Space vs Time, Cache vs Main Memory

Marc Moreno Maza

University of Western Ontario, London, Ontario (Canada)

CS 4435 - CS 9624

Plan

1. Space vs Time
2. Cache vs Main Memory

Plan

1. Space vs Time
2. Cache vs Main Memory

Pebbling games and computing (1/2)

- A computation with **input** and **output** values can be modelled in various ways: **directed acyclic graph (DAG)**, **straight-line program (SLP)**.
- By computation, we mean the execution of a program, not a program itself, similarly to the **instruction stream DAG** of a `Cilk++` program.
- Thus, we assume that all operations (additions, multiplications) to be performed are precisely known.

Pebbling games and computing (2/2)

- Our purpose is then on how computer resources are used to realize this computatiuon. To do so, we make use of **pebbling games** on DAGs.
- From now on we consider a connected directed acyclic graph $G = (V, E)$:
  - Each vertex represents an **operation** and its **result**.
  - An edge from a vertex v_1 to a vertex v_2 indicates that the result of v_1 is needed for performing the operation of v_2.
  - A vertex v of G is an **input** (resp. **output**) if it has no predecessors (resp, no successors).
  - The sets of inputs and outputs are respectively denoted by $I(G)$ and $O(G)$. Note that these sets are disjoint.

The red pebble game (1/2)

The **red** pebble game is played on a directed and connected acyclic graph $G = (V, E)$ using four rules:

(R_1) **Input rule:** A pebble can be placed on an input vertex at any time.

(R_2) **Output rule:** Each output vertex must be pebbled at least once.

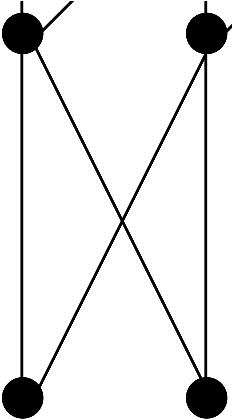
(R_3) **Compute rule:** A pebble can be placed on or moved to any non-input vertex if all of its immediate predecessors carry pebbles.

(R_4) **Delete rule:** pebble can be removed at any time.

The red pebble game (2/2)

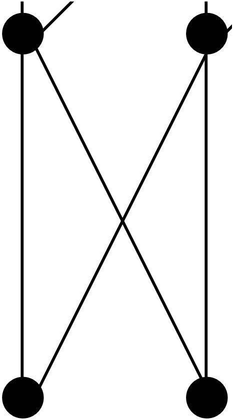
- A **pebbling strategy** determines sequence of rules invoked on vertices of a graph.
- A strategy uses **space** S if it uses at most S pebbles. It uses **time** T if the **input rule** and **compute rule** are invoked T times in total.
- The minimum space $S_{\min}$ to pebble the graph G is the smallest space of any strategy that pebbles G.
- We shall see that the FFT graph exhibits a tradeoff between space and time: the time required when the minimum space is used is strictly more than that required when more space is available.

FFT graph for 2 input nodes



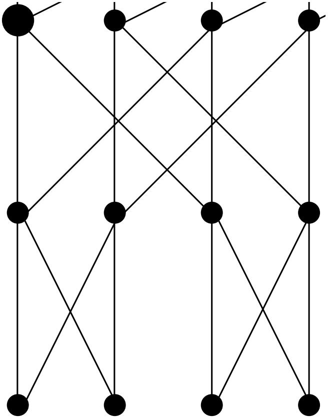
What is $S_{\min}$? What is T when $S = S_{\min}$? What is T when $S = S_{\min} + 1$?

FFT graph for 2 input nodes



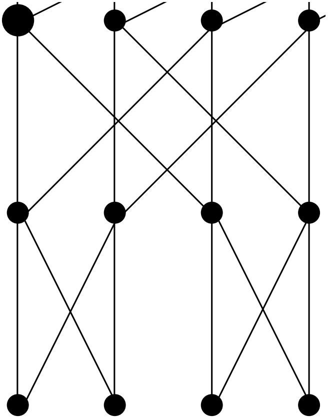
We have $S_{\min} = 2$. Moreover $S = 2 \implies T \geq 5$ while $S = 3 \implies T \geq 4$.

FFT graph for 4 input nodes



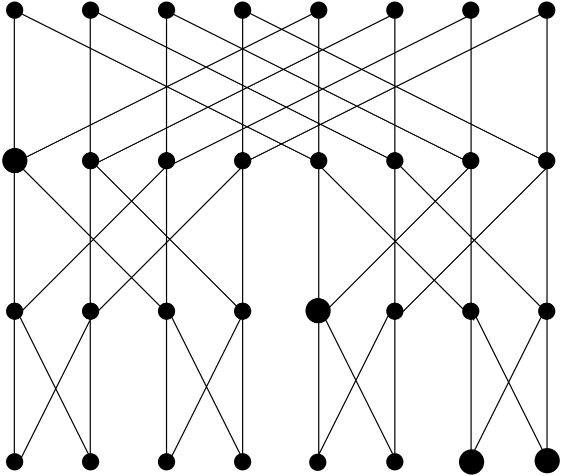
What is $S_{\min}$?

FFT graph for 4 input nodes



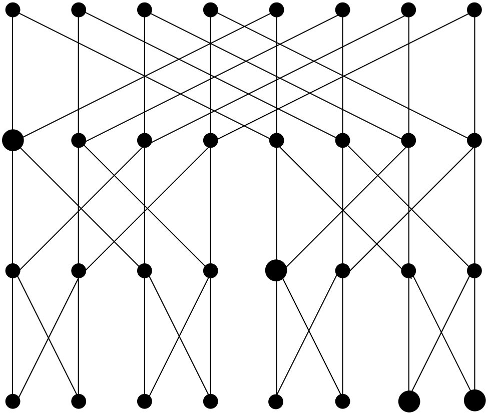
We have $S_{\min} = 3$.

FFT graph for 8 input nodes



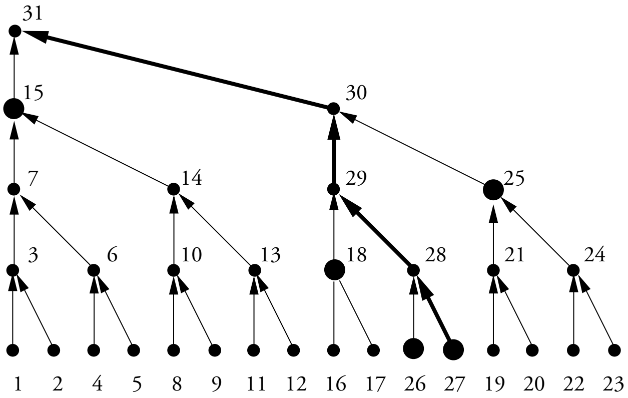
What is $S_{\min}$?

FFT graph for 8 input nodes



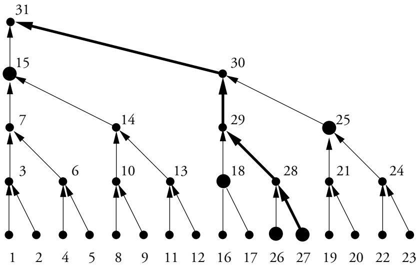
We have $S_{\min} = 4$. More generally, for the FFT graph on $n = 2^k$ inputs we have $S_{\min} = k + 1$.

Pebbling a complete binary tree (1/4)



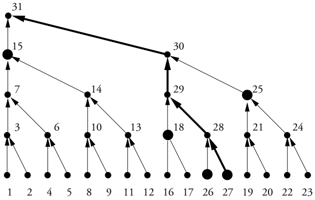
What is $S_{\min}$?

Pebbling a complete binary tree (2/4)



Theorem. The complete balanced binary on $n = 2^k$ inputs has $S_{\min} = k + 1 = \log_2(n) + 1$. It can be pebbled in time $T = 2n - 1$ steps, but no fewer.

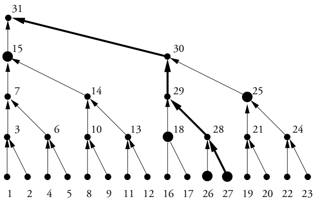
Pebbling a complete binary tree (3/4)



Proof (1/2).

1. Each path has $k+1$ vertices.
2. Initially each path from an input to the output is free of pebbles.
3. Finally, a pebble is on the output and thus all paths contain a pebble.
4. **Therefore**, there is a last time at which a path is open.
5. When placing a pebble on last input, all paths from other inputs to vertices on the path carry 1 pebble; moreover there are k such paths.
6. **Therefore**, we have $S_{\min} \geq k+1$.

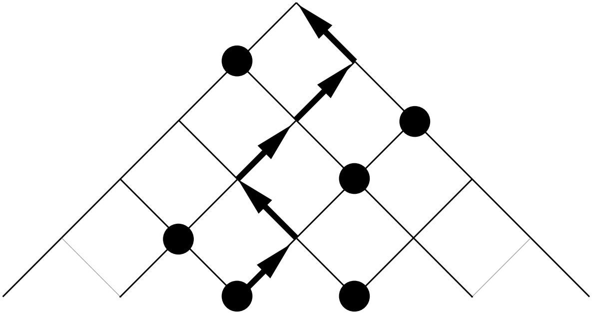
Pebbling a complete binary tree (4/4)



Proof (2/2).

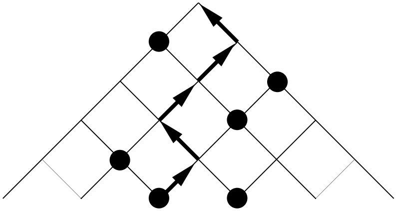
1. We prove by induction that $S_{\min} = k + 1$ holds and that $T = 2n - 1$ holds for $S = S_{\min}$.
2. The property is true for $n = 1$, that is, for $k = 0$.
3. Assume $n \geq 1$. We do the left subtree in time $2(n/2) - 1$ using k pebbles (by induction) and leave one pebble at its root.
4. We do the rigth subtree in time $2(n/2) - 1$ using k pebbles too.
5. **Therefore**, we have: $T = 2((n/2) - 2 + 1$.

Pebbling the pyramid graph (1/4)



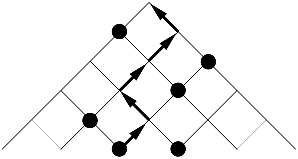
What is $S_{\min}$?

Pebbling the pyramid graph (2/4)



Theorem. For the pyramid graph $P(m)$ on m inputs, we have $S_{\min} = m$. With $S = S_{\min}$ pebbles, the graph $P(m)$ can be pebbled in time $T = n$, where $n = m(m+1)/2$ is the number of vertices of $P(m)$.

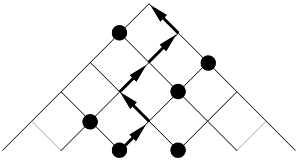
Pebbling the pyramid graph (3/4)



Proof (1/2).

1. The last open path argument can be used to show that $S_{\min} \geq m$ holds.
2. To pebble $P(m)$ with m pebbles, place pebbles on all inputs.
3. Move the leftmost pebble up one level.

Pebbling the pyramid graph (4/4)

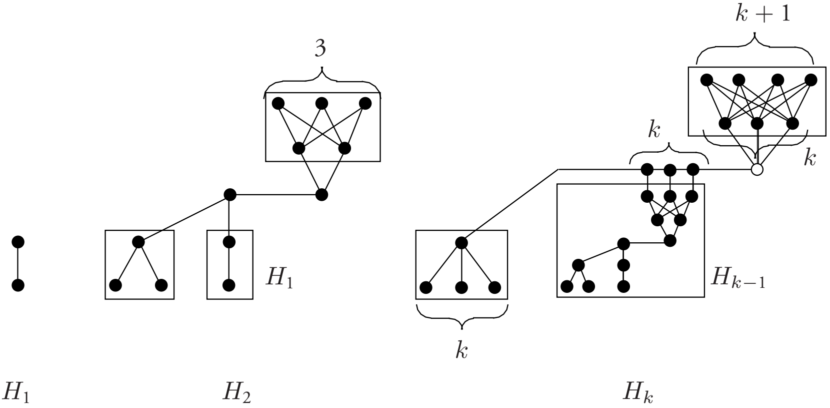


Proof (2/2).

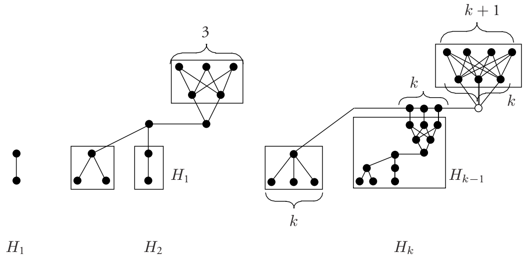
1. Now all vertices one level up can be pebbled using $m - 1$ pebbles.
2. Repeat this procedure at all subsequent levels.
3. Each vertex is pebbled once.

Observe that $S_{\min}$ is about $\sqrt{n}$, where n is the number of vertices of $P(m)$, which is much larger than for binary trees.

Extreme Tradeoffs (1/7)

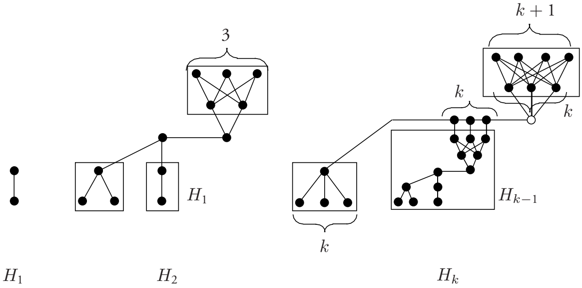


Extreme Tradeoffs (2/7)



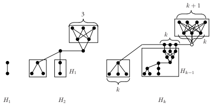
For $k \geq 3$, the grap H_k is of a k- input tree, $T(k)$, a **spine**, $S(k)$, of k vertices connected to the k outputs of H_{k-1}, an **open** vertex, and a complete bipartite graph $BP(k)$, with k inputs and $k+1$ outputs.

Extreme Tradeoffs (3/7)



What is $S_{\min}(H_1)$? $S_{\min}(H_2)$? $S_{\min}(H_k)$, for $k \geq 3$?

Extreme Tradeoffs (4/7)

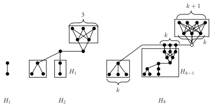


Lemma. $S_{\min}(H_k) = k$ for all $k \geq 1$.

Proof .

1. Since the tree $T(k)$ needs k pebbles, we have $S_{\min}(H_k) \geq k$.
2. We show k pebbles suffice assuming outputs of H_{k-1} can be pebbled in succession with $k-1$ pebbles.
3. Advance pebble to tree output, use $k-1$ pebbles on H_{k-1} to pebble its $k-1$ outputs and advance pebbles along spine.
4. Advance pebble to open vertex.
5. Put k pebbles on the $BP(k)$ inputs and then pebble one output of $BP(k)$.
6. Repeat the whole process for each additional output of $BP(k)$.

Extreme Tradeoffs (5/7)



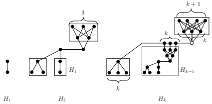
Lemma. H_k has $N(k) = 2k^2 + 5k - 6$ vertices, for all $k \geq 2$.

Proof .

1. Base case: $N(2) = 12 = 8 + 10 - 6$.
2. By induction:

$$\begin{array}{rcl} N(k) & = & N(k-1) + (k+1) + k + 1 + (k + k + 1) \\ & = & N(k-1) + 4k + 3 \\ & = & 2(k-1)^2 + 5(k-1) - 6 + 4k + 3 \\ & = & 2k^2 + 5k - 6 \end{array}$$

Extreme Tradeoffs (6/7)

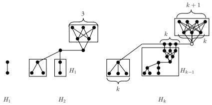


Lemma. H_k requires at least $(k+1)!$ steps to be pebbled with $S = S_{\min}(H_k)$ pebbles.

Proof .

1. When $S = S_{\min}(H_k)$, the subgraph H_{k-1} must be repebbled $k+1$ times.
2. Indeed, pebbling one output of $BP(k)$, removes all pebbles from H_{k-1}.

Extreme Tradeoffs (7/7)



Lemma. H_k can be pebbled in $N(k)$ steps with $k+1$ pebbles.

Proof .

1. Inductive Hypothesis: When $k+1$ pebbles are used, assume all outputs of H_{k-1} can be pebbled in succession using $k+1$ pebbles without repebbling any vertices.
2. We advance k pebbles to the inputs of the $BP(k)$ without repebbling any vertices.
3. The remaining pebble is used to pebble outputs of the $BP(k)$ in succession.

Plan

The Red-Blue Pebble Game (1/3)

The **red**-**blue** pebble game is played on a directed and connected acyclic graph $G = (V, E)$.

- At any point of the game, some vertices have **red** pebbles, others have **blue**, others have pebbles of both types, others have no pebbles.
- A **configuration** is a pair of subsets (R, B) of the vertex set V such that any vertex $v \in R$ (resp. $v \in B$) has a **blue** pebble (resp. **red** pebble).
- The **initial configuration** is the one given by $(\emptyset, I(G))$.
- The **final configuration** is the one given by $(\emptyset, O(G))$.

The Red-Blue Pebble Game (2/3)

The rules of the **red**-**blue** pebble game are

(R_1) **Input rule:** A **red** pebble may be placed on any vertex that has a **blue** pebble.

(R_2) **Output rule:** A **blue** pebble may be placed on any vertex that has a **red** pebble.

(R_3) **Compute rule:** If all immediate predecessors of a vertex v have **red** pebbles then a **red** pebble may be placed on v.

(R_4) **Delete rule:** A pebble **red** or **blue** may be removed at any time from any vertex.

The Red-Blue Pebble Game (3/3)

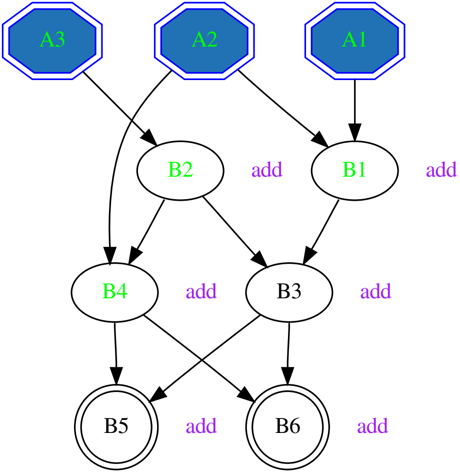
Key concepts:

- A **transition** is an ordered pair of configurations, the second of which follows from the first according to one of the rules (R_1) to (R_4).
- A **caculation** is a sequence of configurations, each successive pair of which form a transition.
- A **complete caculation** is one that begins with the initial configuration and ends with the final configuration.

Application to cache complexity (1/7)

- A graph on which the **red**-**blue** pebble game is played can model a computation performed on a two-level memory structure, say, a fast memory (or cache) and a slow memory.
- Each vertex represents an **operation** and its **result**.
- An edge from a vertex v_1 to a vertex v_2 indicates that the result of v_1 is needed for performing the operation of v_2.
- An operation can be performed only if **all operands reside in cache** (or fast memory).
- The maximum allowable number of **red** (or **blue**) pebbles on the graph at any point in the game corresponds to the number of the **red**-**blue** pebble game is words available for use in the fast (or slow) memory, respectively.

Application to cache complexity (2/7)



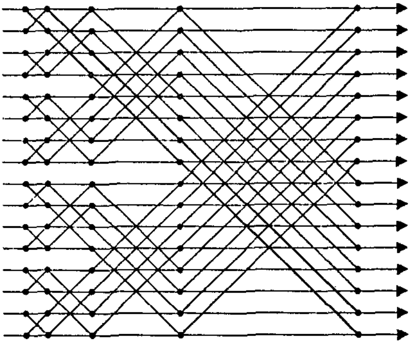
Application to cache complexity (3/7)

- Placing a **red** pebble using Rule (R_3) corresponds to performing an operation and storing the result in cache
- Placing a **blue** pebble using Rule (R_2) corresponds to storing a copy of a result (currently in the fast memory) into the slow memory.
- Placing a **red** pebble using Rule (R_1) corresponds to retrieving a copy of a result (currently in the slow memory) into the fast memory.
- Removing a red or **red** or **blue** pebble using Rule (R_4) means freeing a memory location in the fast or slow memory respectively.

Application to cache complexity (4/7)

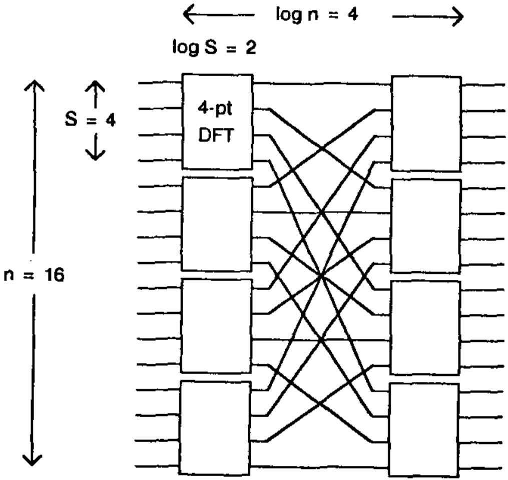
- In what follows, the fast memory can only hold S words, where S is a constant, while the slow memory is arbitrarily large.
- For any given connected DAG, we are interested in the **I/O time**, denoted by Q, which is the minimum number of transitions according to Rules (R_1) or (R_2) required by any complete calculation.
- In the original work of (J.W. Hong, H.T. Kung, 1981) a "static problem" is associated with the **red**-**blue** pebble game, the *S-Partitioning Problem*. Then lower bounds for the S-Partitioning Problem lead to lower bounds for the **red**-**blue** pebble game.
- To establish bounds like those (but weaker) of (J.W. Hong, H.T. Kung, 1981) we will follow a simpler approach due to J.E. Savage (see his book *Models of Computations*) reducing to the **red** pebble game.

Application to cache complexity (5/7)



Theorem. Assume $S \geq 3$. For the n-point FFT graph we have $Q \log(S) \in \Omega(n \log(n))$. Moreover, there is a pebbling strategy for which $Q \log(S) \in \Theta(n \log(n))$ holds.

Application to cache complexity (6/7)



Decomposing the 16-point FFT graph with $n = 16$ and $S = 4$.

Application to cache complexity (7/7)

Theorem. For any DAG G encoding an algorithm multiplying two square matrices of order n (based on an $\Theta(n^3)$-work algorithm) and for every pebbling strategy $\mathcal{P}$ for G in the **red**-**blue** pebble game that uses $S \geq 3$ **red** pebbles, the I/O-time satisfies the following lower bound:

$$Q_{\mathcal{P}} \in \Omega(n^3/\sqrt{S}).$$

Furthermore, for $S \geq 3$, there exists a pebbling strategy for which we have:

$$Q_{\mathcal{P}} \in \theta(n^3/\sqrt{S}).$$

The S-Partitioning Problem (1/6)

Let $G = (V, E)$ be a directed and connected acyclic graph. Let $X, Y \subset V$ be two proper subsets of V, hence $X \neq \emptyset$ and $Y \neq \emptyset$ hold.

- A subset $D \subset V$ is a **dominator set** for X if for every path from a vectex of $I(G)$ to a vertex of X has at least one vertex in D.
- The **minimum set** of X is the set of vertices $v \in X$ such that none of the successors of v belongs to X.
- We say that Y **depends on** X whenever there exists $(v, w) \in X \times Y$ such that $(v, w) \in E$ holds.

The S-Partitioning Problem (2/6)

Let $G = (V, E)$ be a directed and connected acyclic graph and S be a positive integer. A partition $\{V_1, \ldots, V_h\}$ of V is called an S-partition of G if the following properties hold for each $i = 1 \cdots h$:

1. V_i admits a **dominator set** D_i with $| D_i | \leq S$,
2. the **minimum set** M_i of V_i satisfies $| M_i | \leq S$,
3. There is **no cyclic dependence** among $V_1, \ldots, V_h$.

The S-Partitioning Problem (3/6)

Consider a **red**-**blue** pebble game on G using at most S **red** pebbles.

Denote by C a complete calculation. There exists an integer $h \geq 2$ and a sequence of h consecutive subcalculations $C_1, C_2, \ldots, C_h$ such that the following holds:

- for each $i = 1 \cdots (h-1)$, the subcalculation C_i has exactly S transitions using Rules (R_1) or (R_2),
- C_h has at most S transitions using Rules (R_1) or (R_2),

The S-Partitioning Problem (4/6)

For each $i = 1 \cdots (h-1)$, define V_i to be the largest subset of V with the following properties:

- During the subcalculation C_i each vertex of V_i receives a **red** pebble thanks to Rules (R_1) or (R_3).
- At the end of the subcalculation C_i each vertex of $v \in V_i$
  - either has **red** pebbles or **blue** pebbles that are placed on v during C_i,
  - or v has a successor in V_i
- v does not belong to any V_j for $j = (i+1) \cdots h$.

Lemma. The set $\{V_1, V_2, \ldots, V_h\}$ is a $2S$-partition of V.

The S-Partitioning Problem (5/6)

Theorem. Let $G = (V, E)$ be a directed and connected acyclic graph. Any complete calculation **red**-**blue** pebble game on G using at most S **red** pebbles is associated with a $2S$- partiton of G such that

$$S\,h \geq q \geq S\,(h-1).$$

where q is the I/O time required by the calculation and h is the number of parts in the $2S$- partiton.

Sketch of proof.

1. $\{V_1, V_2, \ldots, V_h\}$ (as defined before) is a $2S$-partition of V,
2. For each $i = 1 \cdots (h-1)$, exactly S transitions with Rules (R_1) or (R_2) correspond to V_i,
3. No more than S transitions with Rules (R_1) or (R_2) correspond to V_h.
4. The inequalities follow.

The S-Partitioning Problem (6/6)

Corollary. Let $G = (V, E)$ be a directed and connected acyclic graph. Let $P(2S)$ be the minimum number of parts in a $2S$-partition of V. Then we have:

$$Q \geq S\,(P(2S) - 1).$$

Using this Corollary, lower bounds for $P(2S)$ translate into lower bounds for Q.

Reduction to the **red** pebble game (1/4)

The *S*-**span** of the DAG $G = (V, E)$, denoted by $\rho(S, G)$, is

- the maximum number of vertices of G that can be pebbled with S red pebbles in the **red** pebble game,
- maximized over all initial placements of S red pebbles,
- which means that the initialization rule is disallowed.

Theorem. We have: $\lceil Q/S \rceil \ \rho(2S, G) \ \ \geq \ \mid V \mid - \mid I(G) \mid$.

Reduction to the **red** pebble game (2/4)

Theorem. We have: $\lceil Q/S \rceil \; \rho(2S, G) \quad \geq \quad | \; V \; |$.

Sketch of proof (1/3).

1. Let $\mathcal{P}$ be a pebbling strategy with S pebbles.
2. Divide $\mathcal{P}$ into consecutive sequential sub-pebblings (or calculations) $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_h$, where each sub-pebbling has S I/O operations (rules (R_1) and (R_2)) except possibly the last one.
3. Thus we have $h = \lceil Q/S \rceil$.
4. We shall exhibit an upper bound R to the number of vertices of G pebbled with **red** pebbles in any sub-pebbling $\mathcal{P}_j$
5. This will satisfy $h\, R \geq | \; V \; |$.

Reduction to the **red** pebble game (3/4)

Theorem. We have: $\lceil Q/S \rceil \, \rho(2S, G) \;\; \geq \;\; | \, V \, |$.

Sketch of proof (2/3).

1. The upper bound on R is developed by adding S new red pebbles and showing that we may use these new pebbles to move all I/O operations in each sub-pebbling $\mathcal{P}_j$ to either the beginning or the end of the sub-pebbling without changing the number of computation steps or I/O operations.
2. Consider a vertex v carrying a **red** pebble at some time during $\mathcal{P}_j$ and which is pebbled for the first time with a **blue** pebble during $\mathcal{P}_j$.
3. Instead of pebbling v with a **blue** pebble, we use a **new red** pebble to keep **red** until its last output operation which is preserved and moved to the end of $\mathcal{P}_j$,.

Reduction to the **red** pebble game (4/4)

Theorem. We have: $\lceil Q/S \rceil \, \rho(2S, G) \;\; \geq \;\; |\, V \,|$.

Sketch of proof (3/3).

1. Consider a vertex v carrying a **blue** pebble at the start of $\mathcal{P}_j$ and that is given a **red** during $\mathcal{P}_j$; consider the first pebbling of this kind for v.
2. Then, we use a **new red** pebble instead.
3. This allows us to move this input operation at the beginning of $\mathcal{P}_j$, without violating the precedence conditions of G.
4. It follows that that the number of vertices that are pebbled with **red** pebbles during the **computations** of $\mathcal{P}_j$ is $\rho(2S, G)$.