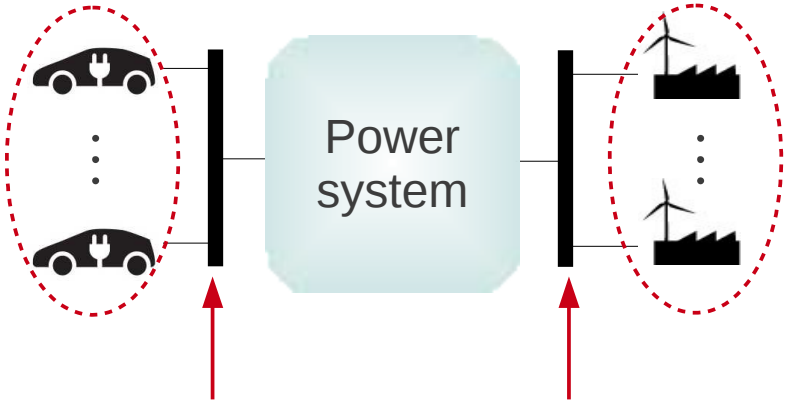
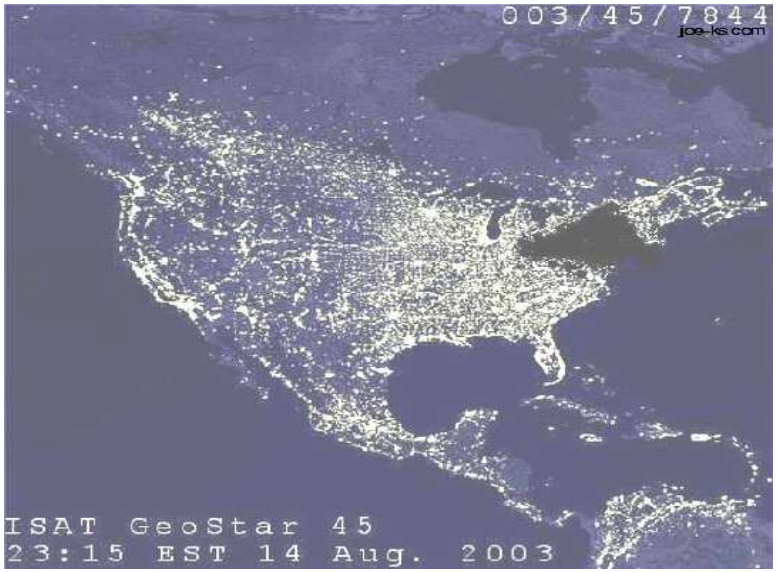
GPU-based techniques for power system analysis

Manuel Marin
University of Perpignan
France

A. Classic power system

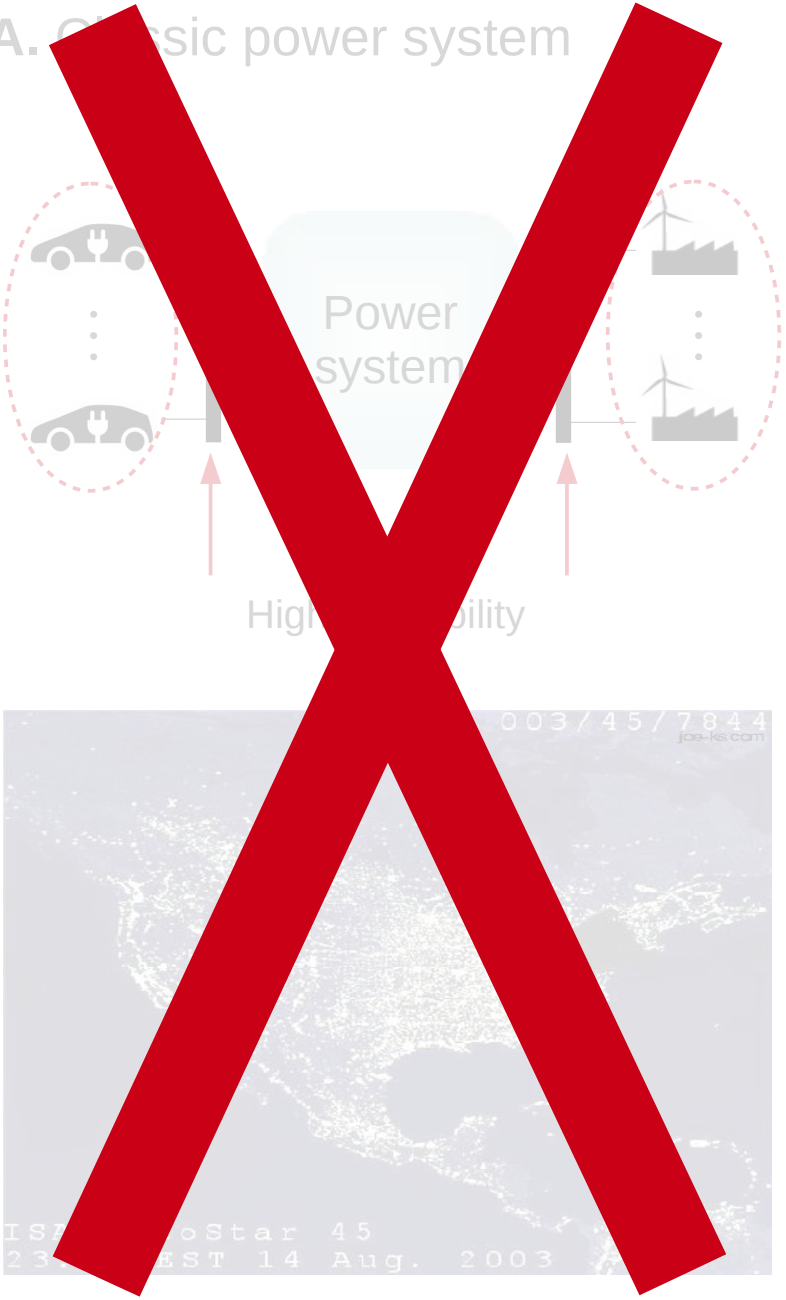


High availability

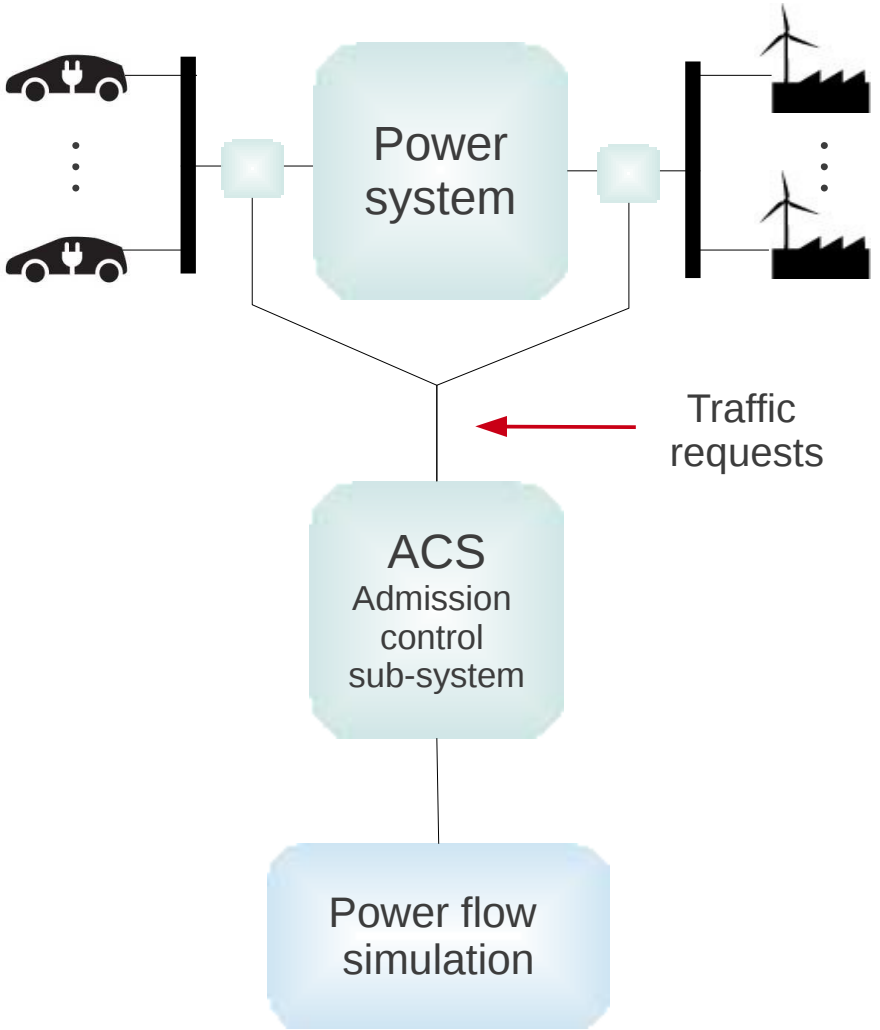


B. Smart grid

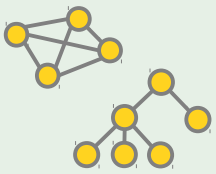
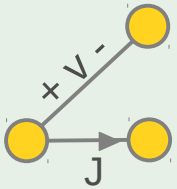
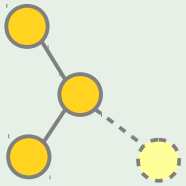
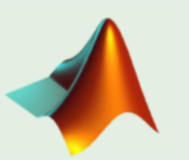
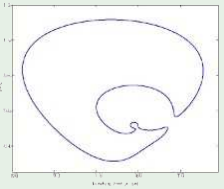
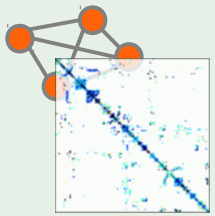
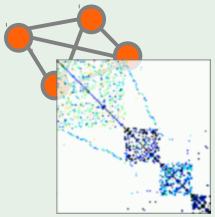
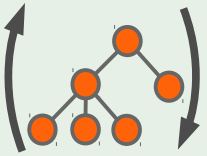
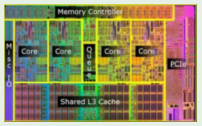
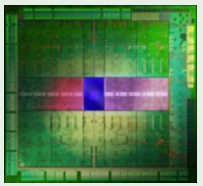
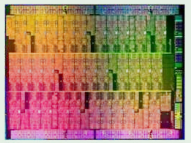
A. Classic power system



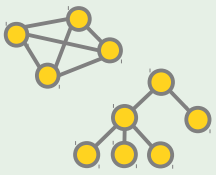
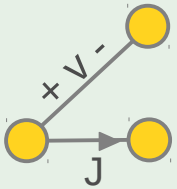
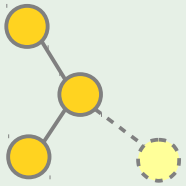
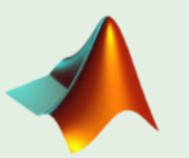
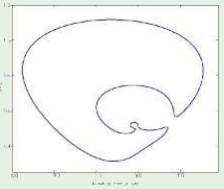
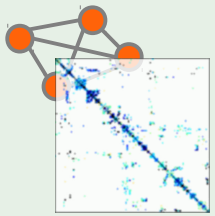
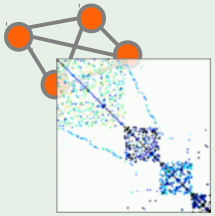
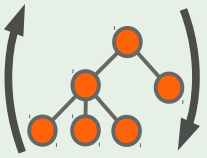
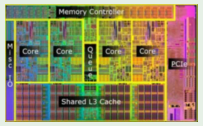
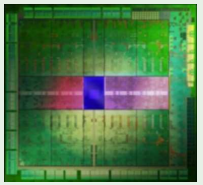
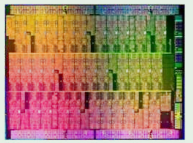
B. Smart grid



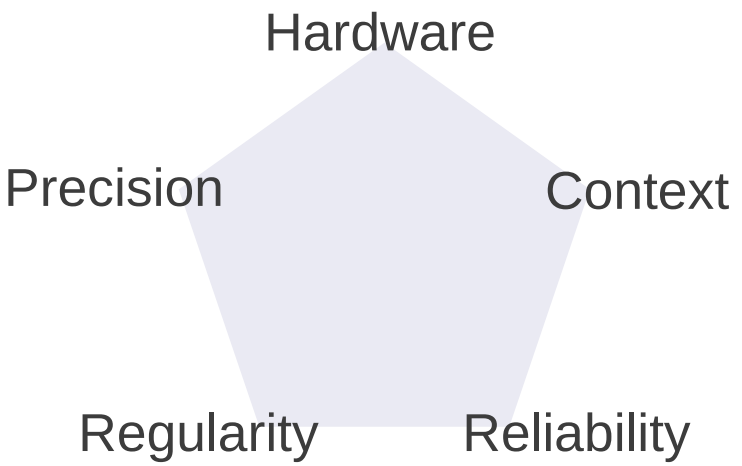
Power flow simulation

Scenario	 Topology	 Values	 Evolution
Software	 Matlab/Simulink	CYME	 Dome
Algorithm	 Newton-Raphson	 Gauss-Seidel	 BFS
Architecture	 CPU	 GPU	 Intel MIC

Power flow simulation

Scenario	 Topology	 Values	 Evolution
Software	 Matlab/Simulink	CYME Dome	
Algorithm	 Newton-Raphson	 Gauss-Seidel	 BFS
Architecture	 CPU	 GPU	 Intel MIC

Modeling



Two talks

- Regularity versus load-balancing on GPU for treefix computations

- FuzzyGPU: a fuzzy arithmetic library for GPU

ICCS 2013: Computation at the Frontiers of Science
Barcelona, June 5-7 2013

Regularity versus load-balancing on GPU for treefix computations

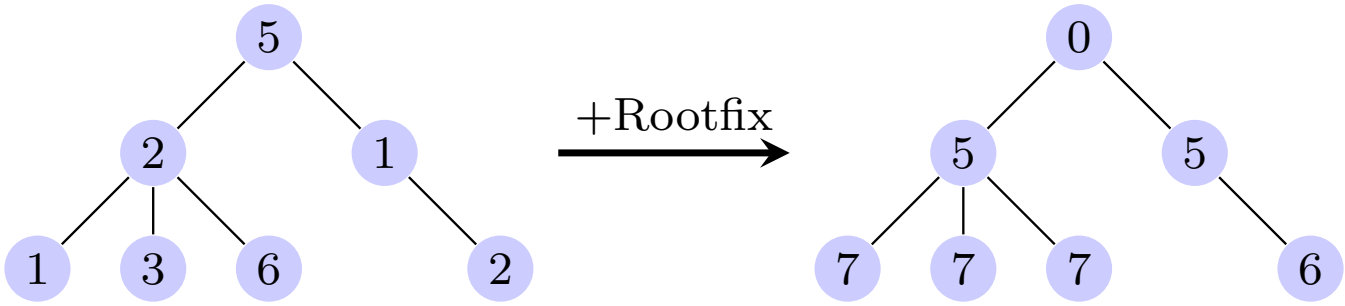
David Defour, **Manuel Marin**

University of Perpignan Via Domitia, DALI,
University Montpellier 2, LIRMM,
CNRS UMR 5506, France

Outline

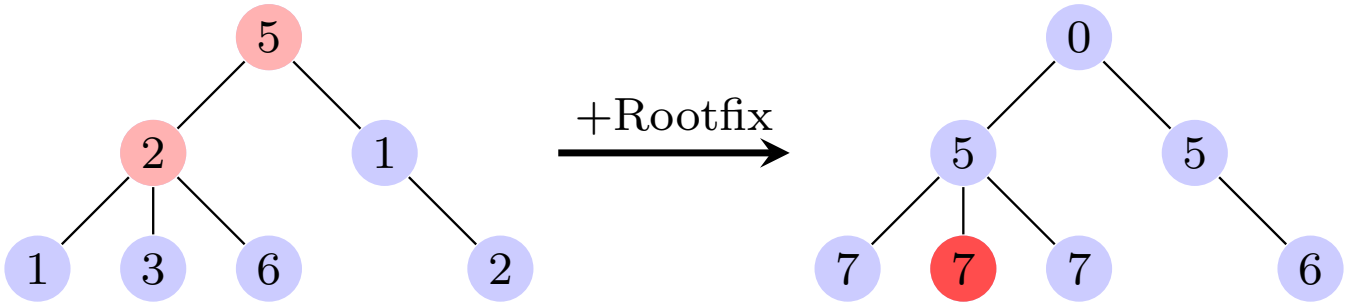
Presentation of +Rootfix

Given a weighted tree, *+Rootfix* returns to each vertex the sum of all its ancestors.



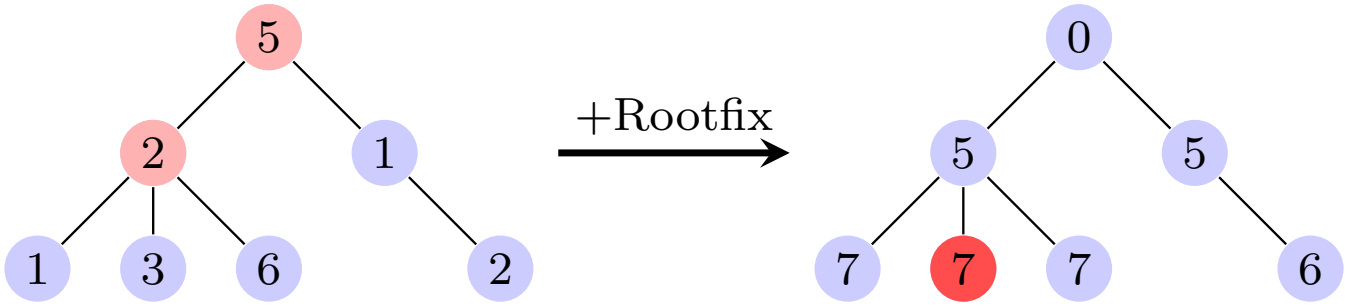
Presentation of +Rootfix

Given a weighted tree, *+Rootfix* returns to each vertex the sum of all its ancestors.



Presentation of +Rootfix

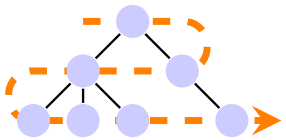
Given a weighted tree, *+Rootfix* returns to each vertex the sum of all its ancestors.



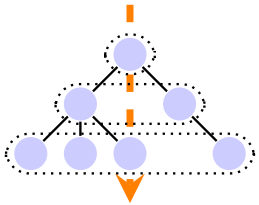
This operation appears in many problems including load flow, parsimony score (phylogenetics) and others.

Different ways of computing +Rootfix

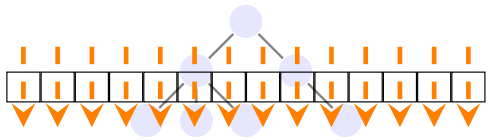
Method	Regularity	GPU-friendly	Memory overhead	Operation overhead
Sequential				
Load-balancing				
Regular				



Sequential



Load-balancing



Regular

+Rootfix using load-balancing

+Rootfix load-balancing algorithm[1]

Input: Adjacency lists A_i, weights array W, queues.
Output: Results array $Output$.

```
1:  Output[0] ← 0
2:  inQ ← {}
3:  inQ.LockedEnqueue(0)
4:  while inQ != {} do
5:      outQ ← {}
6:      for i in inQ do in parallel
7:          for j in A_i do in parallel
8:              Output[j] ← Output[i] + W[i]
9:              outQ.LockedEnqueue(j)
10:     inQ ← outQ
```

[1] Duane Merrill et al. "Scalable GPU graph traversal". In: *SIGPLAN Not.* 47.8 (Feb. 2012).

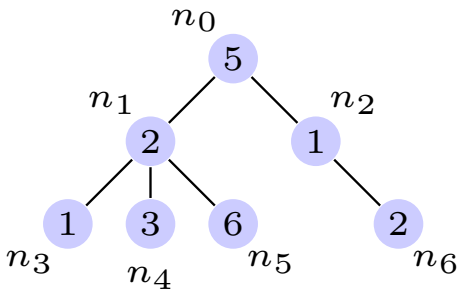
+Rootfix using load-balancing

+Rootfix load-balancing algorithm[1]

Input: Adjacency lists A_i, weights array W, queues.
Output: Results array $Output$.

```
1:  Output[0] ← 0
2:  inQ ← {}
3:  inQ.LockedEnqueue(0)
4:  while inQ != {} do
5:      outQ ← {}
6:      for i in inQ do in parallel
7:          for j in A_i do in parallel
8:              Output[j] ← Output[i] + W[i]
9:              outQ.LockedEnqueue(j)
10:     inQ ← outQ
```

$A_0 = \{1, 2\}$
$A_1 = \{3, 4, 5\}$
$A_2 = \{6\}$
$A_3 = \{\}$
$A_4 = \{\}$
$A_5 = \{\}$
$A_6 = \{\}$
$W = [5, 2, 1, 1, 3, 6, 2]$



[1] Duane Merrill et al. "Scalable GPU graph traversal". In: *SIGPLAN Not.* 47.8 (Feb. 2012).

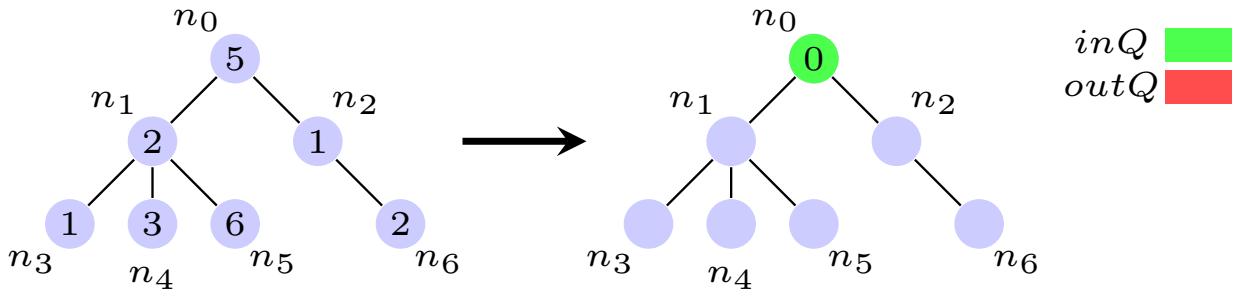
+Rootfix using load-balancing

+Rootfix load-balancing algorithm[1]

Input: Adjacency lists A_i, weights array W, queues.
Output: Results array $Output$.

```
1:  Output[0] ← 0
2:  inQ ← {}
3:  inQ.LockedEnqueue(0)
4:  while inQ != {} do
5:      outQ ← {}
6:      for i in inQ do in parallel
7:          for j in A_i do in parallel
8:              Output[j] ← Output[i] + W[i]
9:              outQ.LockedEnqueue(j)
10:     inQ ← outQ
```

$A_0 = \{1, 2\}$
$A_1 = \{3, 4, 5\}$
$A_2 = \{6\}$
$A_3 = \{\}$
$A_4 = \{\}$
$A_5 = \{\}$
$A_6 = \{\}$
$W = [5, 2, 1, 1, 3, 6, 2]$



[1] Duane Merrill et al. "Scalable GPU graph traversal". In: *SIGPLAN Not.* 47.8 (Feb. 2012).

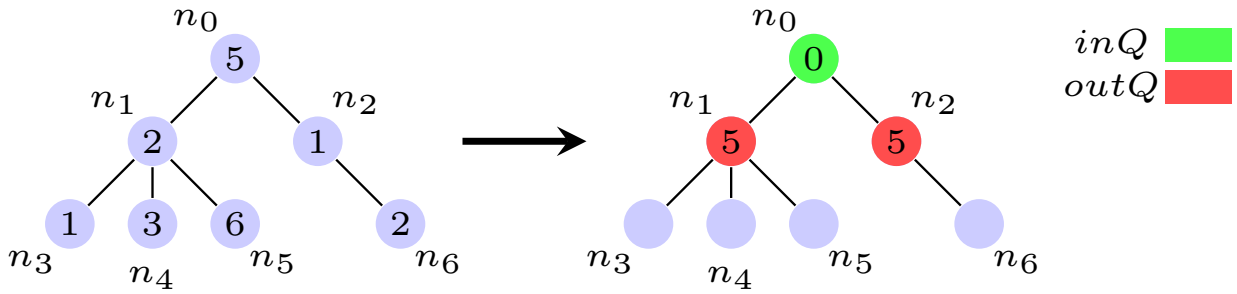
+Rootfix using load-balancing

+Rootfix load-balancing algorithm[1]

Input: Adjacency lists A_i, weights array W, queues.
Output: Results array $Output$.

```
1:  Output[0] ← 0
2:  inQ ← {}
3:  inQ.LockedEnqueue(0)
4:  while inQ != {} do
5:      outQ ← {}
6:      for i in inQ do in parallel
7:          for j in A_i do in parallel
8:              Output[j] ← Output[i] + W[i]
9:              outQ.LockedEnqueue(j)
10:     inQ ← outQ
```

$A_0 = \{1, 2\}$
$A_1 = \{3, 4, 5\}$
$A_2 = \{6\}$
$A_3 = \{\}$
$A_4 = \{\}$
$A_5 = \{\}$
$A_6 = \{\}$
$W = [5, 2, 1, 1, 3, 6, 2]$



[1] Duane Merrill et al. "Scalable GPU graph traversal". In: *SIGPLAN Not.* 47.8 (Feb. 2012).

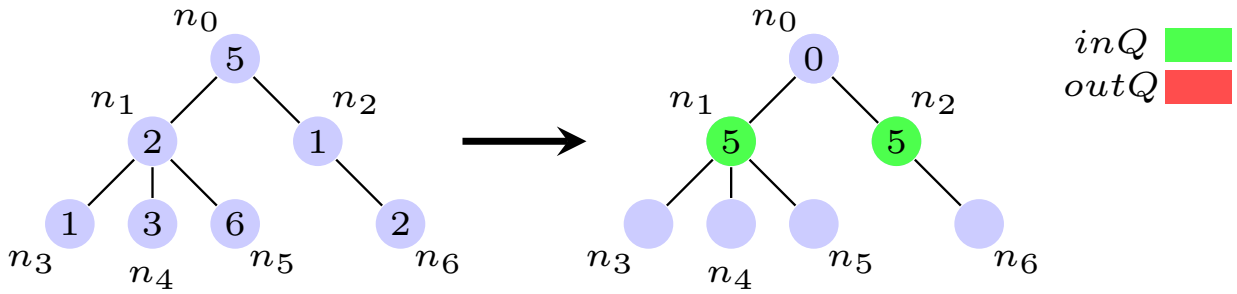
+Rootfix using load-balancing

+Rootfix load-balancing algorithm[1]

Input: Adjacency lists A_i, weights array W, queues.
Output: Results array $Output$.

```
1:  Output[0] ← 0
2:  inQ ← {}
3:  inQ.LockedEnqueue(0)
4:  while inQ != {} do
5:      outQ ← {}
6:      for i in inQ do in parallel
7:          for j in A_i do in parallel
8:              Output[j] ← Output[i] + W[i]
9:              outQ.LockedEnqueue(j)
10:     inQ ← outQ
```

$A_0 = \{1, 2\}$
$A_1 = \{3, 4, 5\}$
$A_2 = \{6\}$
$A_3 = \{\}$
$A_4 = \{\}$
$A_5 = \{\}$
$A_6 = \{\}$
$W = [5, 2, 1, 1, 3, 6, 2]$



[1] Duane Merrill et al. "Scalable GPU graph traversal". In: *SIGPLAN Not.* 47.8 (Feb. 2012).

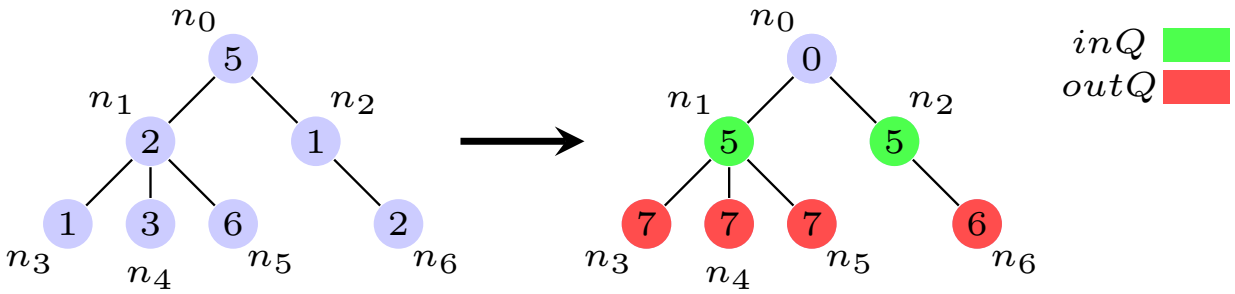
+Rootfix using load-balancing

+Rootfix load-balancing algorithm[1]

Input: Adjacency lists A_i, weights array W, queues.
Output: Results array $Output$.

```
1:  Output[0] ← 0
2:  inQ ← {}
3:  inQ.LockedEnqueue(0)
4:  while inQ != {} do
5:      outQ ← {}
6:      for i in inQ do in parallel
7:          for j in A_i do in parallel
8:              Output[j] ← Output[i] + W[i]
9:              outQ.LockedEnqueue(j)
10:     inQ ← outQ
```

$A_0 = \{1, 2\}$
$A_1 = \{3, 4, 5\}$
$A_2 = \{6\}$
$A_3 = \{\}$
$A_4 = \{\}$
$A_5 = \{\}$
$A_6 = \{\}$
$W = [5, 2, 1, 1, 3, 6, 2]$



[1] Duane Merrill et al. "Scalable GPU graph traversal". In: *SIGPLAN Not.* 47.8 (Feb. 2012).

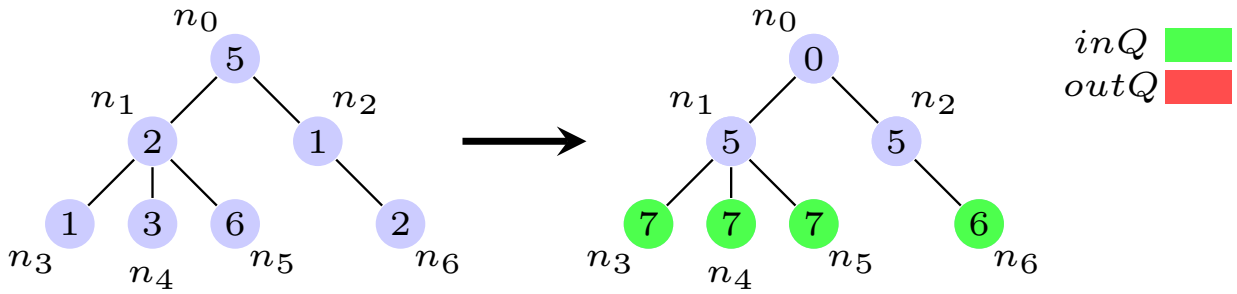
+Rootfix using load-balancing

+Rootfix load-balancing algorithm[1]

Input: Adjacency lists A_i, weights array W, queues.
Output: Results array $Output$.

```
1:  Output[0] ← 0
2:  inQ ← {}
3:  inQ.LockedEnqueue(0)
4:  while inQ != {} do
5:      outQ ← {}
6:      for i in inQ do in parallel
7:          for j in A_i do in parallel
8:              Output[j] ← Output[i] + W[i]
9:              outQ.LockedEnqueue(j)
10:     inQ ← outQ
```

$A_0 = \{1, 2\}$
$A_1 = \{3, 4, 5\}$
$A_2 = \{6\}$
$A_3 = \{\}$
$A_4 = \{\}$
$A_5 = \{\}$
$A_6 = \{\}$
$W = [5, 2, 1, 1, 3, 6, 2]$



[1] Duane Merrill et al. "Scalable GPU graph traversal". In: *SIGPLAN Not.* 47.8 (Feb. 2012).

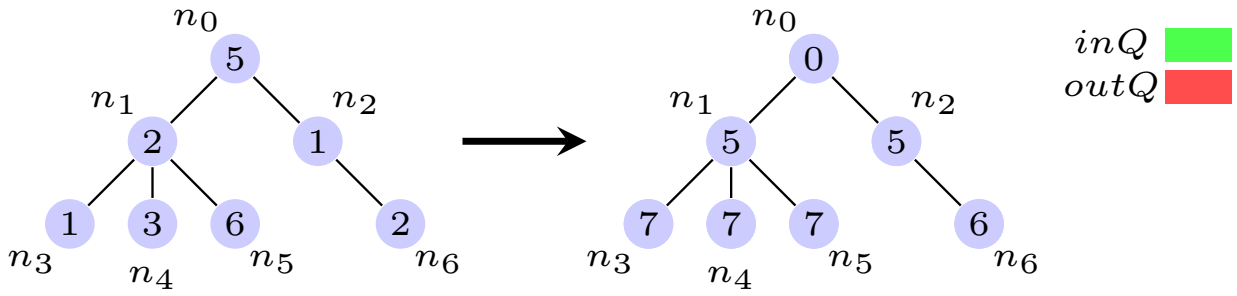
+Rootfix using load-balancing

+Rootfix load-balancing algorithm[1]

Input: Adjacency lists A_i, weights array W, queues.
Output: Results array $Output$.

```
1:  Output[0] ← 0
2:  inQ ← {}
3:  inQ.LockedEnqueue(0)
4:  while inQ != {} do
5:      outQ ← {}
6:      for i in inQ do in parallel
7:          for j in A_i do in parallel
8:              Output[j] ← Output[i] + W[i]
9:              outQ.LockedEnqueue(j)
10:     inQ ← outQ
```

$A_0 = \{1, 2\}$
$A_1 = \{3, 4, 5\}$
$A_2 = \{6\}$
$A_3 = \{\}$
$A_4 = \{\}$
$A_5 = \{\}$
$A_6 = \{\}$
$W = [5, 2, 1, 1, 3, 6, 2]$



[1] Duane Merrill et al. "Scalable GPU graph traversal". In: *SIGPLAN Not.* 47.8 (Feb. 2012).

+Rootfix using regularity

+Rootfix regular algorithm[2]

Input: Nodes list N, weigths array W, braces array B, working array E.
Output: Results array $Rootfix$.

1: **for** i **in** N **do in parallel**
2: $\quad E[B[_(i]] \leftarrow W[i]$
3: $\quad E[B[i_)]] \leftarrow -W[i]$
4: Run an inplace +scan on E
5: **for** i **in** N **do in parallel**
6: $\quad Rootfix[i] \longleftarrow E[B[_(i]]$

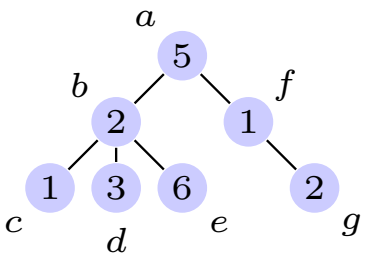
[2] Guy E. Blelloch. *Vector models for data-parallel computing*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 0-262-02313-X.

+Rootfix using regularity

+Rootfix regular algorithm[2]

Input: Nodes list N, weigths array W, braces array B, working array E.
Output: Results array $Rootfix$.

1: **for** i **in** N **do in parallel**
2: $\quad E[B[{}_{(}i]] \leftarrow W[i]$
3: $\quad E[B[i_{)}]] \leftarrow -W[i]$
4: Run an inplace +scan on E
5: **for** i **in** N **do in parallel**
6: $\quad Rootfix[i] \longleftarrow E[B[{}_{(}i]]$



N	a	b	c	d	e	f	g
W	5	2	1	3	6	1	2

B	$({}_{(}a$	${}_{(}b$	$({}_{(}c$	$c_{)})$	${}_{(}d$	$d_{)})$	$({}_{(}e$	$e_{)})$	$b_{)})$	${}_{(}f$	$({}_{(}g$	$g_{)})$	$f_{)})$	$a_{)})$
E														

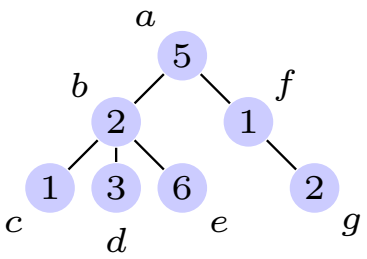
[2] Guy E. Blelloch. *Vector models for data-parallel computing*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 0-262-02313-X.

+Rootfix using regularity

+Rootfix regular algorithm[2]

Input: Nodes list N, weigths array W, braces array B, working array E.
Output: Results array $Rootfix$.

1: **for** i **in** N **do in parallel**
2: $\quad E[B[_{(}i]] \leftarrow W[i]$
3: $\quad E[B[i_{)}]] \leftarrow -W[i]$
4: Run an inplace +scan on E
5: **for** i **in** N **do in parallel**
6: $\quad Rootfix[i] \longleftarrow E[B[_{(}i]]$



N	a	b	c	d	e	f	g
W	5	2	1	3	6	1	2

B	$(_a$	$(_b$	$(_c$	$c_)$	$(_d$	$d_)$	$(_e$	$e_)$	$b_)$	$(_f$	$(_g$	$g_)$	$f_)$	$a_)$
E	5													-5

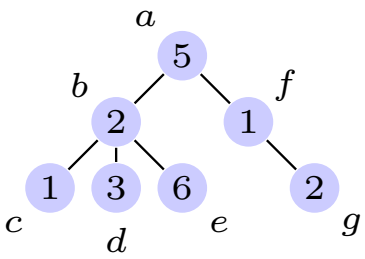
[2] Guy E. Blelloch. *Vector models for data-parallel computing*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 0-262-02313-X.

+Rootfix using regularity

+Rootfix regular algorithm[2]

Input: Nodes list N, weigths array W, braces array B, working array E.
Output: Results array $Rootfix$.

1: **for** i **in** N **do in parallel**
2: $\quad E[B[_{(}i]] \leftarrow W[i]$
3: $\quad E[B[i_{)}]] \leftarrow -W[i]$
4: Run an inplace +scan on E
5: **for** i **in** N **do in parallel**
6: $\quad Rootfix[i] \longleftarrow E[B[_{(}i]]$



N	a	b	c	d	e	f	g
W	5	2	1	3	6	1	2

B	$(a$	$(b$	$(c$	$c)$	$(d$	$d)$	$(e$	$e)$	$b)$	$(f$	$(g$	$g)$	$f)$	$a)$
E	5	2	1	-1	3	-3	6	-6	-2	1	2	-2	-1	-5

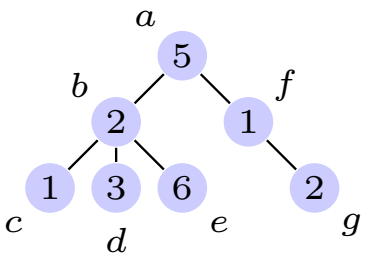
[2] Guy E. Blelloch. *Vector models for data-parallel computing*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 0-262-02313-X.

+Rootfix using regularity

+Rootfix regular algorithm[2]

Input: Nodes list N, weigths array W, braces array B, working array E.
Output: Results array $Rootfix$.

1: **for** i **in** N **do in parallel**
2: $\quad E[B[_{(}i]] \leftarrow W[i]$
3: $\quad E[B[i_{)}]] \leftarrow -W[i]$
4: Run an inplace +scan on E
5: **for** i **in** N **do in parallel**
6: $\quad Rootfix[i] \longleftarrow E[B[_{(}i]]$



N	a	b	c	d	e	f	g
W	5	2	1	3	6	1	2

B	$(a$	$(b$	$(c$	$c)$	$(d$	$d)$	$(e$	$e)$	$b)$	$(f$	$(g$	$g)$	$f)$	$a)$
E	5	2	1	-1	3	-3	6	-6	-2	1	2	-2	-1	-5
	0	5	7	8	7	10	7	13	7	5	6	8	6	5

+scan

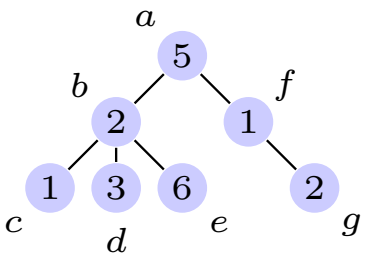
[2] Guy E. Blelloch. *Vector models for data-parallel computing*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 0-262-02313-X.

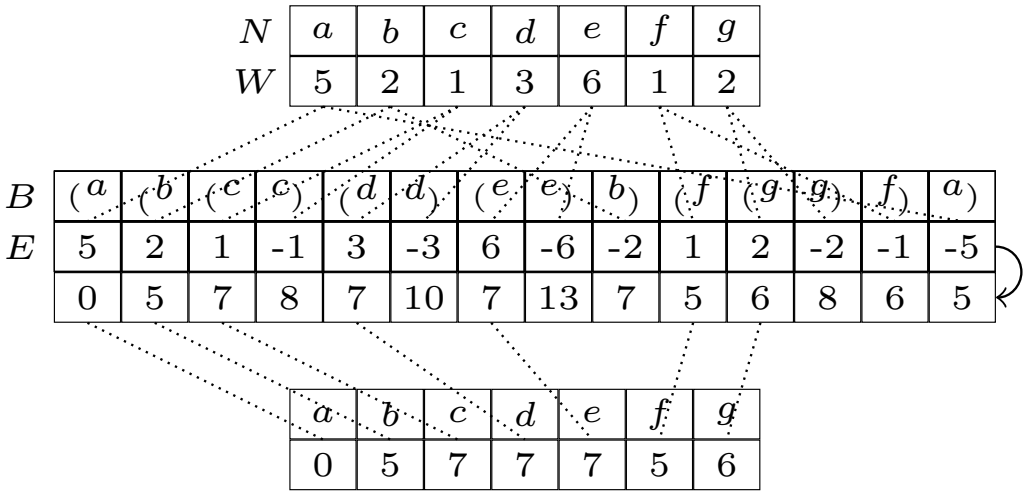
+Rootfix using regularity

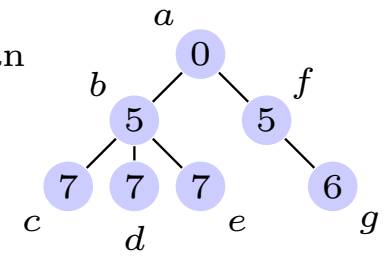
+Rootfix regular algorithm[2]

Input: Nodes list N, weigths array W, braces array B, working array E.
Output: Results array $Rootfix$.

1: **for** i **in** N **do in parallel**
2: $\quad E[B[_{(}i]] \leftarrow W[i]$
3: $\quad E[B[i_{)}]] \leftarrow -W[i]$
4: Run an inplace +scan on E
5: **for** i **in** N **do in parallel**
6: $\quad Rootfix[i] \longleftarrow E[B[_{(}i]]$







[2] Guy E. Blelloch. *Vector models for data-parallel computing*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 0-262-02313-X.

Performance tests

- CUDA implementation of the load-balancing algorithm.
- OpenCL implementation of the regular algorithm.
- Using an NVIDIA GeForce GTX670 GPU.

Suite of benchmark trees[3]

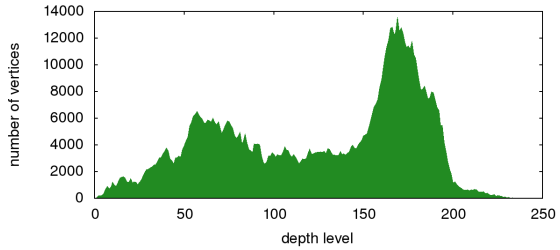
Name	Nb. of vertices	Depth
af_shell9	504855	490
audikw1	943695	236
ldoor	952203	784
af_shell10	1508065	1098
G3_circuit	1585478	705
kkt_power	2063494	36
nlpkkt120	3542400	123
cage15	5154859	81
nlpkkt160	8345600	163
nlpkkt200	16240000	203

[3] *10th DIMACS Implementation Challenge.* http://www.cc.gatech.edu/dimacs10/index.shtml. 2012.

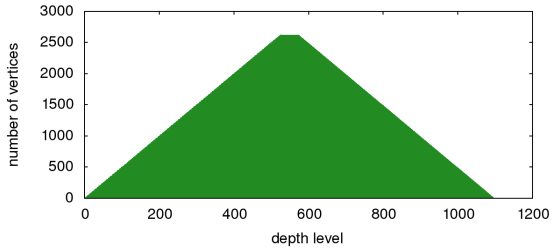
Performance tests

- CUDA implementation of the load-balancing algorithm.
- OpenCL implementation of the regular algorithm.
- Using an NVIDIA GeForce GTX670 GPU.

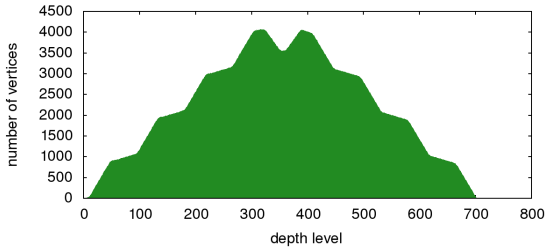
Suite of benchmark trees[3]



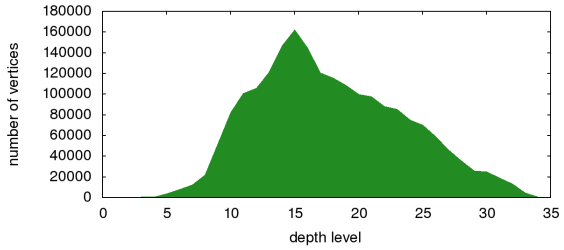
audikw1



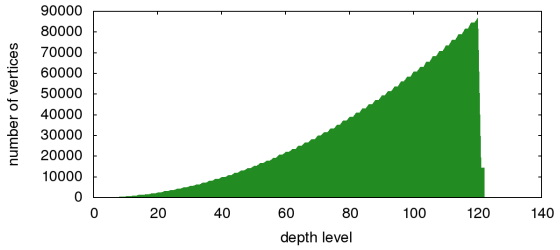
af_shell10



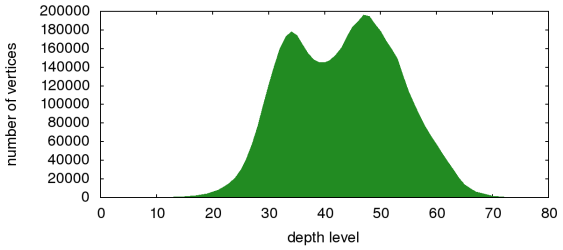
G3_circuit



kkt_power



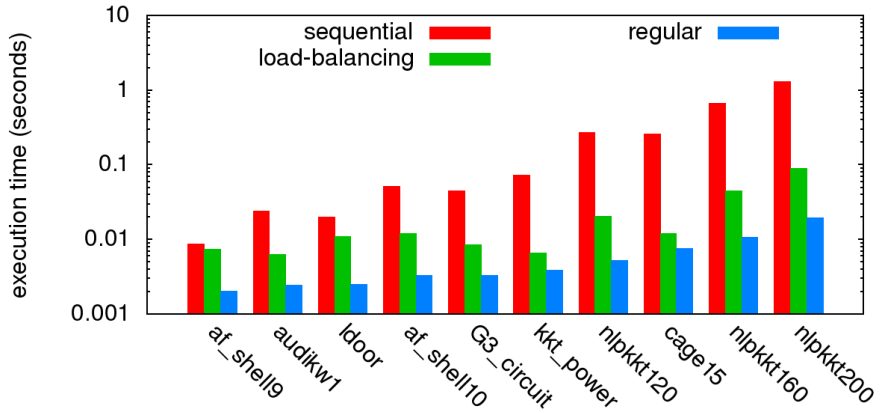
nlpkkt120



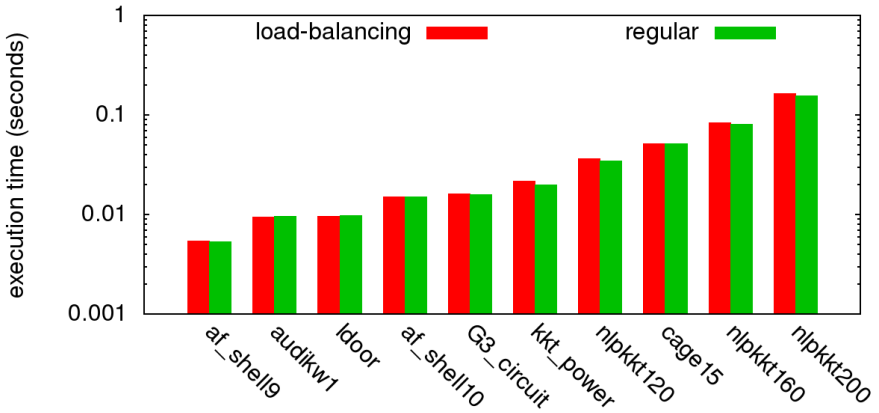
cage15

[3] *10th DIMACS Implementation Challenge.* http://www.cc.gatech.edu/dimacs10/index.shtml. 2012.

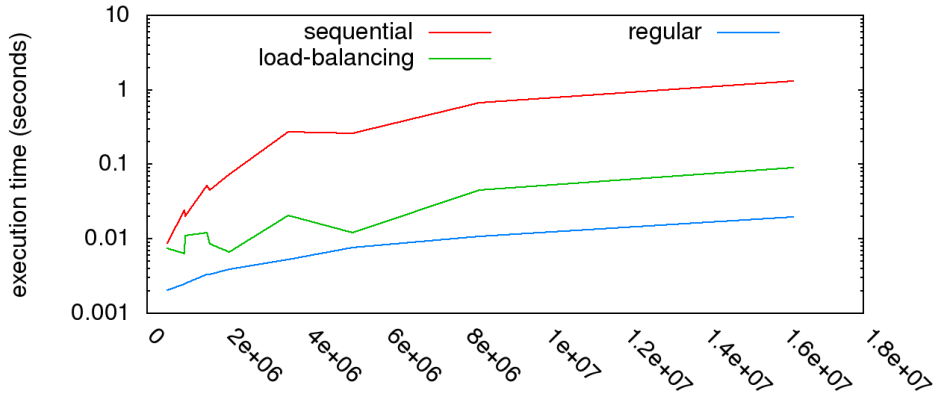
Related performance on GPU



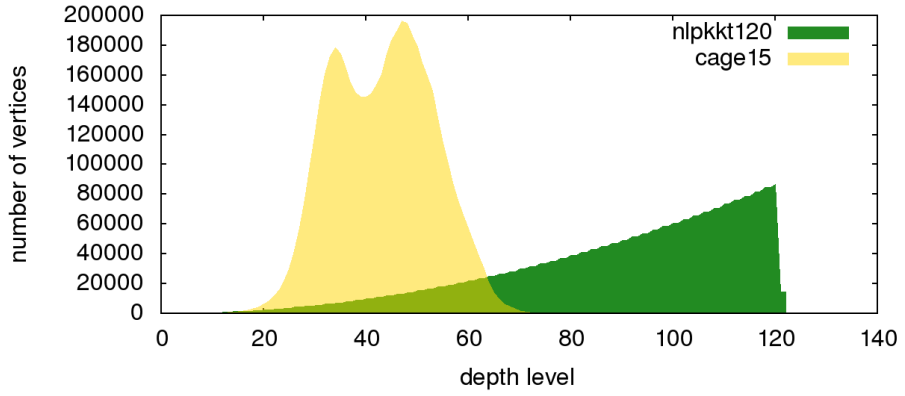
Computation time



Data transfer time

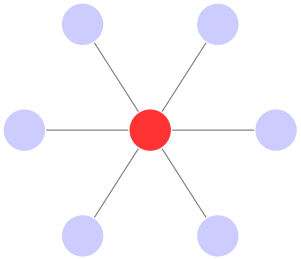


Computation time

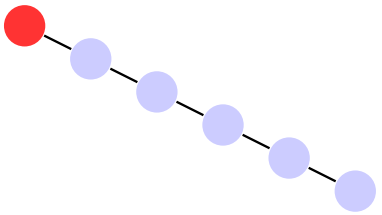


Vertex distribution comparison

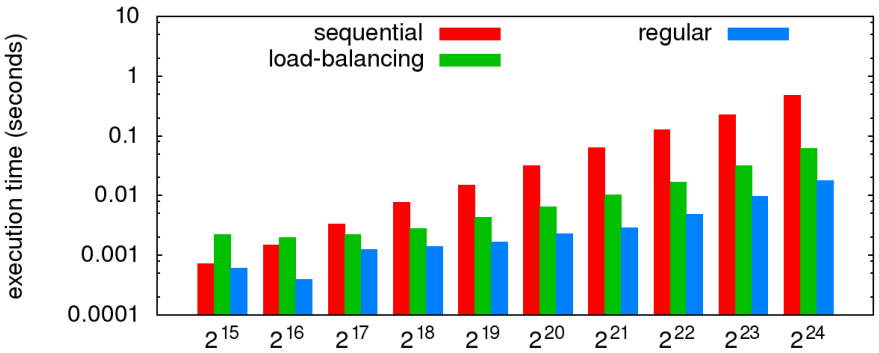
Special topologies



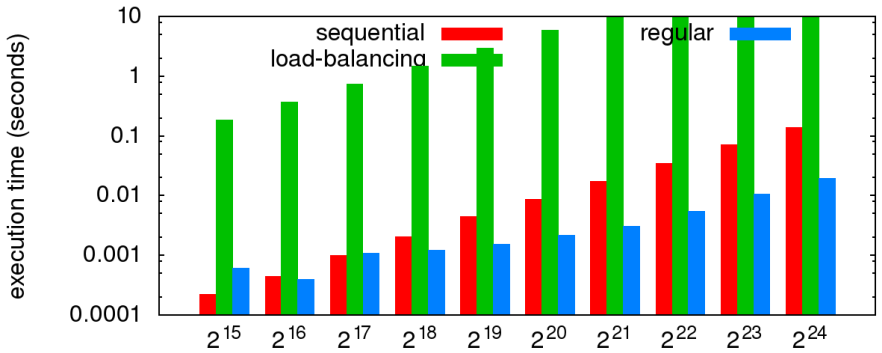
Star



Caterpillar

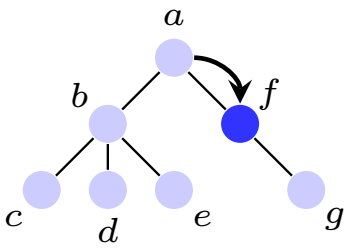


Star computation time

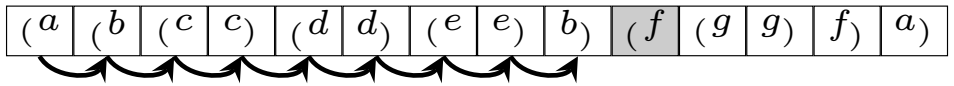


Caterpillar computation time

Accuracy issues



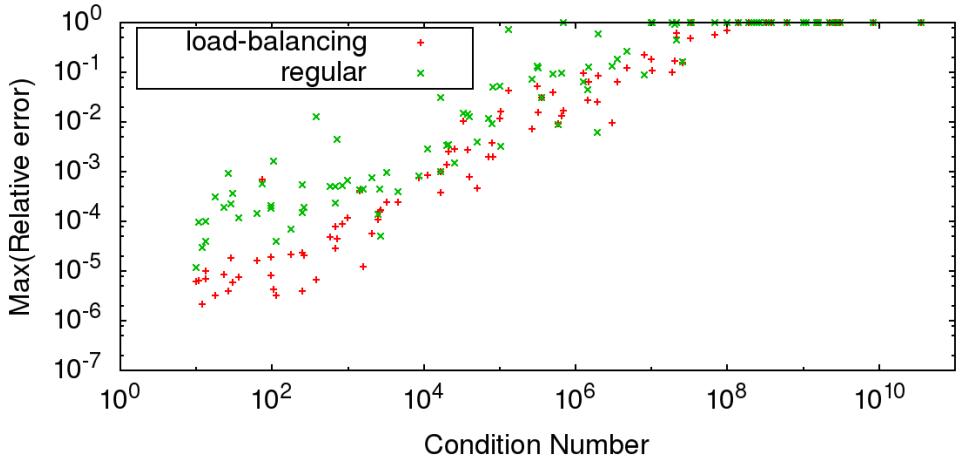
Load-balancing algorithm



Regular algorithm

Different number of operations when performing +Rootfix with different algorithms.

$$e = \frac{|\hat{y} - y|}{|y|}$$



$$C = \frac{\sum_{i=0}^{n-1} |x_i|}{\left| \sum_{i=0}^{n-1} x_i \right|}$$

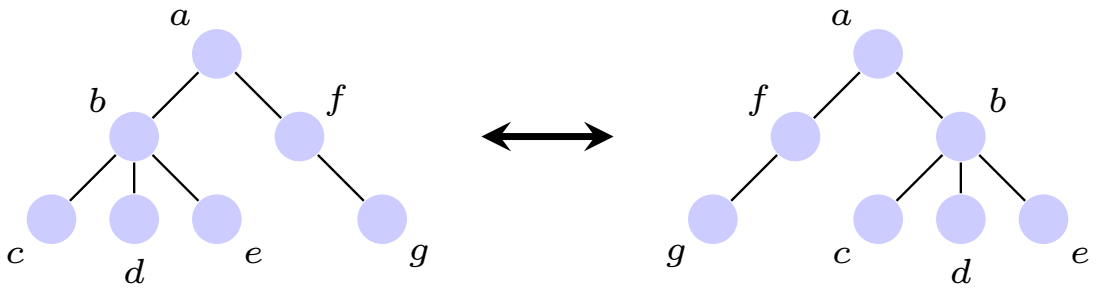
Related accuracy of load-balancing and regular +Rootfix.

Conclusion and future work

- The regular approach is always faster than the load-balancing approach on this type of problem.
- The regular approach is indifferent to topology, whereas the load-balancing approach is strongly correlated with topology.
- When the application involves floating-point data, the regular approach may drive accuracy issues.

Conclusion and future work

- The regular approach is always faster than the load-balancing approach on this type of problem.
- The regular approach is indifferent to topology, whereas the load-balancing approach is strongly correlated with topology.
- When the application involves floating-point data, the regular approach may drive accuracy issues.
    - These issues can be reduced by finding equivalent representations that minimize the impact of cancellation.



FuzzyGPU
a fuzzy arithmetic library for GPU

David Defour, Manuel Marin

University of Perpignan Via Domitia, DALI,
University Montpellier 2, LIRMM,
CNRS UMR 5506, France

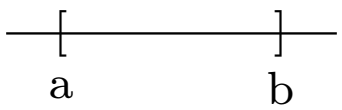
Outline

Introduction

- Uncertainty in expert systems can be efficiently modeled using fuzzy numbers.
- GPUs, which are powerful vector coprocessors, present additional features that can improve fuzzy calculations.
- We present the development of a fuzzy arithmetic library for GPU, introducing two representation formats.
  - Lower-upper encoding for generic fuzzy numbers.
  - Midpoint-radius encoding for symmetric fuzzy numbers and extra GPU performance.

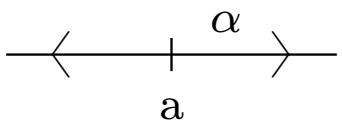
Interval representation

Lower-upper



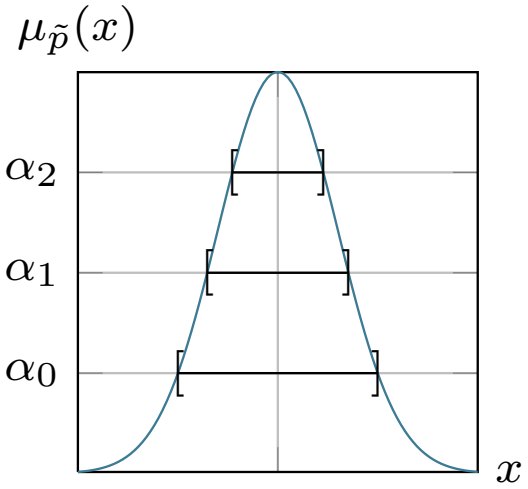
$$
\begin{aligned}
[\ a,b\ ] + [\ c,d\ ] &= [\ \underline{a+b}, \overline{c+d}\ ] \\
[\ a,b\ ] - [\ c,d\ ] &= [\ \underline{a-d}, \overline{b-c}\ ] \\
[\ a,b\ ] \times [\ c,d\ ] &= [\ \min(\underline{ac}, \underline{bd}, \underline{ad}, \underline{bc}), \max(\overline{ac}, \overline{bd}, \overline{ad}, \overline{bc})\ ]
\end{aligned}
$$

Midpoint-radius

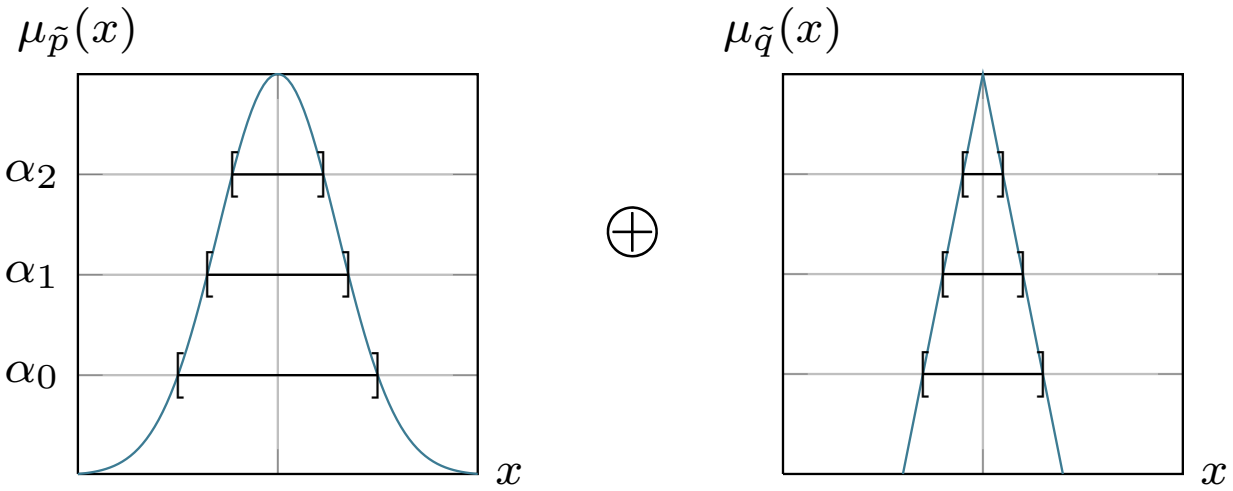


$$
\begin{aligned}
\langle\ a,\alpha\ \rangle + \langle\ b,\beta\ \rangle &= \langle\ a+b, \overline{\epsilon'|a+b| + \alpha + \beta}\ \rangle \\
\langle\ a,\alpha\ \rangle - \langle\ b,\beta\ \rangle &= \langle\ a-b, \overline{\epsilon'|a-b| + \alpha + \beta}\ \rangle \\
\langle\ a,\alpha\ \rangle \times \langle\ b,\beta\ \rangle &= \langle\ ab, \overline{\eta + \epsilon'|ab| + (|a| + \alpha)\beta + \alpha|b|}\ \rangle
\end{aligned}
$$

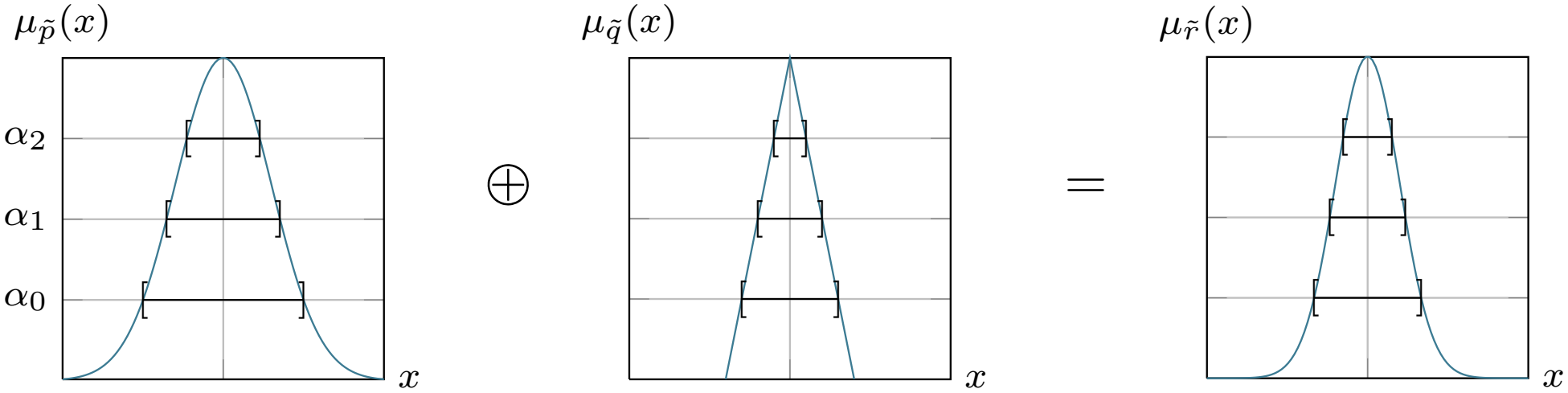
The α-cut concept



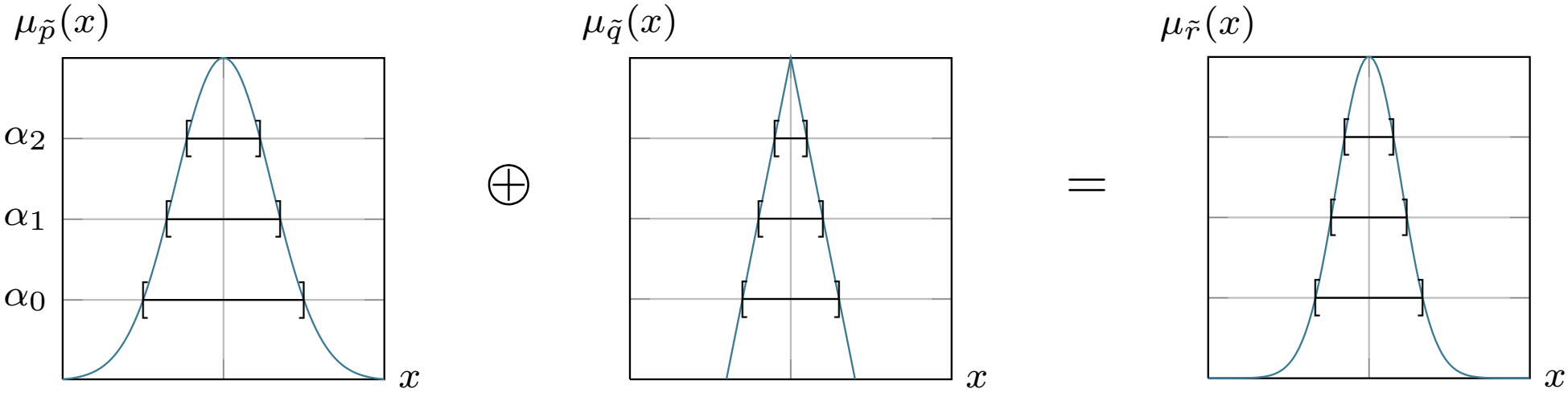
The α-cut concept



The α-cut concept



The α-cut concept



Theorem

I) *If $\tilde{p}$ is a symmetric fuzzy number and $\langle m_i, \rho_i \rangle, \langle m_j, \rho_j \rangle$ two α-cuts, then $m_i = m_j$.*

II) *If $\tilde{p}, \tilde{q}$ are symmetric fuzzy numbers and $\oplus \in \{+, -, \times\}$, then $\tilde{r} = \tilde{p} \oplus \tilde{q}$ is also symmetric.*

Fuzzy encoding

Lower-upper

```
template
<class T, int N>
class lu_fuzzy{
    T low[N];
    T up[N];
};
```

$2N \times \text{sizeof}(T)$ bytes.

Lower-upper fuzzy multiplication

Input: Lower-upper fuzzy operands p and q.
Output: Fuzzy result r.
for i **from** 0 **to** $N-1$ **do**
$r.low[i] \leftarrow min4(p.low[i] \times_{\triangledown} q.low[i], \ldots, p.up[i] \times_{\triangledown} q.up[i])$
$r.up[i] \leftarrow max4(p.low[i] \times_{\triangle} q.low[i], \ldots, p.up[i] \times_{\triangle} q.up[i])$

$14N$ operations.

Fuzzy encoding

Lower-upper

```
template
<class T, int N>
class lu_fuzzy{
    T low[N];
    T up[N];
};
```

$2N \times \text{sizeof}(T)$ bytes.

Lower-upper fuzzy multiplication

Input: Lower-upper fuzzy operands p and q.
Output: Fuzzy result r.
  for i **from** 0 **to** $N-1$ **do**
    $r.low[i] \leftarrow min4(p.low[i] \times_{\triangledown} q.low[i], \ldots, p.up[i] \times_{\triangledown} q.up[i])$
    $r.up[i] \leftarrow max4(p.low[i] \times_{\triangle} q.low[i], \ldots, p.up[i] \times_{\triangle} q.up[i])$

$14N$ operations.

Midpoint-radius

```
template
<class T, int N>
class mr_fuzzy{
    T mp;
    T rad[N];
};
```

$(1 + N) \times \text{sizeof}(T)$ bytes.

Midpoint-radius fuzzy multiplication

Input: Midpoint-radius fuzzy operands p and q.
Output: Fuzzy result r.
  $r.mp \leftarrow p.mp \times q.mp$
  for i **from** 0 **to** $N-1$ **do**
    $r.rad[i] \leftarrow eta +_{\triangle} eps \times_{\triangle} |r.mp| +_{\triangle} \ldots +_{\triangle} p.rad[i] \times_{\triangle} |q.mp|$

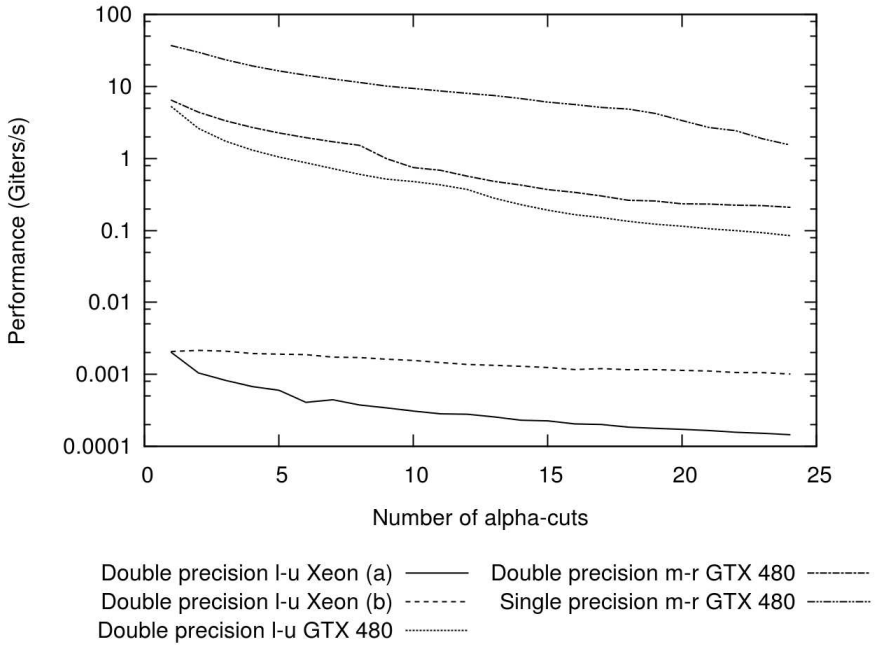
$5 + 5N$ operations.

AXPY kernel

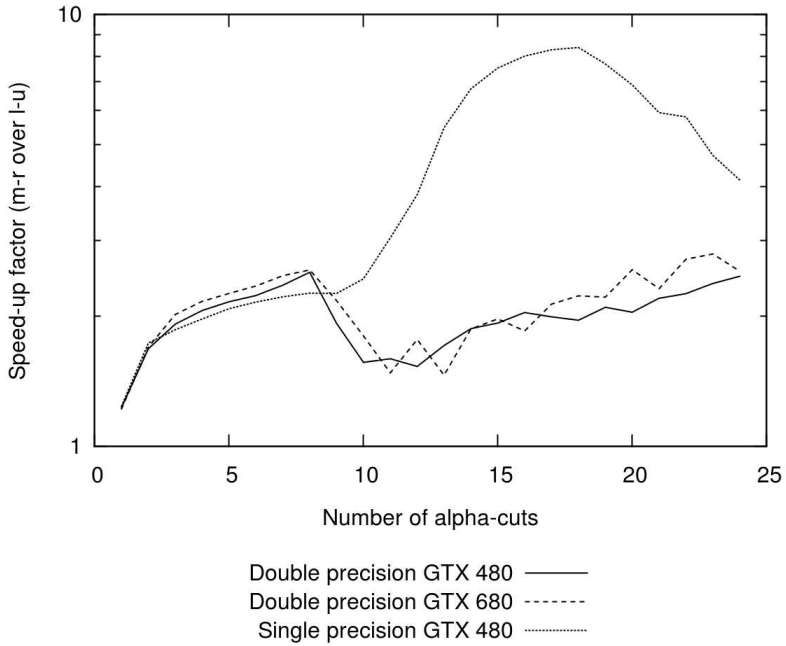
```
typedef lu_fuzzy<double,4> FUZZY;
__global__ void axpy(int iters, FUZZY a, FUZZY b, FUZZY *output){
    FUZZY c;
    for (int i = 0; i < iters; i++)
        c = a * c + b;
    output[blockIdx.x * blockDim.x + threadIdx.x] = c;
}
```

AXPY kernel

```
typedef lu_fuzzy<double,4> FUZZY;
__global__ void axpy(int iters, FUZZY a, FUZZY b, FUZZY *output){
    FUZZY c;
    for (int i = 0; i < iters; i++)
        c = a * c + b;
    output[blockIdx.x * blockDim.x + threadIdx.x] = c;
}
```



Related performance



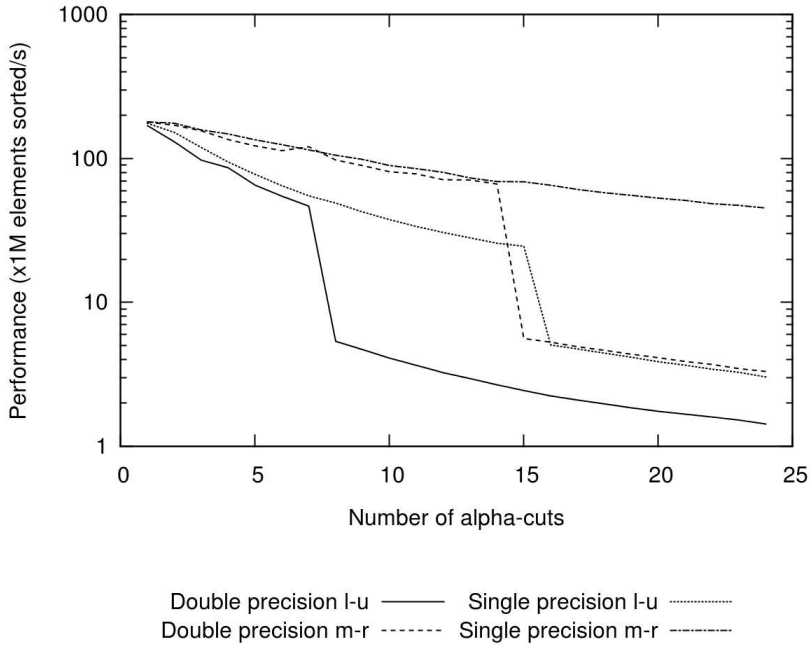
Algorithmic gain

Thrust sort

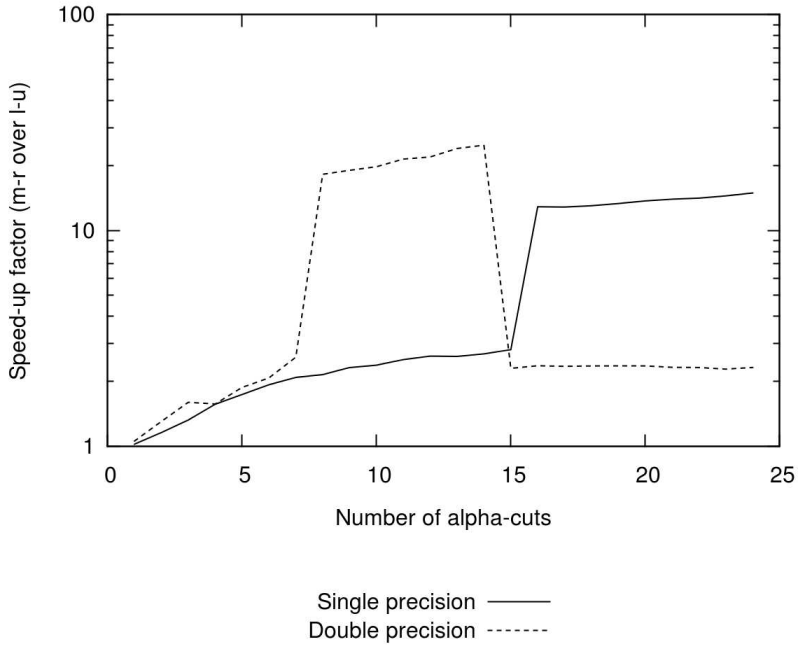
```
typedef mr_fuzzy<float,8> FUZZY;
void main(){
    thrust::device_vector<FUZZY> d_a(len);
    thrust::device_vector<unsigned int> keys(len);
    ...
    thrust::sort_by_key(keys.begin(), keys.end(), d_a.begin());
}
```

Thrust sort

```
typedef mr_fuzzy<float,8> FUZZY;
void main(){
    thrust::device_vector<FUZZY> d_a(len);
    thrust::device_vector<unsigned int> keys(len);
    ...
    thrust::sort_by_key(keys.begin(), keys.end(), d_a.begin());
}
```



Related performance



Algorithmic gain

Conclusion

- Different GPU features can be combined in order to accelerate fuzzy calculations.
- The representation format is relevant. Midpoint-radius fuzzy allows to perform 2x to 20x more interval operations per second than lower-upper fuzzy, depending on the application.
- Future work will consider numerical analysis and extending the library.
  - Accuracy of the midpoint-radius format.
  - Division, mathematical functions, type conversion, symmetric envelope.