A GPU Implementation of the ASP Computation

A. Dovier[1] A. Formisano[2] **E. Pontelli**[3] F. Vella[4]

1. Università di Udine
2. Università di Perugia
3. **New Mexico State University**
4. Sapienza Università Roma, CNR, NVIDIA

PADL-2016 — St. Petersburg, FL, USA, January 2016

General Purpose GPU

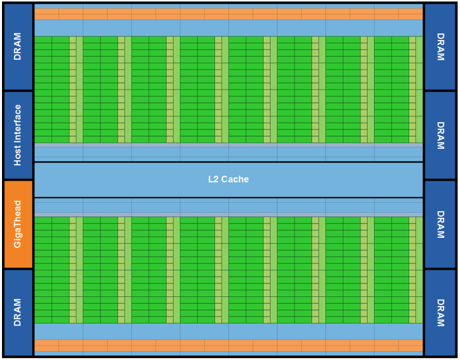
- Graphic Processing Units (GPUs) are parallel processor originally conceived for graphic processing
- In the last years GPUs evolved towards a more flexible architecture
- This enables the use of GPUs for general purpose programming:

  GPU-computing

- GPUs offer great efficiency and high performance (if carefully programmed...)

How it looks like...

Under the hood — The architectural scheme

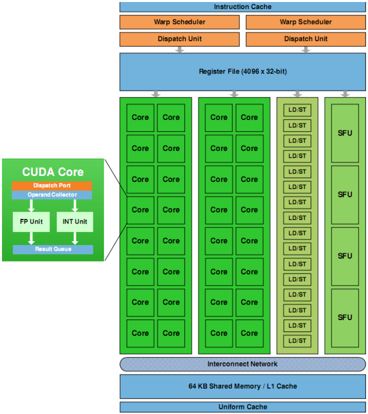


- Fermi's 16 SM are positioned around a common L2 cache.
- Each SM is a vertical rectangular strip that contains
  - an orange portion (scheduler and dispatch),
  - a green portion (execution units),
  - light blue portions (register file and L1 cache).

Zoom in: A streaming multiprocessor

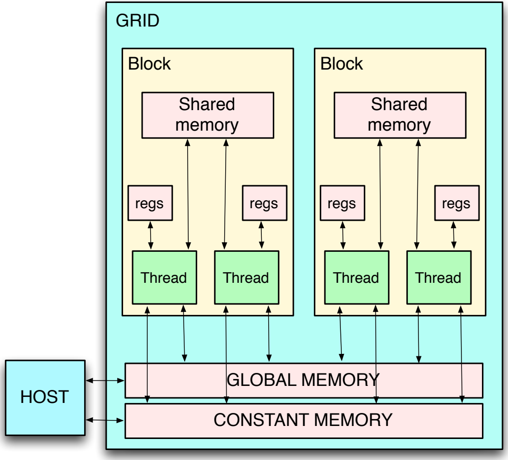
Each SM includes:

- 32 CUDA cores
- Fully pipelined Int and FP ALU
- 16 Load/Store Units (16 threads per clock)
- 4 Special Function Units
- Registers, cache...



Execution model and memory hierarchy (CUDA-style)

- Each core executes a thread
  - registers
  - local memory
- Block: a group of threads
  - shared memory
  - synchronization support
  - 3d grid (e.g., $1K \times 1K \times 64$)
- Grid: a group of blocks
  - global memory
  - 3d grid (e.g., $64K \times 64K \times 64K$)
  - constant, texture mem.
- Warp: 32 threads
  - works in lock-step SIMT parallelism



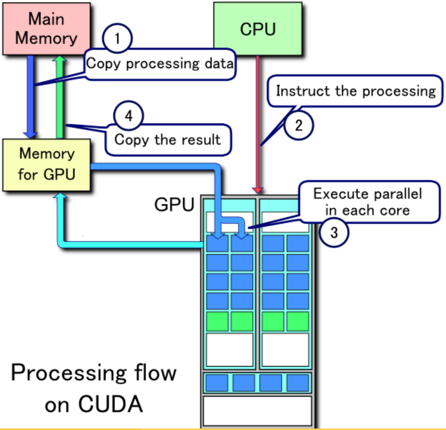
Execution model (CUDA-style)

The computation can proceed on the host and on the device

- The programmer writes a kernel that will be run on the device
- Each thread executes an instance of the kernel

The host instructs the device:

1. copy data, host⇒device
2. kernel call
3. kernel execution on GPU
4. retrieve results, host⇐device



GPUs for ASP?

The idea: to design an ASP-solver that

- exploits GPUs and the CUDA framework
  $\Rightarrow$ massive parallelism mostly for deterministic components of the computation
- adopts a "nogood-driven" approach
  $\Rightarrow$ SAT/ASP technology, heuristics, learning,...
- relies on ASP-computations
  $\Rightarrow$ focus on completion nogoods

Inspired by successes in CUD@SAT

ASP programs

An ASP program Π is composed of rules of the form

$$r: \quad p \leftarrow a_1, \ldots, a_m, not\ b_{m+1}, \ldots, not\ b_n$$

$$\leftarrow a_1, \ldots, a_m, not\ b_{m+1}, \ldots, not\ b_n$$

- p and $\{a_1, \ldots, a_m, not\ b_{m+1}, \ldots, not\ b_n\}$ are denoted by $head(r)$ and $body(r)$, resp.
- $\{a_1, \ldots, a_m\}$ is denoted by $body^+(r)$
- $\{b_{m+1}, \ldots, b_n\}$ is denoted by $body^-(r)$

- Semantics ASP program Π is given in terms of answer sets
- A set M of atoms is an answer set for Π if it is the least Herbrand model of the reduct Π^M

ASP-computation for a program Π

It is a sequence of sets of atoms $I_0 = \emptyset, I_1, I_2, \ldots$ such that

- $I_i \subseteq I_{i+1}$ for all $i \geq 0$ (Persistence of Beliefs)
- $I_\infty = \bigcup_{i=0}^{\infty} I_i$ is such that $T_\Pi(I_\infty) = I_\infty$ (Convergence)
- $I_{i+1} \subseteq T_\Pi(I_i)$ for all $i \geq 0$ (Revision)
- if $p \in I_{i+1} \setminus I_i$ then there is a rule $p \leftarrow body$ in Π such that $I_j \models body$ for each $j \geq i$ (Persistence of Reason)

M is an answer set of Π iff there exists an ASP-computation s.t. converges to M, namely, $M = \bigcup_{i=0}^{\infty} I_i$

L. Liu, E. Pontelli, T. Son, M. Truszczynski: Logic programs with abstract constraint atoms: The role of computations. Art. Int. 174(3-4):295-315 (2010)

Completion and completion-nogoods

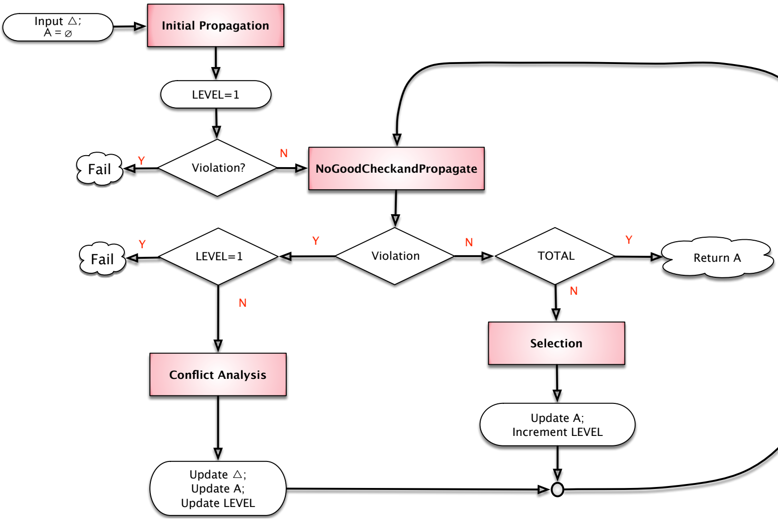
Given a program Π, its completion Π_{cc} is defined as:

$$\Pi_{cc} = \begin{array}{l} \left\{\beta_r \leftrightarrow \bigwedge_{a \in body^+(r)} a \wedge \bigwedge_{b \in body^-(r)} \neg b \mid r \in \Pi\right\} \cup \\ \left\{p \leftrightarrow \bigvee_{r \in body_\Pi(p)} \beta_r \mid p \in atom(\Pi)\right\} \end{array}$$

Π_{cc} can be "compiled" into a collection $\Delta_{\Pi_{cc}}$ of nogoods of the forms:

- $\{not\ \beta_r\} \cup \{a \mid a \in body^+(r)\} \cup \{not\ b \mid b \in body^-(r)\}$
- $\{\beta_r, not\ a\}$ for each $a \in body^+(r)$ and $\{\beta_r, b\}$ for each $b \in body^-(r)$
- $\{not\ p, \beta_r\}$ for each $r \in body_\Pi(p)$, for each head p in Π
- $\{p\} \cup \{not\ \beta_r \mid r \in body_\Pi(p)\}$, for each head p in Π

Ingredients for a nogood-driven solver



Ingredients for a nogood-driven solver

- Assigned atom: Tp or Fp
- (Partial) Assignment: consistent set of assigned atoms
- Nogood: consistent set of assigned atoms

Ingredients for a nogood-driven solver

- Preprocessing: parses the input; computes the completion nogoods, dependency graph, statistics for heuristics; data transfer to the device, ...
- Selection: performs a step in an ASP-computation, to select next branching atom (decision step)
- Propagation: propagates the consequences of decision steps (specific kernels for short nogoods, long nogoods, ...)
- Nogood-Check: looks for violations of nogoods
- Conflict-Analysis: in case of conflict, learns new nogoods
- Backjumping: in case a conflicting partial assignment is reached, updates the device data structures consequently

Blue tasks run on the device. The host performs I/O, some preprocessing, data transfers to/from the device

Basic schema of the CUDA application

```
current_dl := 1; A := ∅                                        ▷ Initial decision level and assignment
(A, Violation) := InitialPropagation(A, Δ)
if (Violation is true) then return no answer set
else
    loop
        (Δ_A, Violation) := NoGoodCheckAndPropagate(A, Δ)      ▷ Conflict(s) detection
        A := A ∪ Δ_A;
        if (Violation is true) ∧ (current_dl = 1) then return no answer set
        else if (Violation is true) then
            (current_dl, δ) = ConflictAnalysis(Δ, A)           ▷ Learning (possibly multiple) and
            Δ := Δ ∪ {δ};  A := A \ {p̄ ∈ A | current_dl < dl(p̄)}                     ▷ backjump
        end if
        if (A is not total) then
            (p̄, OneSel) := Selection(Δ, A)                     ▷ Step in ASP-computation
            if (OneSel is true) then  current_dl++;  dl(p̄) := current_dl;  A := A ∪ {p̄}
            else  A := A ∪ {Fp : p is unassigned}
            end if
        else  return A^T ∩ atom(Π)
        end if
    end loop
end if
```

Some Ideas on How to Develop the Kernels

CPU

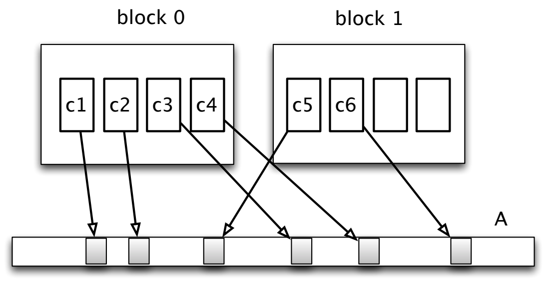
- CPU computes Δ and dependency graph
- Transfers Δ to GPU

Some Ideas on How to Develop the Kernels

InitialPropagation

- Process all unary nogoods in Δ
- One thread per unitary nogood
  - $\lceil \frac{\#UnitaryNogoods}{TPB} \rceil$ blocks
  - Each thread assigns A[p] to the opposite sign as the unitary nogood

InitialPropagation



NoGoodCheckAndPropagate

Problem

Given a partial model A and nogood δ

- Check if δ violated by A
- Check if $\delta \setminus A = \{X\}$

NoGoodCheckAndPropagate

Problem

Given a partial model A and nogood δ

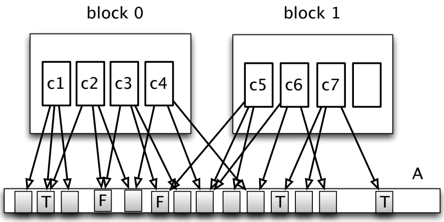
- Check if δ violated by A
- Check if $\delta \setminus A = \{X\}$

General Idea

- One thread per nogood
  - First Phase: original nogoods; only "activated" by recent assignment
  - Second Phase: all learned nogoods
- Three kernels per phase
  - All nogoods of cardinality 2
  - All nogoods of cardinality 3
  - All nogoods of greater cardinality

NoGoodCheckAndPropagate

- One block per assigned atom
- One thread per nogood relevant to assigned atom
- Need to iterate procedure

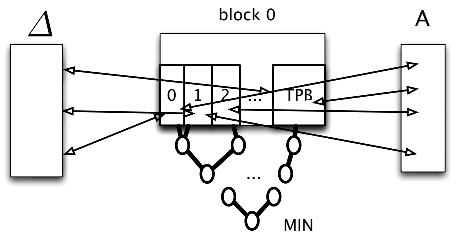


- 7 nogoods of cardinality 3; TPB=4
- *c2* and *c3* satisfied
- *c7* needs to propagate

Other Parallelized Procedures

Selection

- One Thread per unassigned atom p
- For each rule $r : \beta_r \leftarrow \tau_r, \eta_r$ with $head(r) = p$:
  - if $T\tau_r \in A$ and $F\eta_r \notin A$ then rule is applicable
- Determine rank each p that has applicable rules
- Select applicable rule with highest rank (logarithmic reduction)
- Logarithmic parallel reduction to determine rule with best rank



Other Parallelized Procedures

ConflictAnalysis

- First Kernel:
  - one thread per nogood
  - determines if nogood is violated
  - logarithmic reduction to determine nogood δ with oldest most recently assigned atom
- Second Kernel:
  - determine nogood that can resolve with δ (parallel)
  - resolution process to determine learned clause (sequential)

Glimpse at the results

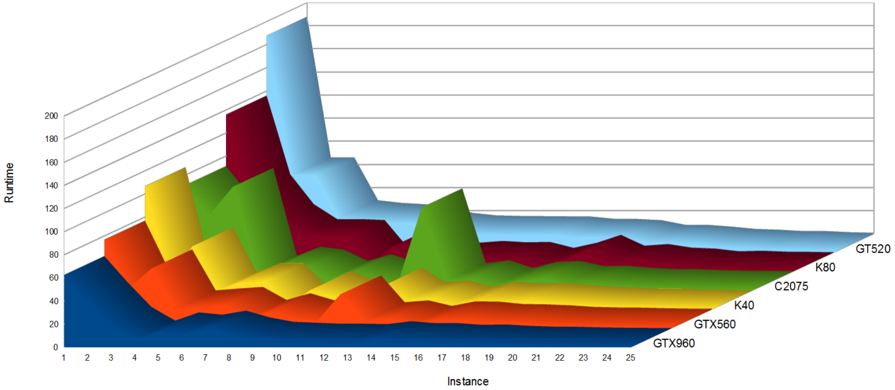
The results of experimentation with different GPUs are encouraging

- Performance scales with the computing power of the GPUs
  - number of cores
  - GPU clock
  - memory clock
- the prototype cannot compete with the state-of-the-art solvers
- but much has to be done in improving various aspects of the solver

Glimpse at the results

INSTANCE	GT 520	GTX 560	GTX 960	C2075	K80	K40	clasp*
0001-stablemarriage-0-0	11.73	6.84	**4.68**	9.41	15.52	6.04	t.o.
0001-visitall-14-1	65.99	51.97	**18.56**	89.87	42.08	54.74	0.02
0002-stablemarriage-0-0	15.34	6.69	**4.97**	7.12	8.75	6.15	t.o.
0003-stablemarriage-0-0	12.68	7.15	**4.66**	8.49	8.72	7.62	t.o.
0003-visitall-14-1	66.07	35.04	39.61	65.97	67.83	**25.11**	0.01
0004-stablemarriage-0-0	14.87	8.02	**3.80**	9.76	9.28	8.78	t.o.
0005-stablemarriage-0-0	15.19	29.55	**4.09**	72.01	10.11	19.70	t.o.
0007-graph_colouring-125-0	29.00	16.51	**6.84**	13.86	28.90	16.00	44.71
0007-stablemarriage-0-0	12.79	3.17	6.27	**3.15**	4.23	3.40	t.o.
0008-stablemarriage-0-0	7.64	4.53	**3.40**	5.18	7.58	5.01	t.o.
0009-labyrinth-11-0	6.08	3.60	**2.26**	3.39	4.45	3.69	0.71
0009-stablemarriage-0-0	7.80	4.88	**3.16**	4.90	5.97	6.58	t.o.
0010-graph_colouring-125-0	3.44	1.83	1.52	2.13	**1.24**	1.60	8.22
0039-labyrinth-11-0	24.39	8.33	15.45	9.38	4.03	**3.30**	0.02
0061-ppm-70-0	2.19	1.08	**0.56**	0.90	0.94	0.77	0.05
0072-ppm-70-0	2.25	1.57	**0.99**	1.38	1.76	1.63	0.03
0121-ppm-120-0	15.79	8.16	**5.69**	8.19	10.86	8.94	0.31
0128-ppm-120-0	0.70	0.64	0.25	0.37	0.34	**0.24**	0.03
0129-ppm-120-0	14.96	6.25	**4.19**	7.26	8.99	7.18	0.08
0130-ppm-90-0	4.00	2.23	**1.63**	2.32	3.60	2.48	0.01
0153-ppm-90-0	1.18	0.89	**0.44**	0.66	0.71	0.58	0.02
0167-sokoban-15-1	25.43	19.48	**11.83**	18.99	28.24	23.59	0.01
0345-sokoban-17-1	187.87	76.86	**62.54**	91.30	135.95	106.73	0.93
0482-sokoban-15-1	26.67	18.20	**13.88**	21.58	29.09	23.60	0.24
0589-sokoban-15-1	17.92	14.08	**9.65**	15.18	21.35	16.83	0.07
SUM	591.97	337.55	**230.92**	472.75	460.52	360.29	

Glimpse at the results



Future Work

- Exhaustive exploration of the tail of the search
- Conflict-driven learning is expensive
- Relaxing the ASP computation and explore alternative selection strategies

THANKS

Questions?