RDFS Reasoning on Massively Parallel Hardware

Norman Heino, Jeff Z. Pan

University of Leipzig,
Germany
AKSW Research Group

University of Aberdeen,
United Kingdom
Dept. of Computing Science

RDFS reasoning

- Interpreting RDFS vocabulary can give rise to new triples
- Model-theoretic semantic conditions from RDF semantics document
- Incomplete set of rules
    - we use subset with exactly two antecedents

(5)	*p* rdfs:subPropertyOf *q*	& *q* rdfs:subPropertyOf *r*	⟹ *p* rdfs:subPropertyOf *r*
(11)	*C* rdfs:subClassOf *D*	& *D* rdfs:subClassOf *E*	⟹ *C* rdfs:subClassOf *E*
(2)	*s p o*	& *p* rdfs:domain *D*	⟹ *s* rdf:type *D*
(3)	*s p o*	& *p* rdfs:range *R*	⟹ *o* rdf:type *R*
(7)	*s p o*	& *p* rdfs:subPropertyOf *q*	⟹ *s q o*
(9)	*s* rdf:type *B*	& *B* rdfs:subClassOf *C*	⟹ *s* rdf:type *C*

RDFS reasoning

(5)	p `rdfs:subPropertyOf` q	& q `rdfs:subPropertyOf` r	$\Longrightarrow$ p `rdfs:subPropertyOf` r
(11)	C `rdfs:subClassOf` D	& D `rdfs:subClassOf` E	$\Longrightarrow$ C `rdfs:subClassOf` E
(2)	$s\ p\ o$	& p `rdfs:domain` D	$\Longrightarrow$ s `rdf:type` D
(3)	$s\ p\ o$	& p `rdfs:range` R	$\Longrightarrow$ o `rdf:type` R
(7)	$s\ p\ o$	& p `rdfs:subPropertyOf` q	$\Longrightarrow$ $s\ q\ o$
(9)	s `rdf:type` B	& B `rdfs:subClassOf` C	$\Longrightarrow$ s `rdf:type` C

Preliminaries

(5)	p `rdfs:subPropertyOf` q	& q `rdfs:subPropertyOf` r	$\Longrightarrow$ p `rdfs:subPropertyOf` r
(11)	C `rdfs:subClassOf` D	& D `rdfs:subClassOf` E	$\Longrightarrow$ C `rdfs:subClassOf` E
(2)	$s\ p\ o$	& p `rdfs:domain` D	$\Longrightarrow$ s `rdf:type` D
(3)	$s\ p\ o$	& p `rdfs:range` R	$\Longrightarrow$ o `rdf:type` R
(7)	$s\ p\ o$	& p `rdfs:subPropertyOf` q	$\Longrightarrow$ $s\ q\ o$
(9)	s `rdf:type` B	& B `rdfs:subClassOf` C	$\Longrightarrow$ s `rdf:type` C

Preliminaries

(5)	p rdfs:subPropertyOf q	& q rdfs:subPropertyOf r	$\Longrightarrow$ p rdfs:subPropertyOf r
(11)	C rdfs:subClassOf D	& D rdfs:subClassOf E	$\Longrightarrow$ C rdfs:subClassOf E
(2)	$s\ p\ o$	& p rdfs:domain D	$\Longrightarrow$ s rdf:type D
(3)	$s\ p\ o$	& p rdfs:range R	$\Longrightarrow$ o rdf:type R
(7)	$s\ p\ o$	& p rdfs:subPropertyOf q	$\Longrightarrow$ $s\ q\ o$
(9)	s rdf:type B	& B rdfs:subClassOf C	$\Longrightarrow$ s rdf:type C

- No rules with ‚trivial‘ entailments

Preliminaries

(5)	p `rdfs:subPropertyOf` q	& q `rdfs:subPropertyOf` r	$\Longrightarrow$	p `rdfs:subPropertyOf` r
(11)	C `rdfs:subClassOf` D	& D `rdfs:subClassOf` E	$\Longrightarrow$	C `rdfs:subClassOf` E
(2)	$s\ p\ o$	& p `rdfs:domain` D	$\Longrightarrow$	s `rdf:type` D
(3)	$s\ p\ o$	& p `rdfs:range` R	$\Longrightarrow$	o `rdf:type` R
(7)	$s\ p\ o$	& p `rdfs:subPropertyOf` q	$\Longrightarrow$	$s\ q\ o$
(9)	s `rdf:type` B	& B `rdfs:subClassOf` C	$\Longrightarrow$	s `rdf:type` C

- No rules with ‚trivial‘ entailments
- Ignore non-authorative statements

Preliminaries

(5)	p `rdfs:subPropertyOf` q	& q `rdfs:subPropertyOf` r	$\Longrightarrow$ p `rdfs:subPropertyOf` r
(11)	C `rdfs:subClassOf` D	& D `rdfs:subClassOf` E	$\Longrightarrow$ C `rdfs:subClassOf` E
(2)	$s\ p\ o$	& p `rdfs:domain` D	$\Longrightarrow$ s `rdf:type` D
(3)	$s\ p\ o$	& p `rdfs:range` R	$\Longrightarrow$ o `rdf:type` R
(7)	$s\ p\ o$	& p `rdfs:subPropertyOf` q	$\Longrightarrow$ $s\ q\ o$
(9)	s `rdf:type` B	& B `rdfs:subClassOf` C	$\Longrightarrow$ s `rdf:type` C

- No rules with ‚trivial‘ entailments
- Ignore non-authorative statements
- Results obtained w/o axiomatic triples

Special cases

(7)-1	s langProp_1 o	&	langProp_1 `rdfs:subPropertyOf` langProp_2	$\Longrightarrow$	s langProp_2 o
(7)-2	s langProp o	&	langProp `rdfs:subPropertyOf` q	$\Longrightarrow$	$s\ q\ o$
(7)-3	$s\ p\ o$	&	p `rdfs:subPropertyOf` langProp	$\Longrightarrow$	s langProp o
(7)-4	$s\ q\ o$	&	p `rdfs:subPropertyOf` q	$\Longrightarrow$	$s\ q\ o$

Special cases

~~(7)-1~~	~~s langProp$_1$ o~~	~~&~~	~~langProp$_1$ `rdfs:subPropertyOf` langProp$_2$~~	~~$\Longrightarrow$~~	~~s langProp$_2$ o~~
~~(7)-2~~	~~s langProp o~~	~~&~~	~~langProp `rdfs:subPropertyOf` q~~	~~$\Longrightarrow$~~	~~$s\ q\ o$~~
(7)-3	$s\ p\ o$	&	p `rdfs:subPropertyOf` langProp	$\Longrightarrow$	s langProp o
(7)-4	$s\ q\ o$	&	p `rdfs:subPropertyOf` q	$\Longrightarrow$	$s\ q\ o$

Parallel RDFS reasoning

Parallel RDFS reasoning

- Cluster-based approaches
  - MapReduce – Urbani et al. (2009)
  - DHT – Kaoudi et al. (2008)
  - Peer network – Weaver and Hendler et al. (2009)

Parallel RDFS reasoning

- Cluster-based approaches
  - MapReduce – Urbani et al. (2009)
  - DHT – Kaoudi et al. (2008)
  - Peer network – Weaver and Hendler et al. (2009)
- HPC Implementation
  - Cray XMT – Goodman and Mizell (2010)

Parallel RDFS reasoning

- Cluster-based approaches
  - MapReduce – Urbani et al. (2009)
  - DHT – Kaoudi et al. (2008)
  - Peer network – Weaver and Hendler et al. (2009)
- HPC Implementation
  - Cray XMT – Goodman and Mizell (2010)

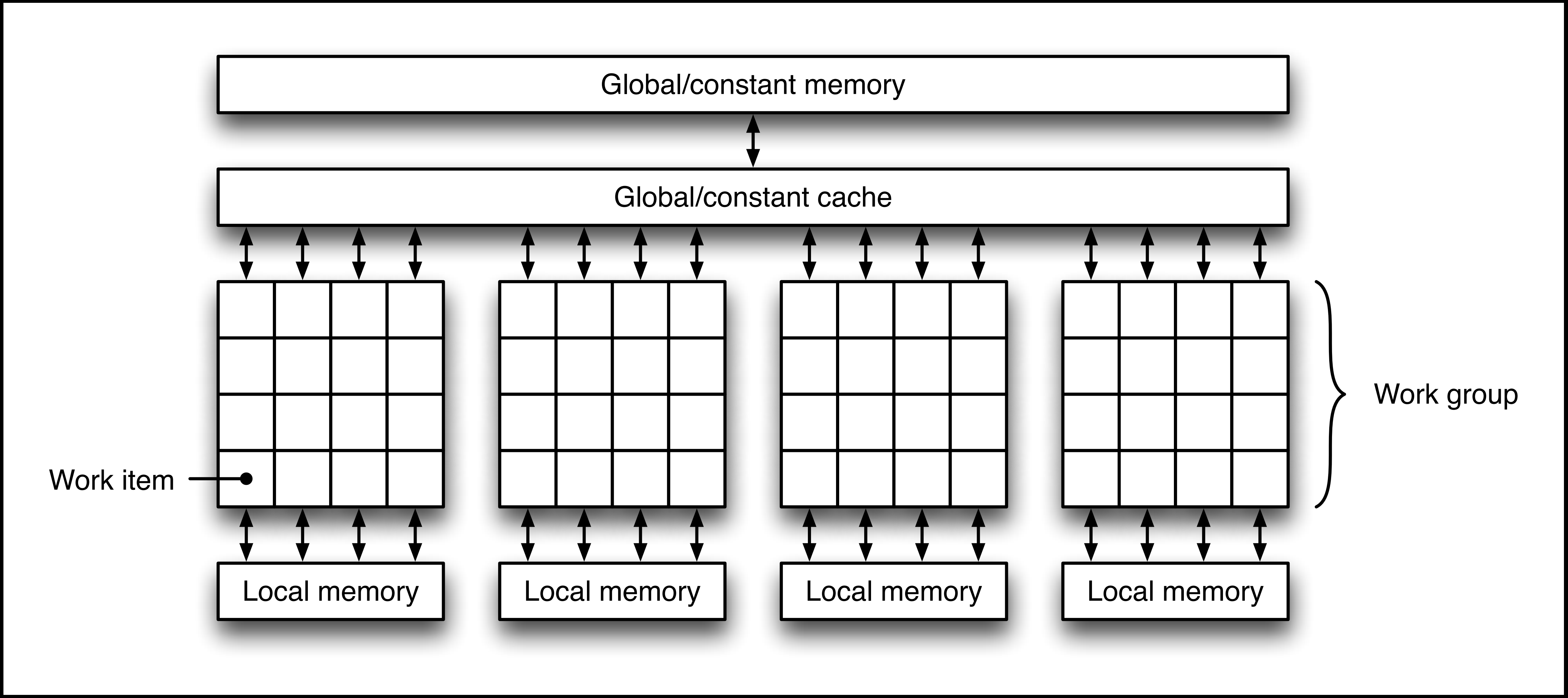
Our platform: single host, massively parallel commodity hardware with shared memory

Challenge: develop algorithm that allows for fine-grained parallelism

OpenCL

- Vendor-agnostic model and API for heterogenous parallel computation
- Khronos standard
- *GPU*, CPU, accellerators (Cell, DSPs)
- Host program controls submission of compute kernels and data to hardware
- Compute kernels written in C99 subset

OpenCL memory model



Big picture

Big picture

1. Parsing

Big picture

1. Parsing
2. Dictionary encoding

Big picture

1. Parsing
2. Dictionary encoding
3. In-memory storage
    - one `std::vector` per column
    - indexed by *hash*(triple)

Big picture

1. Parsing
2. Dictionary encoding
3. In-memory storage
    - one `std::vector` per column
    - indexed by *hash*(triple)
4. Copying data to device (PCIe bus)

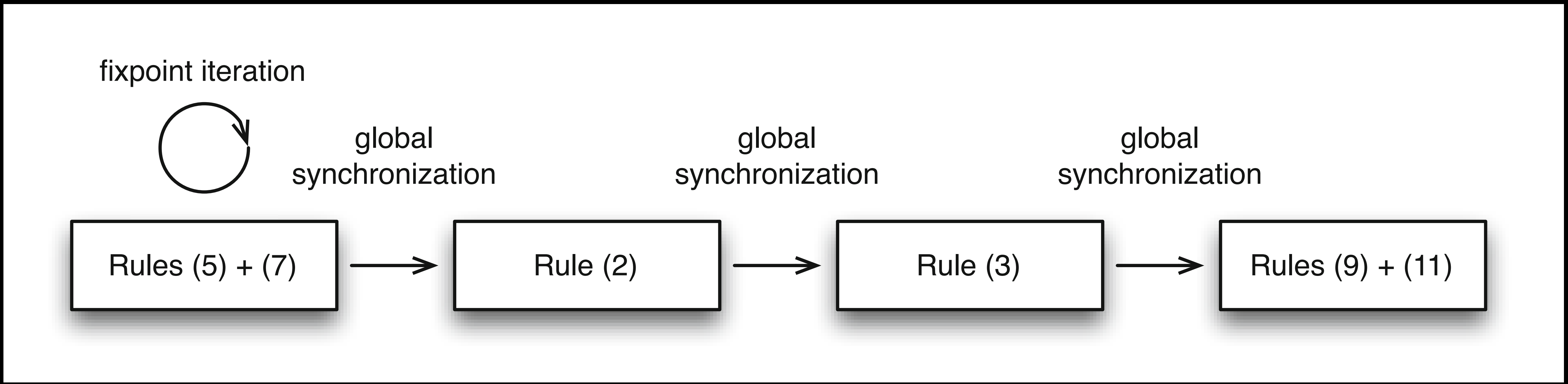
Big picture

1. Parsing
2. Dictionary encoding
3. In-memory storage
    - one `std::vector` per column
    - indexed by *hash*(triple)
4. Copying data to device (PCIe bus)
5. Rule application on device

Big picture

1. Parsing
2. Dictionary encoding
3. In-memory storage
   - one `std::vector` per column
   - indexed by *hash*(triple)
4. Copying data to device (PCIe bus)
5. Rule application on device
6. Copying back, storing results

Rule implementation



- Could produce schema triples
- Independent of (2), could be run in parallel

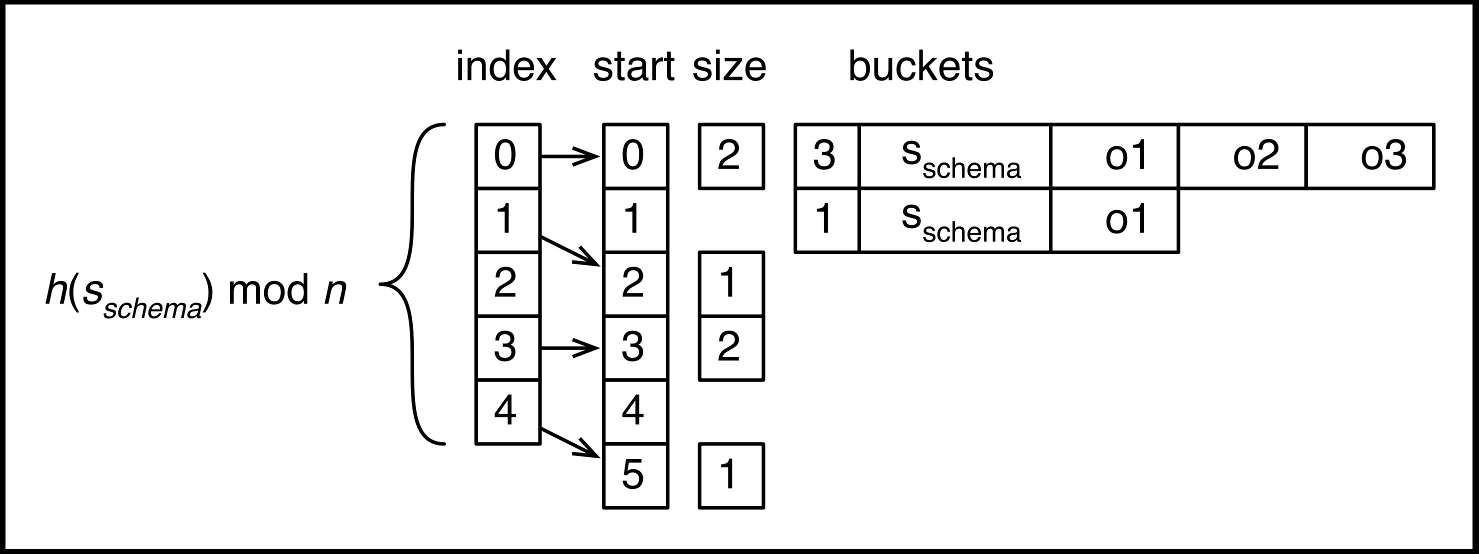
(5)	*p* rdfs:subPropertyOf *q*	& *q* rdfs:subPropertyOf *r*	⟹ *p* rdfs:subPropertyOf *r*
(11)	*C* rdfs:subClassOf *D*	& *D* rdfs:subClassOf *E*	⟹ *C* rdfs:subClassOf *E*
(2)	*s p o*	& *p* rdfs:domain *D*	⟹ *s* rdf:type *D*
(3)	*s p o*	& *p* rdfs:range *R*	⟹ *o* rdf:type *R*
(7)	*s p o*	& *p* rdfs:subPropertyOf *q*	⟹ *s q o*
(9)	*s* rdf:type *B*	& *B* rdfs:subClassOf *C*	⟹ *s* rdf:type *C*

TC-based rules

- Transitive property hierachies have a sparse adjacency matrix
  - Quadratic algorithm (Warshall) is infeasible
- Host-run serial implementation based on work by Nuutila

Join rules

- Join instance triple with schema subject
  - counting results
  - computing index
- Materialize result triples in two passes



The duplicate problem

```
@base <http://example.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<p> rdfs:domain <C> .
<r> rdfs:domain <D> .
<p> rdfs:subPropertyOf <r> .
<C> rdfs:subClassOf <D> .
<A> <p> "01", "02", "03" .
```

(5)	*p* rdfs:subPropertyOf *q*	& *q* rdfs:subPropertyOf *r*	$\Longrightarrow$ *p* rdfs:subPropertyOf *r*
(11)	*C* rdfs:subClassOf *D*	& *D* rdfs:subClassOf *E*	$\Longrightarrow$ *C* rdfs:subClassOf *E*
(2)	*s p o*	& *p* rdfs:domain *D*	$\Longrightarrow$ *s* rdf:type *D*
(3)	*s p o*	& *p* rdfs:range *R*	$\Longrightarrow$ *o* rdf:type *R*
(7)	*s p o*	& *p* rdfs:subPropertyOf *q*	$\Longrightarrow$ *s q o*
(9)	*s* rdf:type *B*	& *B* rdfs:subClassOf *C*	$\Longrightarrow$ *s* rdf:type *C*

The duplicate problem

```
@base <http://example.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<p> rdfs:domain <C> .
<r> rdfs:domain <D> .
<p> rdfs:subPropertyOf <r> .
<C> rdfs:subClassOf <D> .
<A> <p> "01", "02", "03" .
```

```
<A> rdf:type <C>
```

(5)	p `rdfs:subPropertyOf` q	& q `rdfs:subPropertyOf` r	$\Longrightarrow$ p `rdfs:subPropertyOf` r
(11)	C `rdfs:subClassOf` D	& D `rdfs:subClassOf` E	$\Longrightarrow$ C `rdfs:subClassOf` E
(2)	$s\ p\ o$	& p `rdfs:domain` D	$\Longrightarrow$ s `rdf:type` D
(3)	$s\ p\ o$	& p `rdfs:range` R	$\Longrightarrow$ o `rdf:type` R
(7)	$s\ p\ o$	& p `rdfs:subPropertyOf` q	$\Longrightarrow$ $s\ q\ o$
(9)	s `rdf:type` B	& B `rdfs:subClassOf` C	$\Longrightarrow$ s `rdf:type` C

The duplicate problem

```
@base <http://example.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<p> rdfs:domain <C> .
<r> rdfs:domain <D> .
<p> rdfs:subPropertyOf <r> .
<C> rdfs:subClassOf <D> .
<A> <p> "01", "02", "03" .
```

```
<A> rdf:type <C>
```

```
<A> rdf:type <D>
```

(5)	*p* rdfs:subPropertyOf *q*	& *q* rdfs:subPropertyOf *r*	⟹ *p* rdfs:subPropertyOf *r*
(11)	*C* rdfs:subClassOf *D*	& *D* rdfs:subClassOf *E*	⟹ *C* rdfs:subClassOf *E*
(2)	*s p o*	& *p* rdfs:domain *D*	⟹ *s* rdf:type *D*
(3)	*s p o*	& *p* rdfs:range *R*	⟹ *o* rdf:type *R*
(7)	*s p o*	& *p* rdfs:subPropertyOf *q*	⟹ *s q o*
(9)	*s* rdf:type *B*	& *B* rdfs:subClassOf *C*	⟹ *s* rdf:type *C*

The duplicate problem

```
@base <http://example.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<p> rdfs:domain <C> .
<r> rdfs:domain <D> .
<p> rdfs:subPropertyOf <r> .
<C> rdfs:subClassOf <D> .
<A> <p> "01", "02", "03" .
```

```
<A> rdf:type <C>
```

```
<A> rdf:type <D>
```

(5)	*p* rdfs:subPropertyOf *q*	& *q* rdfs:subPropertyOf *r*	$\Longrightarrow$ *p* rdfs:subPropertyOf *r*
(11)	*C* rdfs:subClassOf *D*	& *D* rdfs:subClassOf *E*	$\Longrightarrow$ *C* rdfs:subClassOf *E*
(2)	*s p o*	& *p* rdfs:domain *D*	$\Longrightarrow$ *s* rdf:type *D*
(3)	*s p o*	& *p* rdfs:range *R*	$\Longrightarrow$ *o* rdf:type *R*
(7)	*s p o*	& *p* rdfs:subPropertyOf *q*	$\Longrightarrow$ *s q o*
(9)	*s* rdf:type *B*	& *B* rdfs:subClassOf *C*	$\Longrightarrow$ *s* rdf:type *C*

Duplicates by rule

	DBpedia			YAGO2 Core		
Rule	Triples	Duplicates	Ratio	Triples	Duplicates	Ratio
(5)	0	0	–	0	19	>
(7)	0	0	–	3,551,361	88,477	0.03
(2)	368,832	7,630,029	21	6,450,781	13,453,038	2.1
(3)	568,715	4,939,870	8.7	409,193	1,511,512	3.7
(11)	259	610	2	3,398,943	366,764	0.1
(9)	0	8,329,278	>	6,685,946	3,173,957	0.5
(11+9)	259	10,398,328	42,162	35,061,599	57,969,000	1.7
all	1,650,607	23,775,152	14	45,766,218	89,370,361	2.0

Duplicates by rule

Rule	DBpedia			YAGO2 Core		
	Triples	Duplicates	Ratio	Triples	Duplicates	Ratio
(5)	0	0	–	0	19	>
(7)	0	0	–	3,551,361	88,477	0.03
(2)	368,832	7,630,029	21	6,450,781	13,453,038	2.1
(3)	568,715	4,939,870	8.7	409,193	1,511,512	3.7
(11)	259	610	2	3,398,943	366,764	0.1
(9)	0	8,329,278	>	6,685,946	3,173,957	0.5
(11+9)	259	10,398,328	42,162	35,061,599	57,969,000	1.7
all	1,650,607	23,775,152	14	45,766,218	89,370,361	2.0

Global duplicate prevention

- Hash table in global device memory
- Each thread materializes triple only if not in hash table
- Table is static (not updated)

Local deduplication

1. Sort triples in local memory
2. Count adjacent duplicates
3. Caclulate new duplicate-free index
4. Zero out duplicates, rearrange unique values

Local dedup example

data vector

index vectors

Local dedup example

data vector

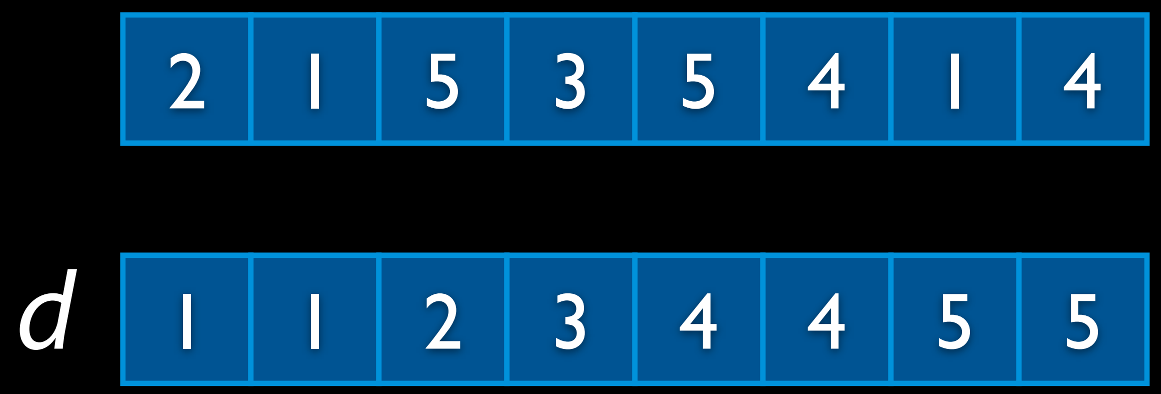
2	1	5	3	5	4	1	4

index vectors

Local dedup example

data vector

index vectors



Local dedup example

data vector

2	1	5	3	5	4	1	4

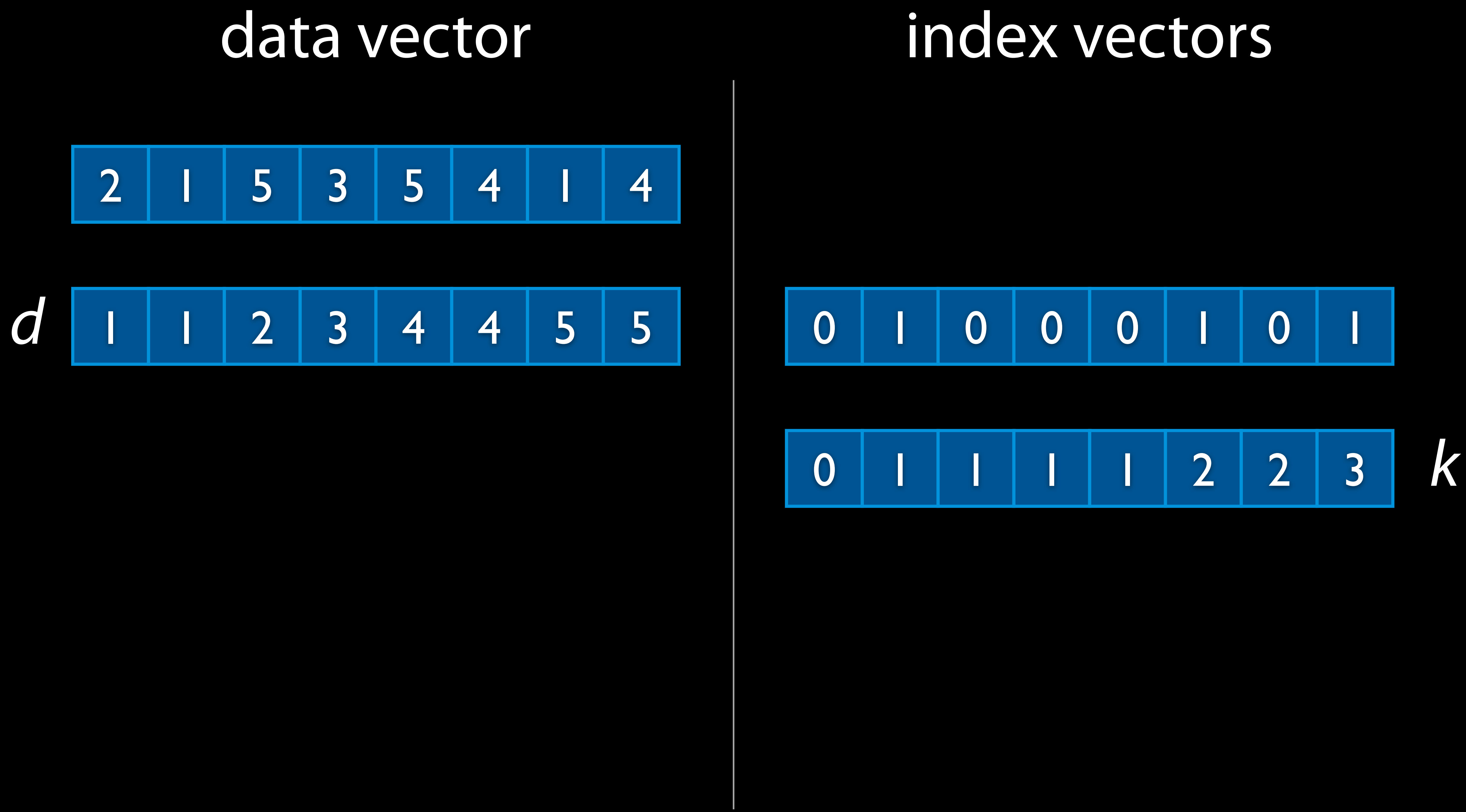
d

1	1	2	3	4	4	5	5

index vectors

0	1	0	0	0	1	0	1

Local dedup example



Local dedup example

data vector

	2	1	5	3	5	4	1	4
d	1	1	2	3	4	4	5	5

index vectors

0	1	0	0	0	1	0	1	
0	1	1	1	1	2	2	3	k

$$r_i = \begin{cases} d_{i+k_i} & i + k_i < |d| \\ 0 & \text{else} \end{cases}$$

Local dedup example

data vector

2	1	5	3	5	4	1	4

d

1	1	2	3	4	4	5	5

r

1	2	3	4	5	0	0	0

$$r_i = \begin{cases} d_{i+k_i} & i + k_i < |d| \\ 0 & \text{else} \end{cases}$$

index vectors

0	1	0	0	0	1	0	1

k

0	1	1	1	1	2	2	3

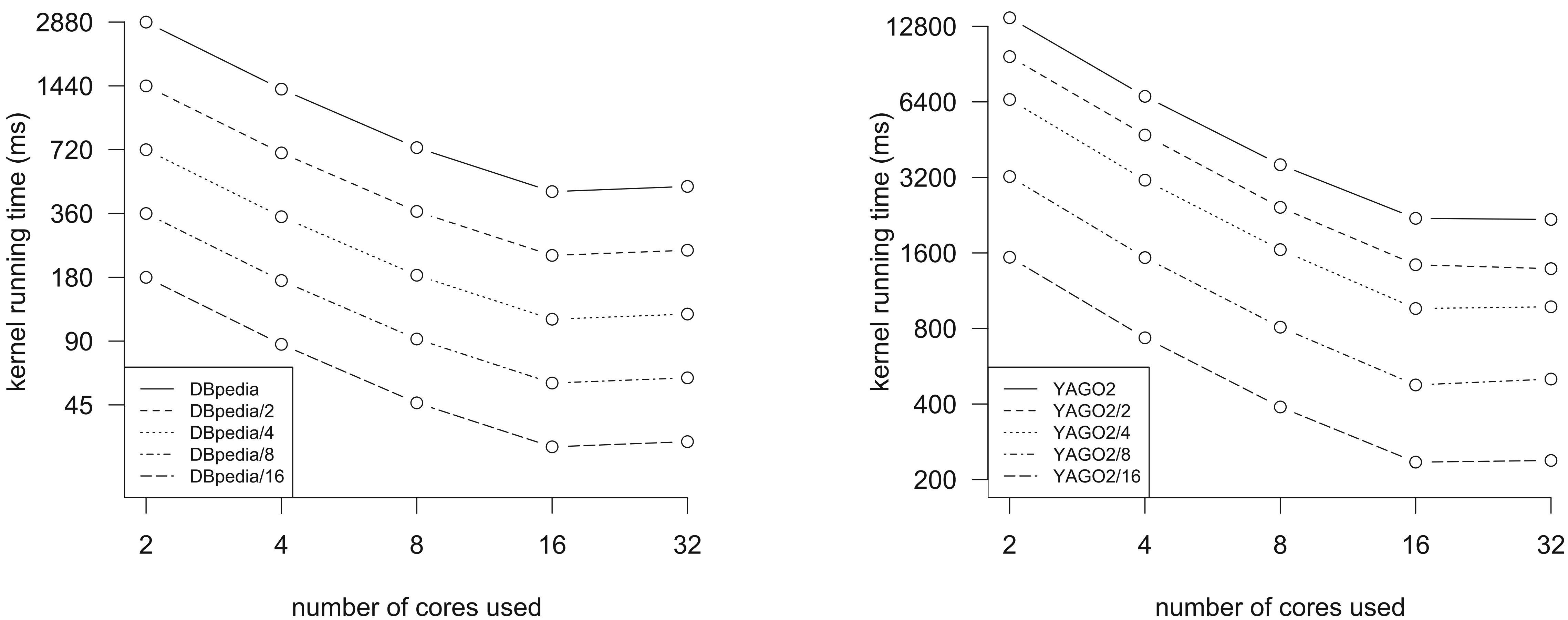
Experiments

- Exp1: study scalability on different levels of hardware parallelism
  - performed on 4 CPU server with 32 CUs
- Exp2: study efficacy of duplicate removal strategies
  - performed on 20 CU GPU device
- Exp3: compare to previous work

Datasets

- DBedia ontology, infobox types, infobox triples:
  - ~26M triples
  - ~1.7M new closure triples
- YAGO2Core
  - ~36M triples
  - ~46M new closure triples
- Both scaled to 1/2, …, 1/16th of instance triples plus all schema triples

Results – Exp1



(a) DBpedia (b) YAGO2 Core

Results – Exp2

Dataset	Strategy	Kernel time (ms)	Closure time (ms)	Duplicates	Speedup
DBpedia	None	28.444	6,884.15	23,775,152	
	L	120.915	6,083.76	12,165,520	13.2 %
	G	52.305	6,635.60	1,511,758	3.7 %
	L+G	117.400	6,557.94	1,057,470	5 %
YAGO2/8	None	25.565	21,625.19	31,552,221	
	L	187.169	19,554.09	2,399,898	10.6 %
	G	53.948	21,622.31	29,357,936	0 %
	L+G	215.947	19,807.66	1,786,753	9.2 %

Results – Exp3

	Input triples	Output triples	Damásio (ms)	Our system (ms)	Speedup
T2	366,490	3,617,532	23,619.90	9,038.89	2.6 $\times$
T6	1,942,887	4,947,407	18,602.43	1,964.49	9.5 $\times$

- T6 also performed on M/R cluster in > 3 min

CPU – GPU comparison

Device	Kernel execution (ms)	Total (ms)
Core i7 3770 (CPU)	647.311	5509.92
Radeon HD 7870 (GPU)	114.683	5881.54

Conclusions/future work

- RDFS reasoning is can be done *massively* parallel
- Shared memory can be used for efficient parallel duplicate reduction
- low ALU:fetch ratio is unvaforable for GPU devices

- Data compression
- Multiple devices
- Reasoning on the FPU
- Complete implementation w.r.t. RDF semantics

Complexity of RDFS reasoning

- RDFS reasoning is in *P,* if *G* does not contain blank nodes (ter Horst, 2002)
- RDFS reasoning is *P*-complete (i.e. in *P* but not in *NC; Patel-Schneider, 2012)*
- NC – problem can be solved in $O(\log^c n)$ time using $O(n^k)$ processors
    - i.e. you can trade parallelism for complexity