Algorithms and datastructures II

Lecture 5: circuit complexity 1/2

Jan Hubička

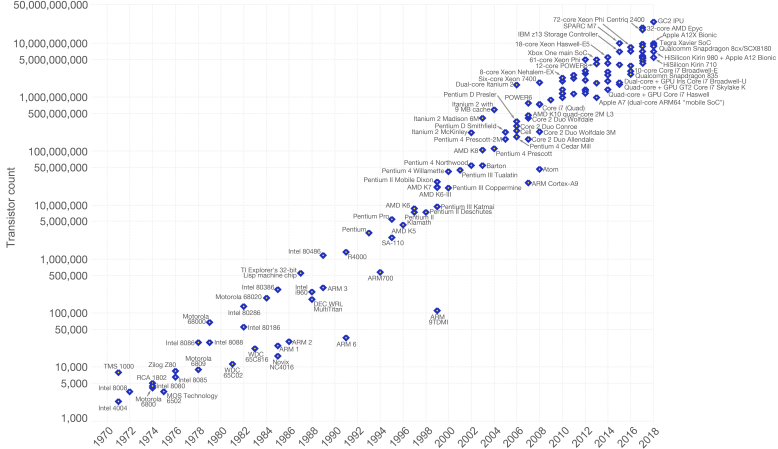
Department of Applied Mathematics
Charles University
Prague

Nov 2 2020

Parallel computing

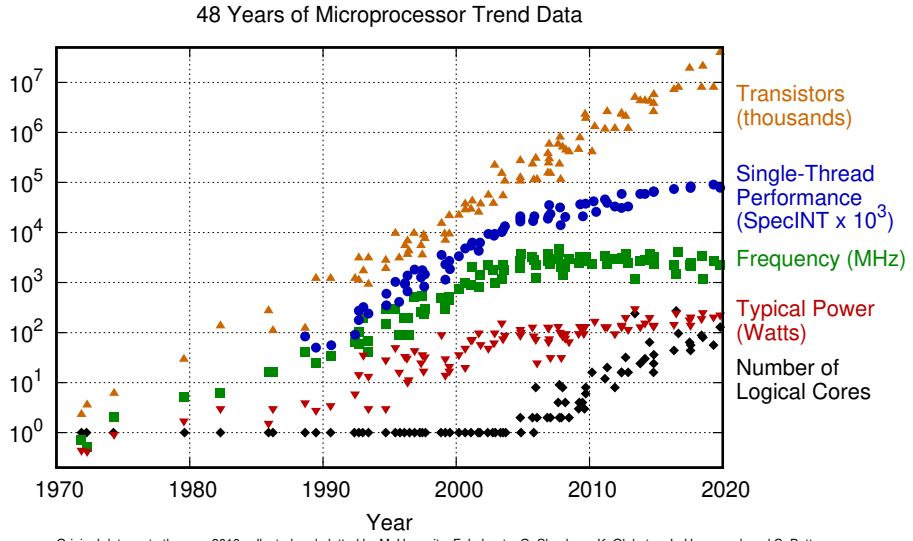
Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Parallel computing



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Computational model: gate

Computational model: circuit

Computational model: example

Compute majority of three 1 bit inputs.

$$\text{maj}(1,1,0) = \text{maj}(1,0,1) = \text{maj}(0,1,1) = \text{maj}(1,1,1) = 1$$

$$\text{maj}(0,0,1) = \text{maj}(0,1,0) = \text{maj}(1,0,0) = \text{maj}(0,0,0) = 0$$

Computational model

Informally circuits consists of gates connected by wires. Every wire transfer a value in alphabet Σ.

Computational model

Informally circuits consists of gates connected by wires. Every wire transfer a value in alphabet Σ.

Definition (Circuit)

Circuit is defined by:

1. Alphabet Σ.
2. Pairwise disjoint set of vertices I (input), O (output) and H (gate).

Computational model

Informally circuits consists of gates connected by wires. Every wire transfer a value in alphabet Σ.

Definition (Circuit)

Circuit is defined by:

1. Alphabet Σ.
2. Pairwise disjoint set of vertices I (input), O (output) and H (gate).
3. Acyclic directed multi-graph (V, E) with $E = I \cup O \cup H$.

Computational model

Informally circuits consists of gates connected by wires. Every wire transfer a value in alphabet Σ.

Definition (Circuit)

Circuit is defined by:

1. Alphabet Σ.
2. Pairwise disjoint set of vertices I (input), O (output) and H (gate).
3. Acyclic directed multi-graph (V, E) with $E = I \cup O \cup H$.
4. Mapping F which assign every gate $g \in H$ of arity $a(g)$ a function $F(h) : \Sigma^{a(g)} \to \Sigma$.

Computational model

Informally circuits consists of gates connected by wires. Every wire transfer a value in alphabet Σ.

Definition (Circuit)

Circuit is defined by:

1. Alphabet Σ.
2. Pairwise disjoint set of vertices I (input), O (output) and H (gate).
3. Acyclic directed multi-graph (V, E) with $E = I \cup O \cup H$.
4. Mapping F which assign every gate $g \in H$ of arity $a(g)$ a function $F(h) : \Sigma^{a(g)} \to \Sigma$.
5. Mapping $z : E \to \mathbb{N}$ which maps every edge to a gate to index which argument it represents.

Computational model

Informally circuits consists of gates connected by wires. Every wire transfer a value in alphabet Σ.

Definition (Circuit)

Circuit is defined by:

1. Alphabet Σ.
2. Pairwise disjoint set of vertices I (input), O (output) and H (gate).
3. Acyclic directed multi-graph (V, E) with $E = I \cup O \cup H$.
4. Mapping F which assign every gate $g \in H$ of arity $a(g)$ a function $F(h) : \Sigma^{a(g)} \to \Sigma$.
5. Mapping $z : E \to \mathbb{N}$ which maps every edge to a gate to index which argument it represents.

Computational model

Informally circuits consists of gates connected by wires. Every wire transfer a value in alphabet Σ.

Definition (Circuit)

Circuit is defined by:

1. Alphabet Σ.
2. Pairwise disjoint set of vertices I (input), O (output) and H (gate).
3. Acyclic directed multi-graph (V, E) with $E = I \cup O \cup H$.
4. Mapping F which assign every gate $g \in H$ of arity $a(g)$ a function $F(h) : \Sigma^{a(g)} \to \Sigma$.
5. Mapping $z : E \to \mathbb{N}$ which maps every edge to a gate to index which argument it represents.

Satisfying:

1. In-degree of vertices in I is 0.
2. In-degree of vertices in O is 1 and out-degree is 0.
3. In-degree of every gate corresponds to its arity. Out-degree of every gate is at least 1.
4. Every input of every gate is assigned.

Computational model: computation

Definition (Computation)

Computation happens in steps (clocks):

1. step 0: All inputs and constants gets assigned value.
2. step $i + 1$: Assign value to all gates and outputs for which all inputs are determined at step i.

Computation ends once all vertices have value assigned.

Computational model: computation

Definition (Computation)

Computation happens in steps (clocks):

1. step 0: All inputs and constants gets assigned value.
2. step $i + 1$: Assign value to all gates and outputs for which all inputs are determined at step i.

Computation ends once all vertices have value assigned.

Definition (Layer)

Layer i consist of all vertices whose value is determined in step i.

Layer is also maximal distance from some input.

Computational model: computation

Definition (Computation)

Computation happens in steps (clocks):

1. step 0: All inputs and constants gets assigned value.
2. step $i + 1$: Assign value to all gates and outputs for which all inputs are determined at step i.

Computation ends once all vertices have value assigned.

Definition (Layer)

Layer i consist of all vertices whose value is determined in step i.

Layer is also maximal distance from some input.

Definition (Time complexity)

Time of the computation of a given circuit is determined by the number of layers.

Computational model: computation

Definition (Computation)

Computation happens in steps (clocks):

1. step 0: All inputs and constants gets assigned value.
2. step $i + 1$: Assign value to all gates and outputs for which all inputs are determined at step i.

Computation ends once all vertices have value assigned.

Definition (Layer)

Layer i consist of all vertices whose value is determined in step i.

Layer is also maximal distance from some input.

Definition (Time complexity)

Time of the computation of a given circuit is determined by the number of layers.

Definition (Space complexity)

Space of the computation is determined by the number of gates in the circuit.

Computational model: program

Definition (Program)

Program is a sequence of circuits for individual sizes of inputs.

Computational model: program

Definition (Program)

Program is a sequence of circuits for individual sizes of inputs.

Remarks:

1. It is necessary to restrict types of allowed gates (or their arity) and Σ. Otherwise every problem can be solved by a single gate.
2. Typically we want network for a given size to be generated by an effective non-parallel algorithm.

We will typically consider boolean circuits where $\Sigma = \{0, 1\}$ and arity is at most 2.

Computational model: example (determine if there is 1 in input)

Addition using boolean circuit

Add $(x_1, x_2, \ldots, x_n; y_1, y_2, \ldots y_n)$

Input: numbers x and y in binary representation.
Output: number $z = x + y$ represented as $z_1, z_2, \ldots z_{n+1}$.

Addition using boolean circuit

Add $(x_1, x_2, \ldots, x_n; y_1, y_2, \ldots y_n)$

Input: numbers x and y in binary representation.
Output: number $z = x + y$ represented as $z_1, z_2, \ldots z_{n+1}$.

We will use temporary carry bit information $c_1, c_2, \ldots c_n$.

Addition using boolean circuit: simple solution

Addition using boolean circuit: blocks and carry bits

Behaviour of block $j \ldots i$ is a function:

$$b(j,i) = \begin{cases} 0 & \text{output carry bit is always 0} \\ \end{cases}$$

Addition using boolean circuit: blocks and carry bits

Behaviour of block $j \ldots i$ is a function:

$$b(j,i) = \begin{cases} 0 & \text{output carry bit is always 0} \\ 1 & \text{output carry bit is always 1} \end{cases}$$

Addition using boolean circuit: blocks and carry bits

Behaviour of block $j \ldots i$ is a function:

$$b(j,i) = \begin{cases} 0 & \text{output carry bit is always 0} \\ 1 & \text{output carry bit is always 1} \\ < & \text{output carry bit is same as input carry bit} \end{cases}$$

Addition using boolean circuit: blocks and carry bits

Behaviour of block $j \ldots i$ is a function:

$$b(j,i) = \begin{cases} 0 & \text{output carry bit is always 0} \\ 1 & \text{output carry bit is always 1} \\ < & \text{output carry bit is same as input carry bit} \end{cases}$$

Behaviour of trivial blocks

Addition using boolean circuit: blocks and carry bits

Behaviour of block $j \ldots i$ is a function:

$$b(j,i) = \begin{cases} 0 & \text{output carry bit is always 0} \\ 1 & \text{output carry bit is always 1} \\ < & \text{output carry bit is same as input carry bit} \end{cases}$$

Behaviour of trivial blocks

Composing behaviours of two adjacent blocks

	0	1	c
0	0	0	0
1	1	1	1
<	0	1	<

Addition using boolean circuit: blocks and carry bits

Behaviour of block $j \ldots i$ is a function:

$$b(j,i) = \begin{cases} 0 & \text{output carry bit is always 0} \\ 1 & \text{output carry bit is always 1} \\ < & \text{output carry bit is same as input carry bit} \end{cases}$$

Behaviour of trivial blocks

Composing behaviours of two adjacent blocks

	0	1	c
0	0	0	0
1	1	1	1
<	0	1	<

Encoding behaviour as a binary function (p, q):

$$(1, *) = < \qquad (0,0) = 0 \qquad (0,1) = 1$$

Addition using boolean circuit: blocks and carry bits

Behaviour of block $j \ldots i$ is a function:

$$b(j,i) = \begin{cases} 0 & \text{output carry bit is always 0} \\ 1 & \text{output carry bit is always 1} \\ < & \text{output carry bit is same as input carry bit} \end{cases}$$

Behaviour of trivial blocks

Composing behaviours of two adjacent blocks

	0	1	c
0	0	0	0
1	1	1	1
<	0	1	<

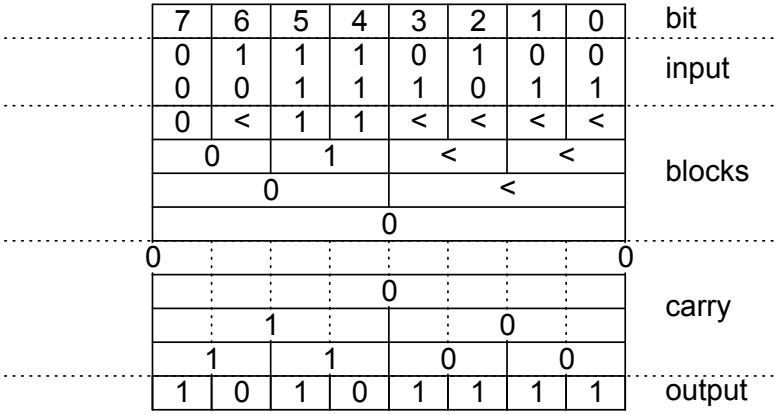
Encoding behaviour as a binary function (p, q):

$$(1, *) = < \qquad (0,0) = 0 \qquad (0,1) = 1$$

Composition:

$$\begin{aligned} p &= p_U \text{ and } p_L \\ q &= (\neg p_U \text{ and } q_L) \text{ or} (p_U \text{ and } q_L) \end{aligned}$$

Addition using boolean circuit: example



Addition using boolean circuit: summary

Basic steps of algorithm:

- Determine behaviour of all canonical blocks
  (block is canonical if its size and position is power of 2)
  1. $\Theta(1)$ layers to determine behaviour of blocks of size 1.
  2. $\Theta(\log n)$ layers to determine behaviour of remaining canonical blocks.

Addition using boolean circuit: summary

Basic steps of algorithm:

- Determine behaviour of all canonical blocks
  (block is canonical if its size and position is power of 2)
  1. $\Theta(1)$ layers to determine behaviour of blocks of size 1.
  2. $\Theta(\log n)$ layers to determine behaviour of remaining canonical blocks.
- Determine carry bits $c_0, c_1, \ldots, c_n$
  1. $\Theta(\log n)$ layers.

Addition using boolean circuit: summary

Basic steps of algorithm:

- Determine behaviour of all canonical blocks
  (block is canonical if its size and position is power of 2)
  1. $\Theta(1)$ layers to determine behaviour of blocks of size 1.
  2. $\Theta(\log n)$ layers to determine behaviour of remaining canonical blocks.
- Determine carry bits $c_0, c_1, \ldots, c_n$
  1. $\Theta(\log n)$ layers.
- Determine final result $z_i = x_i \oplus y_i \oplus c_i$.

Comparator networks: An example

Σ consists of values being sorted.
Comparators are “gates” with two inputs and two outputs.

Comparator networks: An example

Σ consists of values being sorted.
Comparators are “gates” with two inputs and two outputs.
Example: Bubble sort

Comparator networks: An example

Σ consists of values being sorted.
Comparators are “gates” with two inputs and two outputs.
Example: Bubble sort

Next time: an effective sorting algorithm.