CR05: Data Aware Algorithms

Topics covered:

- Pebble game models
- Cache oblivious algorithms
- Memory-aware scheduling
- Communication-avoiding algorithms
- Big-Data algorithms (streaming algorithms)

Teachers:

- Loris Marchal
  (CNRS & ENS Lyon)
- Olivier Beaumont
  (Inria Bordeaux)

Schedule: Wednesday 1:30pm + Thursday 10:15am

Contact: loris.marchal@ens-lyon.fr

CR05: details

Evaluation:

- ~~Two (short) exams~~ Graded homework (40%)
- Group project (60%): 1/2 students,
  paper(s) reading + (optional) code writing,
  report + presentation

Website:

- `http://perso.ens-lyon.fr/loris.marchal/data-aware-algorithms.html`
- Course material available online (right after class)
- Link to other resources

(Your) Participation:

- Class generally goes to fast!
- Stop me with questions, ask for re-explanation
- Especially since half the group is through video
- Partitipation welcomed (and rewarded?)

Part 1: Introduction and Pebble Game models

Introduction and Motivation

Link between Algorithm Design and Data Movement

(Black) Pebble Game and Memory Minimization
- Motivation and rules of the game
- Complexity and variants
- Pebble game on trees
- Space-Time tradeoffs

Introduction & Motivation

- (Fast) Memory: place to store data for compute
- Always been a limited resource (4KB in Appollo 11 computer)
- Not limited anymore ?
  (last iPhone: 4GB, workstations, 1TB)
- But problem size always gets bigger...
- ...And this is rather a question of speed!
- Annual improvements:
  - Time per flop (computation): 59%
  - Data movement:

	Bandwidth	Latency
Network	26%	15%
DRAM	23%	5%

Figures from *Getting up to speed: The future of supercomputing*, 2005, National Academies Press (2004 figure based on data on the period 1988-2002)

Introduction & Motivation

- (Fast) Memory: place to store data for compute
- Always been a limited resource (4KB in Appollo 11 computer)
- Not limited anymore ?
  (last iPhone: 4GB, workstations, 1TB)
- But problem size always gets bigger...
- ...And this is rather a question of speed!
- Annual improvements:
  - Time per flop (computation): 59%
  - Data movement:

	Bandwidth	Latency
Network	26%	15%
DRAM	23%	5%

Figures from *Getting up to speed: The future of supercomputing*, 2005, National Academies Press (2004 figure based on data on the period 1988-2002)

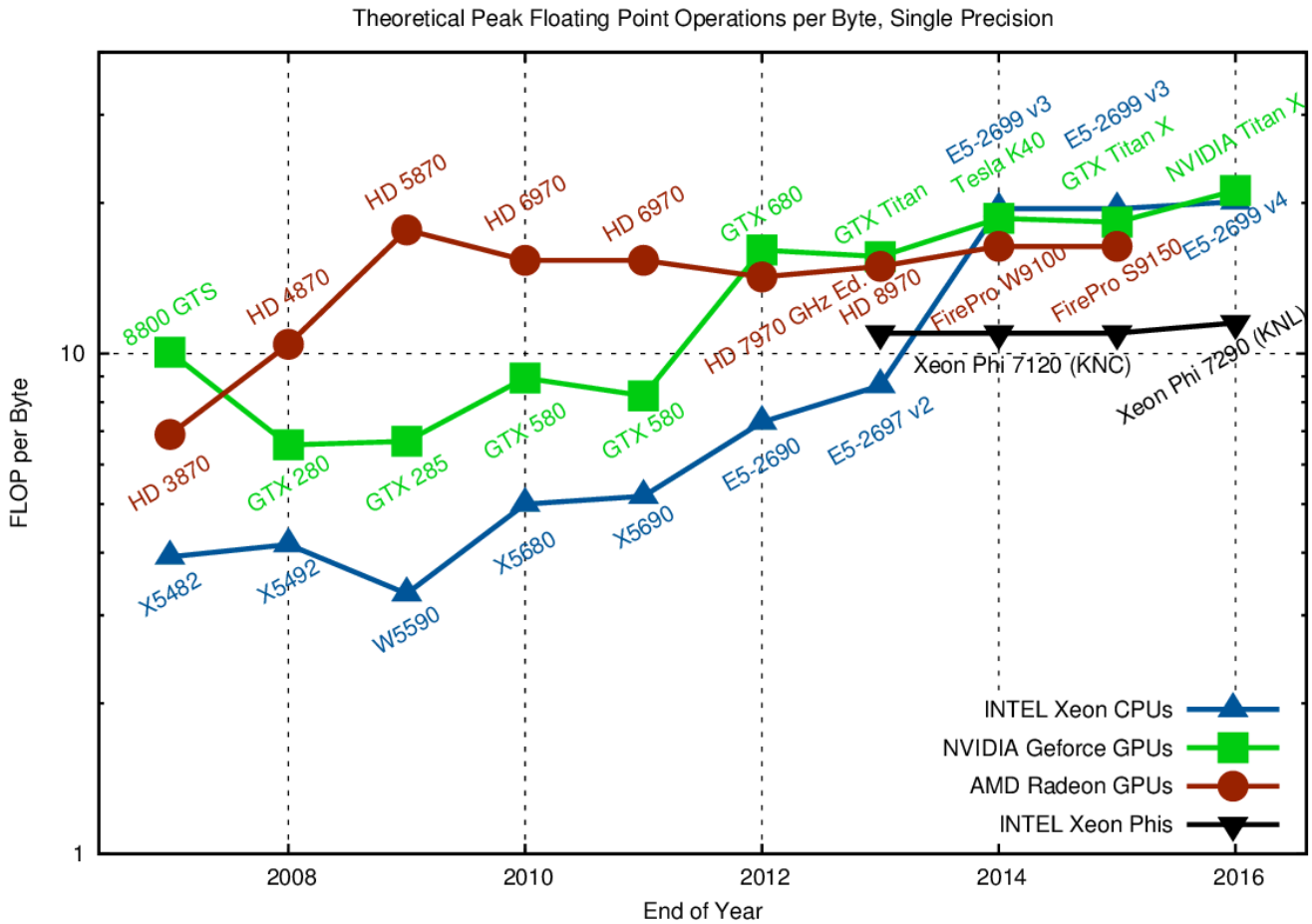
Introduction & Motivation

- (Fast) Memory: place to store data for compute
- Always been a limited resource (4KB in Appollo 11 computer)
- Not limited anymore ?
  (last iPhone: 4GB, workstations, 1TB)
- But problem size always gets bigger...
- ...And this is rather a question of speed!
- Annual improvements:
  - Time per flop (computation): 59%
  - Data movement:

	Bandwidth	Latency
Network	26%	15%
DRAM	23%	5%

Figures from *Getting up to speed: The future of supercomputing*, 2005, National Academies Press (2004 figure based on data on the period 1988-2002)

Flop per byte moved ratio



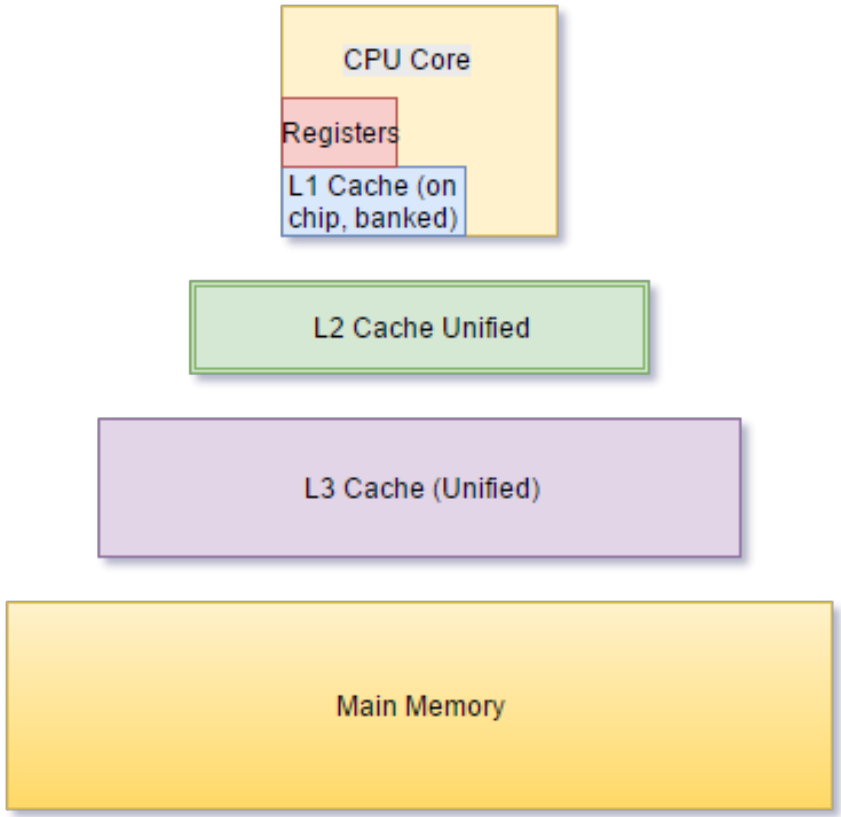
"flop per byte (moved)"

- number of flops perform in the time needed to move a byte
- $\frac{\text{computing speed}}{\text{communication speed}}$

From `http://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/`

Bypass the memory wall

- Time to move the data $>$ Time to compute on the data
- Similar problem in microprocessor design: "memory wall"
- Traditional workaround:
  add a faster but smaller "cache" memory
- Now a hierarchy of caches !



Computing with caches

- Limited amount of fast cache
- Performance sensitive to data locality
- Optimize data reuse
- Avoid data movements between memory and cache(s) (time- and energy-consuming)

In this class: algorithmic approaches to this problem

Computing with caches

- Limited amount of fast cache
- Performance sensitive to data locality
- Optimize data reuse
- Avoid data movements between memory and cache(s) (time- and energy-consuming)

In this class: algorithmic approaches to this problem

Part 1: Introduction and Pebble Game models

Introduction and Motivation

Link between Algorithm Design and Data Movement

(Black) Pebble Game and Memory Minimization
- Motivation and rules of the game
- Complexity and variants
- Pebble game on trees
- Space-Time tradeoffs

Example: matrix-matrix product

- Consider two square matrices A and B (size $n \times n$)
- Compute generalized matrix product: $C \leftarrow C + AB$

Simple-Matrix-Multiply(n, C, A, B)
for $i = 0 \rightarrow n - 1$ **do**
  for $j = 0 \rightarrow n - 1$ **do**
    for $k = 0 \rightarrow n - 1$ **do**
      $C_{i,j} = C_{i,j} + A_{i,k} B_{k,j}$

Assume simple two-level memory model:

- Slow but infinite disk storage
  (where A and B are originally stored)
- Fast and limited memory (size M)

Objective: limit data movement between disk/memory

NB: also applies to other two-level systems (memory/cache, etc.)

Simple algorithm analysis

Simple-Matrix-Multiply(n, C, A, B)
for $i = 0 \rightarrow n - 1$ **do**
  for $j = 0 \rightarrow n - 1$ **do**
    for $k = 0 \rightarrow n - 1$ **do**
      $C_{i,j} = C_{i,j} + A_{i,k} B_{k,j}$

- Assume the memory cannot store half of a matrix: $M < n^2/2$
- Question: How many data movement in this algorithm ?

Simple algorithm analysis

Simple-Matrix-Multiply(n, C, A, B)
for $i = 0 \to n-1$ **do**
  for $j = 0 \to n-1$ **do**
    for $k = 0 \to n-1$ **do**
      $C_{i,j} = C_{i,j} + A_{i,k} B_{k,j}$

- Assume the memory cannot store half of a matrix: $M < n^2/2$
- Question: How many data movement in this algorithm ?

Answer:

- all elements of B accessed during one iteration of the outer loop
- At most half of B stays in memory
- At least $n^2/2$ elements must be read per outer loop
- At least $n^3/2$ read for entire algorithms
- Same order of magnitude of computations: $O(n^3)$
- Very bad data reuse ☹ Question: How to do better ?

Blocked matrix-matrix product

- Divide each matrix into blocks of size $b \times b$:
  $A^b_{i,k}$ is the block of A at position (i, k)
- Perform "coarse-grain" matrix product on blocks
- Perform each block product with previous algorithms

Blocked-Matrix-Multiply(n,A,B,C)
$b \leftarrow \sqrt{M/3}$
for $i = 0, \rightarrow n/b - 1$ **do**
  for $j = 0, \rightarrow n/b - 1$ **do**
    for $k = 0, \rightarrow n/b - 1$ **do**
      Simple-Matrix-Multiply(n, $C^b_{i,j}$, $A^b_{i,k}$, $B^b_{k,j}$)

Blocked matrix-matrix product – Analysis

Blocked-Matrix-Multiply(n,A,B,C)
$b \leftarrow \sqrt{M/3}$
for $i = 0, \rightarrow n/b - 1$ **do**
  for $j = 0, \rightarrow n/b - 1$ **do**
    for $k = 0, \rightarrow n/b - 1$ **do**
      Simple-Matrix-Multiply(n, $C^b_{i,j}$, $A^b_{i,k}$, $B^b_{k,j}$)

Question: Number of data movements ?

Blocked matrix-matrix product – Analysis

Blocked-Matrix-Multiply(n,A,B,C)
$b \leftarrow \sqrt{M/3}$
for $i = 0, \rightarrow n/b - 1$ **do**
  for $j = 0, \rightarrow n/b - 1$ **do**
    for $k = 0, \rightarrow n/b - 1$ **do**
      Simple-Matrix-Multiply(n, $C^b_{i,j}$, $A^b_{i,k}$, $B^b_{k,j}$)

Question: Number of data movements ?

- Iteration of inner loop: 3 blocks of size
  $b \times b = \sqrt{M/3}^3 = M/3$
  $\rightarrow$ fits in memory
- At most $M + M/3$ ($O(M)$) data movements for each inner loop (reading/writing)
- Number of inner iterations:
  $(n/b)^3 = n^3/(M/3) = O(n^3/M\sqrt{M})$
- Total number of data movements: $O(n^3/\sqrt{M})$

Blocked matrix-matrix product – Analysis

Blocked-Matrix-Multiply(n,A,B,C)
$b \leftarrow \sqrt{M/3}$
for $i = 0, \rightarrow n/b - 1$ **do**
  for $j = 0, \rightarrow n/b - 1$ **do**
    for $k = 0, \rightarrow n/b - 1$ **do**
      Simple-Matrix-Multiply(n, $C^b_{i,j}$, $A^b_{i,k}$, $B^b_{k,j}$)

Question: Number of data movements ?

- Iteration of inner loop: 3 blocks of size
  $b \times b = \sqrt{M/3}^3 = M/3$
  $\rightarrow$ fits in memory
- At most $M + M/3$ ($O(M)$) data movements for each inner loop (reading/writing)
- Number of inner iterations:
  $(n/b)^3 = n^3/(M/3) = O(n^3/M\sqrt{M})$
- Total number of data movements: $O(n^3/\sqrt{M})$

Question: Can we do (significantly) better ?

Blocked matrix-matrix product – Analysis

Blocked-Matrix-Multiply(n,A,B,C)
$b \leftarrow \sqrt{M/3}$
for $i = 0, \rightarrow n/b - 1$ **do**
  for $j = 0, \rightarrow n/b - 1$ **do**
    for $k = 0, \rightarrow n/b - 1$ **do**
      Simple-Matrix-Multiply(n, $C^b_{i,j}$, $A^b_{i,k}$, $B^b_{k,j}$)

Question: Number of data movements ?

- Iteration of inner loop: 3 blocks of size
  $b \times b = \sqrt{M/3}^3 = M/3$
  $\rightarrow$ fits in memory
- At most $M + M/3$ ($O(M)$) data movements for each inner loop (reading/writing)
- Number of inner iterations:
  $(n/b)^3 = n^3/(M/3) = O(n^3/M\sqrt{M})$
- Total number of data movements: $O(n^3/\sqrt{M})$

Question: Can we do (significantly) better ? Answer: tomorrow!

Part 1: Introduction and Pebble Game models

Introduction and Motivation

Link between Algorithm Design and Data Movement

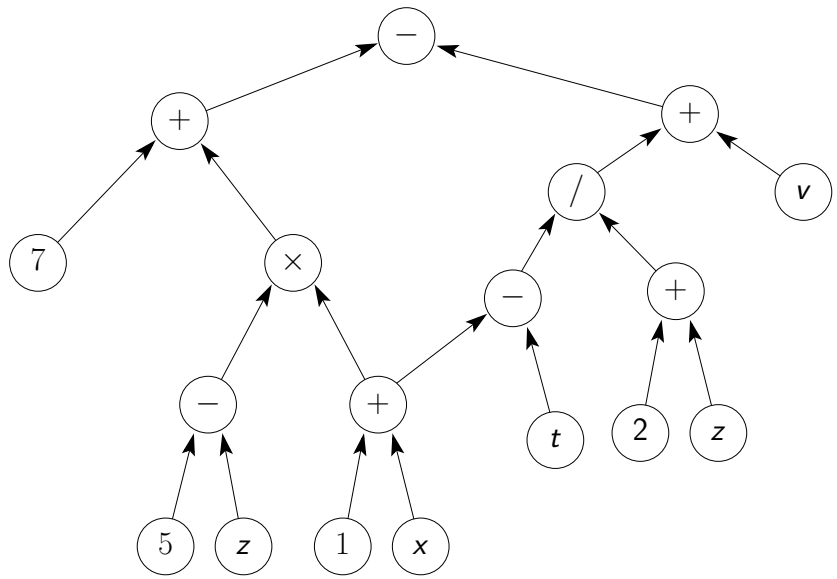
(Black) Pebble Game and Memory Minimization

- Motivation and rules of the game
- Complexity and variants
- Pebble game on trees
- Space-Time tradeoffs

Pebble Game and Register Minimization

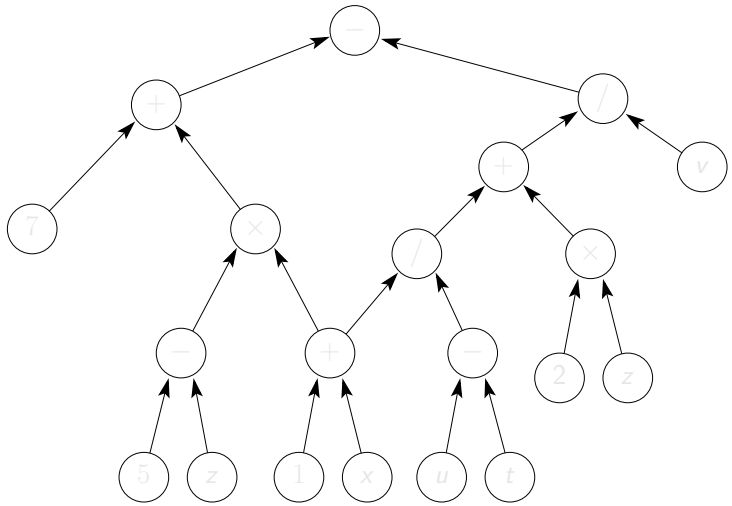
- First model introduced in the 70s
- Motivation: limit the usage of registers for a computation (scarce resource, typically 16/32 for CPUs)
- Registers: at the top of the memory hierarchy
- Restrict to straight-line program: control flow independant from input data
- Modeled as Directed Acyclic Graph:

$$7 + (5 - z) \times (1 + x) - (1 + x - t)/(2 + z) + v$$



(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



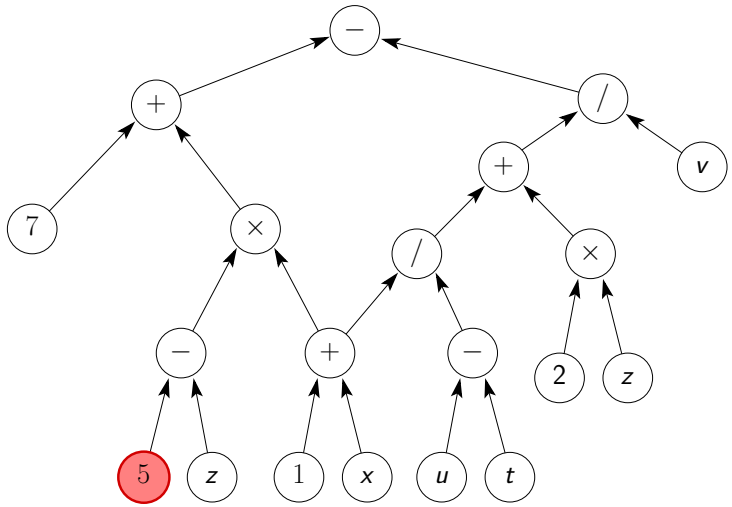
Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



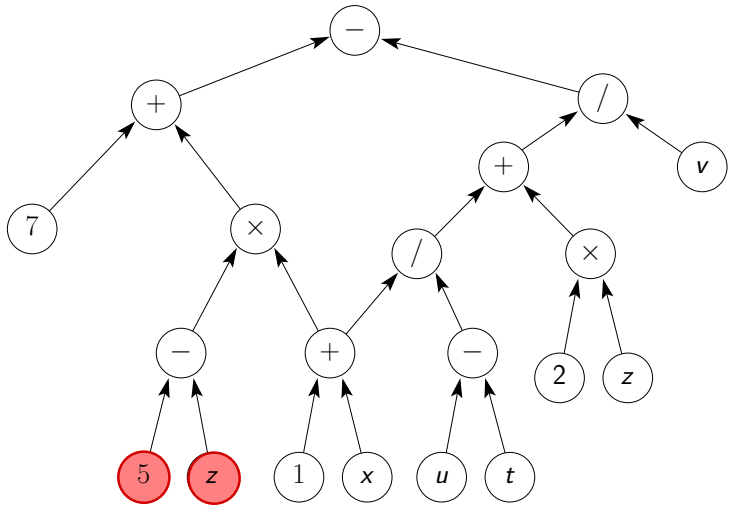
Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



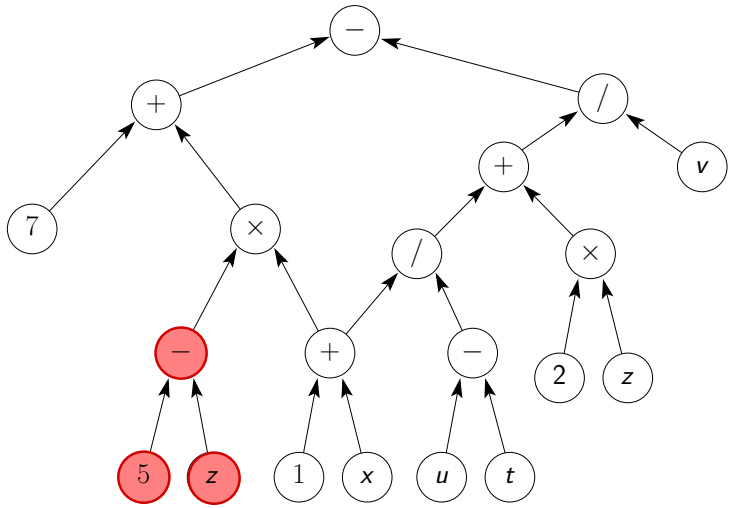
Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



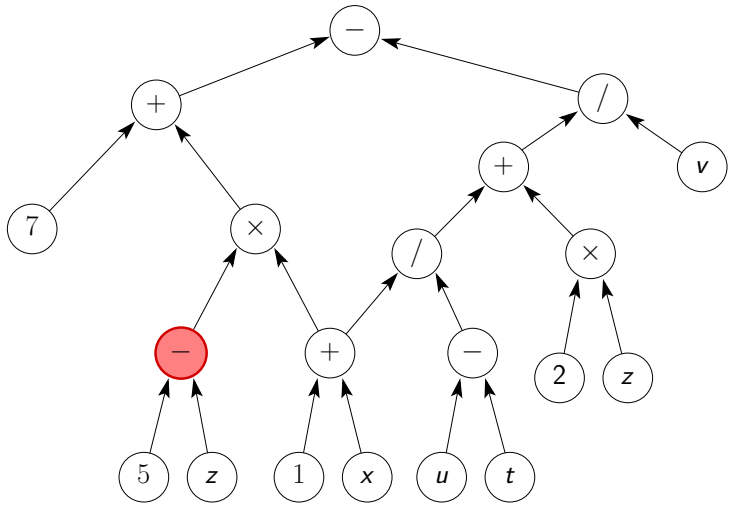
Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



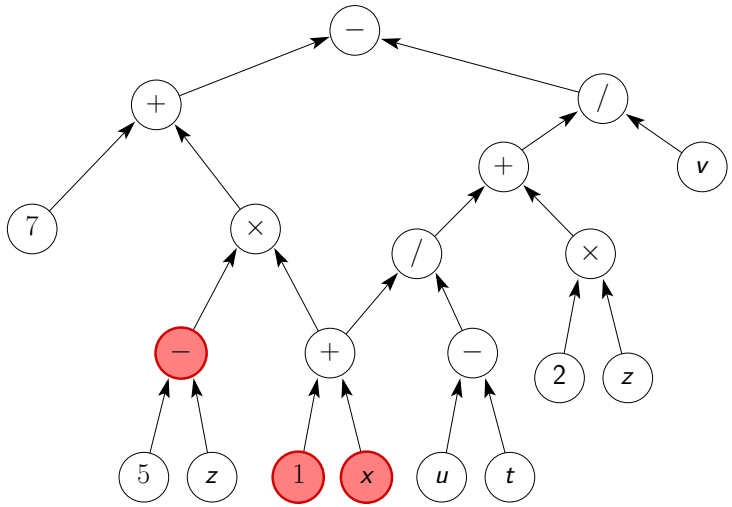
Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



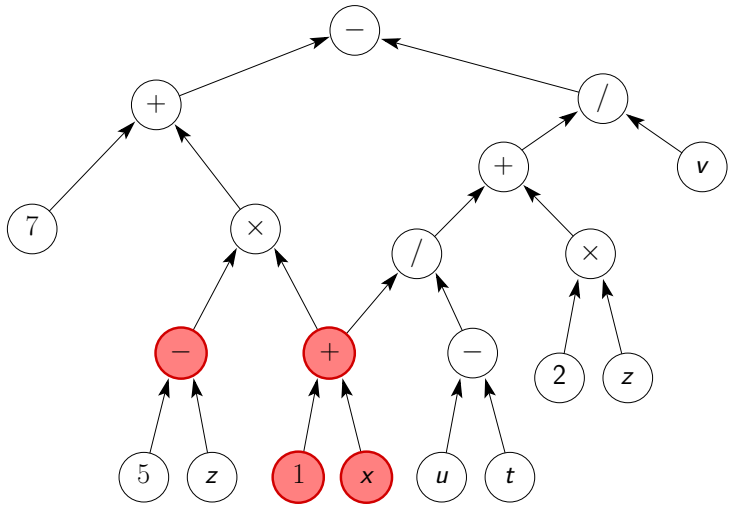
Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



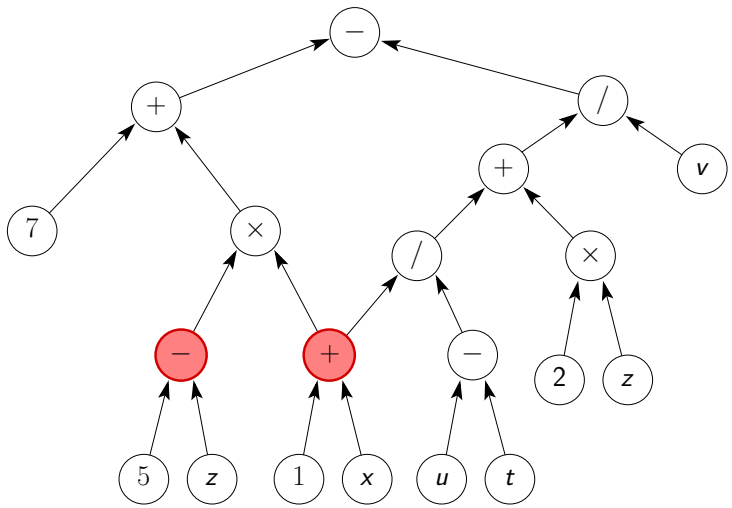
Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



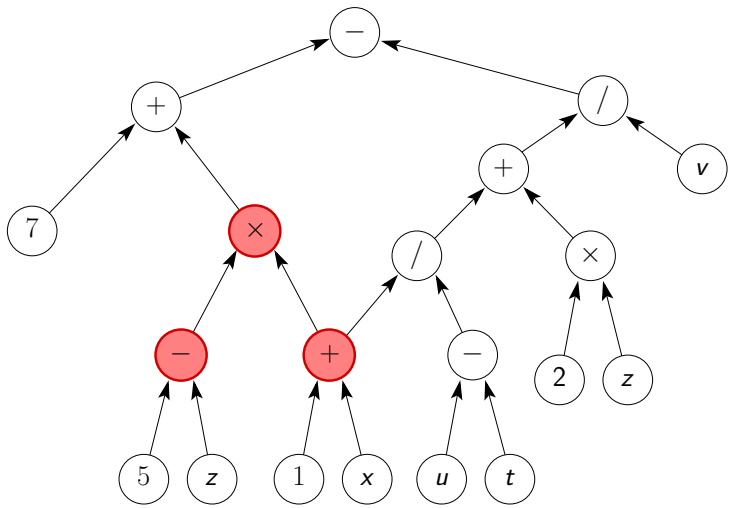
Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



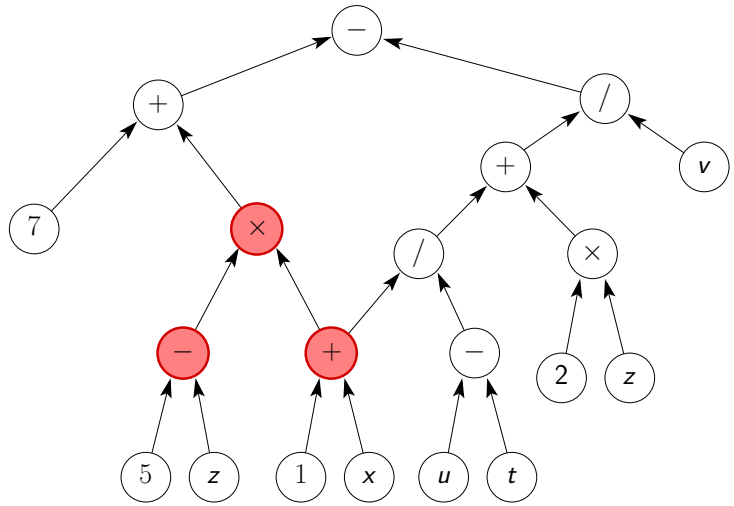
Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Rules

1. A pebble may be removed from a vertex at any time.
2. A pebble may be placed on a source node at any time.
3. If all predecessors of an unpebbled vertex v are pebbled, a pebble may be placed on v.



Analogy with register allocation:

- Rule 2: Load in register
- Rule 3: Compute new value (in new register)

Objective: Pebble all vertices at least once using minimum number of pebbles

(Black) Pebble Game – Complexity and variants

Progressive pebble game:

- Forbid pebbling twice the same vertex
- NP-Hard

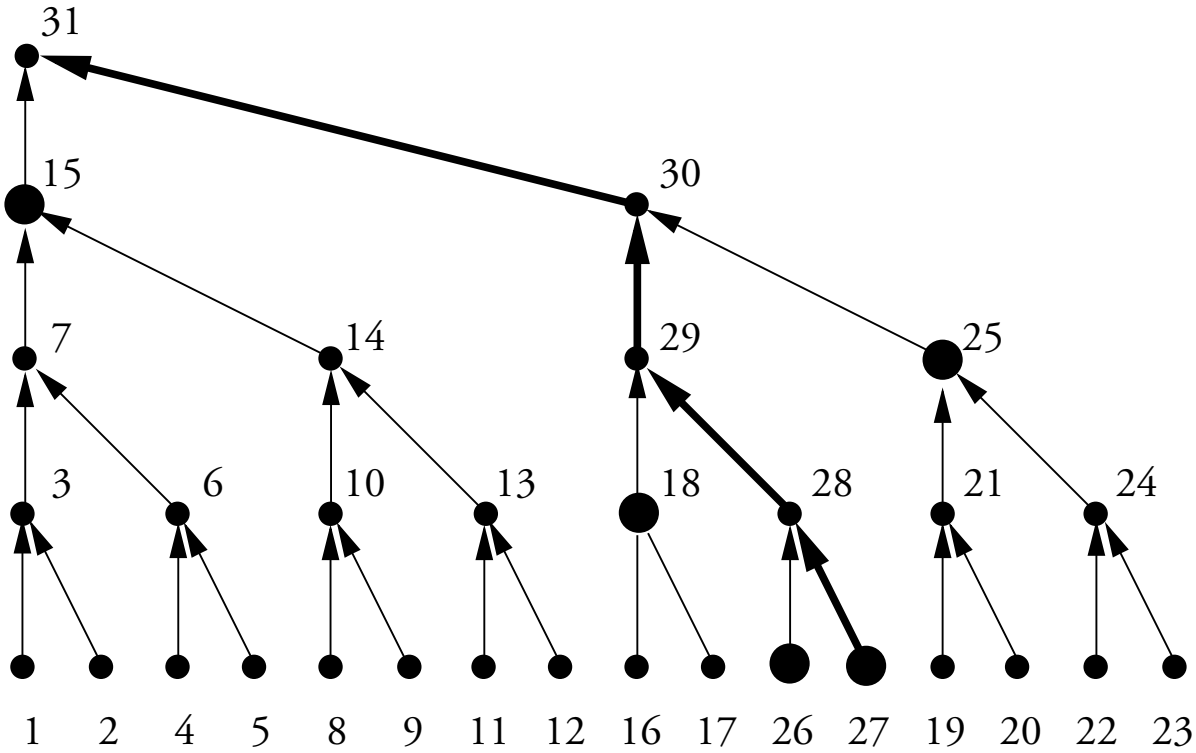
More general problem with re-computation:

- PSpace-complete

Variant with pebble shifting:

- Rule 3 $\rightarrow$ If all predecessors of an unpebbled vertex v are pebbled, a pebble may be shifted from a predecessor to v.
- Decrease the minimum number pebbles required for a graph by at most one (may induce large number of recomputations)

Complete binary tree of depth $k = 4$



Theorem.

Any pebbling strategy (with or without recomputation, without shifting) for the complete balanced binary tree of depth k uses at least $k + 2$ pebbles and $2^{k+1} - 1$ steps. There is a pebbling strategy that reaches both bounds.

NB: All proofs on the board will be made available online.

General trees

Lemma.

Depth-First Traversal are dominant

Depth-First: Totally pebble a subtree, remove all pebbles except on its root, before starting a sibling subtree.

Theorem.

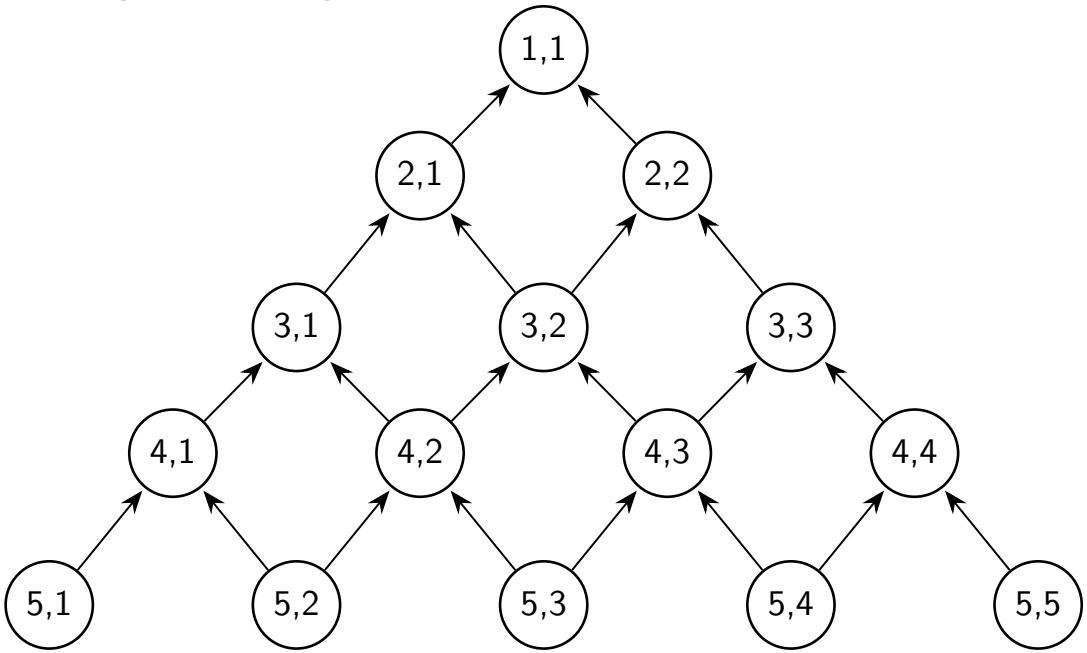
An optimal solution is obtained by ordering subtrees by non-increasing value of $P(i)$, where the peak $P(v)$ of the subtree rooted at v is recursively defined by:

$$P(v) = \begin{cases} 1 & \text{if } v \text{ is a leaf} \\ \max(k+1, \max_{i=1\ldots k} P(c_i) + i - 1) & \end{cases}$$

where $c_1, \ldots c_k$ are the children of v ordered such that $P(c_1) \geq P(c_2) \geq \cdots \geq P(c_k)$.

Homework 1 – deadline Sep. 22

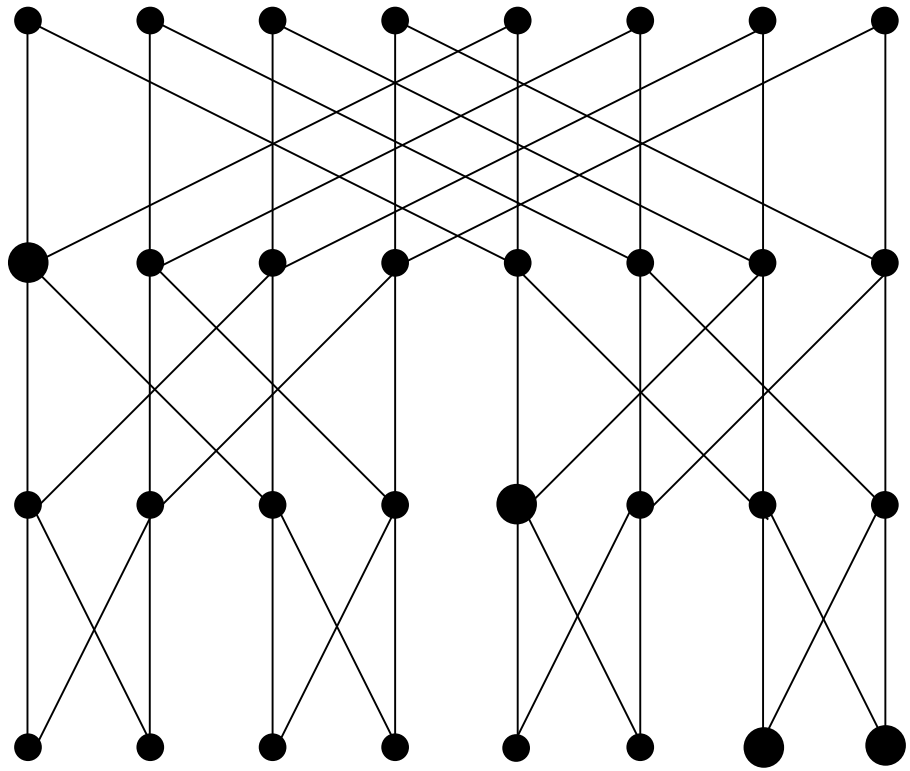
Consider a pyramid graph, obtained by slicing a 2D $n \times n$ mesh along its diagonal.



1. Describe a pebbling strategy (without shift) that pebbles the n-level pyramid using only $n+1$ pebbles (for $n \geq 2$).
2. Prove that any pebbling strategy (without shift) this graph uses at least $n+1$ pebbles (for $n \geq 2$).

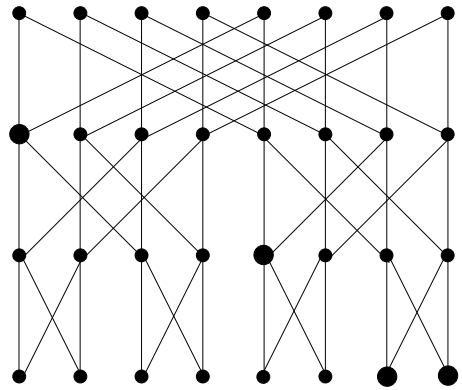
Space-Time tradeoffs – FFT example

- Fast-Fourrier Transform
- Recursive graph based on the "exchange graph" with 2 inputs and 2 outputs



FFT graph with 8 input/output vertices (depth $k = 3$)
$n = 2^k$ vertices at each level

Space-Time tradeoffs – FFT example



Strategy 1:

- Pebble one tree up to one output, then start over
  (variant: pebble two outputs before re-starting)
- Uses $k + 1$ pebbles
  (minimum value since it contains binary tree of depth k)
- Large number of recomputations

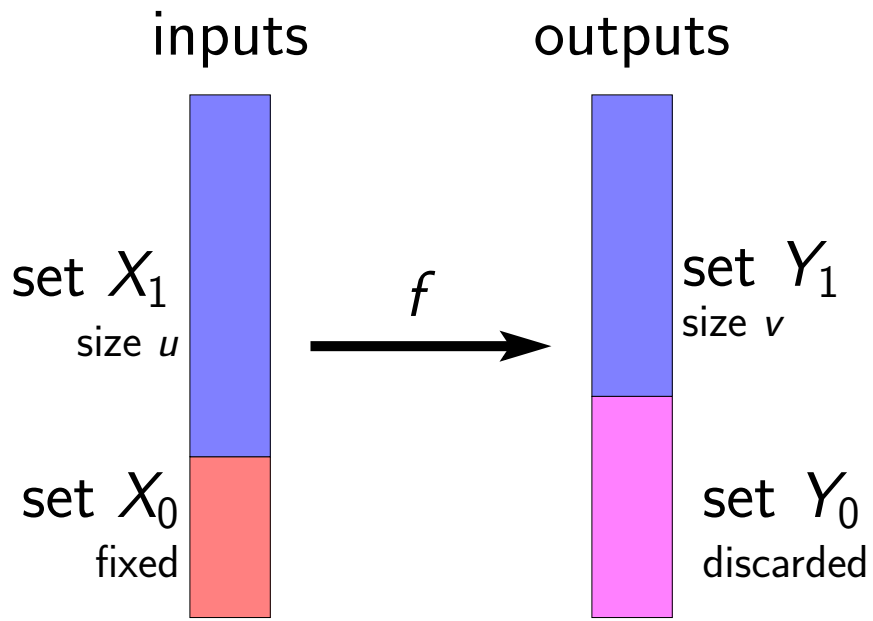
Strategy 2:

- Pebble level by level
- Requires $2n = 2^{k+1}$ pebbles
- No recomputations (minimum number of steps)

Grigoriev's Lower-Bound Method

Definition ($w(u, v)$-flow).

Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$. f has a $w(u, v)$-flow if for all partition (X_0, X_1) of the inputs and all partition (Y_0, Y_1) of the outputs, such that $|X_1| = u$ and $|Y_1| = v$, there exists a subfunction h of f obtained by assigning all variables in X_0 to fixed values and discarding output values in Y_0, such that h has at least $|A|^{w(u,v)}$ distinct points in the image of its domain.



Grigoriev's Lower-Bound Method

Definition ($w(u, v)$-flow).

Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$. f has a $w(u, v)$-flow if for all partition (X_0, X_1) of the inputs and all partition (Y_0, Y_1) of the outputs, such that $|X_1| = u$ and $|Y_1| = v$, there exists a subfunction h of f obtained by assigning all variables in X_0 to fixed values and discarding output values in Y_0, such that h has at least $|A|^{w(u,v)}$ distinct points in the image of its domain.

Definition ((α, n, m, p)-independant function).

f is an (α, n, m, p)-independant function if it has a $w(u, v)$-flow with $w(u, v) \geq v/\alpha - 1$ for all u, v such that $n - u + v \leq p$.

Lemma.

The product of two $N \times N$ matrices is $(1, 2N^2, N^2, N)$-independant.

Interpretation for matrix product:

- Consider y_1 outputs, fix x_0 inputs, with $x_0 + y_1 \leq p = N$
- $\Rightarrow$ large flow of information from the other inputs to Y_1

Lower-bound an application to matrix product

Theorem.

Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ be a function with a $w(u, v)$-flow. Any pebbling of any DAG computing f that takes T steps and S space respects:

$$T \geq \lfloor m/b \rfloor (n - d)$$

for any $b \leq m$ and d the largest integer such that $w(d, b) \leq S$.

Sketch of proof:

- Consider small multiple b of number of pebbles S
- Divide computations into intervals with b outputs pebbled
- Use pigeonhole to prove that we must read a lot of inputs during an interval:
  - If number of read inputs is small
  - Large flow by assumption
  - Large flow from inputs read before the interval
  - These inputs must be pebbled at the beginning of the interval
  - Flow cannot be larger than S, contradiction
- Multiply by the number of intervals

Lower-bound an application to matrix product

Theorem.

Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ be a function with a $w(u, v)$-flow. Any pebbling of any DAG computing f that takes T steps and S space respects:

$$T \geq \lfloor m/b \rfloor (n - d)$$

for any $b \leq m$ and d the largest integer such that $w(d, b) \leq S$.

Corollary

Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ be (α, n, m, p)-independant. For every pebbling of every DAG computing f using S pebbles and T steps, we have

$$\lceil \alpha(S + 1) \rceil T \geq mp/4$$

Sketch of proof:

- Carefully choose b
- Derive an upper bound on d (thanks to the flow)
- Computations on inequalities and case studies

Lower-bound an application to matrix product

Theorem.

Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ be a function with a $w(u, v)$-flow. Any pebbling of any DAG computing f that takes T steps and S space respects:

$$T \geq \lfloor m/b \rfloor (n - d)$$

for any $b \leq m$ and d the largest integer such that $w(d, b) \leq S$.

Corollary

Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ be (α, n, m, p)-independant. For every pebbling of every DAG computing f using S pebbles and T steps, we have

$$\lceil \alpha(S + 1) \rceil T \geq mp/4$$

Theorem.

Every pebbling strategy for any straight-line program computing the multiplication of two $N \times N$ matrices uses a space S and time T respecting the following inequality:

$$(S + 1)T \geq N^3/4$$