Duplicate each bit 3 times and send the duplicates!

- $k = 1$, $n = 3$
- $C(b) = bbb$
- How do we decode a message?

Inefficiency. To correct a single error, we must **triple** the length of the message?!

Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?
Simple Solution. Duplicate each bit 3 times and send the duplicates!

- $k = 1$, $n = 3$
- $C(b) = bbb$
- How do we decode a message?

Inefficiency. To correct a single error, we must **triple** the length of the message?!

A Puzzle. How can we correct a single error more efficiently?

- Don't need to duplicate every bit!
- Idea: use parity checks on *parts* of the string to identify the index where error occurred!

Hamming Codes

How to Locate Errors?

Idea. Use several parity bits!

- Each parity bit detects an error on a part of the input
- Choose parts so that parity checks uniquely specify location of error
- Error may be in one of the parity bits itself!

How to Locate Errors?

Idea. Use several parity bits!

- Each parity bit detects an error on a part of the input
- Choose parts so that parity checks uniquely specify location of error
- Error may be in one of the parity bits itself!

Binary Trick. Blocks of size $n = 7$ bits: $B = B_7B_6B_5B_4B_3B_2B_1$

- Write indices in binary
  - `111, 110, 101, 100, 011, 010, 001`
- Have a parity check for each bit of the index where the error could have occurred
  - was the error at an index whose jth bit is 1?
  - `111, 110, 101, 100, 011, 010, 001` ← j = 1
  - `111, 110, 101, 100, 011, 010, 001` ← j = 2
  - `111, 110, 101, 100, 011, 010, 001` j = 3

How to Locate Errors?

Idea. Use several parity bits!

- Each parity bit detects an error on a part of the input
- Choose parts so that parity checks uniquely specify location of error
- Error may be in one of the parity bits itself!

Binary Trick. Blocks of size $n = 7$ bits: $B = B_7 B_6 B_5 B_4 B_3 B_2 B_1$

- Write indices in binary
  - 111, 110, 101, 100, 011, 010, 001
- Have a parity check for each bit of the index where the error could have occurred
  - was the error at an index whose jth bit is 1?
  - 111, 110, 101, 100, 011, 010, 001
  - 111, 110, 101, 100, 011, 010, 001
  - 111, 110, 101, 100, 011, 010, 001
- **Question.** Where do we store parity/message bits?

(7, 4) Hamming Code

Parity Values. Store parity bits at indices $j = 100_2, 010_2, 001_2$.

- Use other 4 bits for messages

index	111	110	101	100	011	010	001
bit	B_7	B_6	B_5	B_4	B_3	B_2	B_1

Question. Why use these three bits for parity checks?

111, 110, 101, 100, 011, 010, 001

(7, 4) Hamming Code

Parity Values. Store parity bits at indices $j = \texttt{100}_2, \texttt{010}_2, \texttt{001}_2$.

- Use other 4 bits for messages

index	111	110	101	100	011	010	001
bit	B_7	B_6	B_5	B_4	B_3	B_2	B_1

Question. Why use these three bits for parity checks?

111, 110, 101, 100, 011, 010, 001

- They are independent of the other parity checks!
  - 111, 110, 101, 100, 011, 010, 001
  - 111, 110, 101, 100, 011, 010, 001
  - 111, 110, 101, 100, 011, 010, 001

Encoding (7, 4) Hamming Code

Procedure. To encode $m = m_3 m_2 m_1 m_0$:

1. write the bits of m to indices $7, 6, 5, 3$ of the codeword

index	111	110	101	100	011	010	001
bit	m_3	m_2	m_1		m_0		

Encoding (7, 4) Hamming Code

Procedure. To encode $m = m_3 m_2 m_1 m_0$:

1. write the bits of m to indices $7, 6, 5, 3$ of the codeword
2. compute the parity bits:
   - $p_4 = m_3 \oplus m_2 \oplus m_1$
   - $p_2 = m_3 \oplus m_2 \oplus m_0$
   - $p_1 = m_3 \oplus m_1 \oplus m_0$

index	111	110	101	100	011	010	001
bit	m_3	m_2	m_1	p_4	m_0	p_2	p_1

$\oplus$ parity operator (exclusive or)

$0 \oplus 0 = 0$

$1 \oplus 1 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

Encoding (7, 4) Hamming Code

Procedure. To encode $m = m_3 m_2 m_1 m_0$:

1. write the bits of m to indices 7, 6, 5, 3 of the codeword
2. compute the parity bits:
   - $p_4 = m_3 \oplus m_2 \oplus m_1$
   - $p_2 = m_3 \oplus m_2 \oplus m_0$
   - $p_1 = m_3 \oplus m_1 \oplus m_0$

index	111	110	101	100	011	010	001
bit	m_3	m_2	m_1	p_4	m_0	p_2	p_1

Example. Encode the message $m = 1011$

index	111	110	101	100	011	010	001
bit	1	0	1	0	1	0	1

Hamming Code Distance

Recall. **Code distance** is the minimum Hamming distance between any two codewords.

Hamming Code Distance

Recall. **Code distance** is the minimum Hamming distance between any two codewords.

PollEverywhere Question

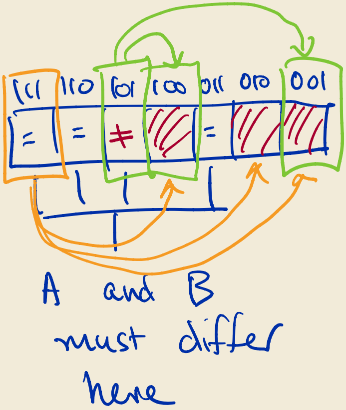
What is the code distance of the (7,4) Hamming code?

pollev.com/comp526

Hamming Code Distance

Recall. Code distance is the minimum Hamming distance between any two codewords.

- Suppose $A = A_7 A_6 A_5 A_4 A_3 A_2 A_1$ and $B = B_7 B_6 B_5 B_4 B_3 B_2 B_1$ are codewords
- A_4, A_2, A_1 determined from other values (similarly for B)
- A and B differ on at least one index $7 = 111_2, 6 = 110_2, 5 = 101_2, 3 = 011_2$
- If A and B differ on exactly one message bit, then two parity bits differ as well
- Check: if A and B differ on two message bits, then at least one parity bit differs as well!



Hamming Code Distance

Recall. Code distance is the minimum Hamming distance between any two codewords.

- Suppose $A = A_7 A_6 A_5 A_4 A_3 A_2 A_1$ and $B = B_7 B_6 B_5 B_4 B_3 B_2 B_1$ are codewords
- A_4, A_2, A_1 determined from other values (similarly for B)
- A and B differ on at least one index $7 = 111_2, 6 = 110_2, 5 = 101_2, 3 = 011_2$
- If A and B differ on exactly one message bit, then two parity bits differ as well
- Check: if A and B differ on two message bits, then at least one parity bit differs as well!

Note. Code distance 3 implies correcting 1 error *might* be possible…

Decoding (7, 4) Hamming Code

Procedure. Given received message $B = B_7 B_6 B_5 B_4 B_3 B_2 B_1$:

1. Compute the parity bits
   - $p_4 = B_7 \oplus B_6 \oplus B_5 \oplus B_4$
   - $p_2 = B_7 \oplus B_6 \oplus B_3 \oplus B_2$
   - $p_1 = B_7 \oplus B_5 \oplus B_3 \oplus B_1$
2. Form index j with binary representation $p_4 p_2 p_1$
3. If $j \neq 0$, form B' by flipping B_j to $1 - B_j$
4. Decode the message $m = B'_7 B'_6 B'_5 B'_3$

Example. Decode the message $B = 1110101$

Question: if B is a valid codeword, then what are the parity bits?

All evaluate to zero! $x \oplus x = 0$

$P_2 = 1$ $P_4 = 1$

Orig codeword: 1010101

1011

$j = P_4 P_2 P_1 = 110_2 = 6$

Decoding (7, 4) Hamming Code

n = codeword size

m = msg size

Procedure. Given received message $B = B_7 B_6 B_5 B_4 B_3 B_2 B_1$:

1. Compute the parity bits
   - $p_4 = B_7 \oplus B_6 \oplus B_5 \oplus B_4$
   - $p_2 = B_7 \oplus B_6 \oplus B_3 \oplus B_2$
   - $p_1 = B_7 \oplus B_5 \oplus B_3 \oplus B_1$
2. Form index j with binary representation $p_4 p_2 p_1$
3. If $j \neq 0$, form B' by flipping B_j to $1 - B_j$
4. Decode the message $m = B'_7 B'_6 B'_5 B'_3$

Example. Decode the message $B = \texttt{1110101}$

- $m = \texttt{1011}$

Note. If $j = 0$, then B is a valid codeword. If $j \neq 0$, then B' is a valid codeword at distance 1 from B.

Error Correction Prospectus

(7, 4) Hamming Codes are **perfect**:

- *m*, *n*, and *d* match the Hamming *lower bound* for block codes

Error Correction Prospectus

(7, 4) Hamming Codes are **perfect**:

- *m*, *n*, and *d* match the Hamming *lower bound* for block codes

$7 = 2^3 - 1$

$4 = 2^3 - 1 - 3$

Generalizations.

- General Hamming codes:
  - Codeword length $n = 2^\ell - 1$ for any ℓ
  - ℓ parity bits
  - Message length $2^\ell - \ell - 1$ message length → ℓ parity bits in codewords
  - All are perfect!

Error Correction Prospectus

(7, 4) Hamming Codes are **perfect**:

- m, n, and d match the Hamming *lower bound* for block codes

Generalizations.

- General Hamming codes:
  - Codeword length $n = 2^\ell - 1$ for any ℓ
  - ℓ parity bits
  - Message length $2^\ell - \ell - 1$ message length
  - All are perfect!
- Other optimal values of m, n, d are generally not known
  - many efficient schemes use algebraic constructions
  - almost all randomly chosen codes are good(!)
  - ongoing research!

Parallel Algorithms

Improving Technology?

Laptop Power.

- My first laptop (ca. 2004)
  - Compaq Presario 2100
  - $900 new ($1,500 with inflation)
  - now < $15 used
- Recent laptop (ca. 2021)
  - Apple MacBook Pro, 2020
  - $1,400 ($1,500 with inflation)
  - Now $800 used

Question. Is my old laptop (in a landfill somewhere) **faster** than my current computer?

Improving Technology?

Laptop Power.

- My first laptop (ca. 2004)
  - Compaq Presario 2100
  - \$900 new (\$1,500 with inflation)
  - now < \$15 used
- Recent laptop (ca. 2021)
  - Apple MacBook Pro, 2020
  - \$1,400 (\$1,500 with inflation)
  - Now \$800 used

PollEverywhere Question

How much *faster* is a new mid/high range laptop computer today than a comparable model from 20 years ago?

pollev.com/comp526

Question. Is my old laptop (in a landfill somewhere) **faster** than my current computer?

Improving Technology?

Laptop Power.

- My first laptop (ca. 2004)
  - Compaq Presario 2100
  - \$900 new (\$1,500 with inflation)
  - now < \$15 used
  - Intel Celeron CPU, **1.6 GHz**
- Recent laptop (ca. 2021)
  - Apple MacBook Pro, 2020
  - \$1,400 (\$1,500 with inflation)
  - Now \$800 used
  - Intel Core i5 CPU, **1.4 GHz**

Question. Is my old laptop (in a landfill somewhere) **faster** than my current computer?

Processor Speed is Not Increasing

Year	Transistors	Clock speed	CPU model
1979	30 k	5 MHz	8088
1985	300 k	20 MHz	386
1989	1 M	20 MHz	486
1995	6 M	200 MHz	Pentium Pro
2000	40 M	2 000 MHz	Pentium 4
2005	100 M	3 000 MHz	2-core Pentium D
2008	700 M	3 000 MHz	8-core Nehalem
2014	6 B	2 000 MHz	18-core Haswell
2017	20 B	3 000 MHz	32-core AMD Epyc
2019	40 B	3 000 MHz	64-core AMD Rome

Processor Speed is Not Increasing

Year	Transistors	Clock speed	CPU model
1979	30 k	5 MHz	8088
1985	300 k	20 MHz	386
1989	1 M	20 MHz	486
1995	6 M	200 MHz	Pentium Pro
2000	40 M	2 000 MHz	Pentium 4
2005	100 M	3 000 MHz	2-core Pentium D
2008	700 M	3 000 MHz	8-core Nehalem
2014	6 B	2 000 MHz	18-core Haswell
2017	20 B	3 000 MHz	32-core AMD Epyc
2019	40 B	3 000 MHz	64-core AMD Rome

But the **number of transistors** is growing exponentially!

Speed vs Throughput

Measuring Performance

- **Processor speed** is the number processor clock cycles per second
- **Latency** of an operation is the time from when the operation starts to when it completes
  - speed determines latency of individual operations
  - speed bounded by physical constraints (e.g. speed of light)

Speed vs Throughput

Measuring Performance

- **Processor speed** is the number processor clock cycles per second
- **Latency** of an operation is the time from when the operation starts to when it completes
  - speed determines latency of individual operations
  - speed bounded by physical constraints (e.g. speed of light)
- **Throughput** is the number of (useful) operations performed each second

Speed vs Throughput

Measuring Performance

- **Processor speed** is the number processor clock cycles per second
- **Latency** of an operation is the time from when the operation starts to when it completes
  - speed determines latency of individual operations
  - speed bounded by physical constraints (e.g. speed of light)
- **Throughput** is the number of (useful) operations performed each second

Speed ≈ Throughput?

- U of L graduates about 6,000 student each year
- $\implies$ each degree takes 1/6,000 year (≈ 88 minutes)

Speed vs Throughput

Measuring Performance

- **Processor speed** is the number processor clock cycles per second
- **Latency** of an operation is the time from when the operation starts to when it completes
  - speed determines latency of individual operations
  - speed bounded by physical constraints (e.g. speed of light)
- **Throughput** is the number of (useful) operations performed each second

Speed $\approx$ Throughput?

- U of L graduates about 6,000 student each year
- $\Longrightarrow$ each degree takes 1/6,000 year ($\approx$ 88 minutes)
- **WRONG!!!**
  - how long does a degree take? — ~ 3 yrs
  - how does U of L have so many graduates?

Parallelism

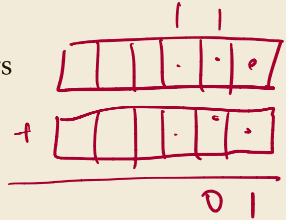
Parallelism is the ability to perform multiple operations simultaneously

Parallelism

Parallelism is the ability to perform multiple operations simultaneously

Examples of parallelism in computers

- Bit level parallelism: adding 32-bit numbers

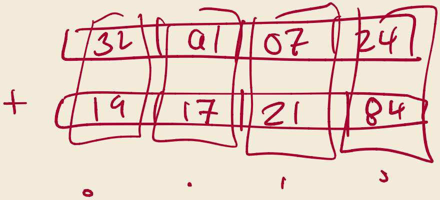


Parallelism

Parallelism is the ability to perform multiple operations simultaneously

Examples of parallelism in computers

- Bit level parallelism: adding 32-bit numbers
- Single Instruction Multiple Data (SIMD) parallelism:
  - *vector* operations in a GPU

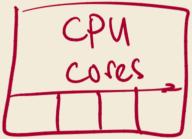


Parallelism

Parallelism is the ability to perform multiple operations simultaneously

Examples of parallelism in computers

- Bit level parallelism: adding 32-bit numbers
- Single Instruction Multiple Data (SIMD) parallelism:
  - *vector* operations in a GPU
- Multiple Instruction Multiple Data (MIMD) parallelism:
  - multicore CPUs



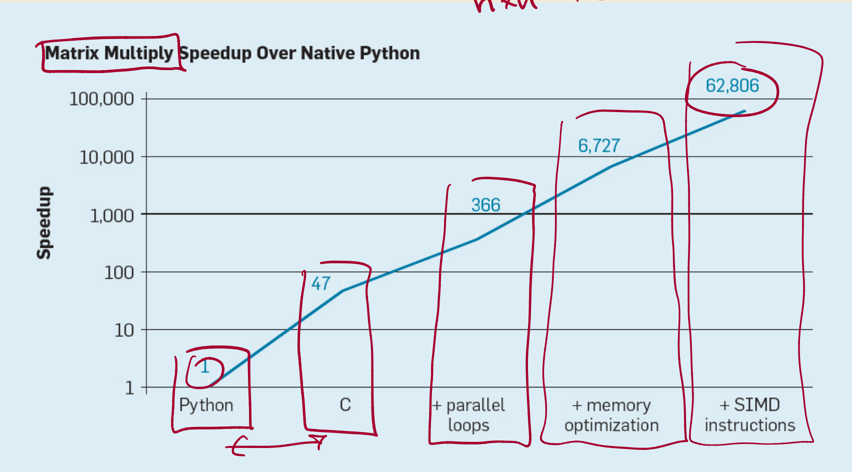
Parallelism

Parallelism is the ability to perform multiple operations simultaneously

Examples of parallelism in computers

- Bit level parallelism: adding 32-bit numbers
- Single Instruction Multiple Data (SIMD) parallelism:
  - *vector* operations in a GPU
- Multiple Instruction Multiple Data (MIMD) parallelism:
  - multicore CPUs
- Distributed/networked computing
  - cluster computing, “cloud” computing, server farms

The Power of Parallelism



Modeling Parallel Computing

Restricted Model: SIMD instructions

- Program = sequence of instructions to be performed
- If *same* operation is performed on multiple data, operations can be performed simultaneously
- Example: A, B, C arrays length n

```
for i = 0 to n-1:
    C[i] = A[i] + B[i]
```

~ $\Theta(n)$

single SIMD operation!

Modeling Parallel Computing

Restricted Model: SIMD instructions

- Program = sequence of instructions to be performed
- If *same* operation is performed on multiple data, operations can be performed simultaneously
- Example:

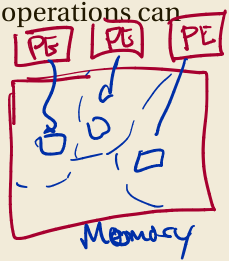
```
for i = 0 to n-1:
    C[i] = A[i] + B[i]
```

Modeling Parallel Computing

Restricted Model: SIMD instructions

- Program = sequence of instructions to be performed
- If *same* operation is performed on multiple data, operations can be performed simultaneously
- Example:

```
for i = 0 to n-1:
    C[i] = A[i] + B[i]
```



General Model: PRAM (Parallel RAM)

- Program can spawn processes/processing elements (PEs) that run in parallel
  - each process is like its own program
- Processes have *shared memory*

Multithreaded computing

Modeling Parallel Computing

Restricted Model: SIMD instructions

- Program = sequence of instructions to be performed
- If *same* operation is performed on multiple data, operations can be performed simultaneously
- Example:

```
for i = 0 to n-1:
    C[i] = A[i] + B[i]
```

General Model: PRAM (Parallel RAM)

- Program can spawn processes/processing elements (PEs) that run in parallel
  - each process is like its own program
- Processes have *shared memory*

Warning. PRAM programs can be *incredibly subtle* to reason

Measuring PRAM Efficiency

Main cost metrics

- **space:** the total amount of accessed memory
- **time:** the number of steps until all processes terminate
  - also known as **depth** or **span**
- **work:** total number of instructions executed by all processes

Measuring PRAM Efficiency

Main cost metrics

- **space:** the total amount of accessed memory
- **time:** the number of steps until all processes terminate
    - also known as **depth** or **span**
- **work:** total number of instructions executed by all processes

Goal:

- minimal span (= time)
- work is (asymptotically) no worse than the best *sequential* algorithm
    - called **work-efficient** algorithms

Models vs Reality

Idealization. The PRAM model does not limit the number of possible PEs (processing elements)

- “multithreaded” computing allows generation of unlimited threads

Models vs Reality

Idealization. The PRAM model does not limit the number of possible PEs (processing elements)

- “multithreaded” computing allows generation of unlimited threads

Reality. More threads does not magically speed up computation

- hardware limits the amount of parallel computation
    - e.g. limited to number of cores

Models vs Reality

Idealization. The PRAM model does not limit the number of possible PEs (processing elements)

- “multithreaded” computing allows generation of unlimited threads

Reality. More threads does not magically speed up computation

- hardware limits the amount of parallel computation
    - e.g. limited to number of cores

Middle Ground (Brent’s Theorem). If an algorithm has span T and work W for an arbitrary number of processors, then the algorithm can be run on a PRAM with p PEs in time $O(T + W/p)$ using work W.

Models vs Reality

Idealization. The PRAM model does not limit the number of possible PEs (processing elements)

- "multithreaded" computing allows generation of unlimited threads

Reality. More threads does not magically speed up computation

- hardware limits the amount of parallel computation
    - e.g. limited to number of cores

Middle Ground (Brent's Theorem). If an algorithm has span T and work W for an arbitrary number of processors, then the algorithm can be run on a PRAM with p PEs in time $O(T + W/p)$ using work W.

- Idea: schedule parallel steps in a "round-robin" fashion on the p PEs.

Enough Generalities!

Parallel Algorithms

- Sorting
  - Sorting Networks (SIMD)
    - sorting short lists
  - Parallel MergeSort
    - sorting long lists
- Searching

Sorting Networks

Comparitors

Recall. In-place sorting algorithms modified the array according to the following pattern:

- check if $A[i]$ and $A[j]$ are out of order
- if so, swap their values

Comparitors

Recall. In-place sorting algorithms modified the array according to the following pattern:

- check if $A[i]$ and $A[j]$ are out of order
- if so, swap their values

Example. InsertionSort

```
procedure InsertionSort(a, n)
    for i = 1, 2, ..., n − 1 do
        j ← i
        while j > 0 and a[j] < a[j − 1] do
            Swap(a, j, j − 1)
            j ← j − 1
        end while
    end for
end procedure
```

Comparitors

Recall. In-place sorting algorithms modified the array according to the following pattern:

- check if $A[i]$ and $A[j]$ are out of order
- if so, swap their values

Example. INSERTIONSORT

```
procedure INSERTIONSORT(a, n)
    for i = 1, 2, ..., n − 1 do
        j ← i
        while j > 0 and a[j] < a[j − 1] do
            SWAP(a, j, j − 1)
            j ← j − 1
        end while
    end for
end procedure
```

Abstract View. A **comparator** is is a PE that takes two values as inputs and returns the values in sorted order.

- $\text{comp}(x, y) = (\min\{x, y\}, \max\{x, y\})$
- all array modifications of INSERTIONSORT can be performed by comparators

Comparitors

Recall. In-place sorting algorithms modified the array according to the following pattern:

- check if $A[i]$ and $A[j]$ are out of order
- if so, swap their values

Example. InsertionSort

```
procedure InsertionSort(a, n)
    for i = 1, 2, ..., n − 1 do
        j ← i
        while j > 0 and a[j] < a[j − 1] do
            Swap(a, j, j − 1)
            j ← j − 1
        end while
    end for
end procedure
```

Abstract View. A **comparator** is is a PE that takes two values as inputs and returns the values in sorted order.

- $\text{comp}(x, y) = (\min\{x, y\}, \max\{x, y\})$
- all array modifications of InsertionSort can be performed by comparators

Question. Which comparator operations of InsertionSort can be performed in parallel (while still ensuring correct output)?

Comparator Networks

Visual Representation.

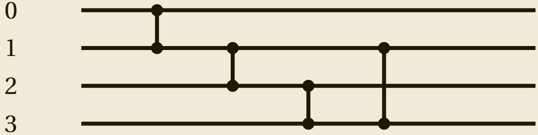
- Inputs/indices are represented by **wires** (horizontal lines)
- Comparators are vertical line segments between wires
  - interpretation: wire between wire i and j performs comp to indices i and j input
- Execution: Scan diagram from left to right and apply comparators in order they are encountered

Comparator Networks

Visual Representation.

- Inputs/indices are represented by **wires** (horizontal lines)
- Comparators are vertical line segments between wires
  - interpretation: wire between wire i and j performs comp to indices i and j input
- Execution: Scan diagram from left to right and apply comparators in order they are encountered

Example. Consider the following comparator network on 4 wires. What is the output on input $[4, 3, 2, 1]$?



Sorting Algorithms to Networks

Consider InsertionSort on inputs of size 5. What are the (possible) comparator operations performed by the algorithm?

- Which comparator operations could be performed *in parallel*?

```
procedure InsertionSort(a, n)
    for i = 1, 2, ..., n − 1 do
        j ← i
        while j > 0 and a[j] < a[j − 1] do
            Swap(a, j, j − 1)
            j ← j − 1
        end while
    end for
end procedure
```

Sorting Network Terminology

Definitions.

- A **comparator network** is defined by a set of wires and a sequence of comparators (left to right).

Sorting Network Terminology

Definitions.

- A **comparator network** is defined by a set of wires and a sequence of comparators (left to right).
- A comparator network is a **sorting network** if for all wire inputs, the resulting outputs are sorted.

Sorting Network Terminology

Definitions.

- A **comparator network** is defined by a set of wires and a sequence of comparators (left to right).
- A comparator network is a **sorting network** if for all wire inputs, the resulting outputs are sorted.
- The **depth** of a comparator network is the maximum number of comparators touched on any path from input to output (including crossed comparators).

Sorting Network Terminology

Definitions.

- A **comparator network** is defined by a set of wires and a sequence of comparators (left to right).
- A comparator network is a **sorting network** if for all wire inputs, the resulting outputs are sorted.
- The **depth** of a comparator network is the maximum number of comparators touched on any path from input to output (including crossed comparators).

Sorting networks and parallel algorithms.

- Each comparator is a process element
- The depth is the span (running time) of the network
- The work is the number of comparators

Sorting Network Terminology

Definitions.

- A **comparator network** is defined by a set of wires and a sequence of comparators (left to right).
- A comparator network is a **sorting network** if for all wire inputs, the resulting outputs are sorted.
- The **depth** of a comparator network is the maximum number of comparators touched on any path from input to output (including crossed comparators).

Sorting networks and parallel algorithms.

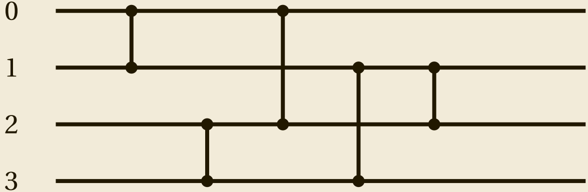
- Each comparator is a process element
- The depth is the span (running time) of the network
- The work is the number of comparators

Question. What is the smallest/shallowest sorting network for a given input size?

- Optimal *size* sorting networks are only known for up to 12 inputs
- Optimal *depth* is only known up to 18 inputs

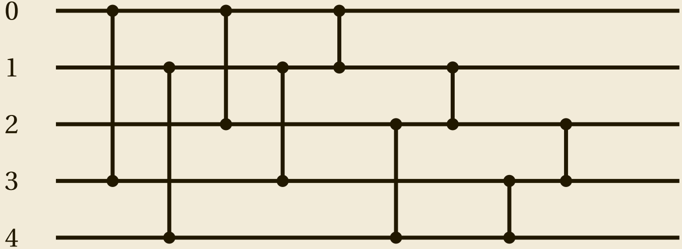
Some Optimal Sorting Networks

Example. $n = 4$ wires. What is the depth?



Some Optimal Sorting Networks

Example. $n = 5$ wires. What is the depth?



Next Time

- More parallel sorting!
- Parallel searching!

Scratch Notes