Optimal Automatic Multipass Shader Partitioning by Dynamic Programming

Alan Heirich
Sony Computer Entertainment America
31 July 2005

Disclaimer

This talk describes GPU architecture research carried out at Sony Computer Entertainment.

It does not describe any commercial product.

In particular, this talk does not discuss the PLAYSTATION 3 nor the RSX.

Outline

The problem:

- Automatically compile large shaders for small GPUs.

The insight:

- This is a classical job-shop scheduling problem.

The proposed solution:

- Dynamic Programming.

Outline++

The problem:

- Automatically compile large shaders for small GPUs.
  - Exhaust registers, interpolants, pending texture requests, ...
  - Goal: optimal solutions, scalable algorithm.

The insight:

- This is a classical job-shop scheduling problem.
  - Job-shop scheduling is NP-hard/complete.
  - *Well-studied problem, many solution algorithms exist.*

The proposed solution:

- Dynamic Programming.
  - Satisfies nonlinear objective function.
  - Optimal and (semi-)scalable.

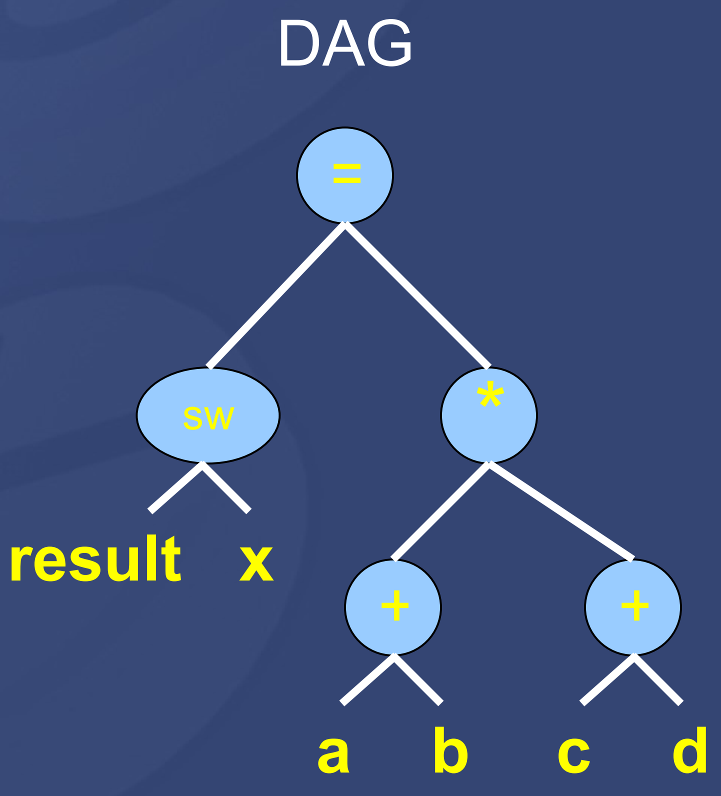
The Problem

Physical resources are limited.

- Rasterized interpolants.
- GP registers.
- Pending texture requests.
- Instruction count.
- etcetera

A very simple example:

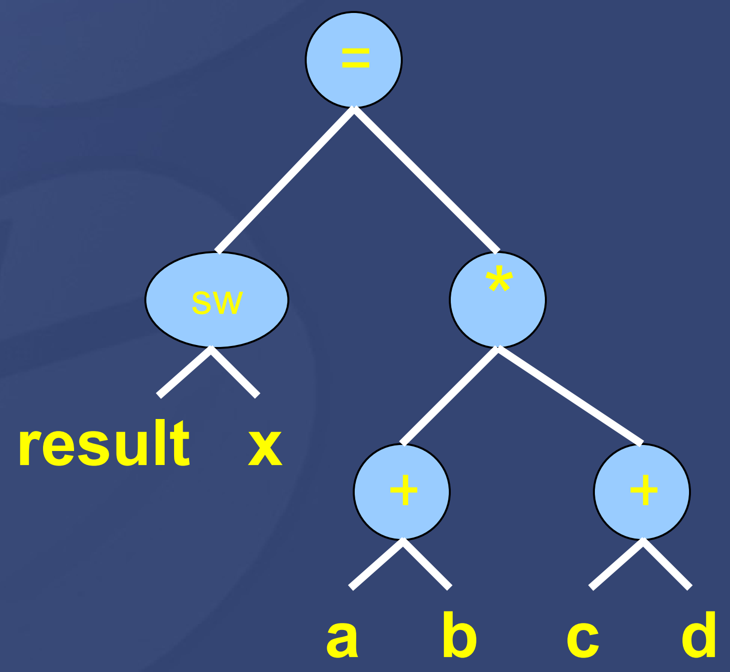
- result.x = (a+b)*(c+d)
- Requires three GP registers
  - Multiple passes with two registers



result.x = (a+b)*(c+d)

R0	R1	
a		Load R0=a
a	b	Load R1=b
a+b	b	R0=+(R0,R1)
a+b	c	Load R1=c
a+b	c	Store R0 → aux
d	c	Load R0=d
c+d	c	R0=+(R0,R1)
c+d	a+b	***New Pass***
(a+b)*(c+d)	a+b	R0=*(R0,R1)
(a+b)*(c+d)	a+b	Store R0 → swizzle(result,x)

DAG



The MPP Problem

Multipass Partitioning Problem [Chan 2002]

Given:
- An input DAG.
- A GPU architecture.

Find:
- A schedule of DAG operations.
- A partition of that schedule into passes.

Such that:
- Schedule observes DAG precedence relations.
- Schedule respects GPU resource limits.
- Runtime of compiled shader is minimal *(optimality).*
  - *(Chan: number of passes is minimal.)*

References

Graphics Hardware 2002:

- *Efficient partitioning of fragment shaders for multipass rendering on programmable graphics hardware.*
  E. Chan, R. Ng, P. Sen, K. Proudfoot, P. Hanrahan

Graphics Hardware 2004:

- *Efficient partitioning of fragment shaders for multiple-output hardware.*
  T. Foley, M. Houston, P. Hanrahan
- *Mio: fast multipass partitioning via priority-based instruction scheduling.*
  A. Riffel, A. Lefohn, K. Vidimce, M. Leone, J. Owens

Requirements: Optimal, Scalable

Nonlinear cost function.

- Depends on current machine state.

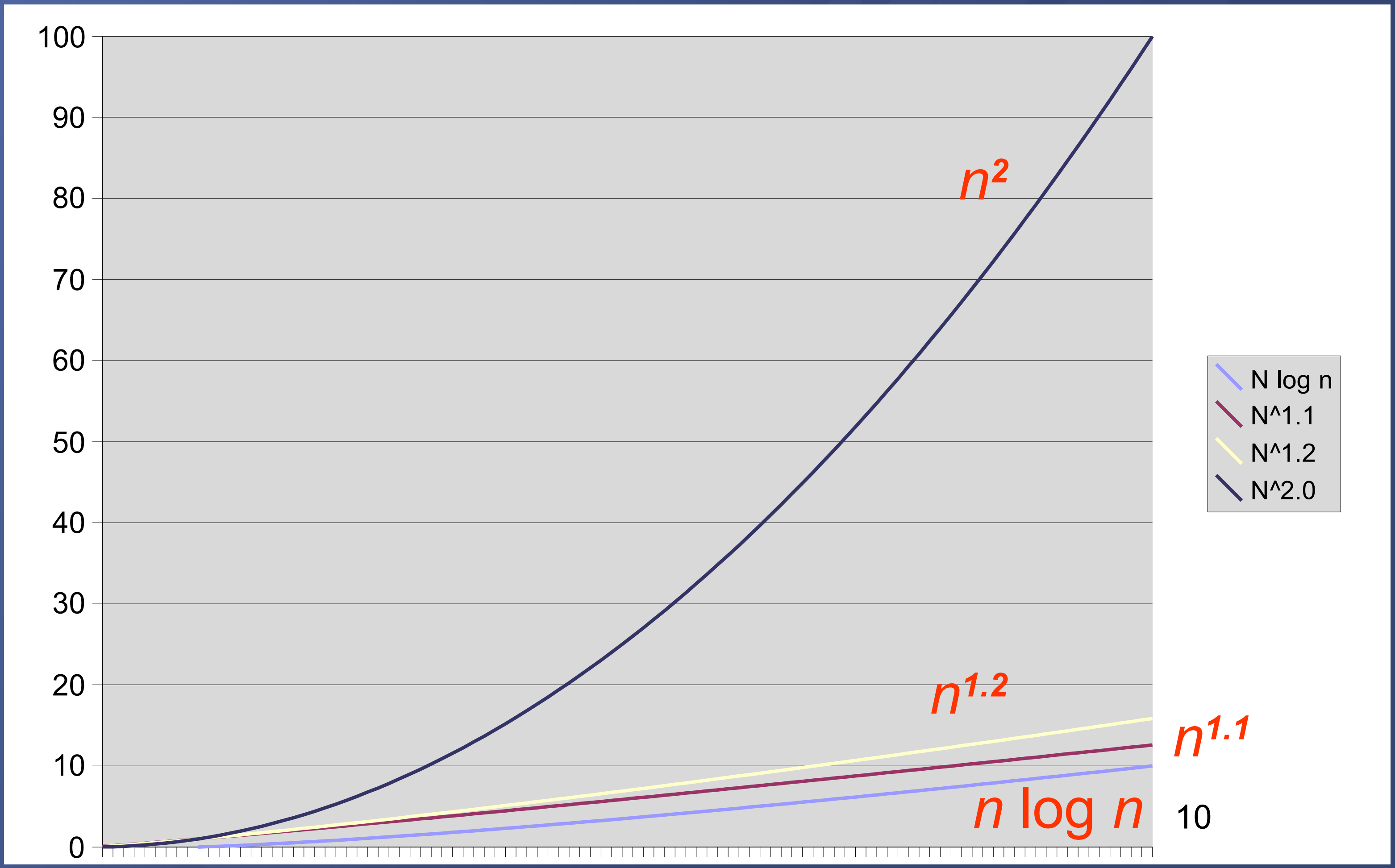
Optimal solutions:

- (Many) fine-grained passes.
- Long shaders.
  - High-dimensional solution space.
  - Many local minima (suboptimal solutions).

Scalable algorithm:

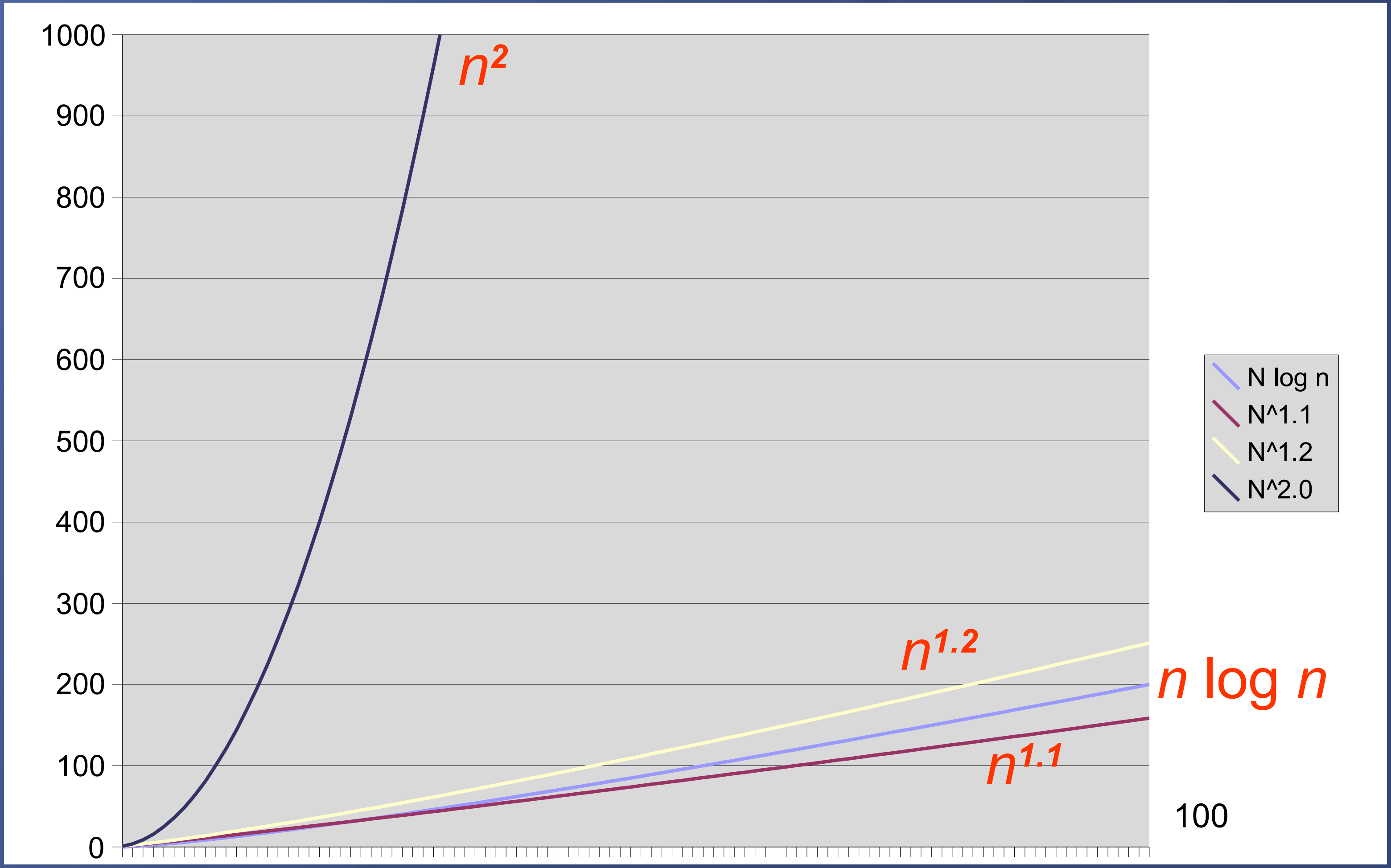
- Compile-time cost must not grow unreasonably.
  - $O(n \log n)$ is scalable.
  - $O(n^2)$ is not scalable.

Scalability, n=10



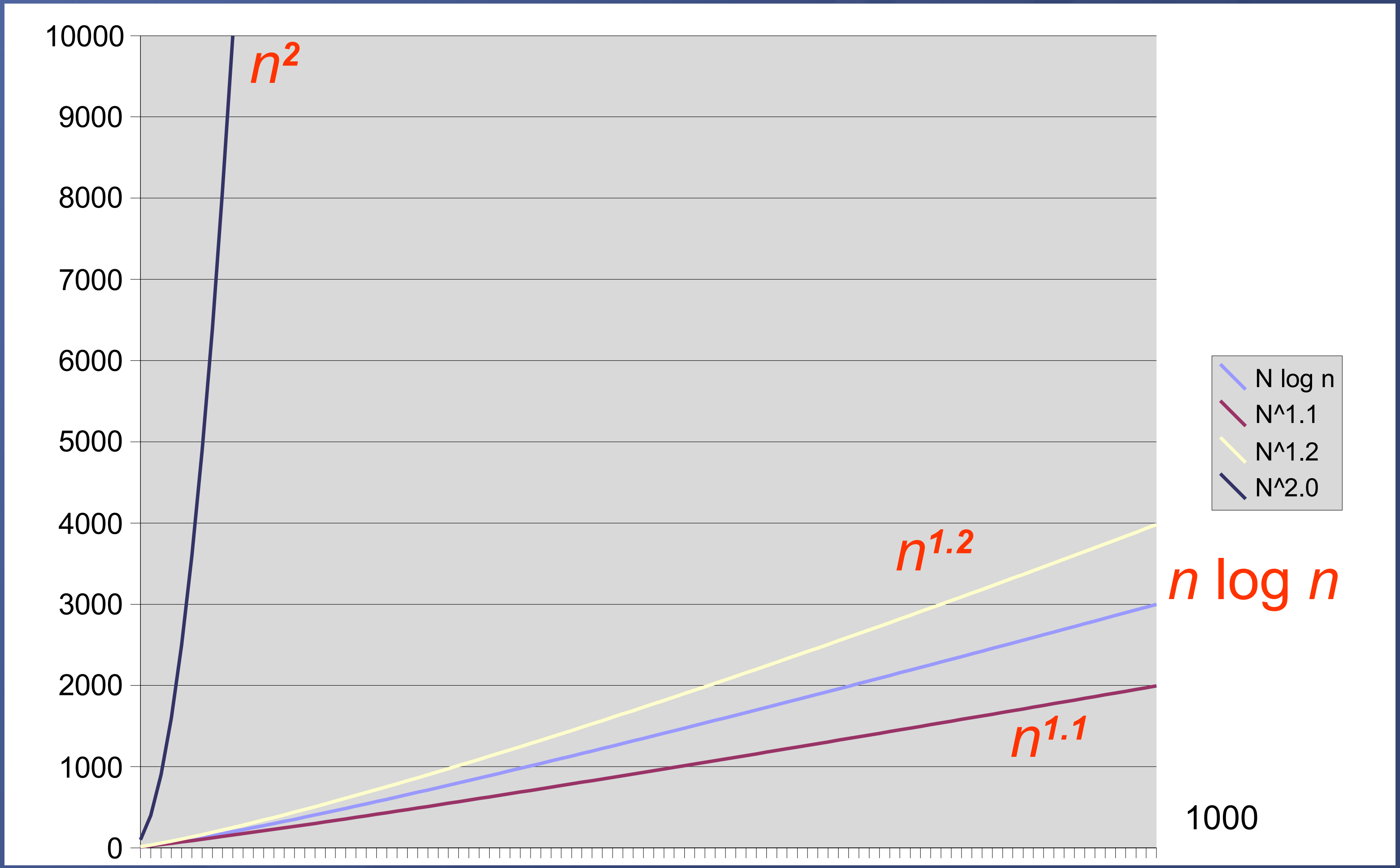
Scalability, n=100

(Current vertex shaders)



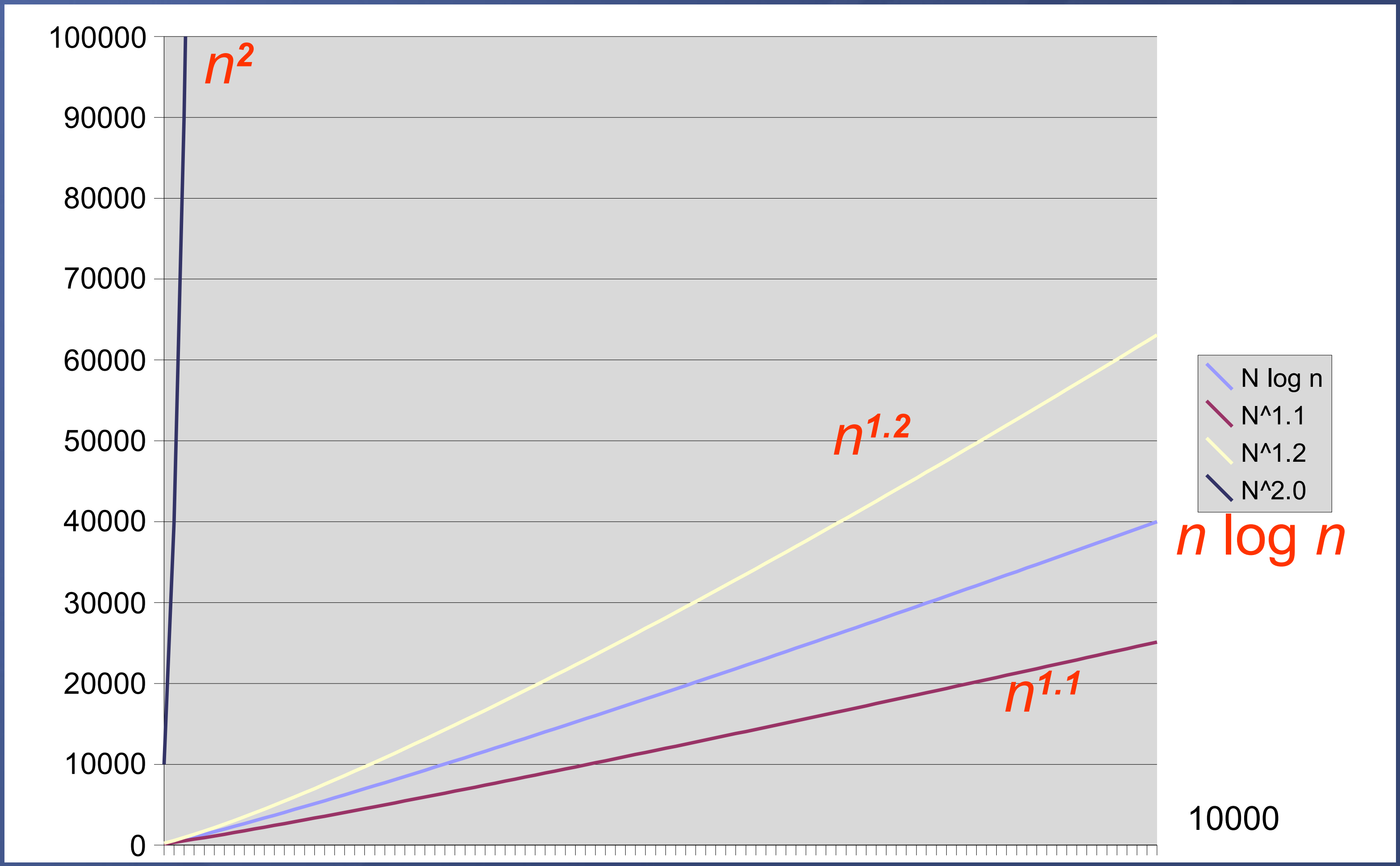
Scalability, n=1000

(Current real-time fragment shaders)



Scalability, n=10000

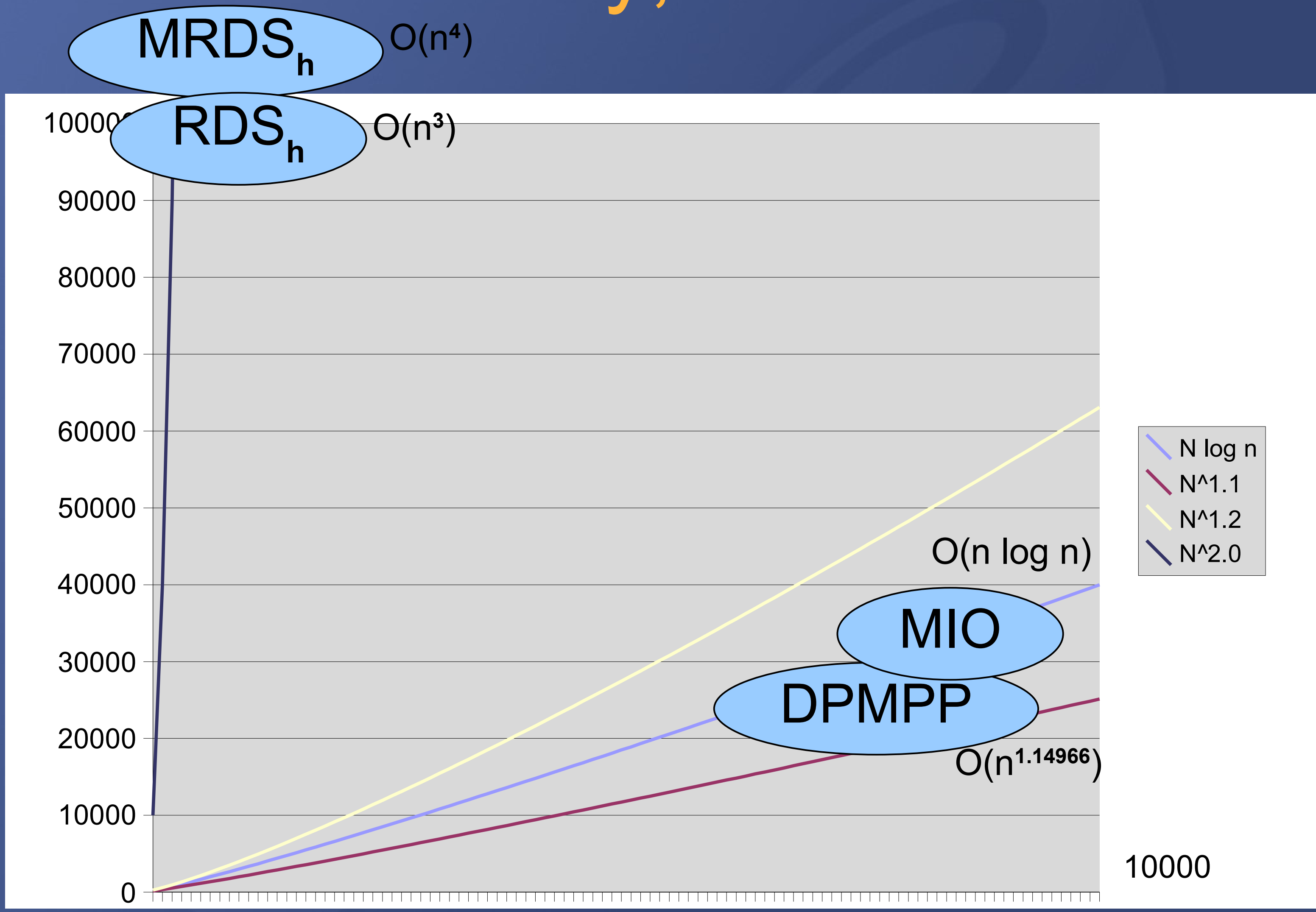
(Current GPGPU fragment shaders)



Three Proposed Solutions

- Minimum cut sets (RDS_h, $MRDS_h$) [Chan 2002, Foley 2004]
  - Graph (DAG) cut sets.
  - Minimize number of cuts.
  - Greedy algorithms.
  - $O(n^3)$, $O(n^4)$, nonscalable.
- List scheduling (MIO) [Riffel 2004]
  - Job scheduling.
  - Minimize instruction count (linear function).
  - Greedy algorithm.
  - $O(n \log n)$, scalable.
- Dynamic programming (DPMPP) [this paper]
  - Job scheduling.
  - Minimize predicted run time (nonlinear function).
  - Globally optimal algorithm.
  - $O(n^{1.14966})$ empirically, (semi-) scalable.

Scalability, n=10000



The Insight: Job Shop Scheduling

An NP-hard multi-stage decision problem.

- A set of time slots and functional units.
- A set of tasks.
- An objective function (cost).

Goal: assign tasks to slots/units to minimize cost.

Examples:

- Compiler instruction selection.
- Airline crew scheduling.
- Factory production planning.
- etcetera

Solving project scheduling problems by minimum cut computations. R. Mohring, A. Schulz, F. Stork, M. Uetz. Management Science (2002), pp. 330-350.

Job Shop Scheduling for MPP

Defined by DAG and GPU architecture.

- A set of n DAG operations (+ “new pass” operation).
- A schedule with n time slots.
- A single GPU processor.
- Cost function predicts quality of compiled code.
  - Predicted execution time (DPMPP).
  - Instruction count (MIO).
  - Number of passes (RDS_h, $MRDS_h$).

Many possible formulations and solution algorithms.

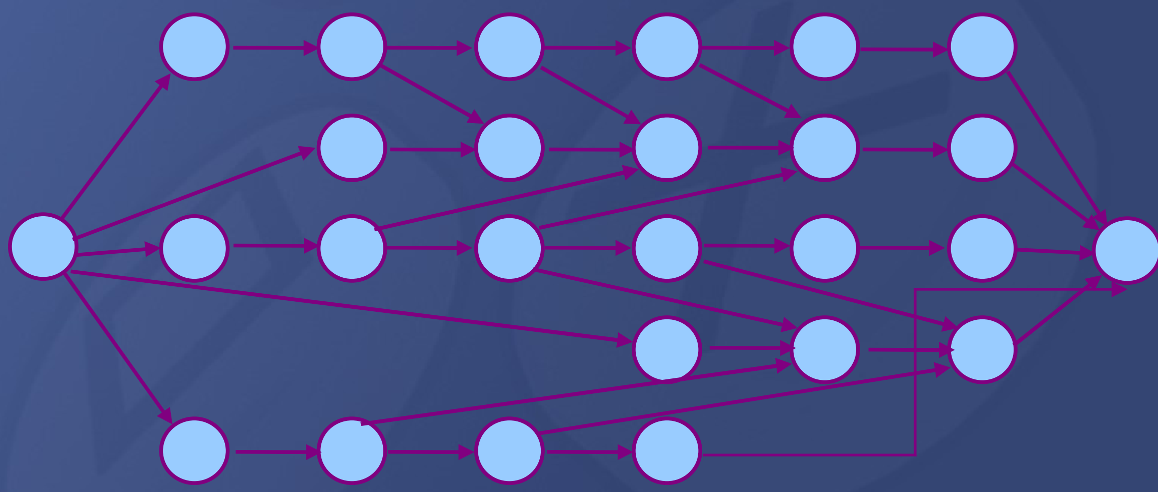
- Integer programming, linear programming, dynamic programming, list scheduling, graph cutting, branch and bound, Satisfiability, ...
- Problem size can often be $O(n^2)$ [nonscalable]

Integer Programming Formulation

- Jobs (tasks) *j, times t; unknowns x.*
- 0-1 decision variables $x_{j,t}$=1 iff job *j* is scheduled at time *t.*
- Costs $w_{j,t}$ = time-dependent cost of job *j* at time *t.*
- Resource requirements $r_{j,k}$ for job *j* of resource *k.*
- Constraints:
  - Precedence: $\Sigma_t\, t(x_{j,t} - x_{i,t}) \quad >= \quad d_{i,j}$
  - Resource: $\Sigma_t\, r_{j,k}(\Sigma^t_{s=t-pj+1}\, x_{j,s}) <= R_k$
  - Uniqueness: $\Sigma_t\, x_{j,t} \quad = \quad 1$
- Objective:
  - Minimize $\Sigma_{j,t}\, w_{j,t} x_{j,t}$ subject to constraints (linear objective).
- Various solvers (simplex, Karmarkar's algorithm, ...).
  - Potentially exponential worst-case time.
  - Easy transformation to SAT (Boolean decision variables).
    - Different solvers (CHAFF, branch and bound, Tabu, ...)
- $\| X \|$ is $O(n^2)$.

Graph Cut Formulation

- See [Mohring 2002] for details.
- Vertices $v_{j,t}$ represents job *j* scheduled at time *t.*
- $v_{j,first(j)}$ ... $v_{j,last(j)}$ *marks all possible times for job j.*
- *Temporal arcs* ($v_{i,t}$, $v_{j,t+d_{ij}}$) *for time lags* $d_{i,j}$ *have infinite capacity.*
- *Assignment arcs* ($v_{j,first(j)}$, $v_{j,first(j)+1}$) *have capacity* $w_{j,t}$.

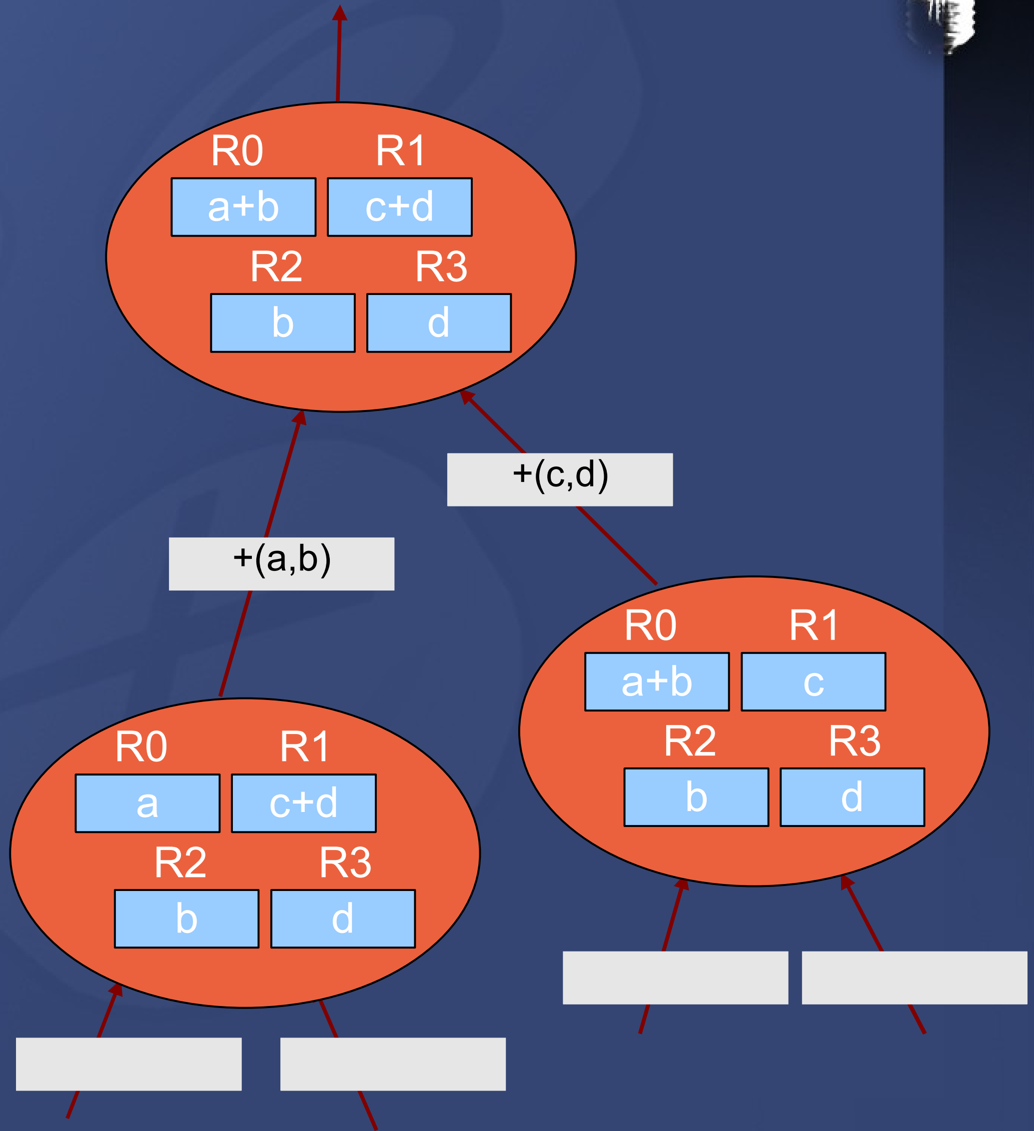


- A minimum cut in this graph defines a minimum cost schedule.
- O(*m* log *m*) time for *m* vertices [but *m is* O(n^2)].

Dynamic Programming Formulation

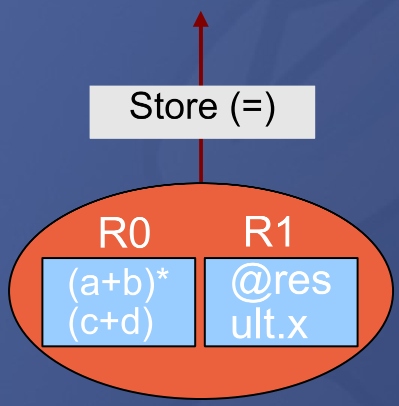
Search tree root is terminal end state at time t=n.

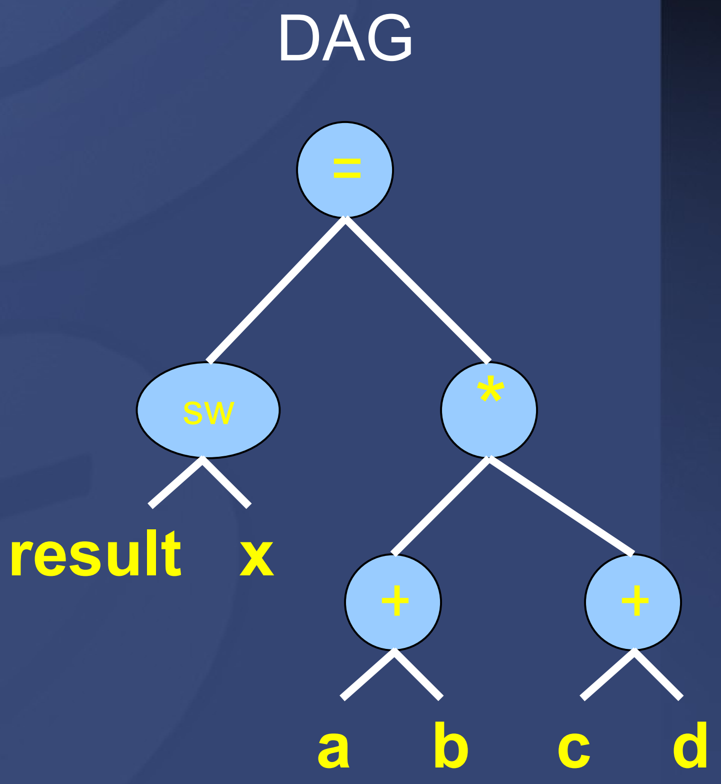
- Vertices are snapshots of machine state.
- Edges are transitions (DAG operation, or "new pass").
  - Generate tree breadth-first.
  - Leaves represent initial states (time t=1).
  - Every path from leaf to root is a valid schedule.
- MPP solution is the lowest-cost path.
  - Time and space are O(n^b) where *b* is the average branching factor.
- Prune maximally.
- (*b* < 1.2) ⇨ (semi-)scalable.



Dynamic Programming Example

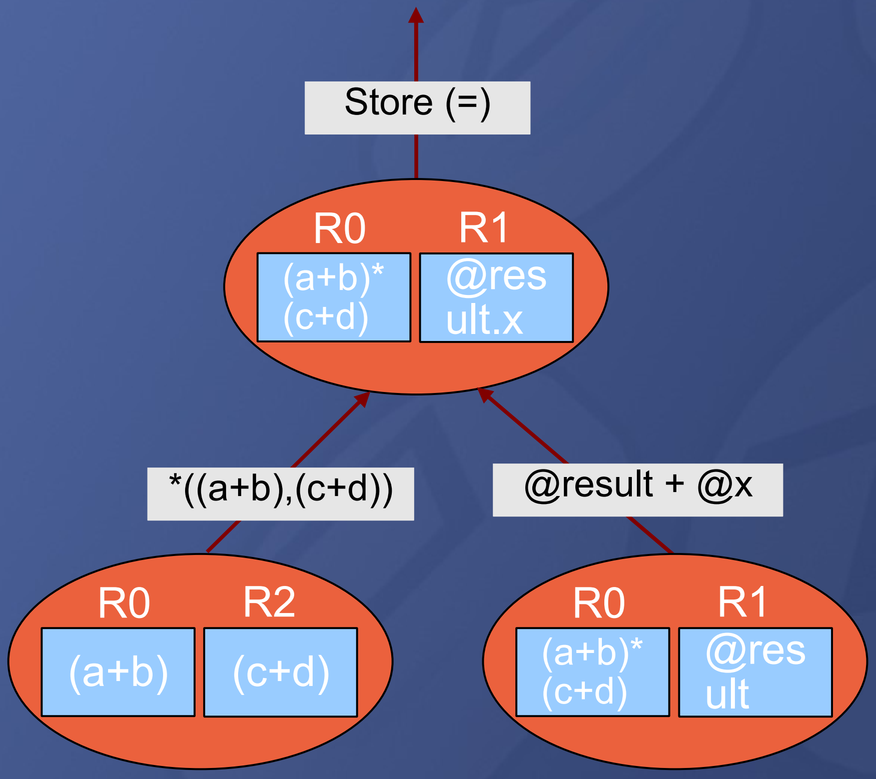
- Root is terminal end state (time *t=n*).

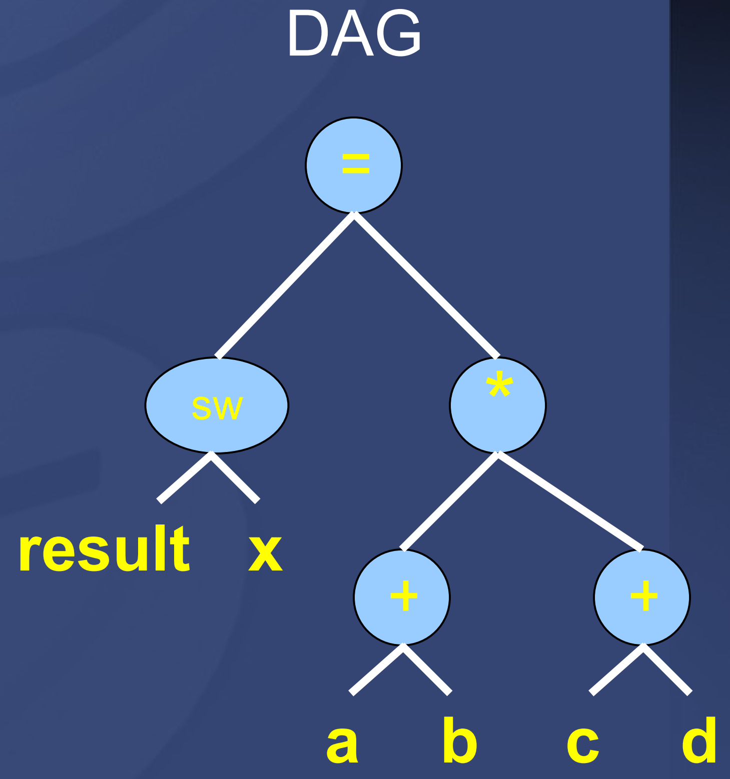




Dynamic Programming Example

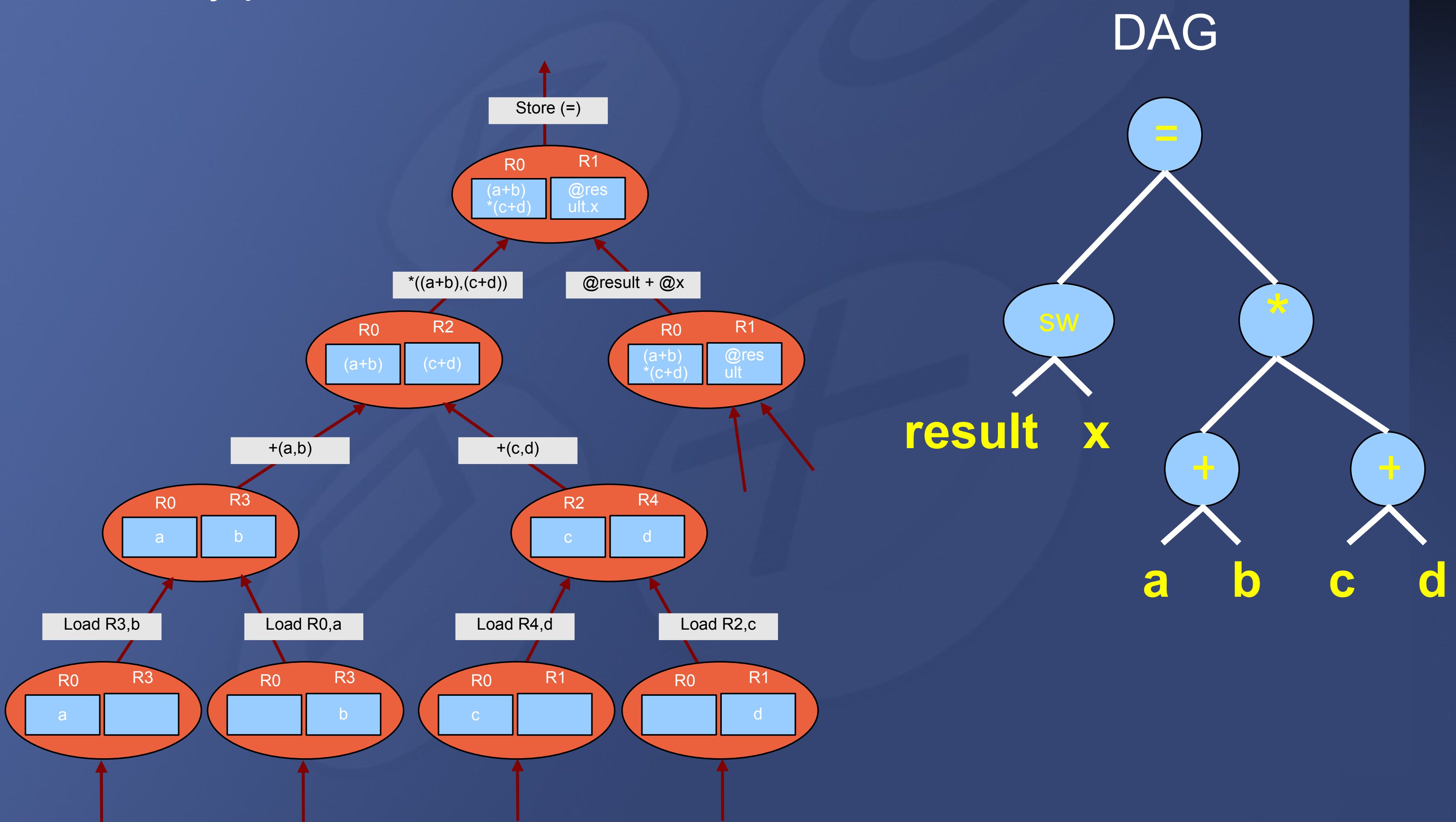
- Generate tree breadth-first.
  Accumulate cost along paths.





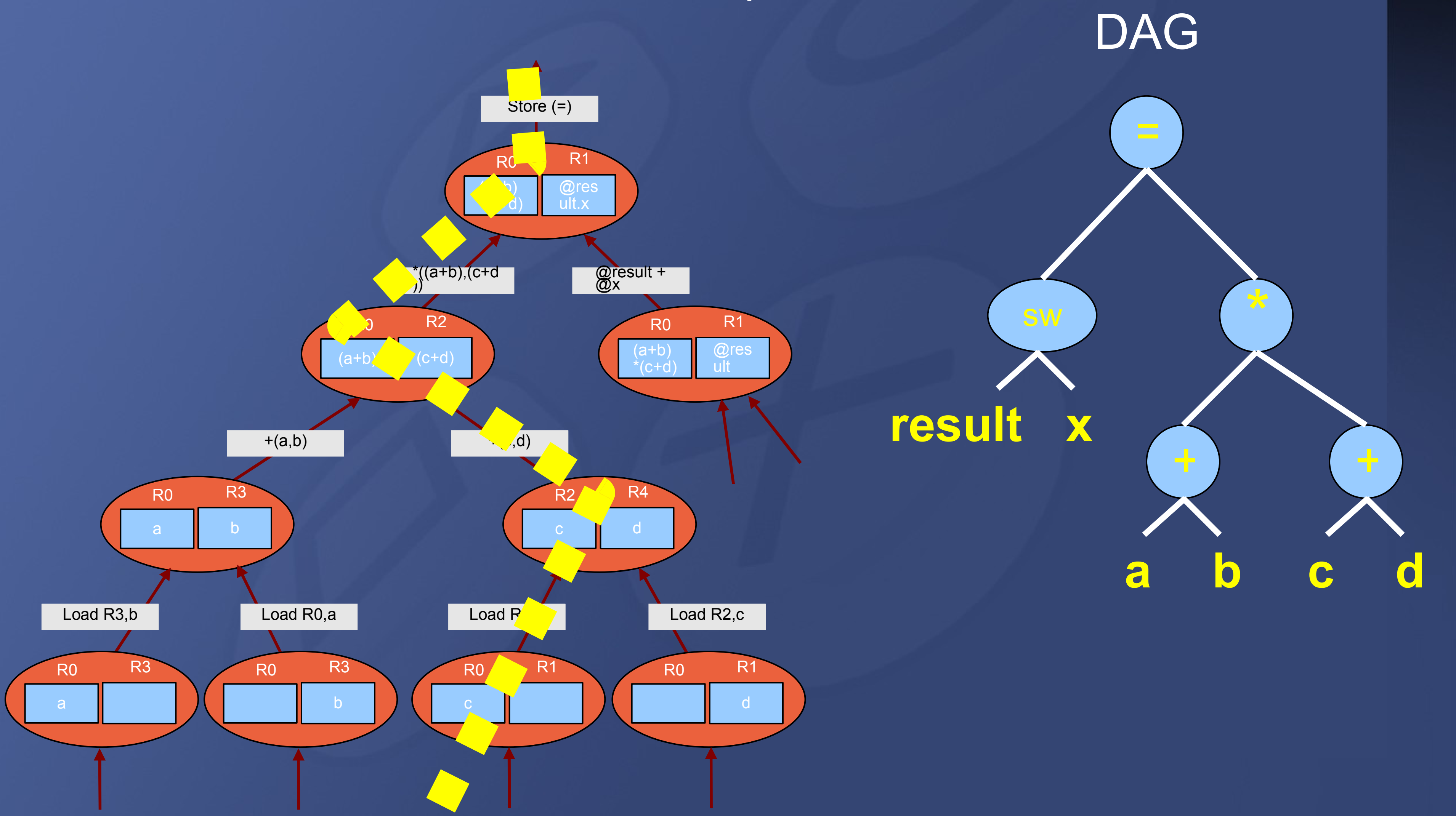
Dynamic Programming Example

- Every path from leaf to root is a valid schedule.



Dynamic Programming Example

- MPP solution is the lowest-cost path.



Key Elements of DP Solution

Solve problem in reverse.
- Start from optimal end state.
- Requires Markov property.

- Prune maximally.
  - Manage complexity.
  - “optimal substructure”.

Retain all useful intermediate states.

Consider all valid paths to find solution.

Markov property

(Stale operands)

R0	R1	
a		Load R0=a
a	b	Load R1=b
a+b	b	R0=+(R0,R1)
a+b	c	Load R1=c
a+b	c	Store R0
d	c	Load R0=d
c+d	c	R0=+(R0,R1)
c+d	a+b	*New Pass*
(a+b)*(c+d)	a+b	R0=*(R0,R1)
(a+b)*(c+d)	a+b	Store R0

R0	R1	
d		Load R0=d
d	c	Load R1=c
c+d	c	R0=+(R0,R1)
c+d	b	Load R1=b
c+d	b	Store R0
a	b	Load R0=a
a+b	b	R0=+(R0,R1)
a+b	c+d	*New Pass*
(a+b)*(c+d)	c+d	R0=*(R0,R1)
(a+b)*(c+d)	c+d	Store R0

Markov property holds for ...

- GP registers
- Rasterized interpolants
- Pending texture requests
- Instruction storage
- etcetera

Nonlinearity and Optimality

Algorithm	Objective	Linearity
RDS_h, $MRDS_h$	*passes*	*linear*
MIO	*instructions*	*linear*
DPMPP	*execution time*	*nonlinear*

GPU cost function can be nonlinear.

- Depends on current machine state.
  - *E.g. pipelined activity due to previous operation.*
- COST($instr_A$)+COST($instr_B$) <> COST($instr_A$,$instr_B$)
  - *Linear objective functions are approximations to reality (e.g. instruction count).*

Nonlinear functions can have many minima.

- Functions for real GPUs are ugly.
- Greedy algorithms become trapped in local minima.
  - Dynamic Programming computes global minima.
  - Dynamic Programming solutions are globally optimal.

Optimal substructure

DP algorithms must avoid search tree explosion.

- Complexity $O(n^b)$, average branching factor b.
- Need to prune search space.
- Strong preference for local branching factor 1 (scalability).

Optimal substructure:

- Compatible with global solution (conservative evaluation).
- Can evaluate locally.

Objective:

- Minimize predicted execution time.
- Approximate by minimizing number of register loads.
  - Locally computable, globally conservative (includes solution).

Implementation:

- Schedule shortest DAG subtree (DPMPP).
  - Schedule is generated in reverse order.
  - Result is ordered largest to smallest.

Algorithm DPMPP

Stage is initially equal to n, and recurses down to 1.

T is initially the set of final transitions to the end state. Subsequently, T is P from the previous stage.

P is the current set of transitions being explored (search tree cross-section)

S_t is the set of locally optimal transitions that could be scheduled before this *t*.

Cost $DPMPP(stage, T)$ {
  $P := \{\}$
  $(\forall t \in T)$ do
    $S_t := \left\{ \begin{array}{l} \text{valid prior transitions of } t \\ \text{with optimal substructure} \end{array} \right\}$
    $min := \infty$
    $(\forall s \in S_t)$ do
      $s.cost := t.cost +$
        $COST(s.operation, s.postcondition)$
      if $(s.cost < min)$
        $min := s.cost$
    enddo s
    $(\forall s \in S_t$ such that $s.cost = min)$ do
      $P := (P \cup s)$ uniquely by precondition
    enddo s
  enddo t
  if $(stage > 1)$
    return $DPMPP(stage - 1, P)$
  else
    return $\min_{p \in P}(p.cost)$ }

Figure 1

Choose shortest subtrees or reverse Sethi-Ullman numbering.

Cost is computed with respect to s.precondition.

Only keep minimum cost path(s) for this *t*.

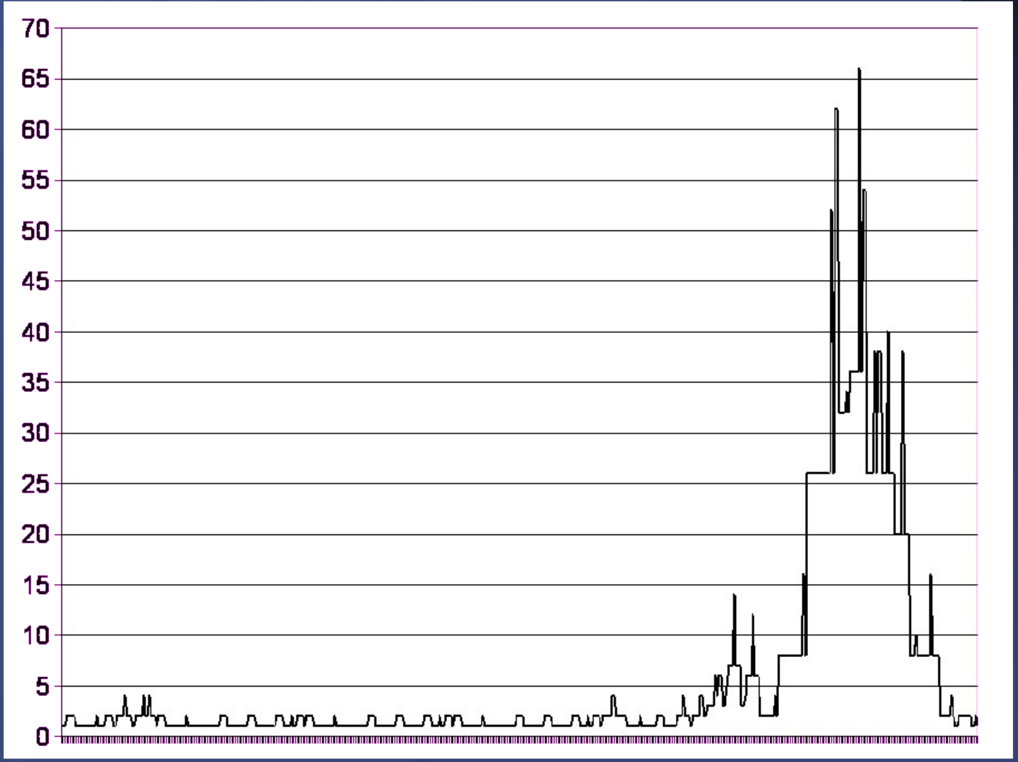
Also discard redundant *s.precondition.*

Continue the breadth-first search of the tree.

Terminate when Stage = 1. The global solution is a path from the cheapest *p* in *P* to the search tree root.

Scalability

- Search width w_i.
- Branching factor $b_i = w_{i+1}/w_i$.
- Area under the curve is equal to n^b where b is the average b_i.
- Observation: b decreased with increasing n.
  - Is this a pattern?
  - Requires that area grow less than unit for each unit increase in n.
- Implication: asymptotic scalability.
  - Don't know.



Search tree width over n=490 stages.
(Real-time fragment shader, b=1.06091).
Figure 2

Optimal Substructure Revisited

Sethi-Ullman numbering.

- Orders DAG nodes by number of subtree register usage.
- Used in algorithm MIO to prioritize operations.
  - Highest numbered nodes first.
  - Schedule generated in order.
- Should be explored for dynamic programming.

Subtree size.

- Monotonic in subtree register usage.
- Used in algorithm DPMPP to prioritize operations.
  - Smallest numbered nodes first.
  - Schedule generated in reverse order.
- Probably less accurate than Sethi-Ullman.
  - Implication: less efficient compilation.

DPMPP and MIO

	MIO	DPMPP
	List Scheduling	DP
Direction:	*forward*	*reverse*
Paradigm:	*greedy*	*optimization*
Optimality:	*local*	*global*
Complexity:	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{1.14966})$
Numbering:	*Sethi – Ullman*	*subtree size*
Critical:	*scheduling priority*	*search pruning*
Policy:	*largest trees first*	*smallest trees last*
Affects:	*run time*	*compile&runtime*

Figure 4

Conclusions

Claims:

- Nonlinear cost functions are required for accuracy.
- Algorithm DPMPP:
  - Is GPU-generic.
  - Supports nonlinear cost functions.
  - Finds globally optimal solutions.
  - Is scalable above $n=10^5$.
  - May be asymptotically scalable.

Remarks:

- DPMPP should use Sethi-Ullman numbering.
- Shader multipassing provides diverse benefits.
  - Some benefits require accurate (i.e. detailed) cost functions.
- Primary challenge is inter-pass data transfer.
  - Challenge: zero (effective) latency transfer mechanisms.
    - e.g. F-Buffer with zero latency.
    - Simpler solutions are possible.