Scheduling applications on GPU and CPU

"Denis Barthou, Julien Jeager Emmanuel Jeannot and Minhaj Khan

LaBRI/INRIA Bordeaux Sud-Ouest/ENSEIRB

Runtime Team

Emmanuel.Jeannot@inria.fr

Introduction

- GPU : widespread component of many computers
- Can accelerate performance
- Appealing device for HPC

Disclaimer

Very, very preliminary work (progress was slower than expected)

Solution for only a part of the problem (suggestion welcome)

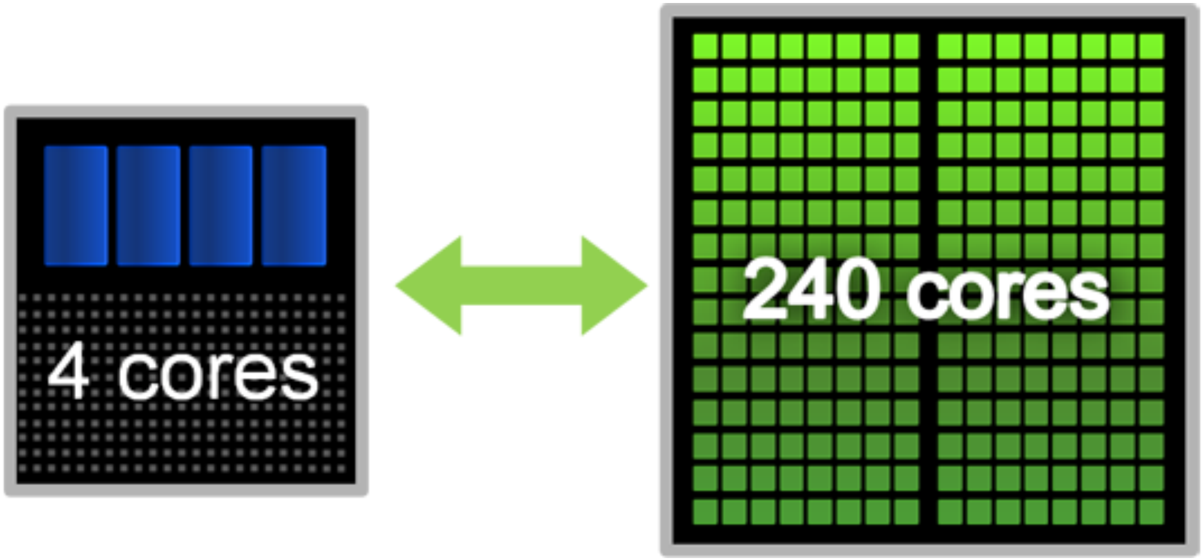
No experimental results

GPU

Many cores (448 CUDA Cores for the Tesla)

Simple programming: vector computation

Simple (no) memory management



GPU Vs CPU

Peak performance : GPU better

Disk, network, memory I/O: must be performed by CPU

CUDA model: CPU controls GPU (no memory management)

Depending on the granularity: performance ratio changes (CPU can be better than GPU for small size data)

Ratio of performance depend on the computation

Unrelated model

CPU+GPU environments

StarPU (http://runtime.bordeaux.inria.fr/StarPU/): unified framework for executing application on CPU, GPU, SPU, etc.

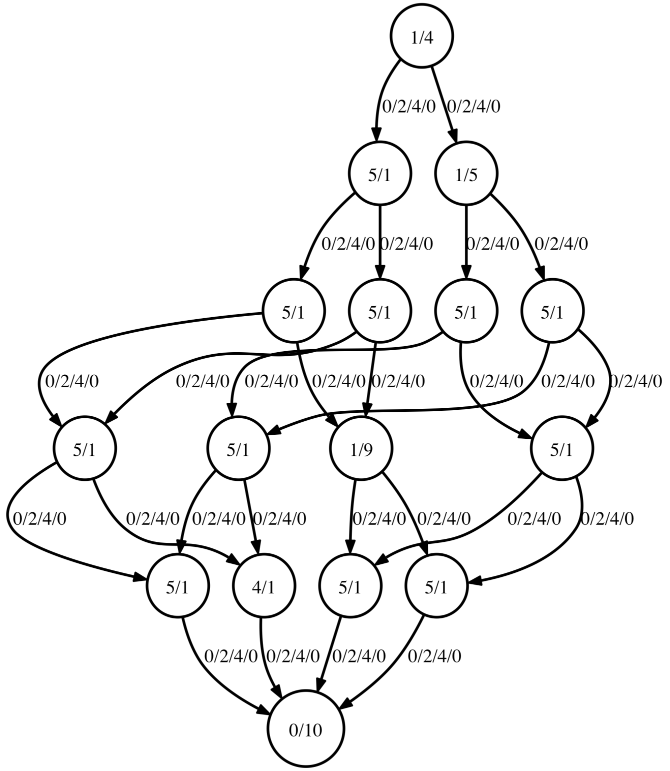
Streamit (http://groups.csail.mit.edu/cag/streamit/): language for streaming application

OpenCL: A language for parallel programming of heterogeneous environments : can derive a DAG from a program

Plasma/Magma (ICL/UTK) : MultiCore/GPU environemnts

Model

- Unrelated model
- Bandwidth different from CPU to GPU and GPU to CPU
- Computation time of kernels (task) : very stable
- A task graph:
  - Edges 4 values (CPU to CPU, CPU to GPU, GPU to CPU and GPU to GPU)
  - Vertex 2 values (CPU or GPU)



Problem

Given : m CPUs and n GPUs:

- Allocate tasks to a resource
- Respect constraints
- Minimize makespan (finish time of last task)

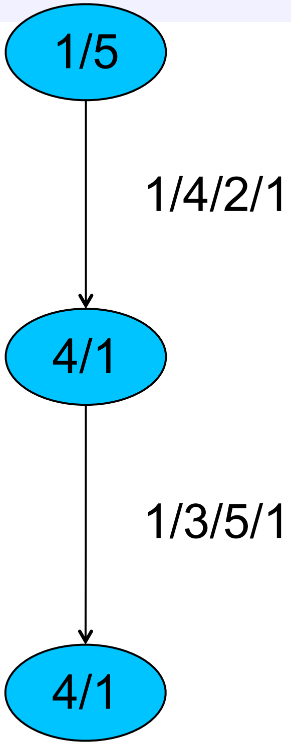
Clustering the graph

- Reactivating the old idea [Sarkar 89]:
  - Clustering the graph for an unbounded number of resources
  - Mapping clusters to GPUs or CPUs to minimize makespan
- Intuition: providing a good clustering should help to built a good schedule

The *spaghetti* algorithm

Graph Contraction

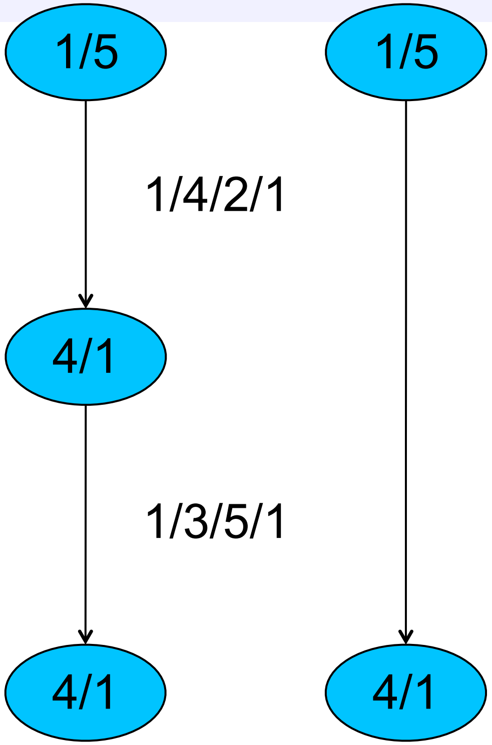
Notation: C→C/C→G/G→C/G→G



The *spaghetti* algorithm

Graph Contraction

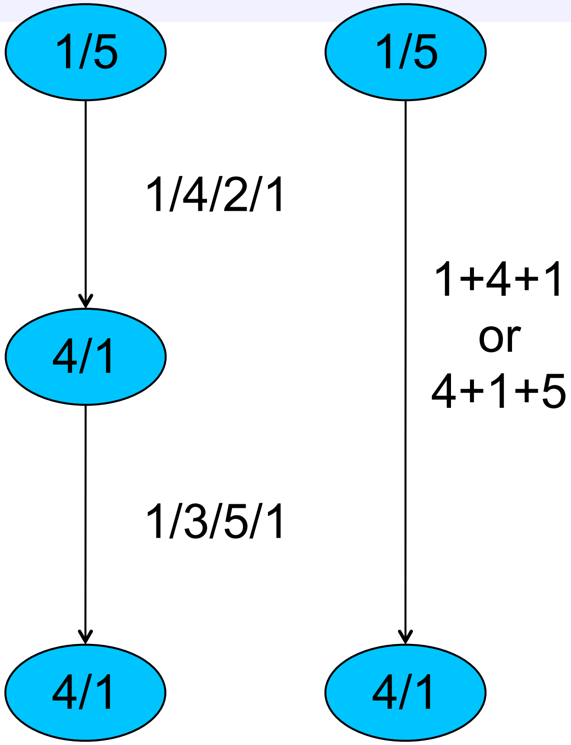
Notation: C→C/C→G/G→C/G→G



The *spaghetti* algorithm

Graph Contraction

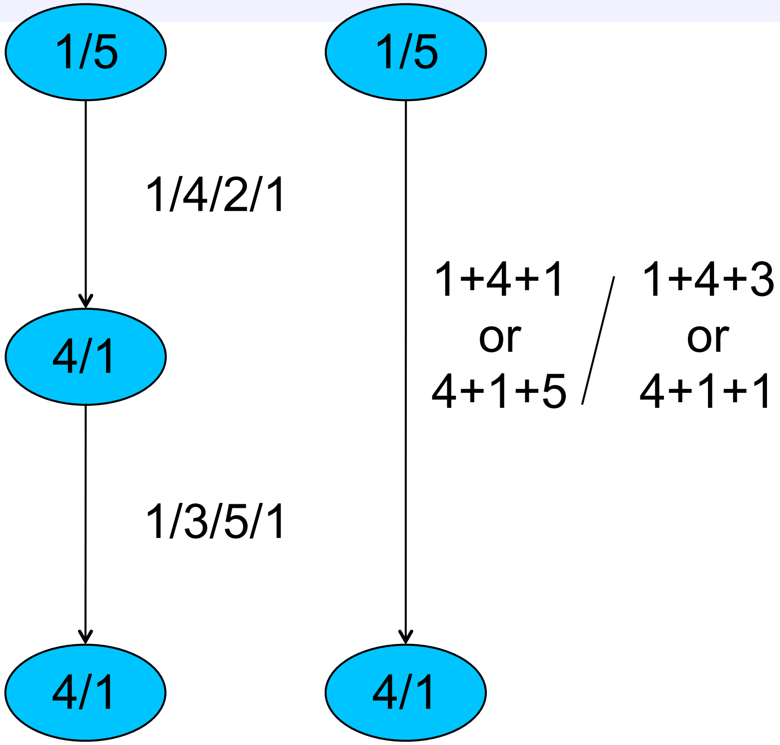
Notation: C→C/C→G/G→C/G→G



The *spaghetti* algorithm

Graph Contraction

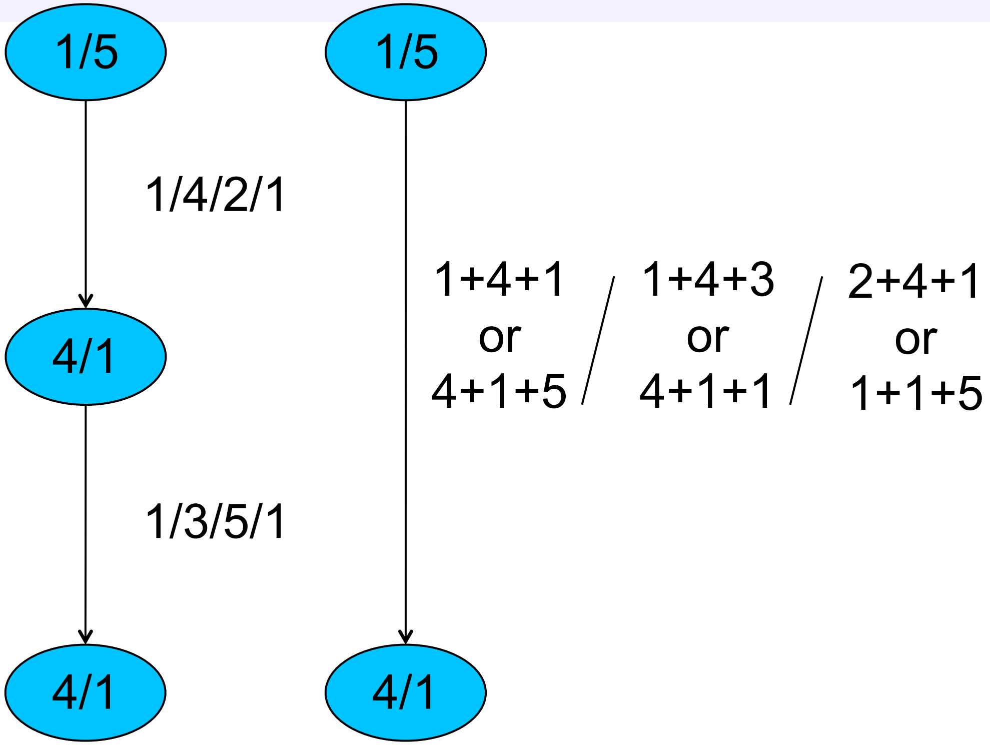
Notation: C→C/C→G/G→C/G→G



The *spaghetti* algorithm

Graph Contraction

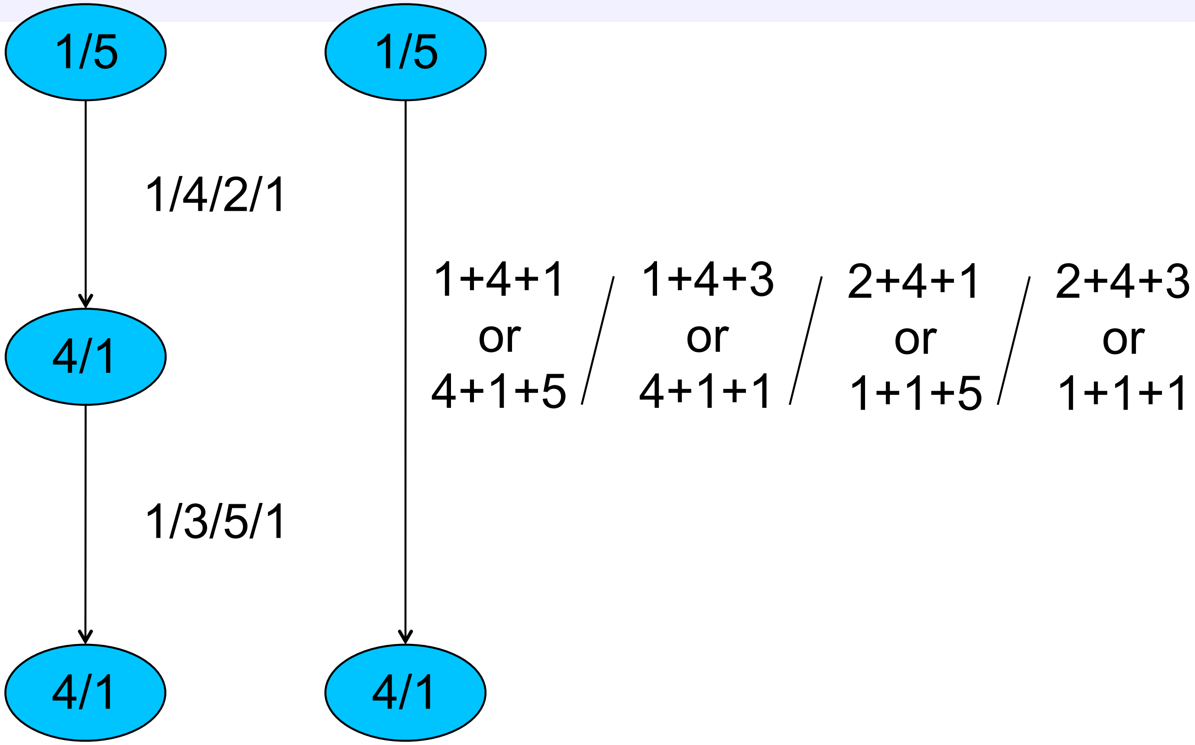
Notation: C→C/C→G/G→C/G→G



The *spaghetti* algorithm

Graph Contraction

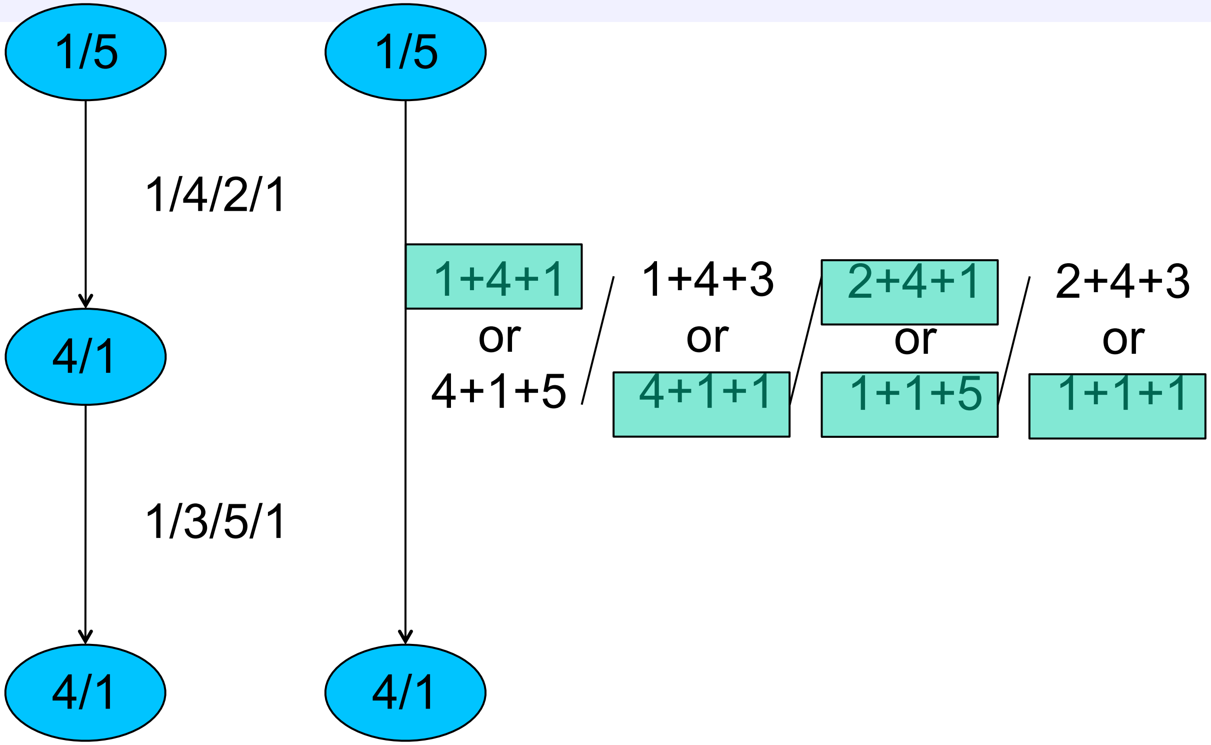
Notation: C→C/C→G/G→C/G→G



The *spaghetti* algorithm

Graph Contraction

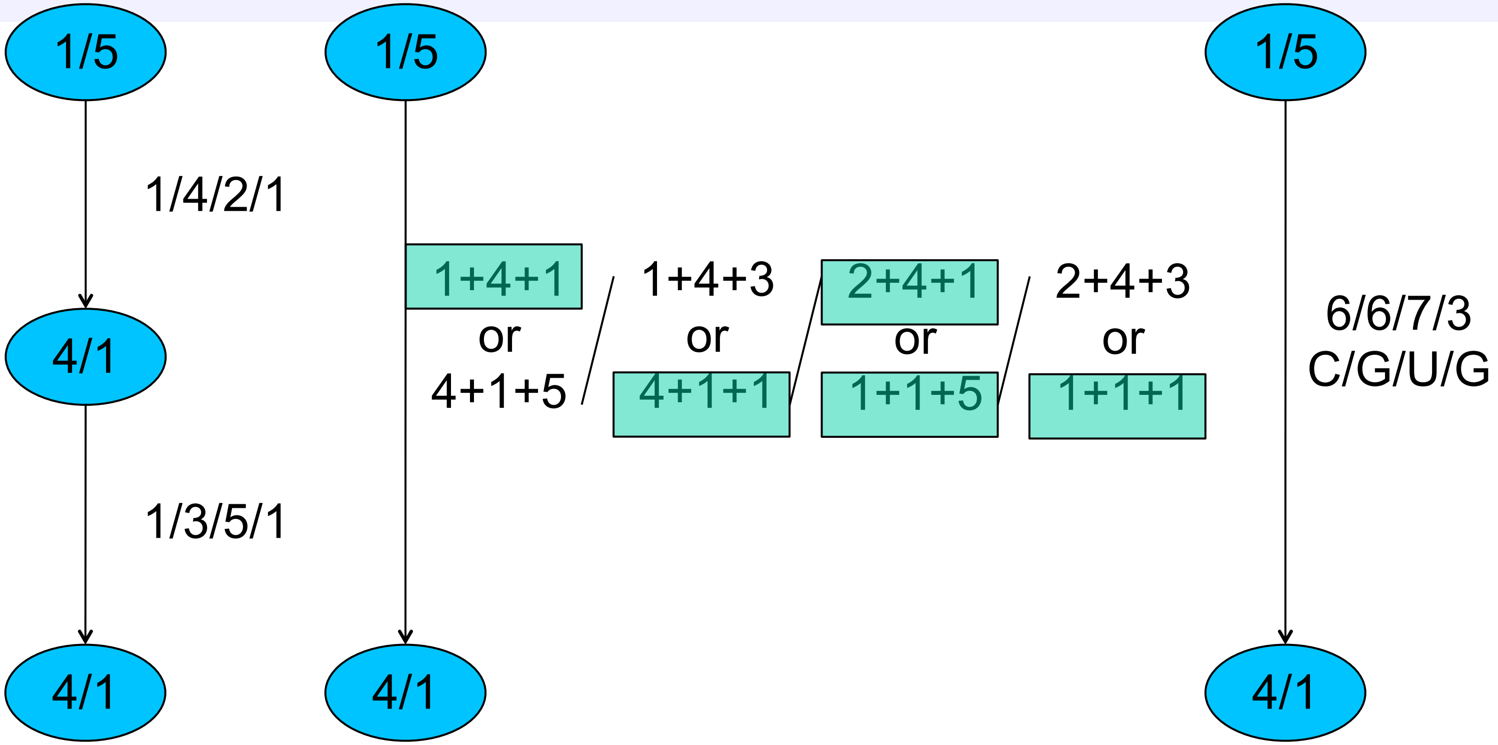
Notation: C→C/C→G/G→C/G→G



The *spaghetti* algorithm

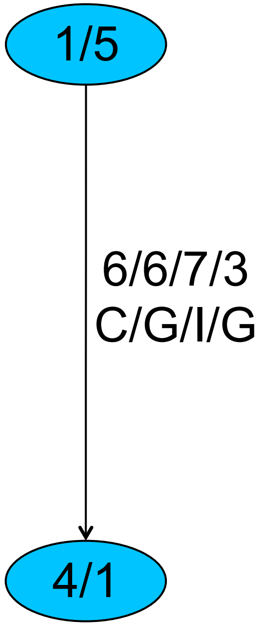
Graph Contraction

Notation: C→C/C→G/G→C/G→G



The *spaghetti* algorithm

- We contract the whole graph, until we have two nodes.
- We keep track of each intermediate possible mapping
- We fix the mapping of the star and end-node
- We derive the mapping of each intermediate node



Best mapping:

- C→C: 11
- C→G : 8
- G→C : 13
- G→G : 9

We map the intermediate node on a GPU

Simple implementation

A=

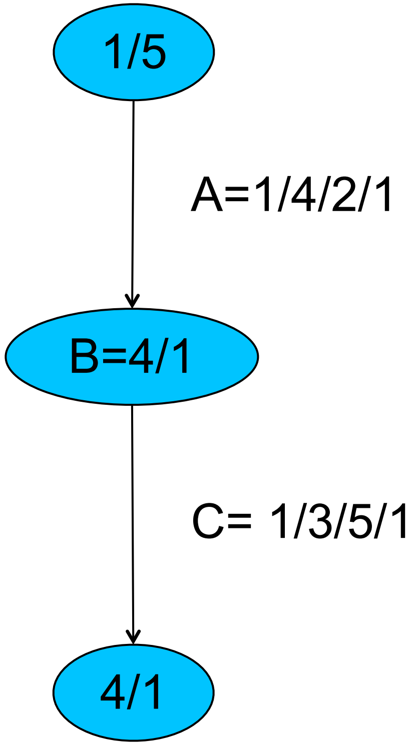
1	2
4	1

B=

4	+∞
+∞	1

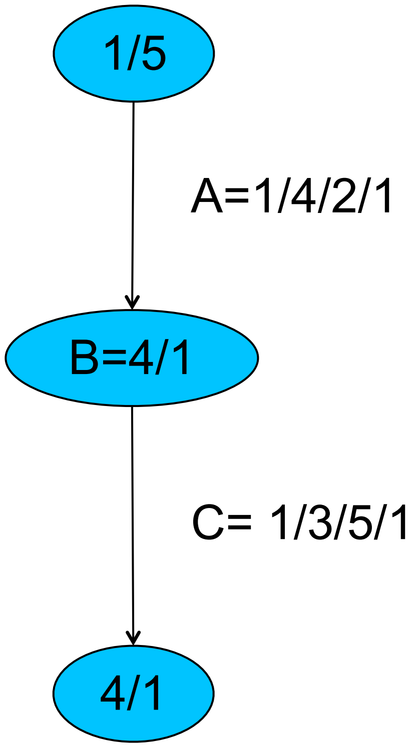
C=

1	3
5	1



Simple implementation

$$A = \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 4 & 1 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|} \hline 4 & +\infty \\ \hline +\infty & 1 \\ \hline \end{array} \qquad C = \begin{array}{|c|c|} \hline 1 & 3 \\ \hline 5 & 1 \\ \hline \end{array}$$



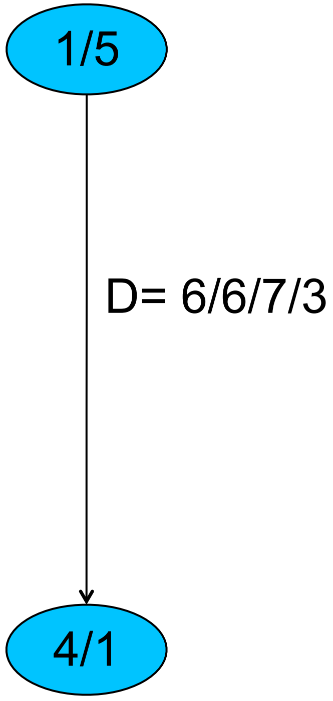
1/5

A=1/4/2/1

B=4/1

C= 1/3/5/1

4/1



1/5

D= 6/6/7/3

4/1

Simple implementation

A=

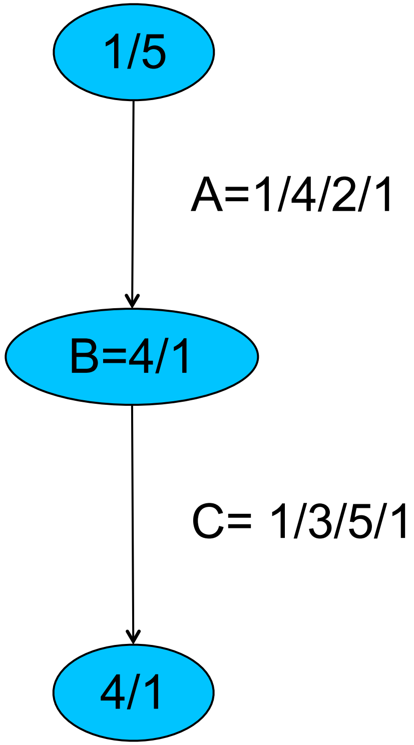
1	2
4	1

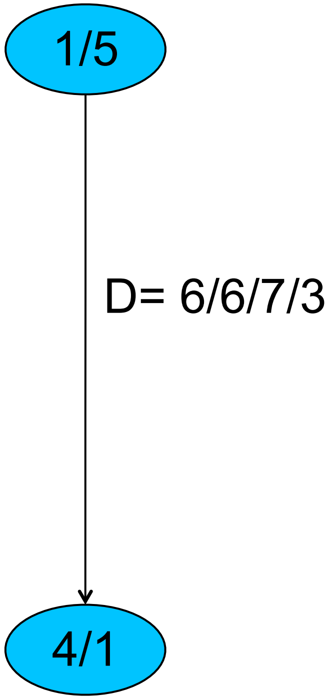
B=

4	+∞
+∞	1

C=

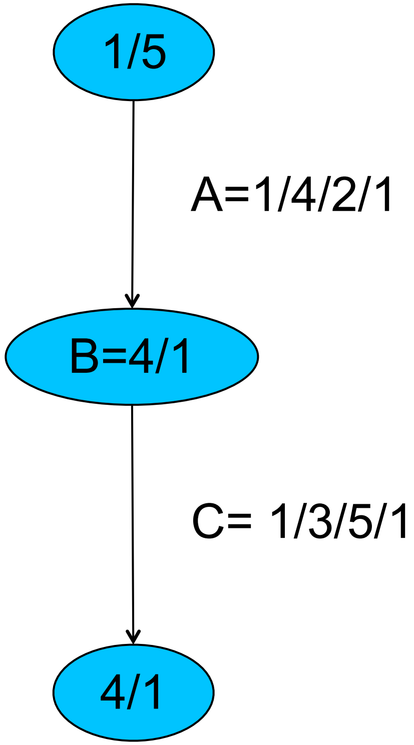
1	3
5	1





D=A*B*C
In the Min/+ algebra

Matrix multiplication in the min/+ algebra



1/5

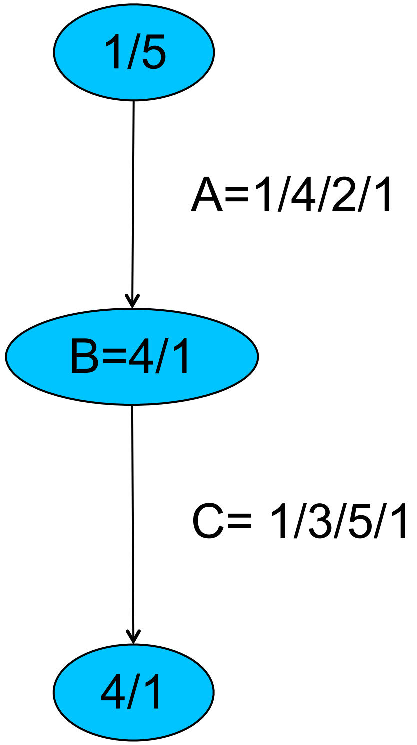
A=1/4/2/1

B=4/1

C= 1/3/5/1

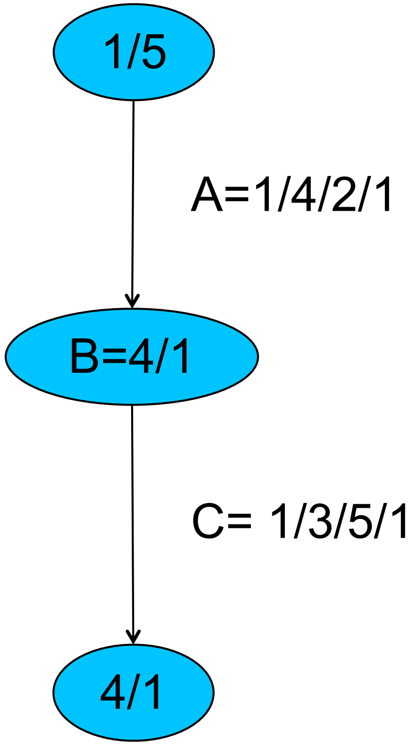
4/1

Matrix multiplication in the min/+ algebra



1	3
5	1

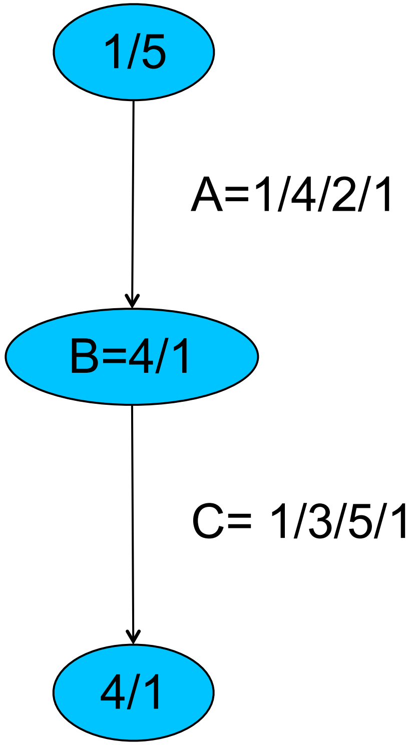
Matrix multiplication in the min/+ algebra



1	3
5	1

4	+∞
+∞	1

Matrix multiplication in the min/+ algebra

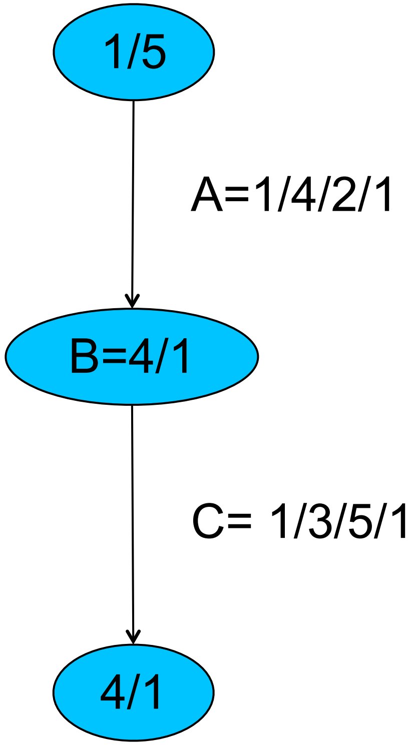


1	3
5	1

4	+∞
+∞	1

4+1	4+3
5+1	1+1

Matrix multiplication in the min/+ algebra



1/5

A=1/4/2/1

B=4/1

C= 1/3/5/1

4/1

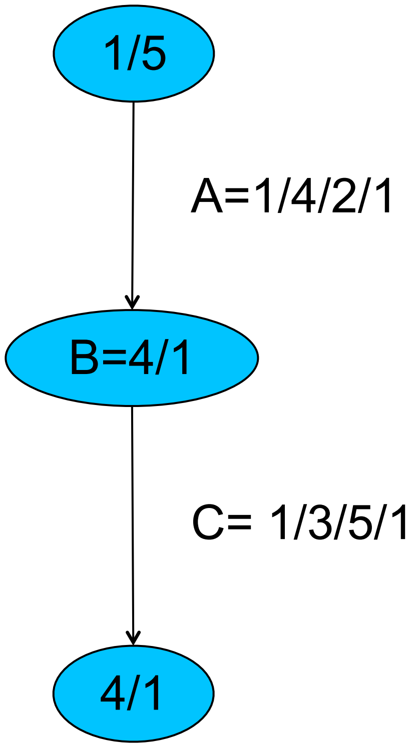
1	3
5	1

4	+∞
+∞	1

4+1	4+3
5+1	1+1

1	4
2	1

Matrix multiplication in the min/+ algebra



1	3
5	1

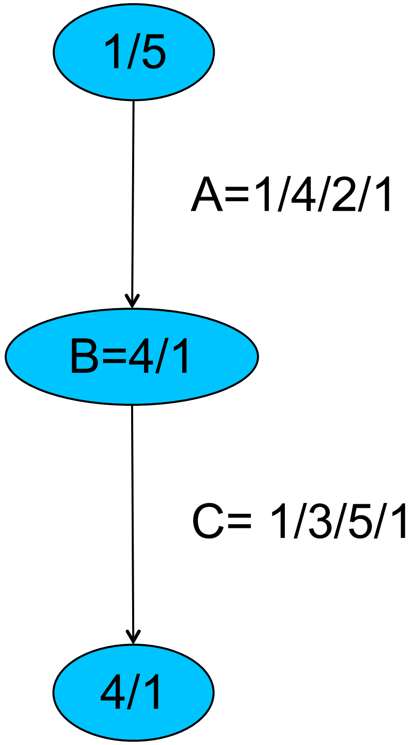
4	+∞
+∞	1

4+1	4+3
5+1	1+1

1	4
2	1

1+4+1 4+5+1	1+4+3 4+1+1
2+4+1 1+5+1	2+4+3 1+1+1

Matrix multiplication in the min/+ algebra



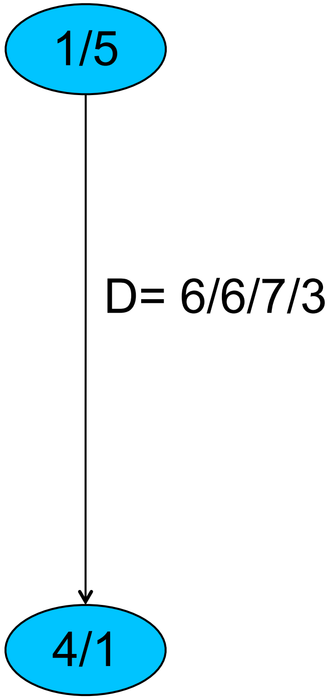
1	3
5	1

4	+∞
+∞	1

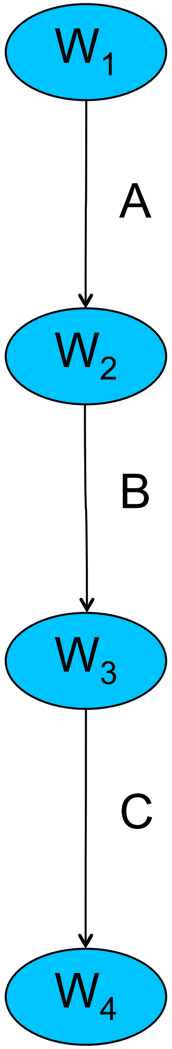
4+1	4+3
5+1	1+1

1	4
2	1

1+4+1 4+5+1	1+4+3 4+1+1
2+4+1 1+5+1	2+4+3 1+1+1



Computation on the whole graph



A,B,…, W_i: 2 by 2 matrices

M
Adjacency matrix

-∞	A	-∞	-∞
-∞	-∞	B	-∞
-∞	-∞	-∞	C
-∞	-∞	-∞	-∞

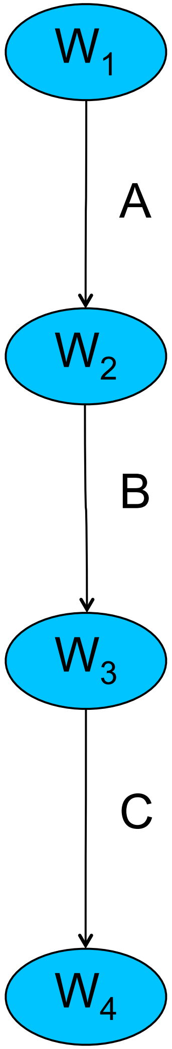
W
Weight matrix

W_1	-∞	-∞	-∞
-∞	W_2	-∞	-∞
-∞	-∞	W_3	-∞
-∞	-∞	-∞	W_4

M^2=MWM (in the max/* algebra)

-∞	-∞	AW_2B	-∞
-∞	-∞	-∞	BW_3C
-∞	-∞	-∞	-∞
-∞	-∞	-∞	-∞

Computation on the whole graph



A,B,…, W_i: 2 by 2 matrices

M
Adjacency matrix

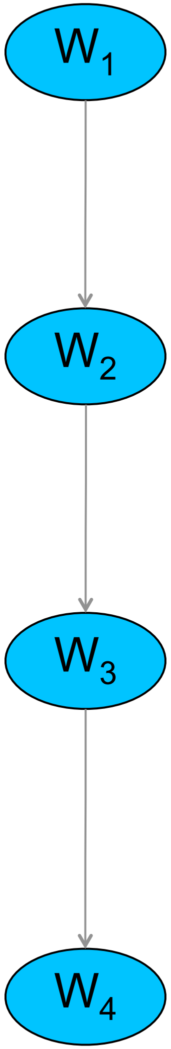
-∞	A	-∞	-∞
-∞	-∞	B	-∞
-∞	-∞	-∞	C
-∞	-∞	-∞	-∞

W
Weight matrix

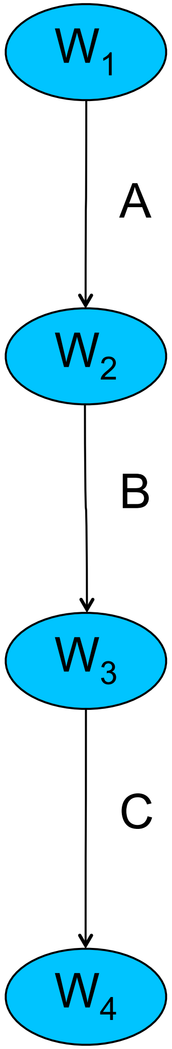
W_1	-∞	-∞	-∞
-∞	W_2	-∞	-∞
-∞	-∞	W_3	-∞
-∞	-∞	-∞	W_4

M^2=MWM (in the max/* algebra)

-∞	-∞	AW_2B	-∞
-∞	-∞	-∞	BW_3C
-∞	-∞	-∞	-∞
-∞	-∞	-∞	-∞



Computation on the whole graph



$A, B, \ldots, W_i$: 2 by 2 matrices

M
Adjacency matrix

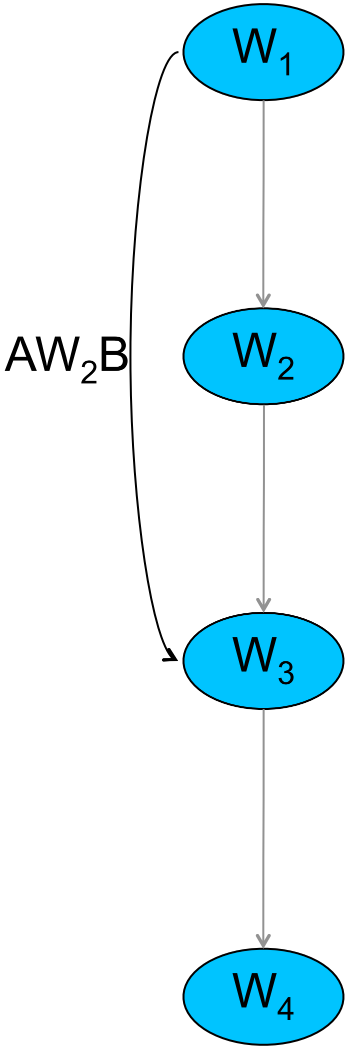
-∞	A	-∞	-∞
-∞	-∞	B	-∞
-∞	-∞	-∞	C
-∞	-∞	-∞	-∞

W
Weight matrix

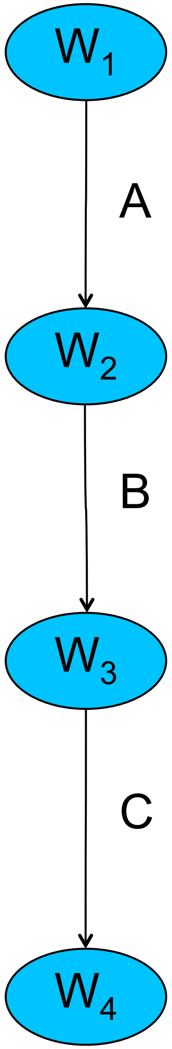
W_1	-∞	-∞	-∞
-∞	W_2	-∞	-∞
-∞	-∞	W_3	-∞
-∞	-∞	-∞	W_4

$M^2 = MWM$ (in the max/* algebra)

-∞	-∞	AW_2B	-∞
-∞	-∞	-∞	BW_3C
-∞	-∞	-∞	-∞
-∞	-∞	-∞	-∞



Computation on the whole graph



A,B,…, W_i: 2 by 2 matrices

M
Adjacency matrix

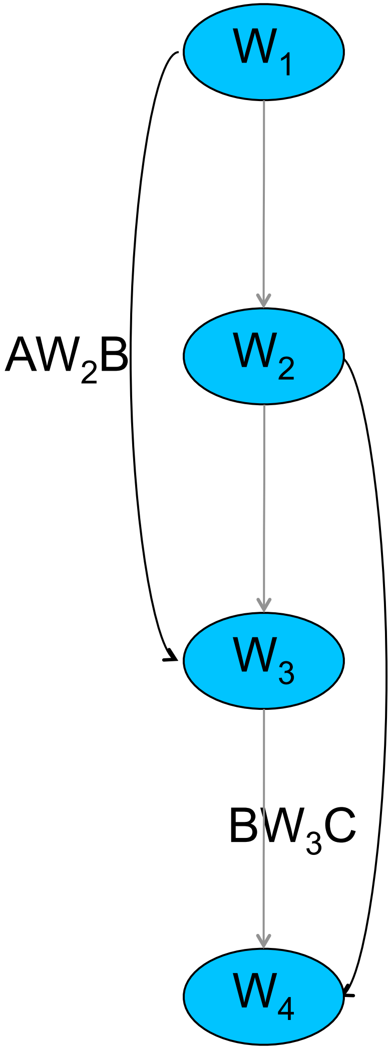
-∞	A	-∞	-∞
-∞	-∞	B	-∞
-∞	-∞	-∞	C
-∞	-∞	-∞	-∞

W
Weight matrix

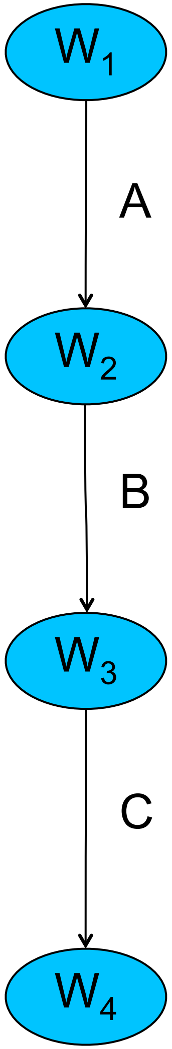
W_1	-∞	-∞	-∞
-∞	W_2	-∞	-∞
-∞	-∞	W_3	-∞
-∞	-∞	-∞	W_4

M^2=MWM (in the max/* algebra)

-∞	-∞	AW_2B	-∞
-∞	-∞	-∞	BW_3C
-∞	-∞	-∞	-∞
-∞	-∞	-∞	-∞



Computation on the whole graph



$A, B, \ldots, W_i$: 2 by 2 matrices

M
Adjacency matrix

-∞	A	-∞	-∞
-∞	-∞	B	-∞
-∞	-∞	-∞	C
-∞	-∞	-∞	-∞

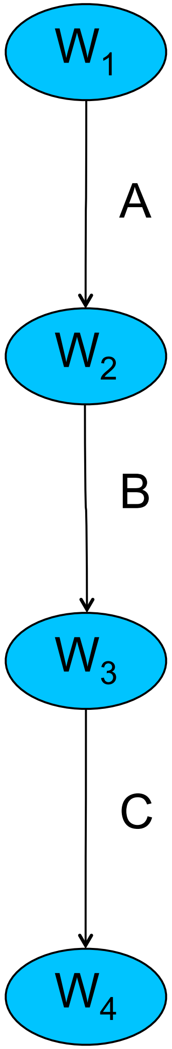
W
Weight matrix

W_1	-∞	-∞	-∞
-∞	W_2	-∞	-∞
-∞	-∞	W_3	-∞
-∞	-∞	-∞	W_4

$M^3=MWM^2$: paths of length 3

In General MWM^{n-1}: paths of length n

Computation on the whole graph



A,B,…, W_i: 2 by 2 matrices

M
Adjacency matrix

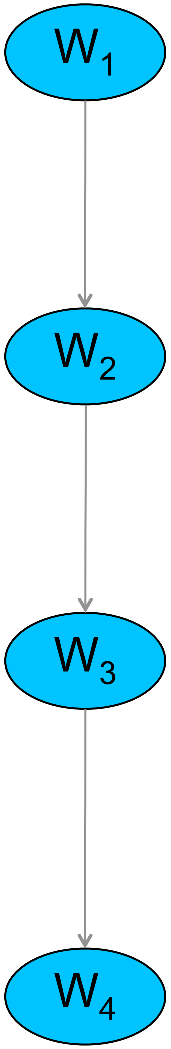
-∞	A	-∞	-∞
-∞	-∞	B	-∞
-∞	-∞	-∞	C
-∞	-∞	-∞	-∞

W
Weight matrix

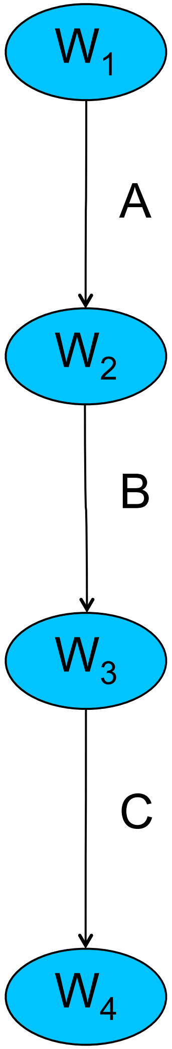
W_1	-∞	-∞	-∞
-∞	W_2	-∞	-∞
-∞	-∞	W_3	-∞
-∞	-∞	-∞	W_4

$M^3=MWM^2$: paths of length 3

In General MWM^{n-1}: paths of length n



Computation on the whole graph



A,B,…, W_i: 2 by 2 matrices

M
Adjacency matrix

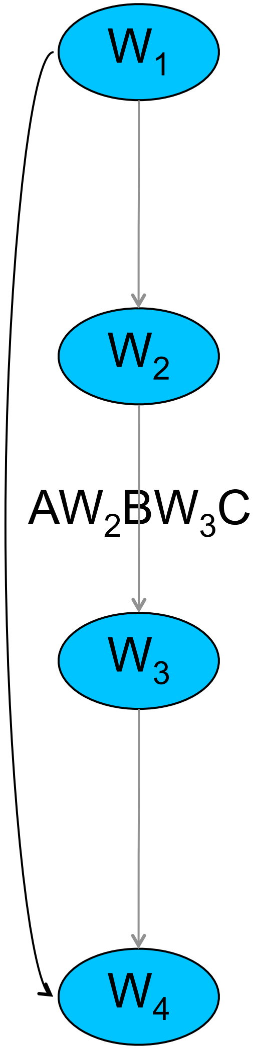
-∞	A	-∞	-∞
-∞	-∞	B	-∞
-∞	-∞	-∞	C
-∞	-∞	-∞	-∞

W
Weight matrix

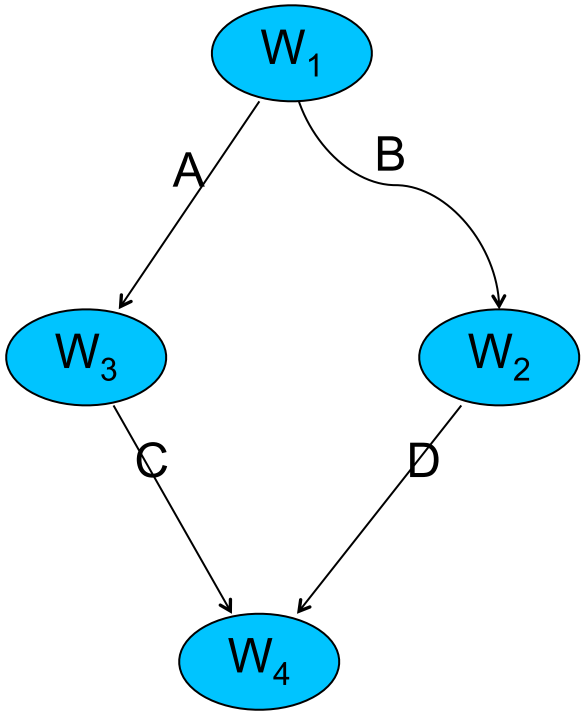
W_1	-∞	-∞	-∞
-∞	W_2	-∞	-∞
-∞	-∞	W_3	-∞
-∞	-∞	-∞	W_4

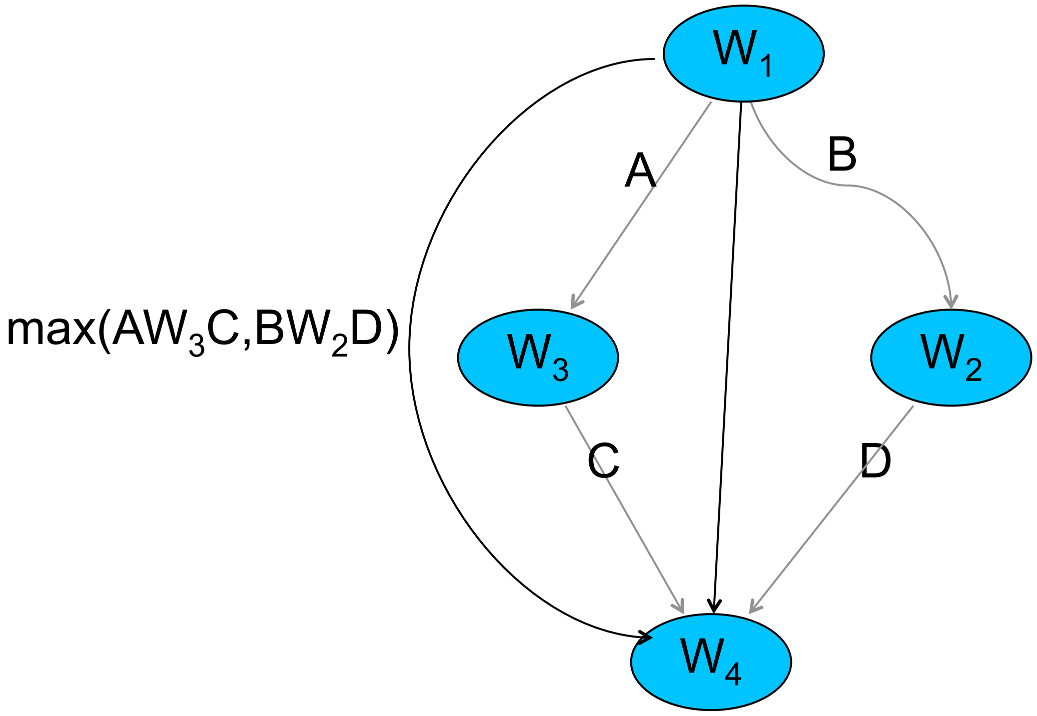
$M^3=MWM^2$: paths of length 3

In General MWM^{n-1}: paths of length n



This also works with concurrent paths





Implementation

Algorithm

1. Compute n the length of the longest path
2. Compute M^n (using the correct algebras), keep track of intermediate decisions.
3. Determine the best mapping depending on the mapping of the *start* and *end* nodes

Advantages:

- Polynomial
- Simple to implement (less bugs, ref. impl)
- Basic operations

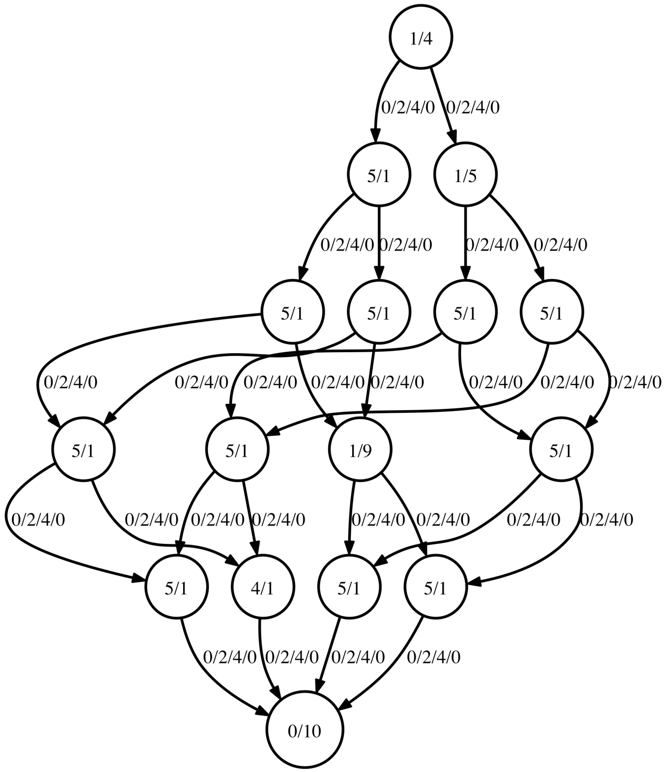
Drawbacks:

- Sub-optimal
- Memory costy

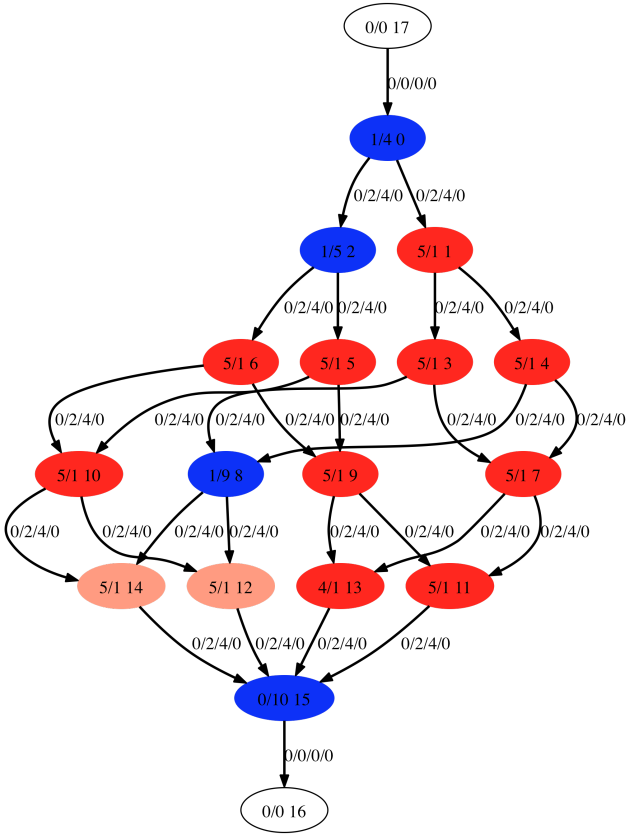
Duplication

Enable duplication in case of a join if it provides better makespan.

Results



1/4
0/2/4/0
0/2/4/0
5/1
1/5
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
5/1
5/1
5/1
5/1
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
5/1
5/1
1/9
5/1
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
5/1
4/1
5/1
5/1
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/10



0/0 17
0/0/0/0
1/4 0
0/2/4/0
0/2/4/0
1/5 2
5/1 1
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
5/1 6
5/1 5
5/1 3
5/1 4
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
5/1 10
1/9 8
5/1 9
5/1 7
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
5/1 14
5/1 12
4/1 13
5/1 11
0/2/4/0
0/2/4/0
0/2/4/0
0/2/4/0
0/10 15
0/0/0/0
0/0 16

What does this algorithm really compute?

What does this algorithm really compute?

A mapping for:

What does this algorithm really compute?

A mapping for:

An unlimited number of GPUs

What does this algorithm really compute?

A mapping for:

- An unlimited number of GPUs
- An unlimited number of CPUs

What does this algorithm really compute?

A mapping for:

An unlimited number of GPUs

An unlimited number of CPUs

No bottleneck for memory transfer

What does this algorithm really compute?

A mapping for:

- An unlimited number of GPUs
- An unlimited number of CPUs
- No bottleneck for memory transfer

In practice: almost all tasks are mapped on GPUs…

Scheduling and Load-balancing

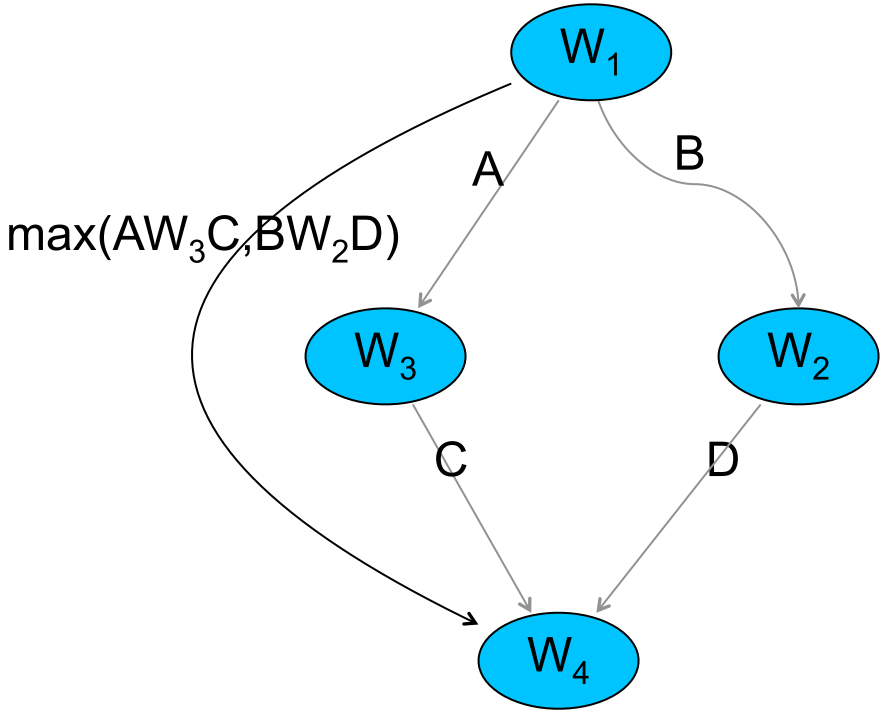
Difficult tasks:

- We make no hypothesis on the ratio CPU/GPU (number performance, etc.)

Different ideas:

- Change tasks mapping based on this ratio (which tasks?)
- Build cluster, and change cluster mapping (which clusters?)
- Apply a greedy algorithm to perform the scheduling (why no only do the greedy algorithm?
- Use undetermined tasks (ok, but we do have many).

Undetermined tasks



Basically : CP computing

W_3 on CP, what about W_2?

In general, the algorithm forces W_2’s mapping

Maybe this mapping has no influence on the critical path?

New version of the algorithm

Same as before but:

Determine the influence of the mapping of non-critical tasks

If no influence : this task can later be scheduled on any resources

Requires (probably) to get rid of the max/*, min/+ algebra

Unanswered questions

Efficient scheduling?

Efficient load balancing?

Mapping assuming unlimited resources: really a good idea?

Mid-term between greedy scheduling and (exponential) linear program

Conclusion

GPU : new resource to execute computation

A real implementation of the urnelated model

Need to take into account memory transfer

A lot of room for interesting scheduling problems