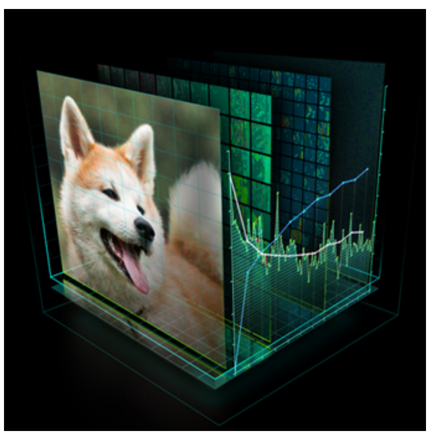
Introduction

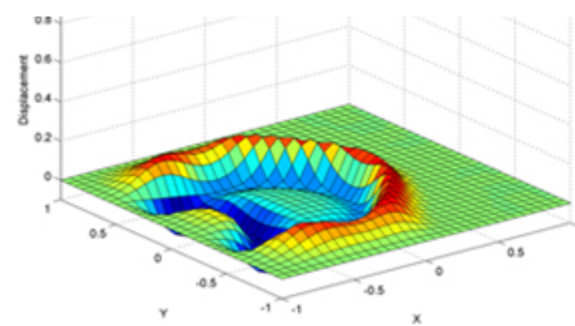
- Every new desktop/laptop is now equipped with a graphic processing unit (GPU).
- GPU = Massively Parallel Architecture.
- For most of their life, such GPUs are idle.

- General Purpose GPU applications:

Bioinformatics



Deep Learning



Numerical Analysis
MathWorks MATLAB

(Distributed) Constraint Optimization

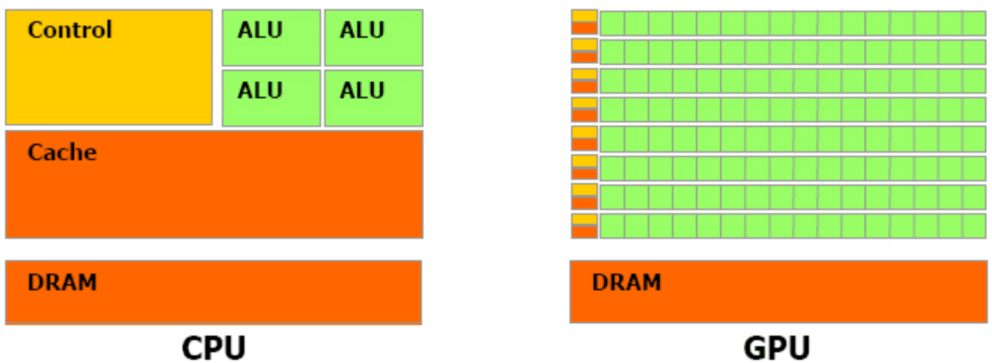
- A *(D)COP* is a tuple <***X***, ***D***, ***F***, (A, α)>, where:
  - ***X*** is a set of variables.
  - ***D*** is a set of finite domains.
  - ***F*** is a set of utility functions: $f_i : \times_{x_j \in scope(f_i)} D_j \mapsto \mathbb{N} \cup \{0, -\infty\}$
  - ***A*** is a set of agents, controlling the variables in ***X***.
  - α maps variables to agents.

  - GOAL: Find a utility maximal assignment.

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \mathbf{F}(\mathbf{x}) = \arg\max_{\mathbf{x}} \sum_{f \in \mathbf{F}} f(\mathbf{x}|_{\text{scope}(f)})$$

Graphical Processing Units (GPUs)

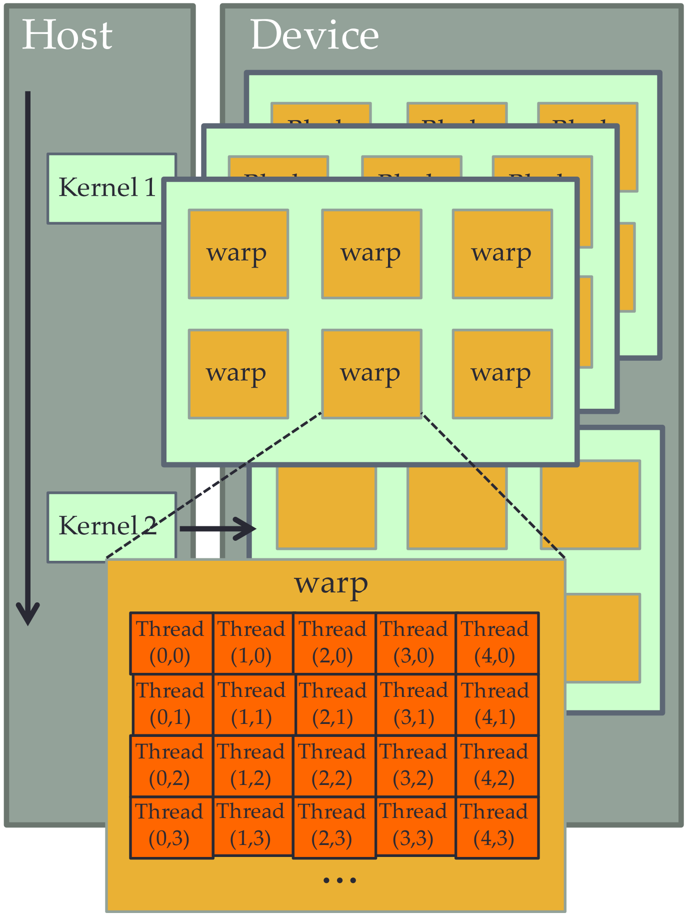
- A GPU is a massive parallel architecture:
  - Thousands of multi-threaded **computing cores**.
  - Very high **memory bandwidths**.
  - ~80% of transistors devoted to data processing rather than caching.



- **However:**
  - GPU cores are slower than CPU cores.
  - GPU memories have different sizes and access times.
  - GPU programming is more challenging and time consuming.

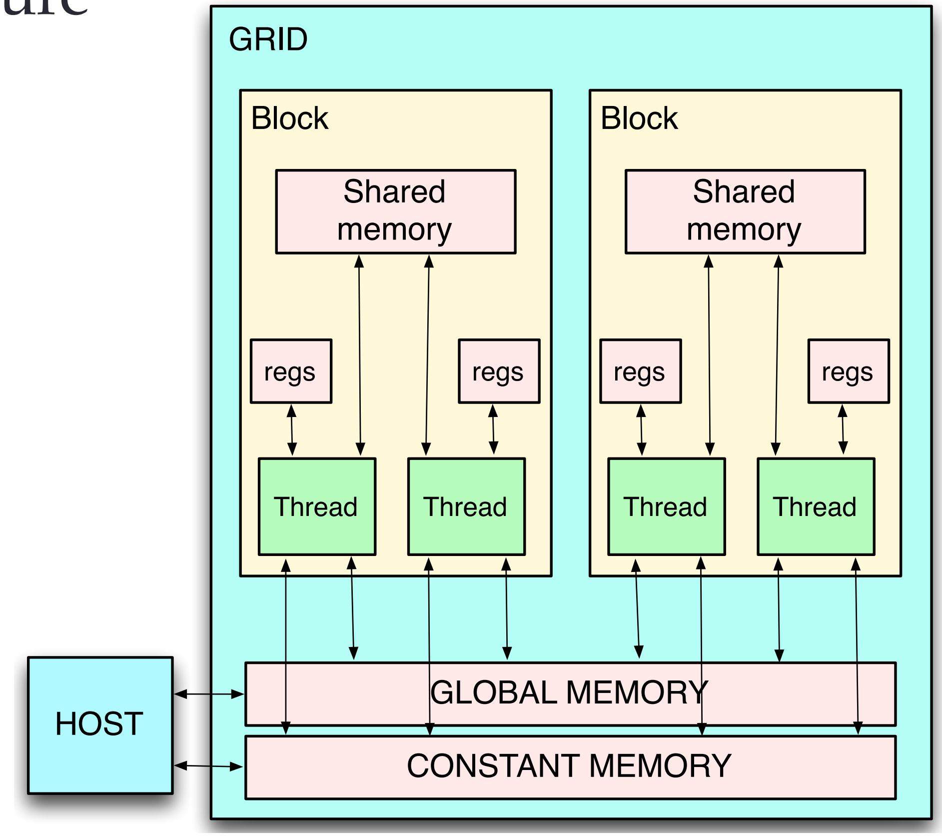
Execution Model

- A Thread is the **basic parallel unit**.
- **Threads** are organized into a Block.
- Several **warps** are scheduled for the execution of a GPU function.
- Several Streaming Multiprocessors, (SD) scheduled in parallel.
- Single Instruction Multiple Thread (SIMT) parallel model.



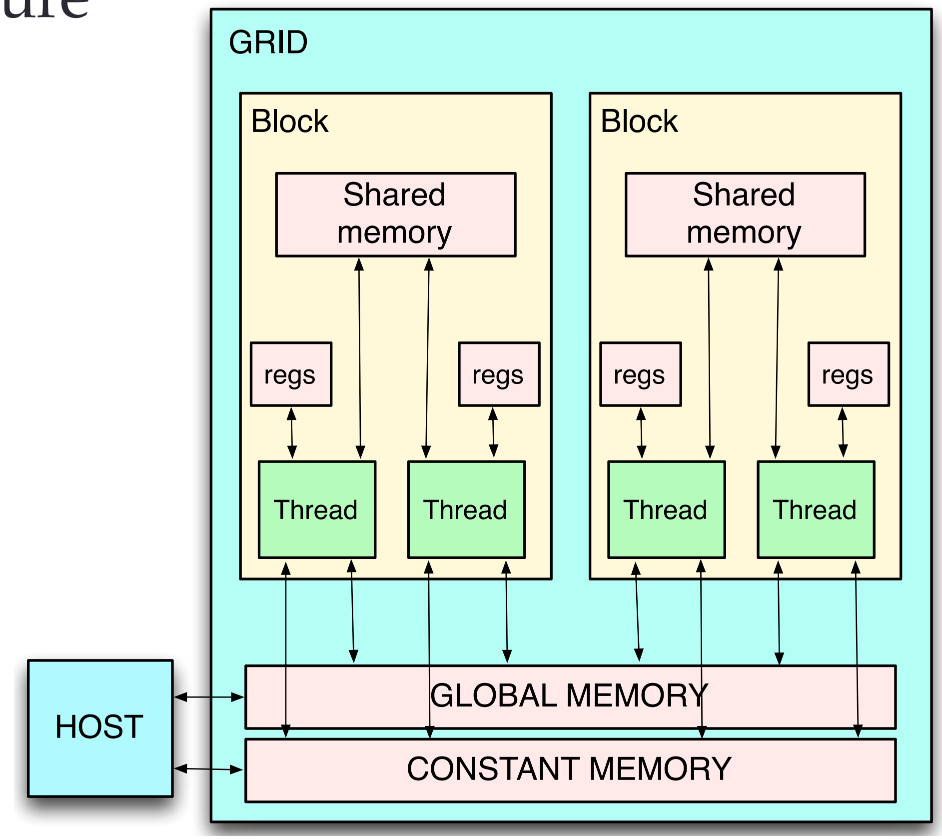
Memory Hierarchy

- The GPU memory architecture is rather involved.
- Registers
- Shared memory
- Global memory



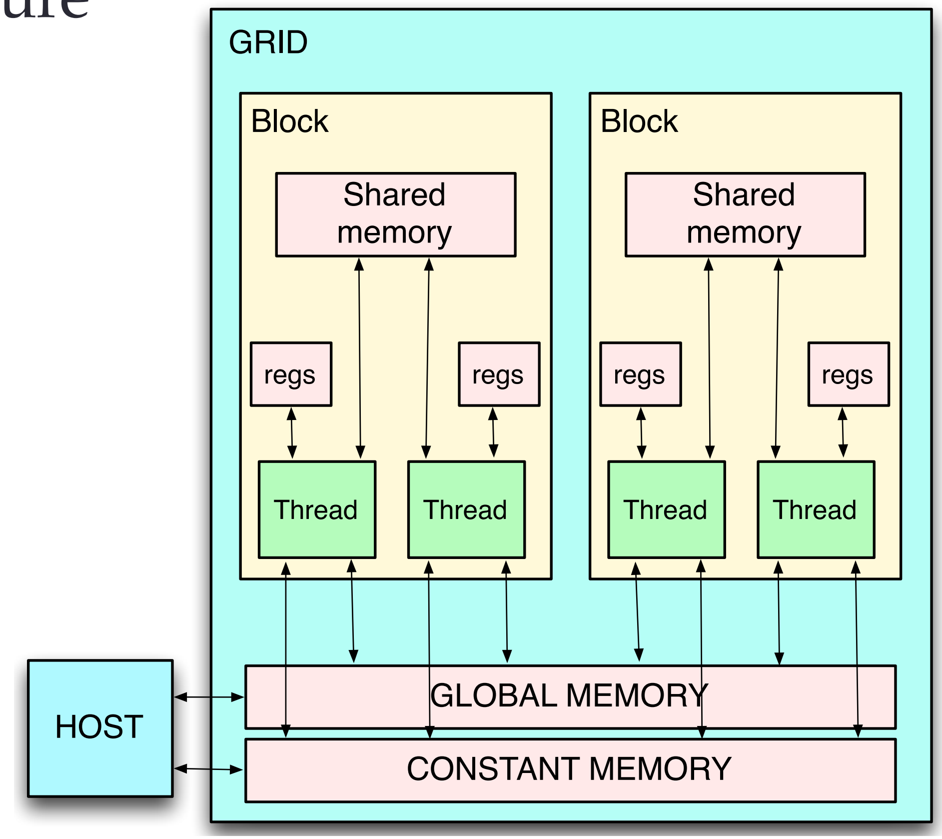
Memory Hierarchy

- The GPU memory architecture is rather involved.
- Registers
  - **Fastest;**
  - Only accessible by a **thread**;
  - Lifetime of a thread.
- Shared memory
- Global memory



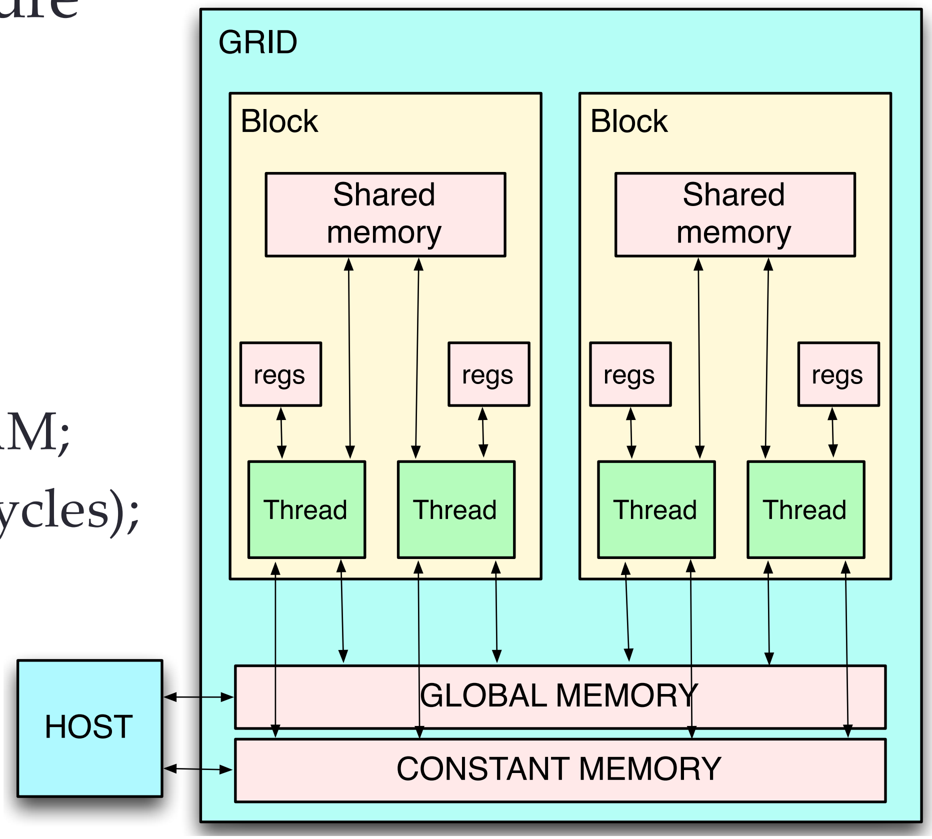
Memory Hierarchy

- The GPU memory architecture is rather involved.
- Registers
- Shared memory
  - **Extremely fast**;
  - Highly parallel;
  - Restricted to a **block**.
- Global memory



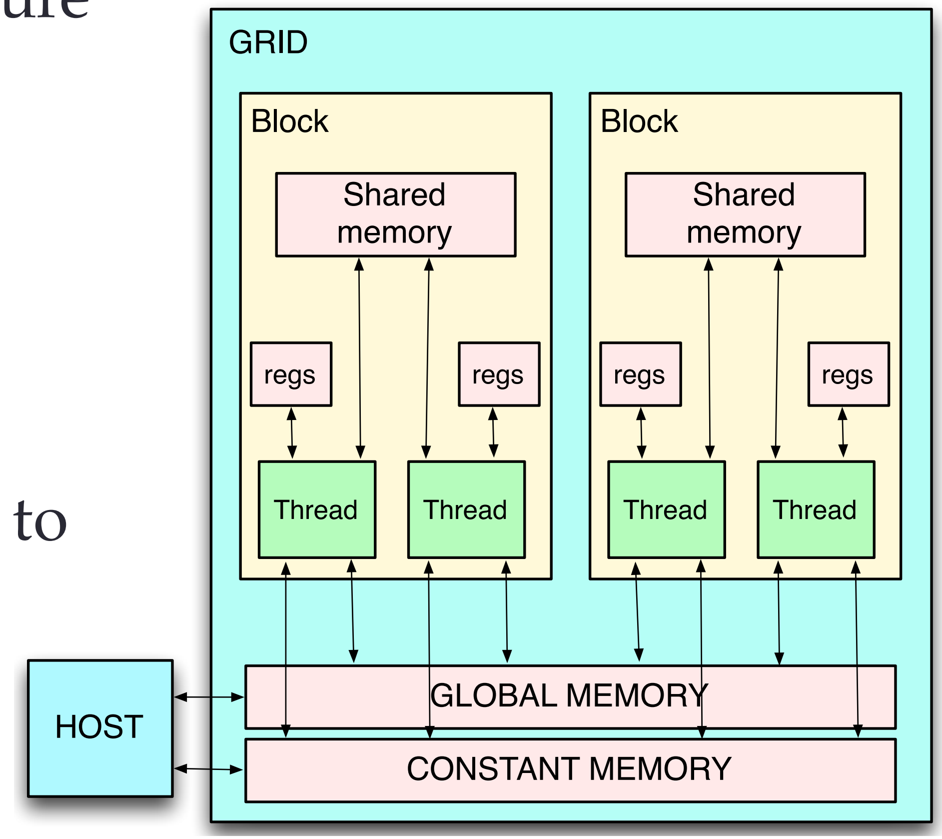
Memory Hierarchy

- The GPU memory architecture is rather involved.
- Registers
- Shared memory
- Global memory
  - Typically implemented in DRAM;
  - **High access latency** (400-800 cycles);
  - Potential of **traffic congestion**.



Memory Hierarchy

- The GPU memory architecture is rather involved.
- Registers
- Shared memory
- Global memory
- **Challenge**: using memory effectively -- likely requires to redesign the algorithm.



CUDA: Compute Unified Device Architecture

Host

Device

CUDA: Compute Unified Device Architecture

Host

Device

cudaMalloc(&deviceV, sizeV);

cudaMemcpy(deviceV, hostV, sizeV, ...)

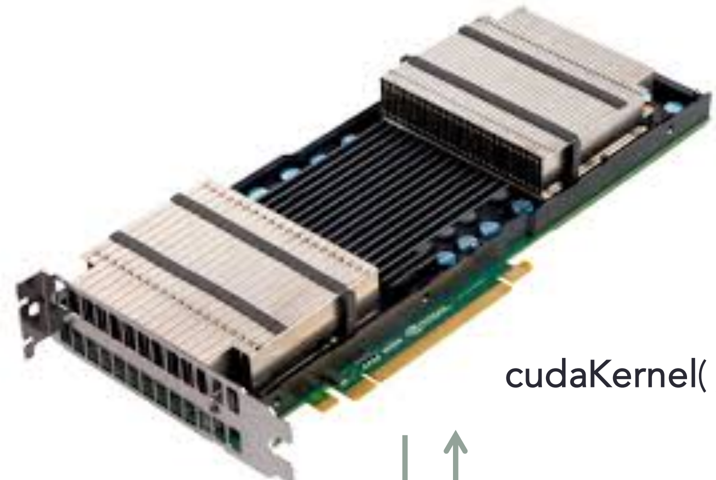
data →

Global Memory

CUDA: Compute Unified Device Architecture

Host

Device



cudaKernel( )

cudaKernel<nThreads, nBlocks>( )

Kernel invocation

Global Memory

CUDA: Compute Unified Device Architecture

Host

Device

cudaMemcpy(hostV, deviceV, sizeV, ...)

data

Global Memory

Bucket Elimination and DPOP

- Dynamic Programming procedures to solve (D)COPs.
- Both procedures rely on the use of two operators:
- **Projection Operator:** $\pi_{-xi}(f_{ij})$

x_i	x_j	U
0	0	5
0	1	8
1	0	20
1	1	3

f_{ij}

$\longrightarrow$

xj	U
0	20
1	8

- **Aggregation Operator:** $f_{ij} + f_{ik}$

Bucket Elimination and DPOP

- Dynamic Programming procedures to solve (D)COPs.
- Both procedures rely on the use of two operators:
- **Projection Operator:** $\pi_{-x_i}(f_{ij})$

x_i	x_j	U
0	0	5
0	1	8
1	0	20
1	1	3

f_{ij}

max(5, **20**)

xj	U
0	20
1	8

- **Aggregation Operator:** $f_{ij} + f_{ik}$

Bucket Elimination and DPOP

- Dynamic Programming procedures to solve (D)COPs.
- Both procedures rely on the use of two operators:
- **Projection Operator:** $\pi_{-xi}(f_{ij})$

x_i	x_j	U
0	0	5
0	1	8
1	0	20
1	1	3

f_{ij}

max(**8**, 3)

xj	U
0	20
1	8

- **Aggregation Operator:** $f_{ij} + f_{ik}$

Bucket Elimination and DPOP

- Dynamic Programming procedures to solve (D)COPs.
- Both procedures rely on the use of two operators:
- **Projection Operator:** $\pi_{-xi}(f_{ij})$
- **Aggregation Operator:** $f_{ij} + f_{ik}$

x_i	x_j	U
0	0	5
0	1	8
1	0	20
1	1	3

x_i	x_k	U
0	0	2
0	1	6
1	0	11
1	1	4

f_{ij}

x_i	x_j	x_k	U
0	0	0	7
0	0	1	11
0	1	0	10
0	1	1	14
	...		

...

Bucket Elimination and DPOP

- Dynamic Programming procedures to solve (D)COPs.
- Both procedures rely on the use of two operators:
- **Projection Operator:** $\pi_{-xi}(f_{ij})$
- **Aggregation Operator:** $f_{ij} + f_{ik}$

x_i	x_j	U
0	0	5
0	1	8
1	0	20
1	1	3

$5 + 2 = 7$

x_i	x_k	U
0	0	2
0	1	6
1	0	11
1	1	4

f_{ij}

x_i	x_j	x_k	U
0	0	0	7
0	0	1	11
0	1	0	10
0	1	1	14
	. . .		

. . .

Bucket Elimination and DPOP

- Dynamic Programming procedures to solve (D)COPs.
- Both procedures rely on the use of two operators:
- **Projection Operator:** $\pi_{-xi}(f_{ij})$
- **Aggregation Operator:** $f_{ij} + f_{ik}$

x_i	x_j	U
0	0	5
0	1	8
1	0	20
1	1	3

5 + 6 = **11**

x_i	x_k	U
0	0	2
0	1	6
1	0	11
1	1	4

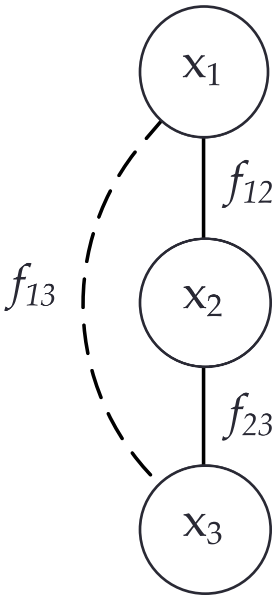
f_{ij}

x_i	x_j	x_k	U
0	0	0	7
0	0	1	11
0	1	0	10
0	1	1	14
	. . .		

. . .

Bucket Elimination and DPOP

1. Imposes an ordering on the problem's variables.

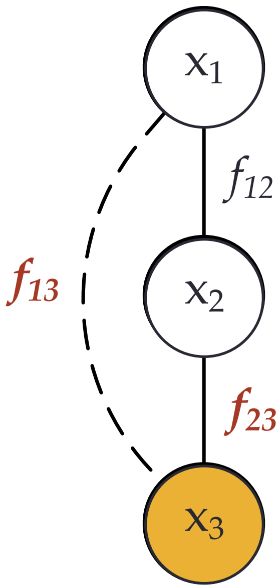


$\boldsymbol{X} = \{x_1, x_2, x_3\}$

$\boldsymbol{F} = \{f_{12}, f_{13}, f_{23}\}$

Bucket Elimination and DPOP

2. Selects the variable x_i with highest priority, and it creates a **bucket:**



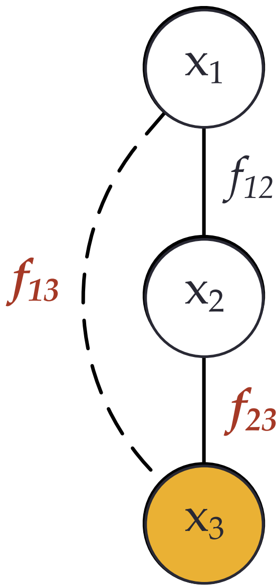
$$B_i = \Big\{ f_j \in \mathbf{F} | x_i \in scope(f_j) \wedge i = \max\{k | x_k \in scope(f_j)\} \Big\}$$

$\boldsymbol{X} = \{x_1, x_2, x_3\}$ $B_3 = \{f_{13}, f_{23}\}$

$\boldsymbol{F} = \{f_{12}, f_{13}, f_{23}\}$

Bucket Elimination and DPOP

3. It computes a new utility function f_i' by aggregating the functions in B_i and projecting out x_i



f_{13}

x_1	x_3	U
0	0	5
0	1	8
1	0	20
1	1	3

f_{23}

x_2	x_3	U
0	0	5
0	1	8
1	0	20
1	1	3

f_3'

x_1	x_2	U	(x_3)
0	0	**max**(5 + 5, 8 + 8) = **16**	1

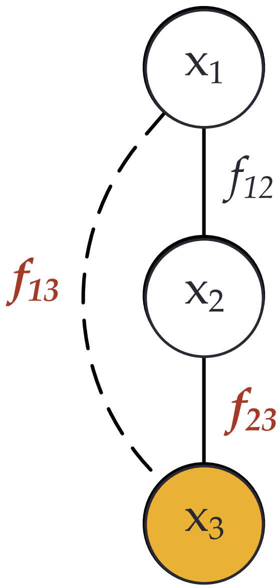
$$f_3' = \pi_{-x3}\,(f_{13} + f_{23})$$

$\boldsymbol{X} = \{x_1, x_2, x_3\}$ $\quad B_3 = \{f_{13}, f_{23}\}$

$\boldsymbol{F} = \{f_{12}, f_{13}, f_{23}\}$

Bucket Elimination and DPOP

3. It computes a new utility function f_i' by aggregating the functions in B_i and projecting out x_i



f_{13}

x_1	x_3	U
0	0	5
0	1	8
1	0	20
1	1	3

f_{23}

x_2	x_3	U
0	0	5
0	1	8
1	0	20
1	1	3

f_3'

x_1	x_2	U	(x_3)
0	0	16	1
0	1	**max**(5 + 20, 8 + 3) = **25**	0

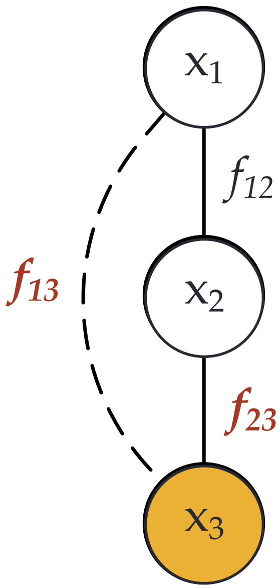
$$f_3' = \pi_{-x3}\,(f_{13} + f_{23})$$

$\boldsymbol{X} = \{x_1, x_2, x_3\}$ $\quad B_3 = \{f_{13}, f_{23}\}$

$\boldsymbol{F} = \{f_{12}, f_{13}, f_{23}\}$

Bucket Elimination and DPOP

3. It computes a new utility function f_i' by aggregating the functions in B_i and projecting out x_i



f_{13}

x1	x3	U
0	0	5
0	1	8
1	0	20
1	1	3

f_{23}

x2	x3	U
0	0	5
0	1	8
1	0	20
1	1	3

f_3'

x1	x2	U	(x3)
0	0	16	1
0	1	25	0
1	0	**max**(20 + 5, 3 + 8) = **25**	0

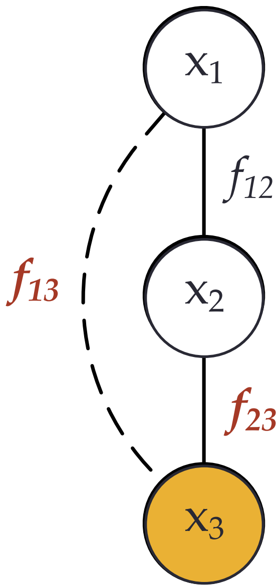
$$f_3' = \pi_{-x3}\,(f_{13} + f_{23})$$

$\boldsymbol{X} = \{x_1, x_2, x_3\}$ $\quad B_3 = \{f_{13}, f_{23}\}$

$\boldsymbol{F} = \{f_{12}, f_{13}, f_{23}\}$

Bucket Elimination and DPOP

3. It computes a new utility function f_i' by aggregating the functions in B_i and projecting out x_i



f_{13}

x1	x3	U
0	0	5
0	1	8
1	0	20
1	1	3

f_{23}

x2	x3	U
0	0	5
0	1	8
1	0	20
1	1	3

f_3'

x1	x2	U	(x3)
0	0	16	1
0	1	25	0
1	0	25	0
1	0	**max**(20 + 20, 3 + 3) = **40**	0

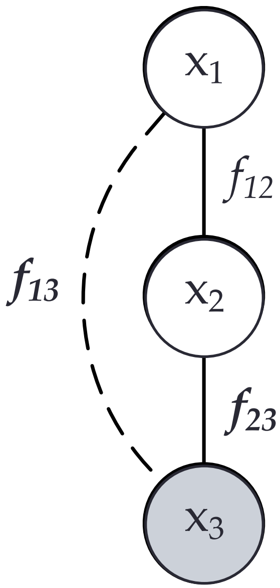
$$f_3' = \pi_{-x3}(f_{13} + f_{23})$$

$\boldsymbol{X} = \{x_1, x_2, x_3\}$ $B_3 = \{f_{13}, f_{23}\}$

$\boldsymbol{F} = \{f_{12}, f_{13}, f_{23}\}$

Bucket Elimination and DPOP

4. It updates the set of variables: $\mathbf{X} \leftarrow \mathbf{X} \setminus \{x_i\}$
5. It updates the set of functions: $\mathbf{F} \leftarrow (\mathbf{F} \cup \{f_i'\}) \setminus B_i$

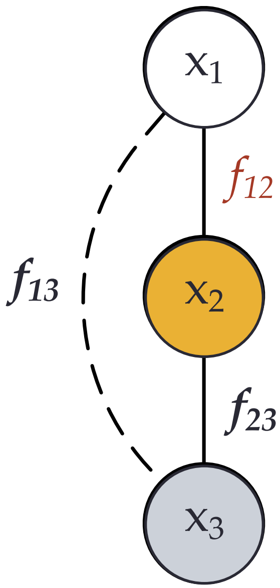


$\boldsymbol{X} = \{x_1, x_2\}$ $\quad$ $B_2 = \{f_{13}, f_3'\}$

$\boldsymbol{F} = \{f_{12}, f_3'\}$

Bucket Elimination and DPOP

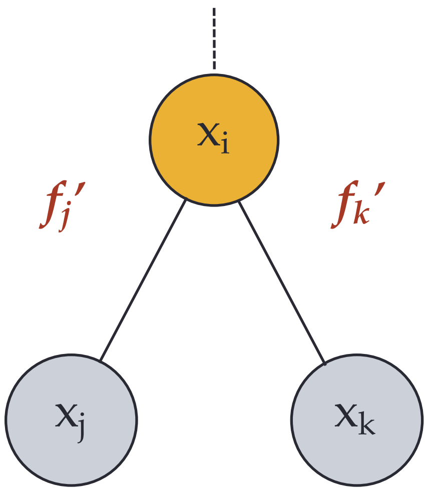
Repeat...



$X = \{x_1, x_2\}$

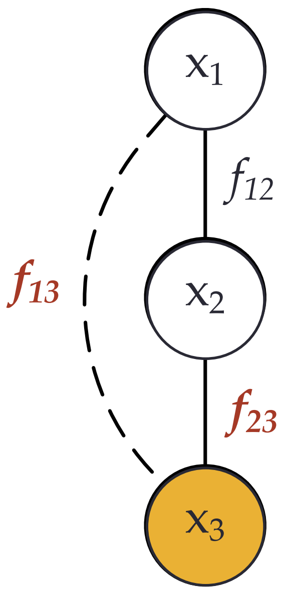
$F = \{f_{12}, f_3'\}$

- **DPOP** is a distributed version of BE.
- It operates on a Pseudotree ordering of the constraint graph.



GPU-(D)BE

- BE and DPOP complexity: $O(d^{w^*})$.
  d = max. domain size; w* = induced width of the constraint graph.
- Can the projection and aggregator operators be executed in parallel?
- Do they fit the SIMT parallel model?



x_1	x_3	U
0	0	5
0	1	8
1	0	20
1	1	3

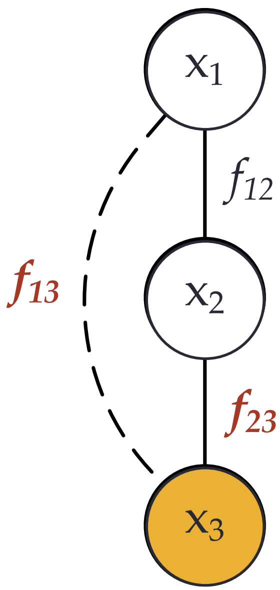
x_2	x_3	U
0	0	5
0	1	8
1	0	20
1	1	3

x_1	x_2	U
0	0	**max**(5 + 5, 8 + 8) = **16**
0	1	**max**(5 + 20, 8 + 3) = **25**
1	0	**max**(20 + 5, 3 + 8) = **25**
1	0	**max**(20 + 20, 3 + 3) = **40**

$$f_3' = \pi_{-x3}\,(f_{13} + f_{23})$$

GPU-(D)BE

- BE and DPOP complexity: $O(d^{w^*})$.
  d = max. domain size; w* = induced width of the constraint graph.
- Can the projection and aggregator operators be executed in parallel?
- Do they fit the SIMT parallel model?
- **Obs.**: The computation of each row of the Utility tables is independent from the computation of other rows.



x_1	x_3	U
0	0	5
0	1	8
1	0	20
1	1	3

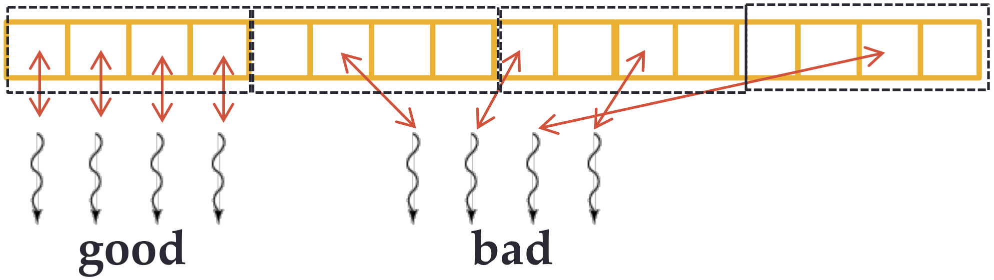
x_2	x_3	U
0	0	5
0	1	8
1	0	20
1	1	3

x_1	x_2	U
0	0	**max**(5 + 5, 8 + 8) = **16**
0	1	**max**(5 + 20, 8 + 3) = **25**
1	0	**max**(20 + 5, 3 + 8) = **25**
1	0	**max**(20 + 20, 3 + 3) = **40**

$$f_3' = \pi_{-x3}(f_{13} + f_{23})$$

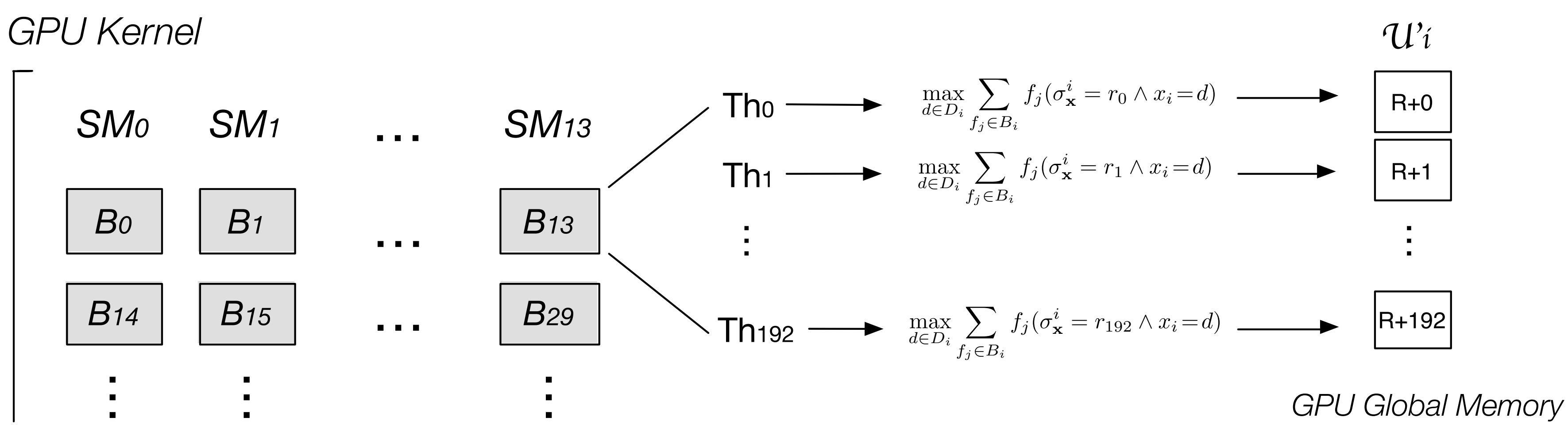
Algorithm design and data structure

- Limit the amount of host-device **data transfers**.
  - Static Entities: require a **single** data transaction.
    - Variables; Domains; Utility functions; Constraint Graph.
  - Dynamic Entities: might require **multiple** data transactions.
    - Utility tables.
- Minimize the accesses to the **global memory**.
  - Padding Utility Tables' rows; Perfect hashing.
- Ensure data accesses are **coalesced**.
  - Mono-dimensional array organization;



Parallel Projection and Aggregation

- Mapping between the f_i' table rows and the *CUDA* blocks:
  - Each **thread** in a block is associated to the computations of one permutation of values in *scope*(f_i').
  - 1 **block** = $64k$ threads ($1 \leq k \leq 16$).
    - k depends on the architecture and it is chosen so to maximize the number of threads that can be scheduled concurrently.
- **Obs.**: Max number of parallel f_i' table rows is $M = |SM| 64k$
  - In our experiments, $|SMs| = 14$ and $k = 3$. Thus $\mathbf{M = 2688}$.

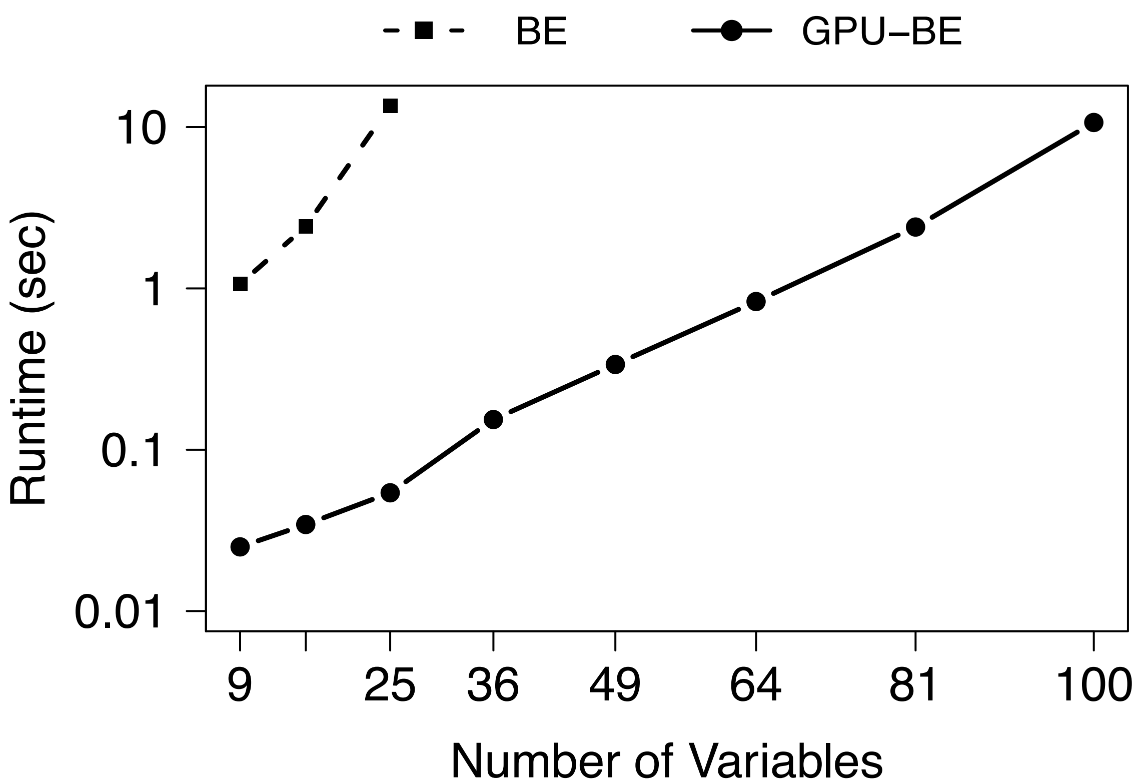


Experimental Results

- |Di| = 5;
- p2 = 0.5;
- timeout = 300s

Regular Grid Networks

- CPU: 2.3GHz, 128 GB RAM
- GPU: 14 SMs, 837MHz.



Speedup

avg. max.: 125.1x

avg. min.: 42.6x

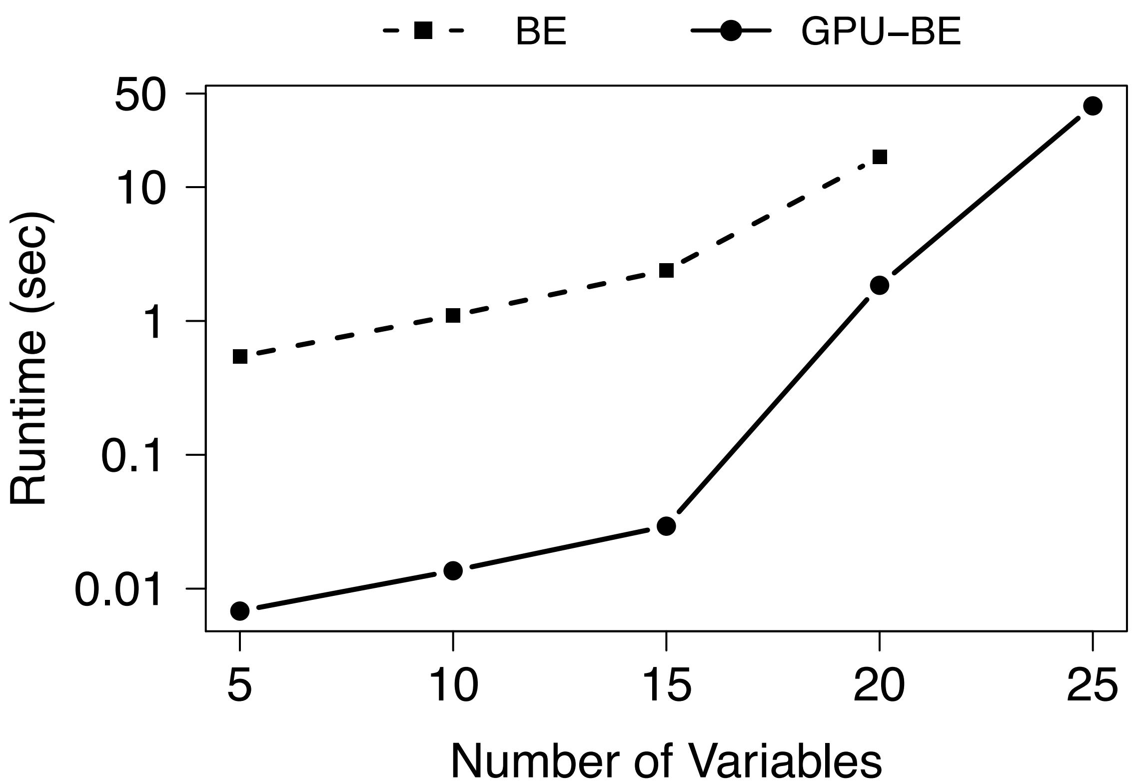
- Similar trends at increasing $|Di|$.
- Similar trends for DPOP vs GPU-DBE

Experimental Results

- |Di| = 5;
- p2 = 0.5; p1 = 0.3;
- timeout = 300s

Random Networks

- CPU: 2.3GHz, 128 GB RAM
- GPU: 14 SMs, 837MHz.



Speedup

avg. max.: 69.3.x

avg. min.: 16.1x

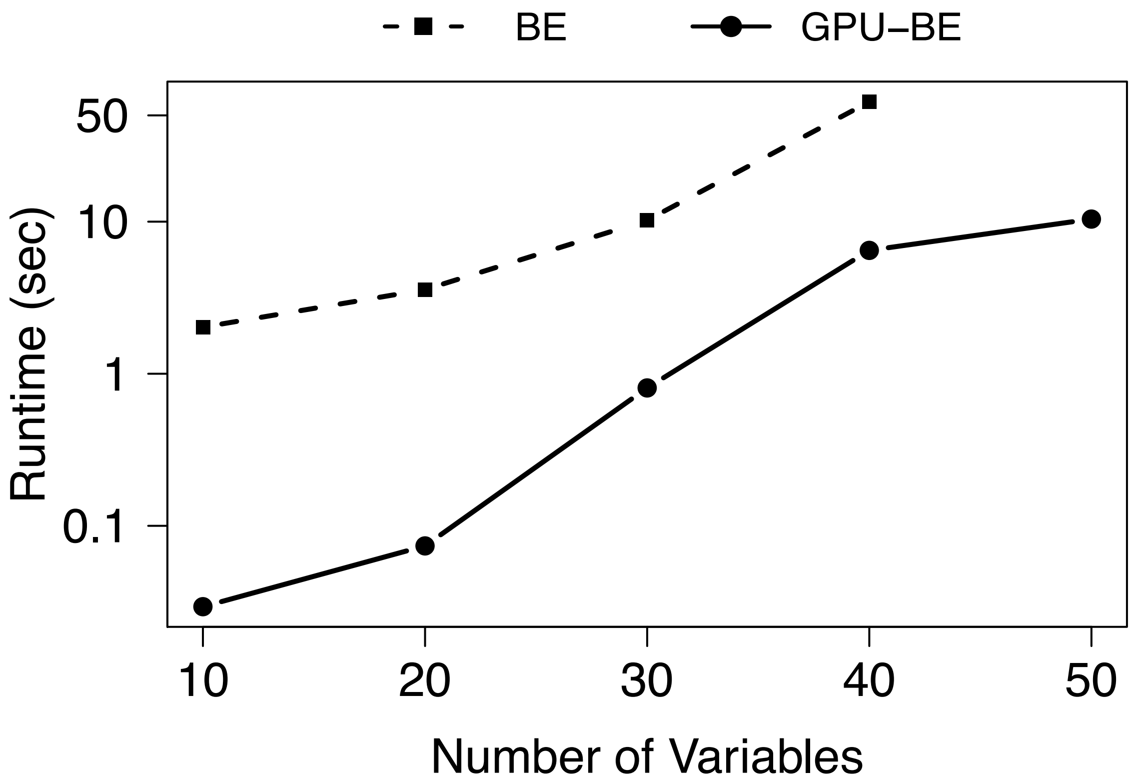
- Similar trends at increasing |*Di*|.
- Similar trends for DPOP vs GPU-DBE

Experimental Results

- |Di| = 5;
- p2 = 0.5;
- timeout = 300s

Scale Free Networks

- CPU: 2.3GHz, 128 GB RAM
- GPU: 14 SMs, 837MHz.



Speedup

avg. max.: 34.9.x

avg. min.: 9.5x

- Similar trends at increasing $|Di|$.
- Similar trends for DPOP vs GPU-DBE

Lesson Learned #1

- The f_i' table size increases exponentially with w^*.
- Limited GPU global memory (2GB).
- f_i' table + B_i tables, to be used in the aggregation operation, might exceed global memory capacity!
- Partition f_i' computations in multiple chunks.

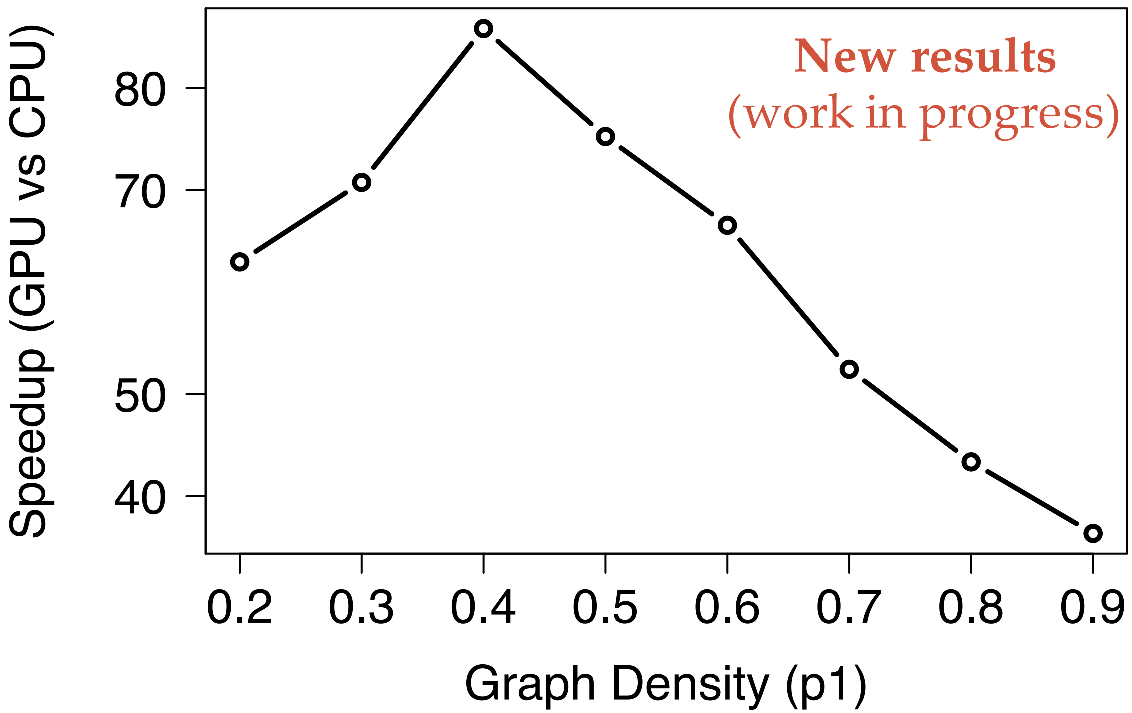
- Alternates GPU and CPU to compute f_i'.
  - GPU: Aggregates the functions in B_i excluding those which do not fit in the global memory.
  - CPU: Aggregates the other functions in B_i;
    Projects out the variable x_i.

Experimental Results

- |A| = 10; |Di| = 5;
- p2 = 0.5;
- timeout = 300s

Random Networks

- CPU: 2.3GHz, 128 GB RAM
- GPU: 14 SMs, 837MHz.



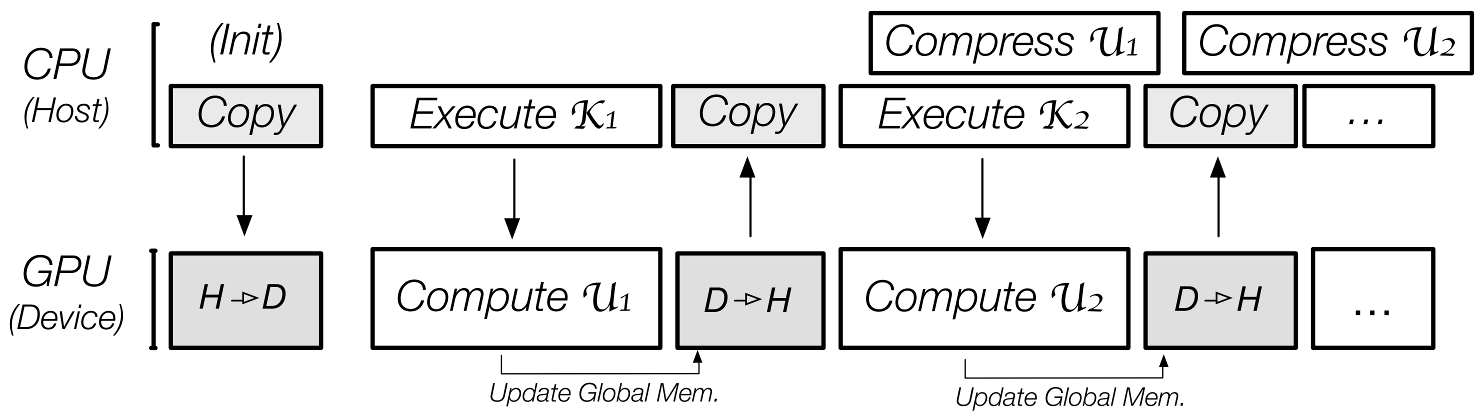
Phase Transition

$p_1 = 0.4$

- small p_1 correspond to smaller w*

Lesson Learned #2

- Host and Device **concurrency**.
- Possible when the f_i' tables are computed in **chunks**.
- It may hide host-device data transfers as byproduct.



Discussion

- Exploiting the integration of CPU and GPU is a key factor to obtain competitive solver performance.

- How to determine good tradeoffs of such integration?
- **GPU**:
  - Repeated, non memory intensive operations;
  - Operations requiring regular memory access;
- **CPU:**
  - Memory intensive operations;

Conclusions

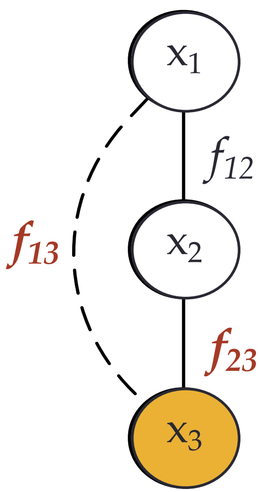
- Exploit GPU-style parallelism from DP-based (D)COPs resolution methods.
- GPU-(D)BE: Exploits GPUs to parallelizes the **aggregation** and **projection** operators.
- Observed different speedup, ranging from 34.9 to 125.1, based on several network topologies.
- Discussed several possible optimization techniques.

- **FUTURE WORK**:
  - Exploit GPUs in DP-based propagators.
  - Investigate GPUs in higher form of consistency.

Exploiting GPUs in Solving (Distributed) Constraint Optimization Problems with Dynamic Programming

F. Fioretto, T. Le, E. Pontelli, W. Yeoh, T. Son

Thank You!



Ferdinando Fioretto
New Mexico State University,
University of Udine
Email: ffiorett@cs.nmsu.edu
Web: www.cs.nmsu.edu/~ffiorett