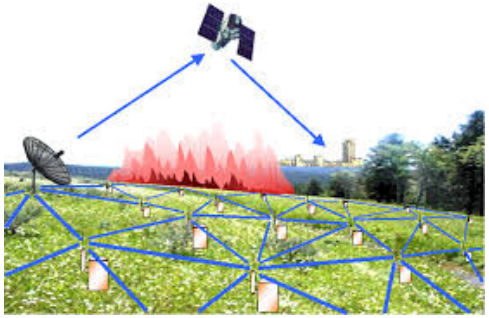
A Dynamic Programming-based MCMC Framework for Solving DCOPs with GPUs

Ferdinando Fioretto[1(2)] (joint work with)
William Yeoh[2] and Enrico Pontelli[2]

[1] University of Michigan
[2] New Mexico State University

CP 2016, Toulouse

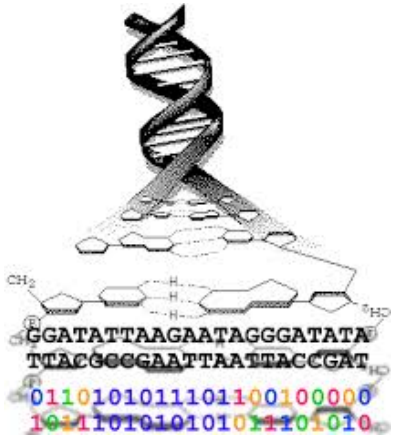
Distributed Discrete Optimization with Preferences



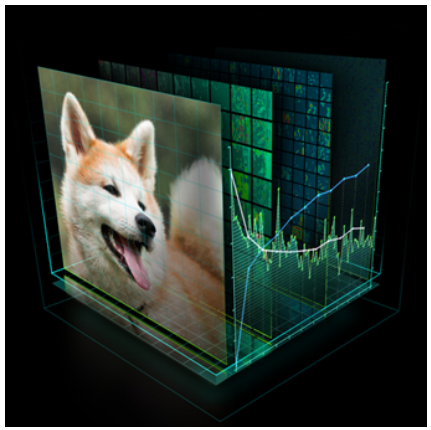
GPUs

- 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.

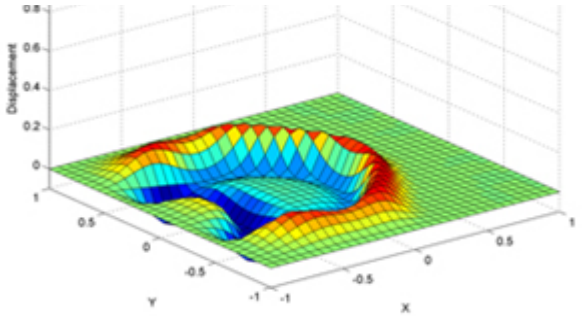
- General Purpose GPU applications:



Bioinformatics



Deep Learning



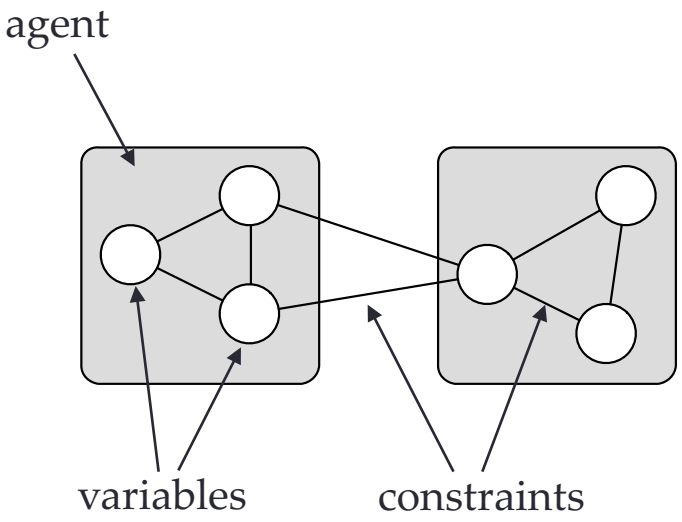
Numerical Analysis
MathWorks MATLAB

Outline

- Introduction
- GPUs
- D-MCMC
- Results
- Conclusions

Multi-Agent Constraint Optimization

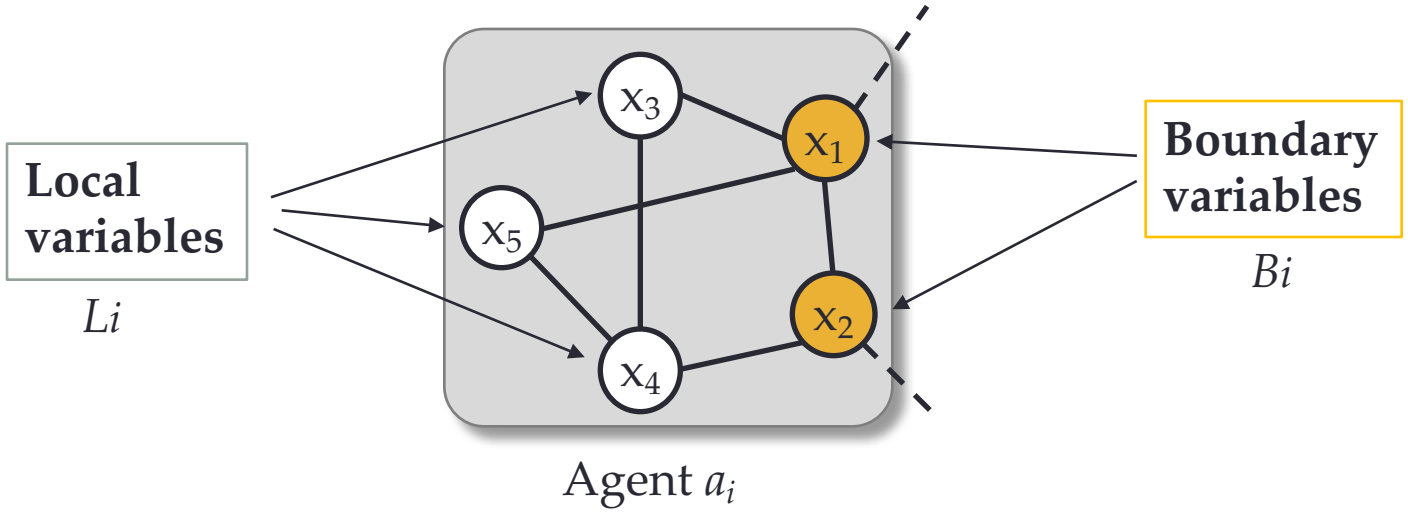
- A *DCOP* is a tuple <***X***, ***D***, ***F***, A, α>, where:
  - ***X*** is a set of variables.
  - ***D*** is a set of finite domains for each variable.
  - ***F*** is a set of constraints between variables.
  - ***A*** is a set of agents, controlling the variables in ***X***.
  - α is a mapping from variables to agents.



x_a	x_b	U
0	0	3
0	1	20
1	0	2
1	1	5

Multi-Agent Constraint Optimization

- A *DCOP* is a tuple <***X***, ***D***, ***F***, A, α>, where:
  - ***X*** is a set of variables.
  - ***D*** is a set of finite domains for each variable.
  - ***F*** is a set of constraints between variables.
  - ***A*** is a set of agents, controlling the variables in ***X***.
  - α is a mapping from variables to agents.



Multi-Agent Constraint Optimization

- A *DCOP* is a tuple <***X***, ***D***, ***F***, A, α>, where:
  - ***X*** is a set of variables.
  - ***D*** is a set of finite domains for each variable.
  - ***F*** is a set of constraints between variables.
  - ***A*** is a set of agents, controlling the variables in ***X***.
  - α is a mapping from variables to agents.

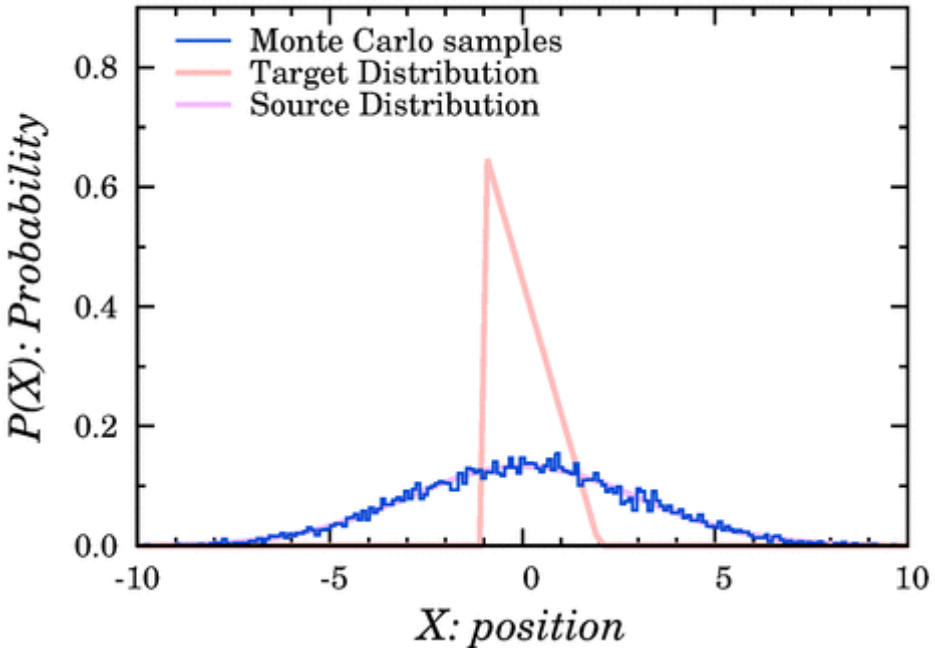
- GOAL: Find a utility maximal assignment.

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \mathbf{F}(\mathbf{x})$$

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

MCMC Sampling

- MCMC algorithms approximate probability distributions.
- They use a proposal distribution to generate a sequence of samples $z^{(1)}$, $z^{(2)}$, … which forms a Marokv Chain.
- The quality of the sample improves as a function of the number of steps.



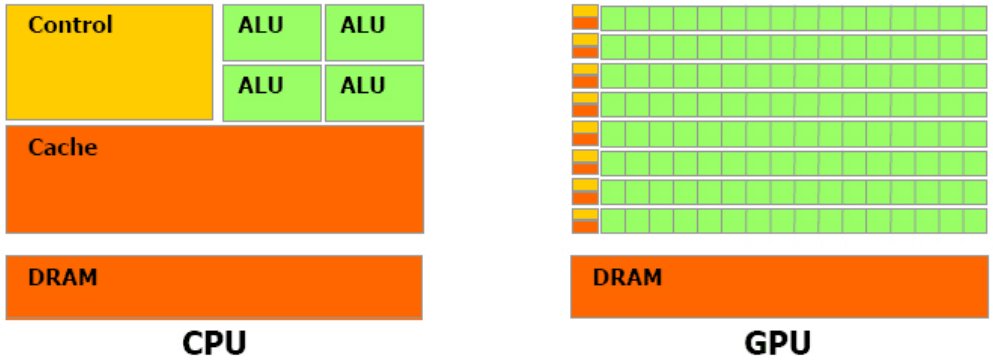
Source: http://xr0038.hatenadiary.jp/

MCMC Sampling

- MCMC sampling algorithms can be used to solve DCOPs. [Nguyen et al., AAMAS 2013]
- MCMC Sampling algorithms can be used to solve the Maximum A Posteriori (MAP) estimation problem.
- The authors provide a mapping from solving a DCOP to solving a MAP.

Graphical Processing Units (GPUs)

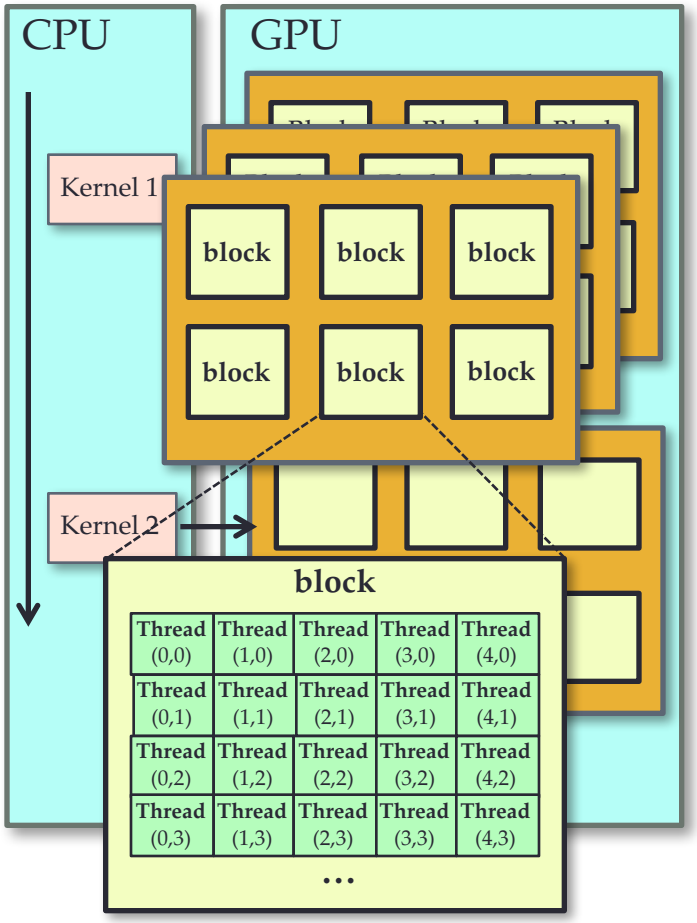
- A GPU is a massive parallel architecture:
  - Thousands of multi-threaded **computing cores**.
  - Very high **memory bandwidths**.
  - ~80% of transistors devoted to data processing rather than caching.



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

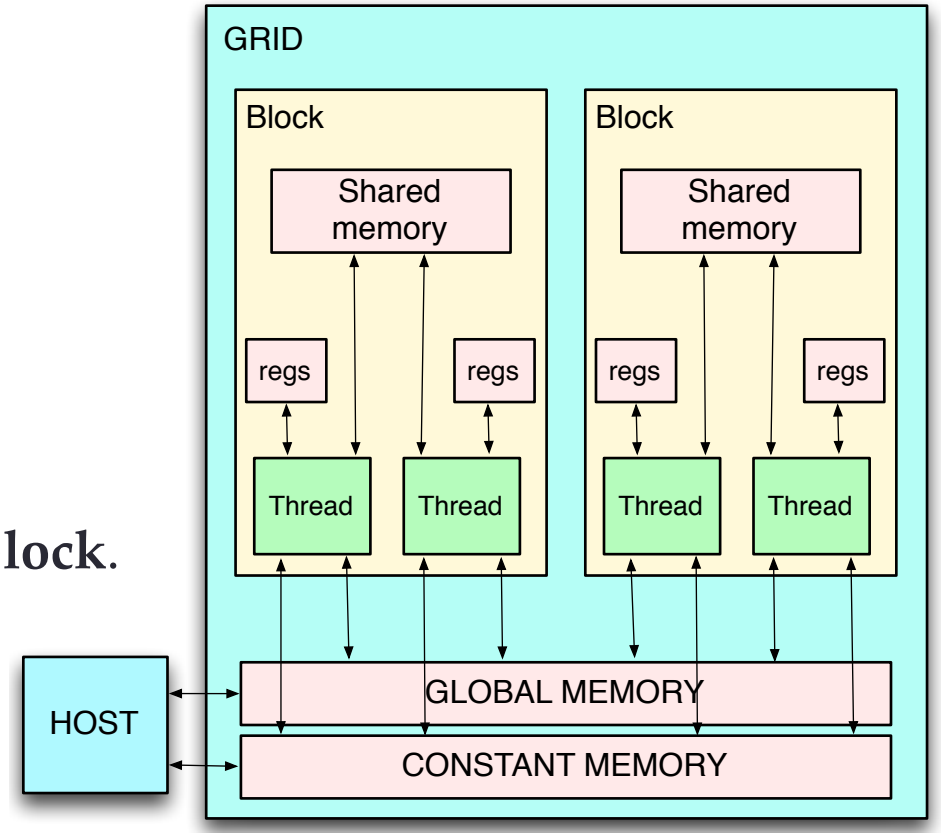
Execution Model

- A **Thread** is the basic parallel unit.
- Identified by a Thread ID.
- **Threads** are organized into **Blocks**.
- Several **Streaming Multiprocessors**, (SD) scheduled in parallel.
- Single Instruction Multiple Thread (SIMT) parallel model.



Memory Hierarchy

- The GPU memory architecture is rather involved.
- Registers
  - **Fastest**.
  - Only accessible by a **thread**.
  - Lifetime of a thread.
- Shared memory
  - Fast.
  - Accessible by all threads in a **block**.
- Global memory
  - **High access latency**
  - Potential of **traffic congestion**.



CUDA: Compute Unified Device Architecture

Host

Device

CUDA: Compute Unified Device Architecture

Host

Device

cudaMalloc(&deviceV, sizeV);

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

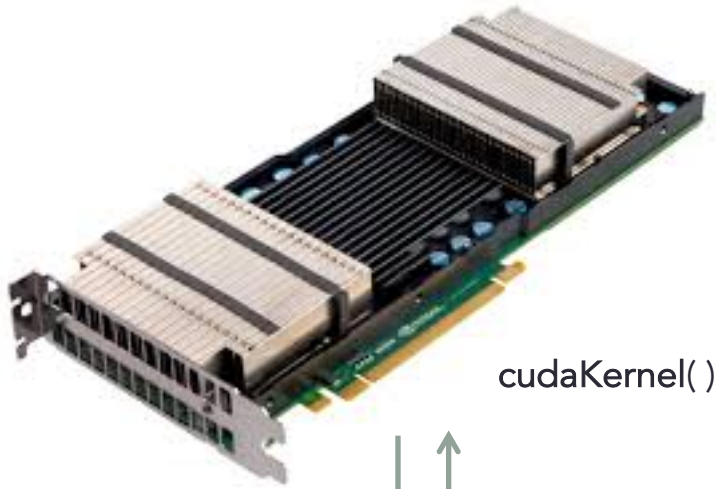
data →

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

D-MCMC: Related Work

[Nguyen et al. AAMAS-2013]

D-Gibbs Sampling

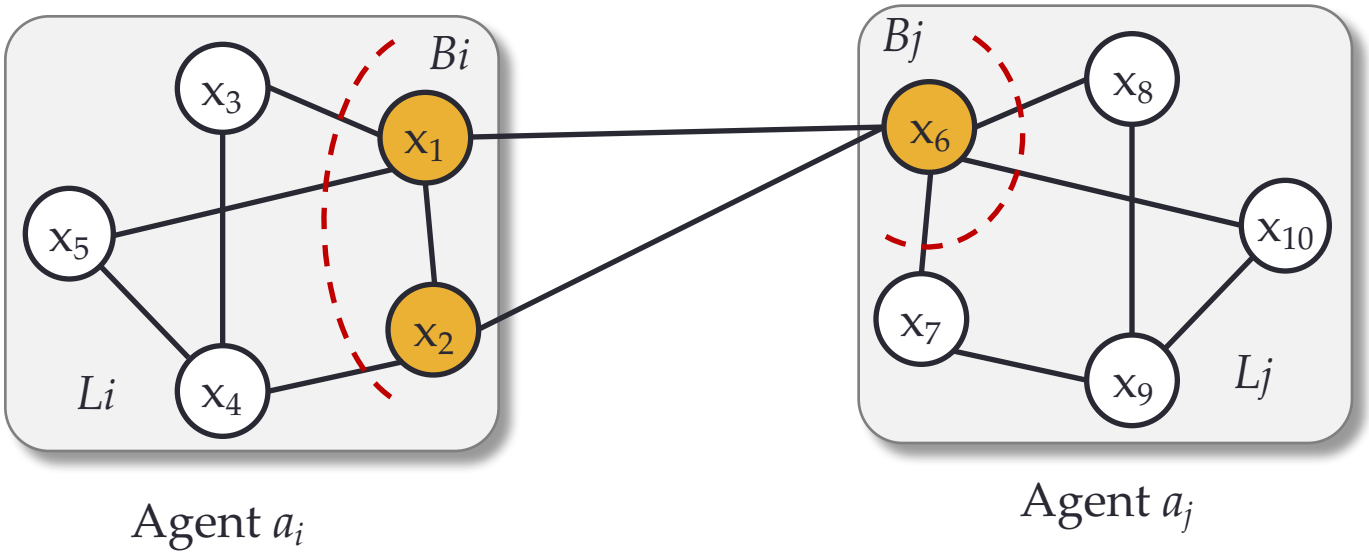
Algorithm 1: GIBBS$(z_1, \ldots, z_n)$

for $i = 1$ **to** n **do**
  $z_i^0 \leftarrow$ INITIALIZE(z_i)
end
for $t = 1$ **to** S **do**
  for $i = 1$ **to** n **do**
    $z_i^t \leftarrow$ SAMPLE$(P(z_i \mid z_1^t, \ldots, z_{i-1}^t, z_{i+1}^{t-1}, \ldots, z_n^{t-1}))$
  end
end

- Computing the normalizing constant can be expensive.
- A lots of sample to converge.

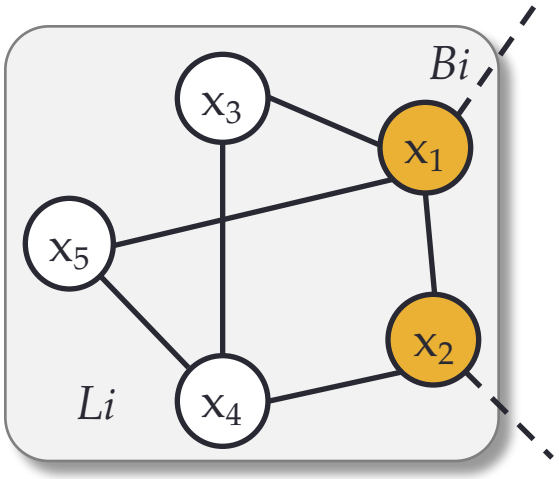
DMCMC

- Each agent controls several variables.
- Given values for its boundary variables each agent can solve its local sub-problem independently from other agents.



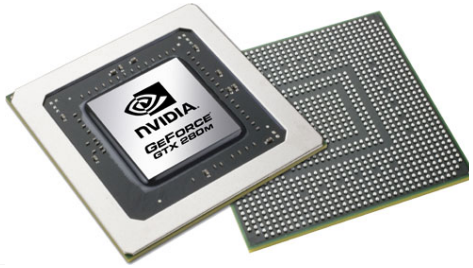
DMCMC

- Each agent controls several variables.
- Given values for its boundary variables.
- Find a solution for the local sub-problem using MCMC algorithms: Gibbs sampling and Metropolis–Hastings.



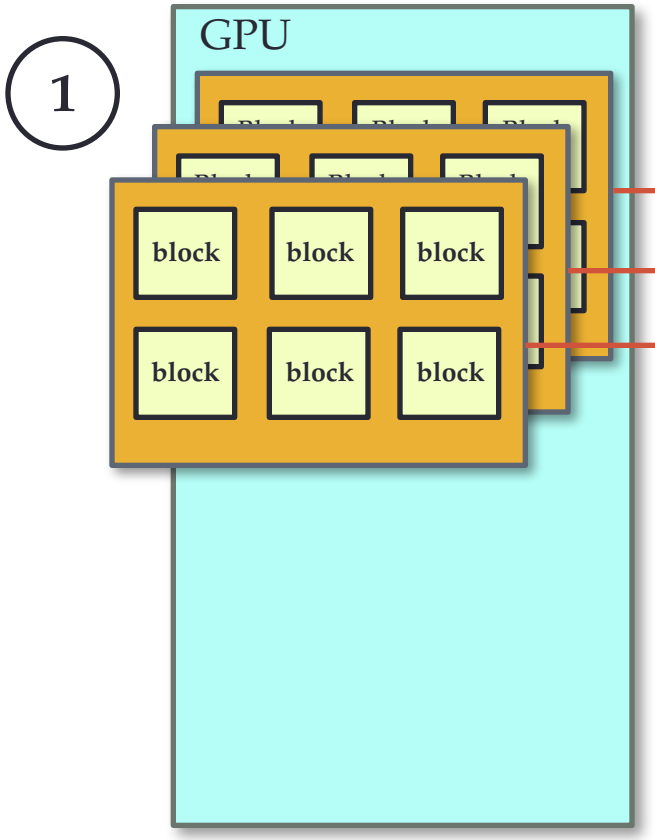
Joint utility table

Bi		*Li*			
x_1	x_2	x_3	x_4	x_5	U
0	0	3	2	5	21
0	1	2	1	4	20
0	2	3	5	1	32



DMCMC: Local Sampling Process

3 Level of Parallelism



Joint utility table

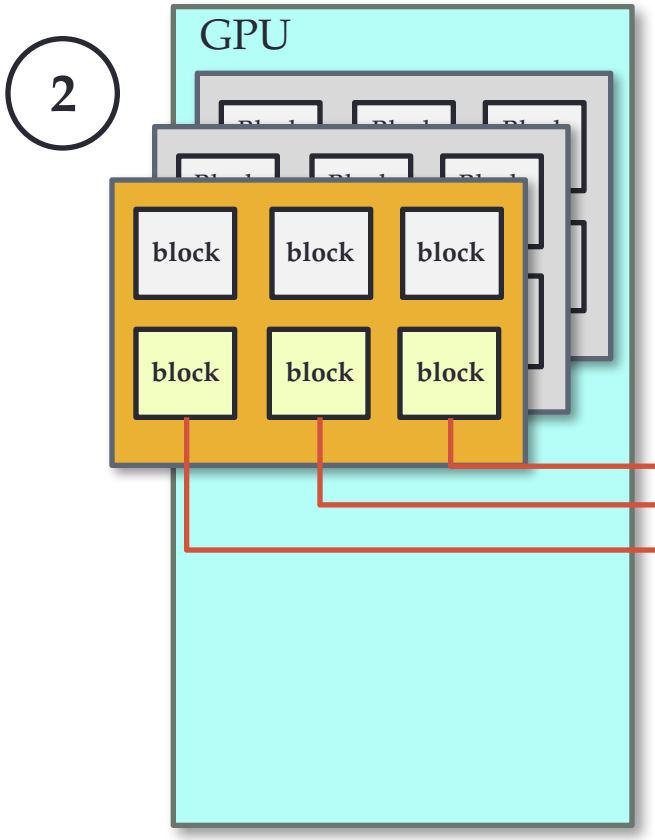
x_1	x_2	x_3	x_4	x_5	U
0	0	3	2	5	21
0	1	2	1	4	20
0	2	3	5	1	32

Each row of the Joint utility table is computed in parallel using several blocks.

[Fioretto et al. CP-15]

DMCMC: Local Sampling Process

3 Level of Parallelism



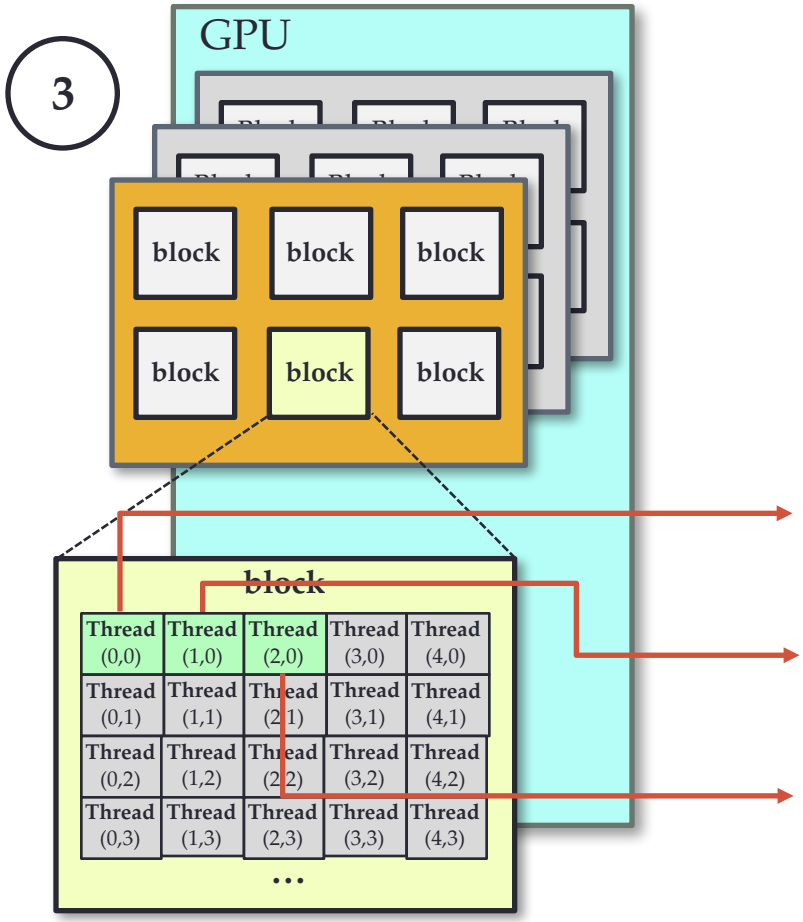
R multiple samples

Joint utility table

x_1	x_2	x_3	x_4	x_5	U
0	0	3	2	5	21
0	0	2	1	4	20
0	0	3	5	1	32
0	1	2	1	4	20
0	1	3	5	1	32
0	1	2	1	4	20
0	2	3	5	1	32
		…			

DMCMC: Local Sampling Process

3 Level of Parallelism



Gibbs Sampling Process

$$q(x_k = 0 \mid x_l \in L_i \setminus \{x_k\}) = \frac{1}{Z_\pi} \exp \sum_{f_j \in \mathcal{F}_i} f_j(\mathbf{z}_{|\mathbf{x}^{f_j}})$$

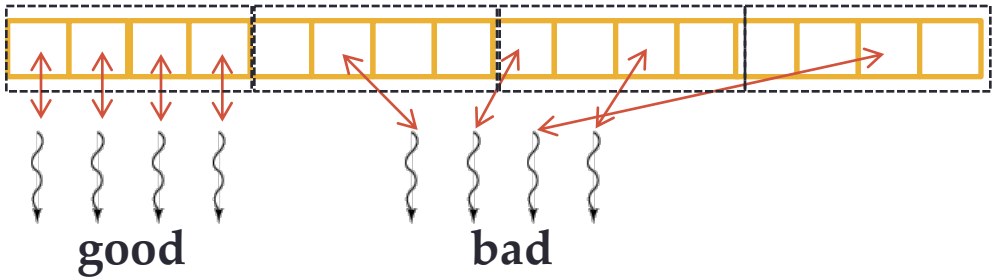
$$q(x_k = 1 \mid x_l \in L_i \setminus \{x_k\}) = \frac{1}{Z_\pi} \exp \sum_{f_j \in \mathcal{F}_i} f_j(\mathbf{z}_{|\mathbf{x}^{f_j}})$$

$$q(x_k = 2 \mid x_l \in L_i \setminus \{x_k\}) = \frac{1}{Z_\pi} \exp \sum_{f_j \in \mathcal{F}_i} f_j(\mathbf{z}_{|\mathbf{x}^{f_j}})$$

...

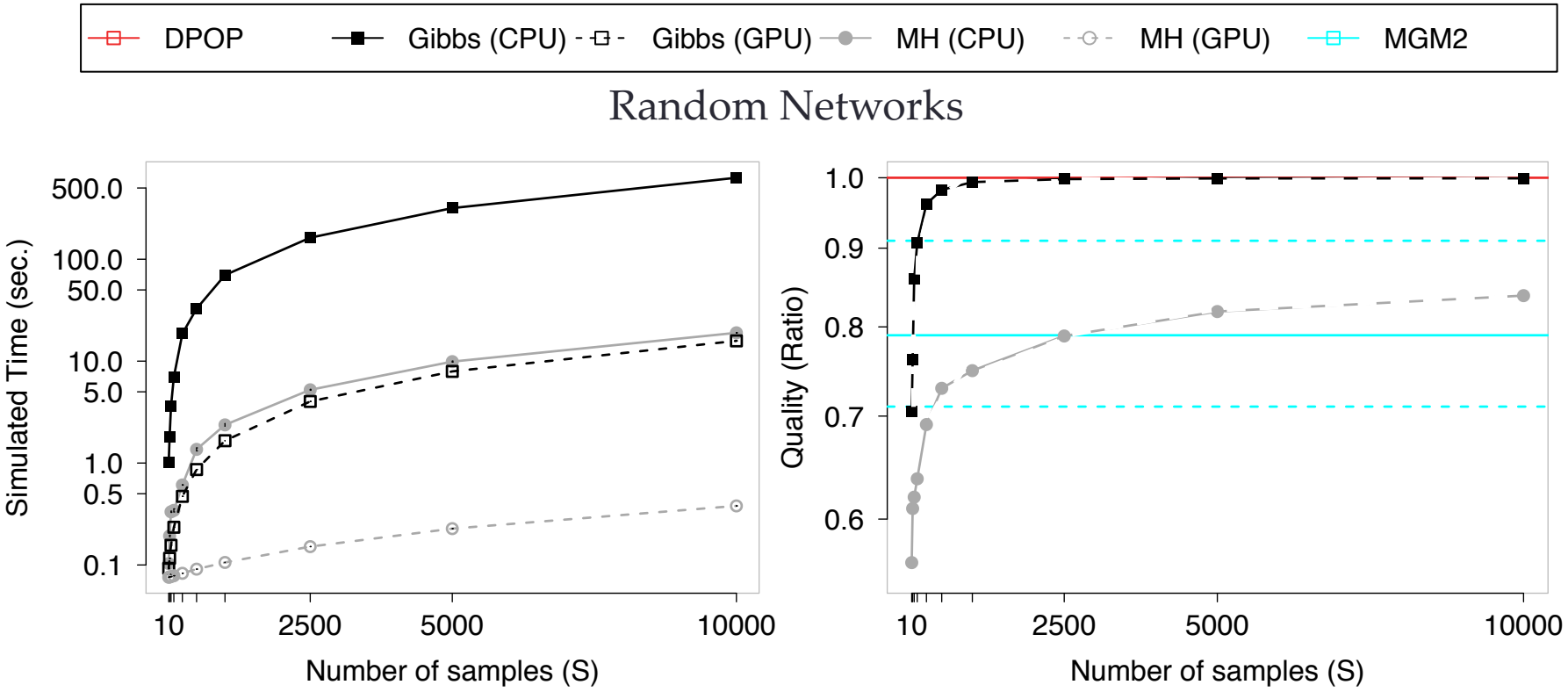
Algorithm design and data structure

- Ensure data accesses are **coalesced**.



- Minimize the accesses to the **global memory**.
  - Padding Utility Tables' rows; Perfect hashing.

Results



Main results:

- **Runtime**: GPU-MCMC algorithms are > 1 order of magnitude faster than CPU-MCMC ones.
- **Quality**: GPU-Gibbs dominates MGM2 for S>100; GPU-MH solutions quality comparable to those of MGM2

Results

Meeting Scheduling

S = 100; R = 10

$\|\mathcal{A}\|$	5			10			25			50		
	wct	*st*	*quality*	*wct*	*st*	*quality*	*wct*	*st*	*quality*	*wct*	*st*	*quality*
DPOP	125.39	94.98	**1661**	*oot*	*oot*	-	*oot*	*oot*	-	*oot*	*oot*	-
MGM	7.435	0.435	1379	11.910	0.446	2766	24.211	0.417	6692	45.771	0.462	13802
MGM2	8.939	0.979	1389	23.903	1.526	2783	56.035	1.629	7116	112.54	1.788	14145
Gibbs_{CPU}	6.146	1.101	1638	12.093	1.190	3.319	31.031	1.347	**8344**	62.411	1.489	**16577**
Gibbs_{GPU}	0.162	0.033	1635	0.301	0.034	**3338**	0.708	0.041	**8344**	1.416	0.048	16550
MH_{CPU}	0.561	0.113	1131	1.091	0.121	2775	2.281	0.176	6921	3.921	0.185	12112
MH_{GPU}	**0.047**	**0.014**	1143	**0.102**	**0.016**	2663	**0.196**	**0.017**	6925	**0.360**	**0.022**	11856

Main results:

- Gibbs on GPU is up to 2 order of magnitude faster than MGM(2) and finds solutions of higher quality.

Conclusions

- Exploit GPU-style parallelism from DP-based DCOPs resolution methods and MCMC sampling.
- **D-MCMC framework**: Decomposes a DCOP into independent sub-problems that can be sampled in parallel with GPUs.
- D-MCMC with Gibbs produces high quality solutions with runtimes up to 2 order of magnitude faster than other state-of-the art incomplete solvers.

- **Future Work:**
  - Exploit similar techniques to solve WCSPs.
  - Extend the proposed method using memory bounded solutions.

Thank you!

References

- [1] D. T. Nguyen, W. Yeoh, and H. C. Lau, "Distributed Gibbs: A Memory-Bounded Sampling-Based DCOP Algorithm, AAMAS, 2013.
- [2] F. Fioretto, T. Le, E. Pontelli, W. Yeoh, and T. Son, "Exploiting GPUs in Solving (Distributed) Constraint Optimization Problems with Dynamic Programming", CP, 2015.