GPU-Accelerated Hypothesis Cover Set Testing for Learning in Logic

Eyad Algahtani and Dimitar Kazakov
University of York

28th ILP Conference, Ferrara, 4 Sep 2018

GPU-Accelerated Hypothesis Cover Set Testing for Learning in Logic

Eyad Algahtani and Dimitar Kazakov
University of York

28th ILP Conference, Ferrara, 4 Sep 2018

Overview

- Classic ILP algorithms combine cover set evaluation with a search algorithm using that result to find the best hypothesis.

Overview

- Classic ILP algorithms combine cover set evaluation with a search algorithm using that result to find the best hypothesis.

Overview (2)

- General-purpose GPU computation allows data-parallelism to be used in finding the cover set of logic hypotheses.

- Our long-term aim is: the efficient implementation of classic ILP-inspired algorithms for the Description Logic (DL) domain.

Description Logics

- Make use of unary predicates (*concepts*) and binary predicates, so called *roles,* e.g.: car1 *in_front_of* car2, car1 *size* short

- Several classes of DL exist depending on their expressivity, e.g. whether they have:

  - ✦ Existential restriction: ∃*drives*.Ferrari

  - ✦ Value restriction: ∀*drives*.Ferrari

  - ✦ Number restrictions: ≥2 *drives*.Ferrari (cf. ‘2 Jags’ Prescott)

  - ✦ Transitive roles, inverse roles, etc.

Description Logics (2)

- Expressivity of a given DL may affect decidability
- DL vs Horn clauses: neither subsumes the other. A good overlap, e.g. these definitions* are equivalent (for explicit types & non-transitive def. of infront/2):

```
eastbound(X):-
    has_car(X,Y), shape(Y, rectangle), infront(Y,Z), load(Z, triangle).
```

$\text{Eastbound} \equiv \text{Train} \sqcap \exists \mathit{has\_car}.(\exists \mathit{shape}.\text{Rectangle} \sqcap \exists \mathit{infront}.(\exists \mathit{load}.\text{Triangle}))$

$\text{Eastbound} \equiv \text{Train} \sqcap \exists \mathit{has\_car}.(\forall \mathit{shape}.\text{Rectangle} \sqcap \forall \mathit{infront}.(\forall \mathit{load}.\text{Triangle}))$

*Horn clause example from:
Meike Schaller. *How does the Representation of Machine Learned Relational Rules affect Human Comprehensibility? A comparative study.* 2017

Relational Learning for Description Logics

- DL-FOIL (Fanizzi, d'Amato, Esposito 2008)
- DL-Learner (Buehmann, Lehmann, Westphal 2016)
- APARELL: Learning ordinal relations (Qomariyah & Kazakov 2017)

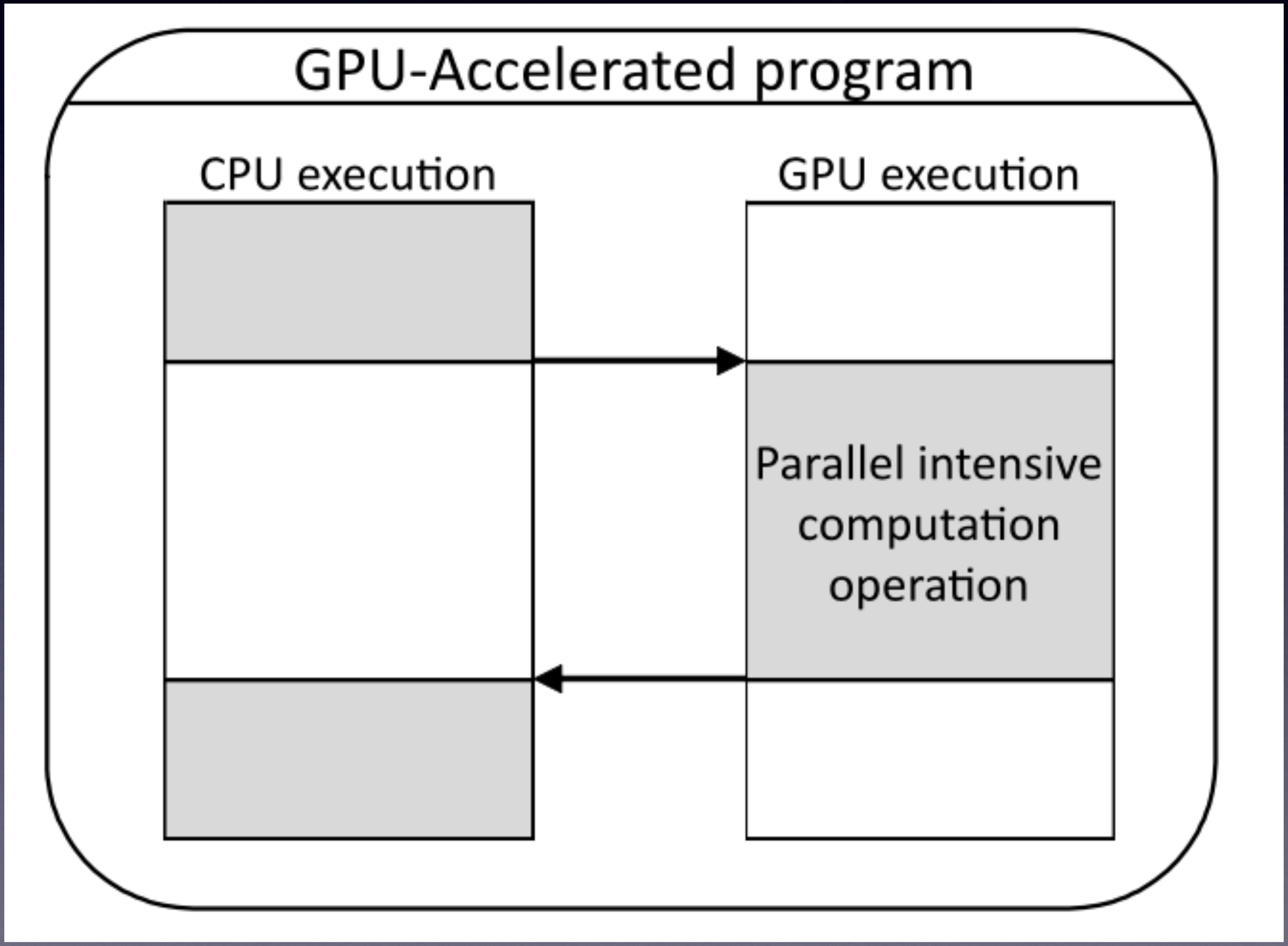
Speeding up ILP Learners

- Avoid redundancy:
  - ✦ Query packs (Blockeel *et al* 2000)
- Parallelise computation (Fonseca *et al* 2005):
  - ✦ animal(X,fish) || animal(X,mammal) || animal(X,bird)
  - ✦ divide the search space among different threads
  - ✦ split the data, learn, merge results

Speeding up Hypothesis Evaluation

- Given a GPU
  - ✦ The CPU can run the search algorithm while
  - ✦ the GPU evaluates individual hypotheses.
- We're using Nvidia GeForce GTX 1070 GPU running CUDA library

CPU ⟷ GPU Interplay



GPU Hypothesis Evaluation for Propositional Data

	C1	C2	C3	C4		Result (C2 AND C3)	
	1	0	0	1	=	0	Individual 1
Thread 1	1	0	1	0	=	0	Individual 2
	1	1	1	1	=	1	Individual 3
	1	0	1	0	=	0	Individual 4
Thread 2	0	1	0	1	=	0	Individual 5
	0	1	1	0	=	1	Individual 6
	0	1	0	1	=	0	Individual 7
Thread 3	0	1	1	0	=	1	Individual 8
	0	1	0	1	=	0	Individual 9
	...	...	...	...	=	...	Individual N

cover set size $=\sum Result_i$

GPU Hypothesis Evaluation for Propositional Data

- Membership of unary predicates/concepts is represented through a 2D binary matrix in the 'global' GPU memory (shared by all threads). There are two such matrices for the ⊕, resp. ⊖ examples of the target concept.
- Data parallelism is used to compute conjunction, disjunction or negation of concepts: the matrix is split up, and a thread assigned to each part.
- A 1D array is used to record the membership of each individual in the hypothesis being tested, e.g. `C1 U C2 U C3`. Lazy evaluation can be used. Total coverage is added up by the GPU using reduction-sum.
- Coverage of ⊕ and ⊖ ex.s of the target concept is counted separately.

GPU Hypothesis Evaluation for Propositional Data

- Each result (e.g. `C1 U C2 U C3`) can be directly memoized by simply adding it as another column to the individuals x concepts matrix. The memory for it needs to be preallocated though to make the process efficient.
  - I.e. if we have N concepts, we need to allocate a combined matrix of size individuals x (N+M).

Sample Operator Pseudocode

Algorithm 1 For a Boolean matrix M (individuals × concepts)

---

procedure PARALLELCONCEPTCONJUNCTIONCOVERSET

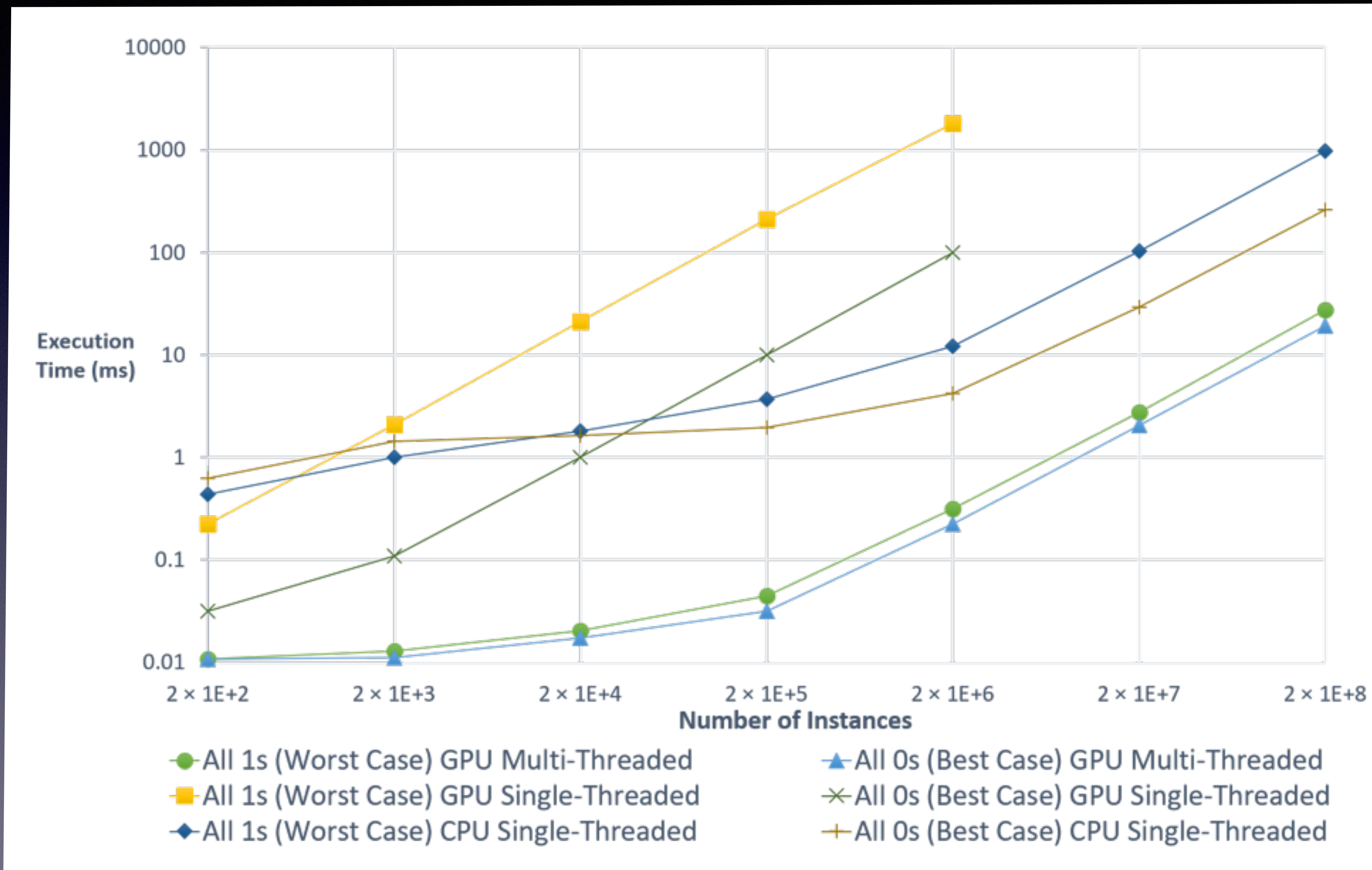
```
set S := list of concepts in conjunction
parallel_foreach thread T_i
|  foreach individual I_j in thread T_i
|  |  set result(row(I_j)) := 1
|  |  foreach concept C_k in set S
|  |  | set result(row(I_j)) := result(row(I_j)) && M(row(I_j),column(C_k))
|  |  | if (result(row(I_j)) == 0) break
|  |  endfor
|  endfor
endfor
```

return Boolean array: result(1..numberOfIndividuals)

Best & Worst Times

Instances (pos+neg)	4 concepts			
	Time (all 1s)		Time (all 0s)	
	conj	disj	conj	disj
2×100	10.69 μs	9.09 μs	10.69 μs	11.20 μs
$2 \times 1,000$	12.83 μs	16.38 μs	11.14 μs	11.81 μs
$2 \times 10,000$	20.35 μs	29.95 μs	17.34 μs	20.35 μs
$2 \times 100,000$	44.16 μs	31.97 μs	31.55 μs	45.02 μs
$2 \times 1,000,000$	315.65 μs	224.74 μs	223.30 μs	314.08 μs
$2 \times 10,000,000$	2.76 ms	2.05 ms	2.06 ms	2.75 ms
$2 \times 100,000,000$	27.55 ms	20.63 ms	19.10 ms	25.45 ms

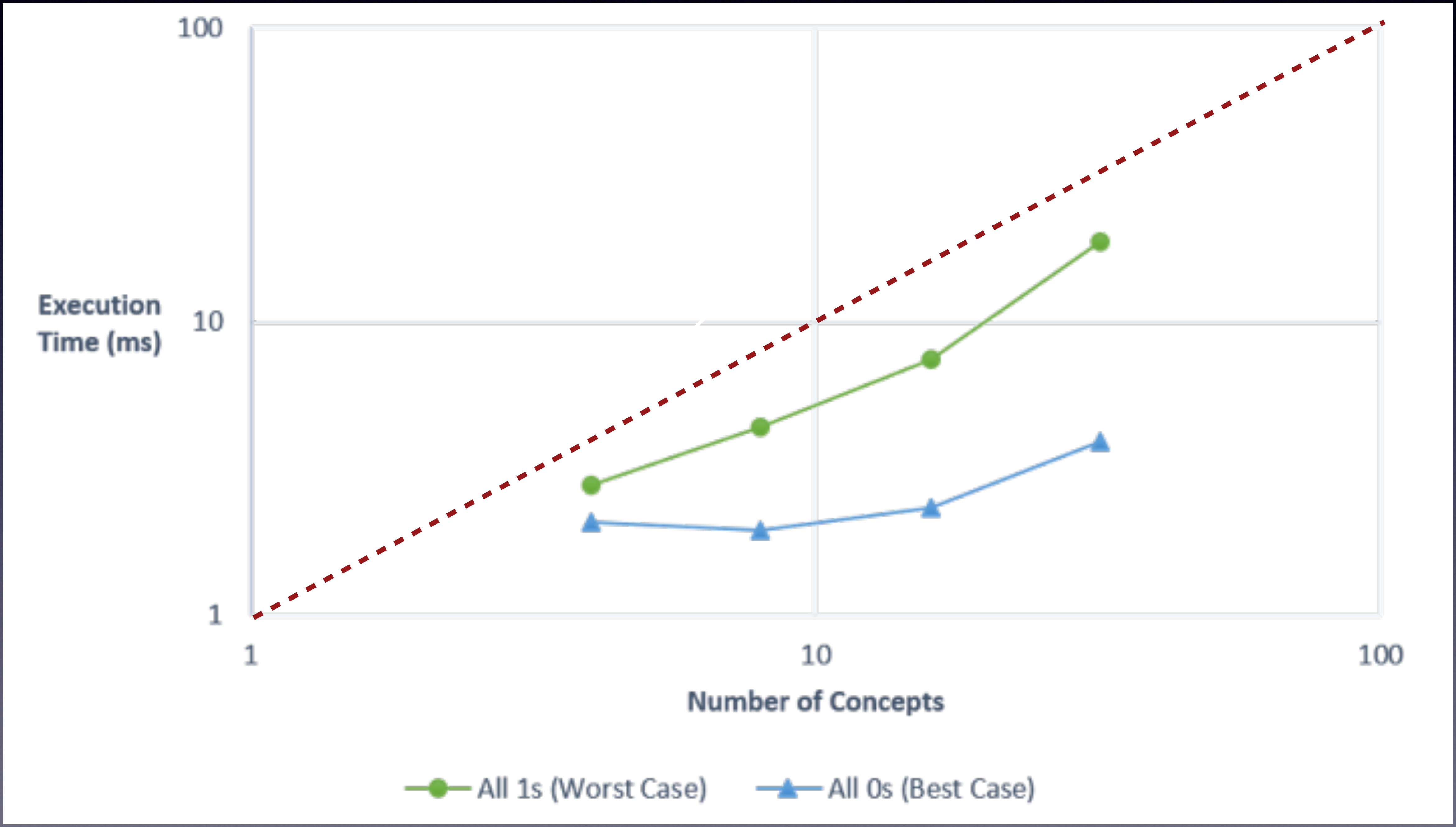
CPU (1 thread) v GPU (1, 32 threads/block)
4 attributes/concepts/unary predicates



GPU *vs* single-thread CPU

- For 2,000,000 individuals and 4 concepts, and the worst case w.r.t. lazy evaluation:
  - 1 thread/block GPU is ~150 times slower
  - 64-thread/block GPU is ~38 times faster

t=f(#concepts)


100
10
1
Execution Time (ms)
1
10
100
Number of Concepts
All 1s (Worst Case)
All 0s (Best Case)

t=f(#concepts)

- Due to lazy evaluation, increasing the number of concepts does not necessarily increase execution time.
- The worst case time complexity increases approx. linearly as hoped for.

Representing Roles on the GPU

Individual A	Role	Individual B
6	1	46
6	1	5
9	2	14
1079	3	78
749	3	43
465	4	89
89	4	700
658	4	133
98	5	1079
…	…	…

Existential Restriction Pseudocode

Algorithm 4 Compute Existential Restriction (∃)

```
procedure ParallelExistentialRestrictiOnConcepts(role,role_start_ind,role_end_ind,S)

Call setAllElementsInResultToVal_inParallel(0)
set S := list of concepts in conjunction
set IndA := list of individualA (individualA values) in the same Role
set IndB := list of individualB (individualB values) in the same Role
parallel foreach thread T_i
| foreach individualA I_A in set IndA
| | set result(row(I_A)) := 1
| | foreach concept C_k in set S
| | | set result(row(I_A)) := result(row(I_A)) && M(row(I_B),column(C_k))
| | | if (result(row(I_A)) == 0) break
| | endfor
| endfor
endfor

return Boolean array: result(1..numberOfIndividuals)
```

Value Restriction Pseudocode

Algorithm 5 Compute Value Restriction (∀)

```
procedure ParallelUniversalRestrictiOnConcepts(role,role_start_ind,role_end_ind,S)

Call setAllElementsInResultToVal_inParallel(1)
set S := list of concepts in conjunction
set IndA := list of individualA (individualA values) in the same Role
set IndB := list of individualB (individualB values) in the same Role
parallel_foreach thread T_i
| foreach individualA I_A in set IndA
| | foreach concept C_k in set S
| | | set result(row(I_A)) := result(row(I_A)) && M(row(I_B),column(C_k))
| | | if (result(row(I_A)) == 0) break
| | endfor
| endfor
endfor

return Boolean array: result(1..numberOfIndividuals)
```

The enD