GK Spring Workshop Waldau:

Pathfinder/MonetDB: A High-Performance Relational Runtime for XQuery

Jan Rittinger

Database & Information Systems Group
University of Konstanz

03/16/2005

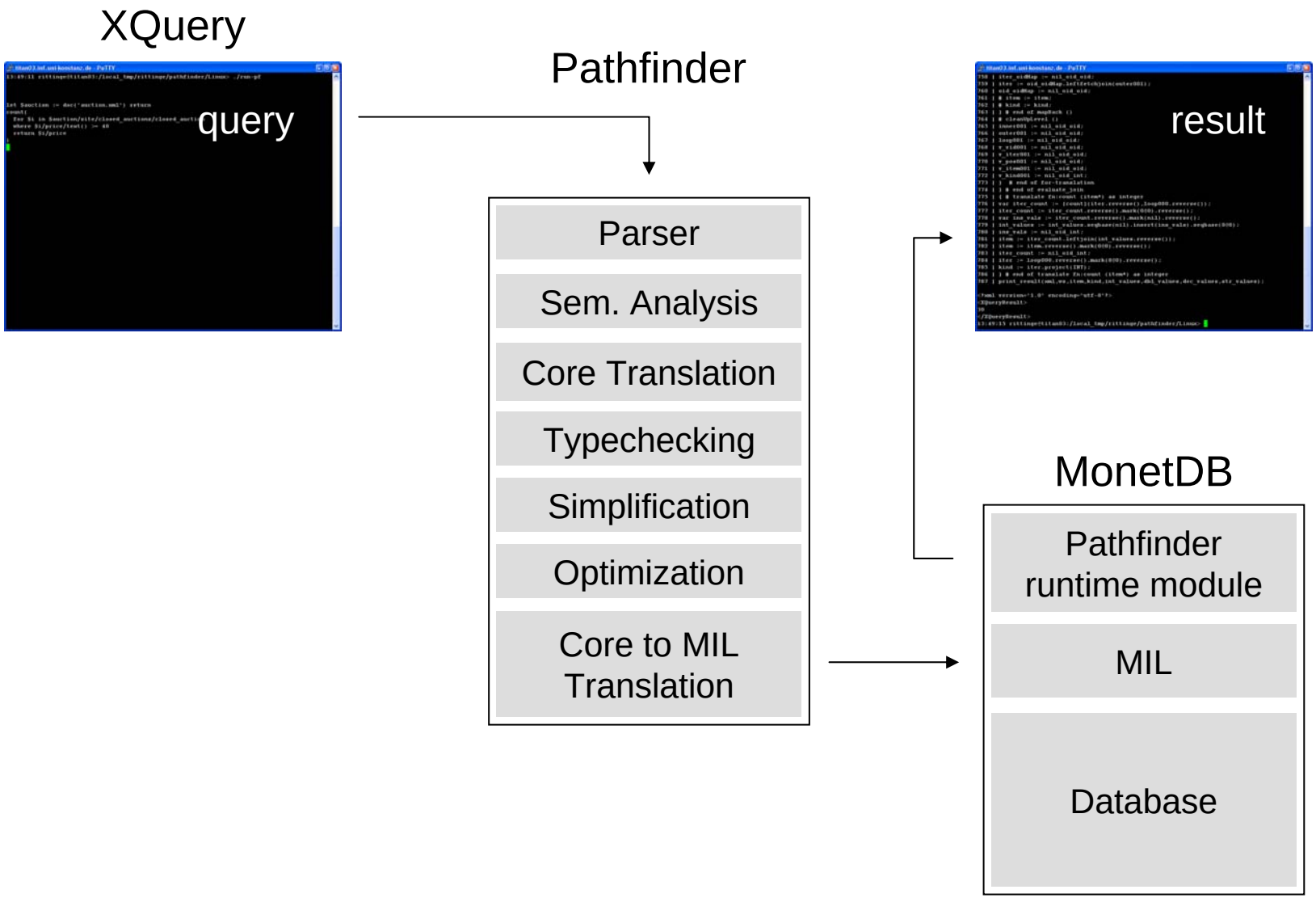
Pathfinder/MonetDB (Query)

titan03.inf.uni-konstanz.de - PuTTY

```
13:49:11 rittinge@titan03:/local_tmp/rittinge/pathfinder/Linux> ./run-pf



let $auction := doc("auction.xml") return
count(
  for $i in $auction/site/closed_auctions/closed_auction
  where $i/price/text() >= 40
  return $i/price
)
```


XQuery
query
Pathfinder
Parser
Sem. Analysis
Core Translation
Typechecking
Simplification
Optimization
Core to MIL Translation
result
MonetDB
Pathfinder runtime module
MIL
Database

Pathfinder/MonetDB (MIL and Output)

Pathfinder/MonetDB: A High-Performance Relational Runtime for XQuery

titan03.inf.uni-konstanz.de - PuTTY

```
758 | iter_oidMap := nil_oid_oid;
759 | iter := oid_oidMap.leftfetchjoin(outer001);
760 | oid_oidMap := nil_oid_oid;
761 | # item := item;
762 | # kind := kind;
763 | } # end of mapBack ()
764 | # cleanUpLevel ()
765 | inner001 := nil_oid_oid;
766 | outer001 := nil_oid_oid;
767 | loop001 := nil_oid_oid;
768 | v_vid001 := nil_oid_oid;
769 | v_iter001 := nil_oid_oid;
770 | v_pos001 := nil_oid_oid;
771 | v_item001 := nil_oid_oid;
772 | v_kind001 := nil_oid_int;
773 | }  # end of for-translation
774 | } # end of evaluate_join
775 | { # translate fn:count (item*) as integer
776 | var iter_count := {count}(iter.reverse(),loop000.reverse());
777 | iter_count := iter_count.reverse().mark(0@0).reverse();
778 | var ins_vals := iter_count.reverse().mark(nil).reverse();
779 | int_values := int_values.seqbase(nil).insert(ins_vals).seqbase(0@0);
780 | ins_vals := nil_oid_int;
781 | item := iter_count.leftjoin(int_values.reverse());
782 | item := item.reverse().mark(0@0).reverse();
783 | iter_count := nil_oid_int;
784 | iter := loop000.reverse().mark(0@0).reverse();
785 | kind := iter.project(INT);
786 | } # end of translate fn:count (item*) as integer
787 | print_result(xml,ws,item,kind,int_values,dbl_values,dec_values,str_values);

<?xml version="1.0" encoding="utf-8"?>
<XQueryResult>
30
</XQueryResult>
13:49:15 rittinge@titan03:/local_tmp/rittinge/pathfinder/Linux>
```

Motivation

Conclusion (Summer School GK - St. Vigil):

- almost every feature of XQuery is supported
- 18 of 20 Xmark queries are running with generated MIL code

but ...

- **performance problems (mostly because of iterative translations)**

  ➔ solution: intelligent loop-lifted version
- a lot of work has still to be done
  (e.g. built-in functions, ...)

Challenges

- mapping makes extensive use of sort operators (mostly indirect by a row numbering operator)
- concepts have to be `loop lifted'
  e.g. path steps, functions, …
  (instead of simple values, sets of values for multiple iterations have to be managed)
- for-loop nesting implicitly creates cartesian products

Idea: sorting

- try to use only order preserving operations
  ⇨ some additional operators are needed
  (e.g., ``merged union'' instead of union)
- keep more order properties (secondary orderings)
  ⇨ only refine sorting

⇨ we can get rid of runtime order properties

Motivation: loop-lifted staircase join

example: How many people are living on each continent?

```
for $a in doc("earth.xml")//continent
    return count($a/country//people)
```

iter	item
1	1

doc("earth.xml")

iter	item
1	1
1	...
1	n

//continent

iter	item
1	1
...	1
n	1

$a

iter	item
1	1
1	...
1	m
...	...
n	1
n	...
n	m

/country

iter	item
1	1
1	...
1	p
...	...
...	...
n	1
n	...
n	p

//people

Idea: loop lifted path steps

- evaluate all iterations during one sequential scan over the document
- use stack with active iter values ⇨ multiple iterations
- maintain ideas of path step algorithms (staircase join - pruning, partitioning, skipping)

Motivation: join recognition

```
s0 [ for $a in (1, 2, 3)
     s1 [ for $b in (30, 20, 10)
          s2 [ where 10 * $a = $b
               return 42
```

independent for-loop execution

⇨ `for $a in (1, 2, 3) return 10 * $a` $\bowtie_{(=)}$ `for $b in (30, 20, 10) return $b`

⇨ intermediate results stay linear

three steps:

- join pattern
- independent execution
- join translation

Join pattern

for $*v* in e_{in}
    return if ($p(e_1,e_2)$) then e_{return} else ()

```
for $a in (1,2,3)
   for $b in (30,20,10)
   where 10 * $a = $b
   return 42
```

≡

```
for $a in (1,2,3)
   for $b in (30,20,10)
   return if (=(10*$a,$b))
          then 42
          else ()
```

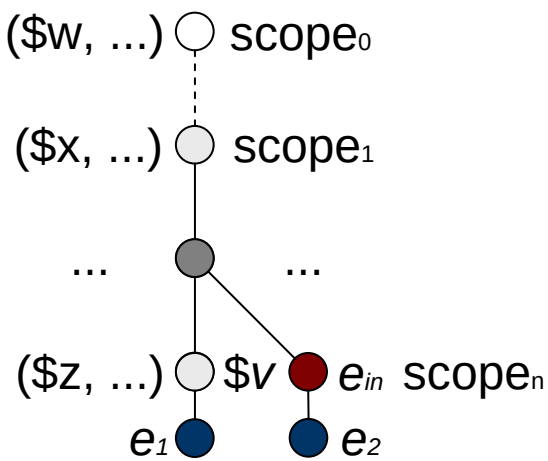
e_{in}: (30,20,10)
p: =
e_1: 10*$a
e_2: $b
e_{return}: 42

Conditions for independent execution

for $\$v$ in e_{in}
    return if ($p(e_1,e_2)$) then e_{return} else ()

three limitations:

- variable $\$v$ appears free in e_2 only
- variables occurring free in e_2 and e_{in} are bound in any enclosing scope, except for the scope that directly encloses the pattern
- p is a supported join predicate of the database



(e_1 and e_2 may be arbitrarly swapped)

Experimental results

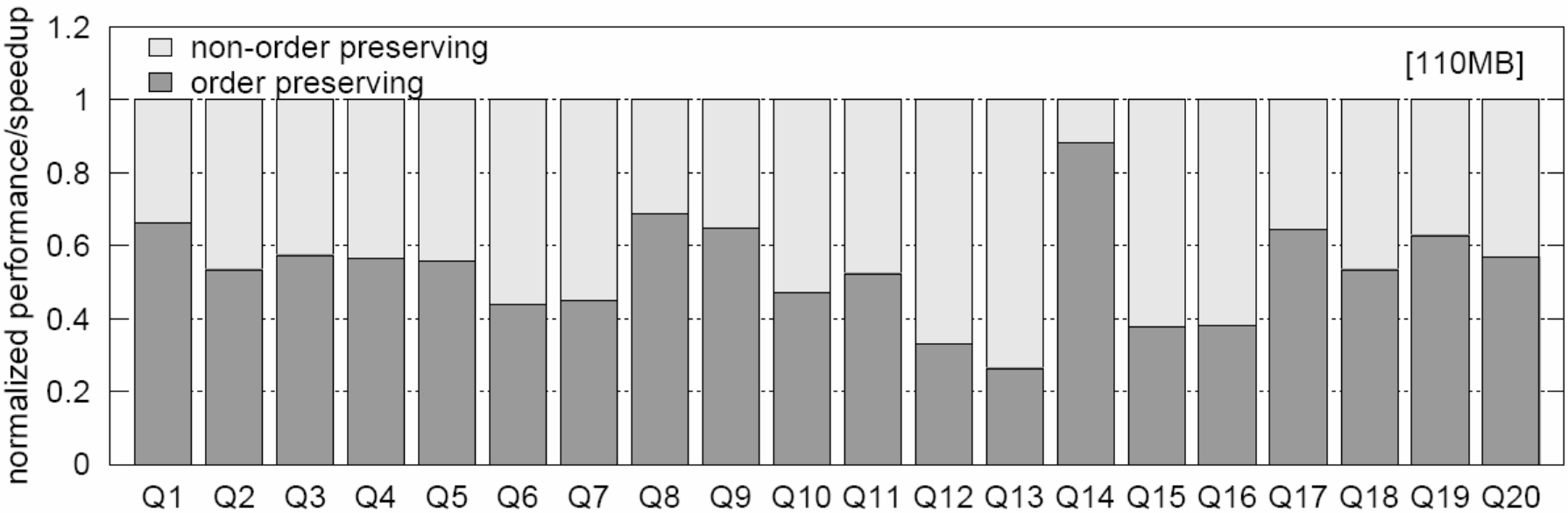
Test Set: Xmark Benchmark (11 MB to 11 GB)

Experimentation Platform: 1.6 GHz AMD Opteron 242 (1MB L2 cache), 8GB RAM

---

order awareness:

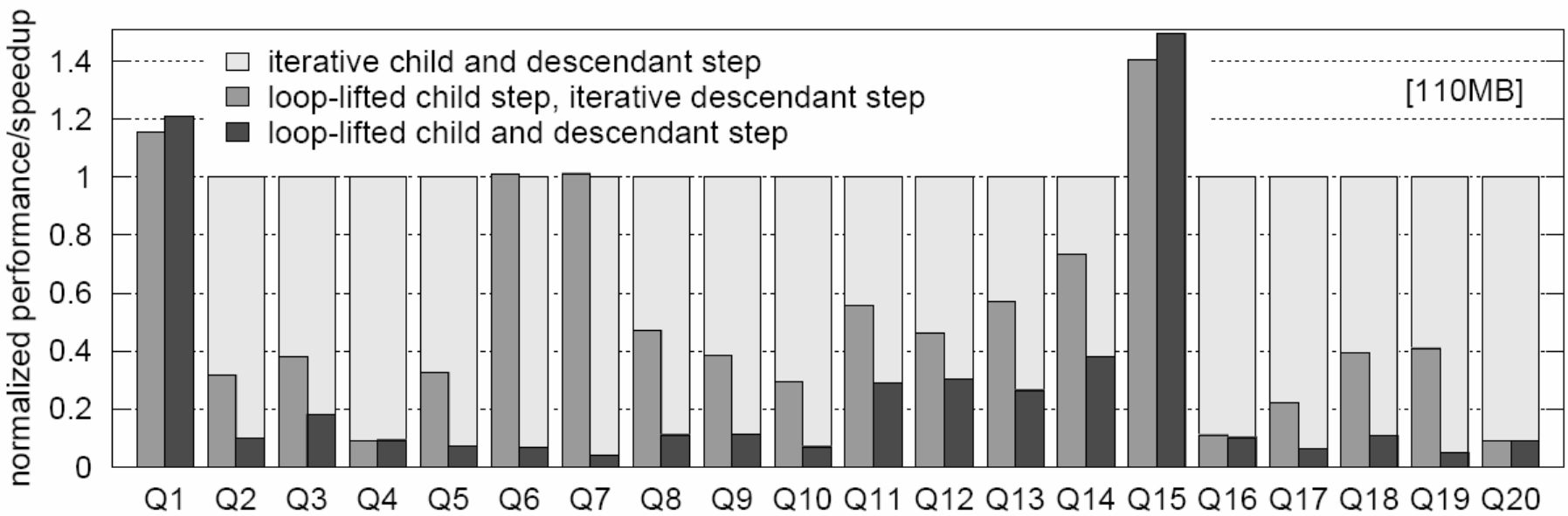
- speedup of factor 2



Experimental results

loop lifting path steps:

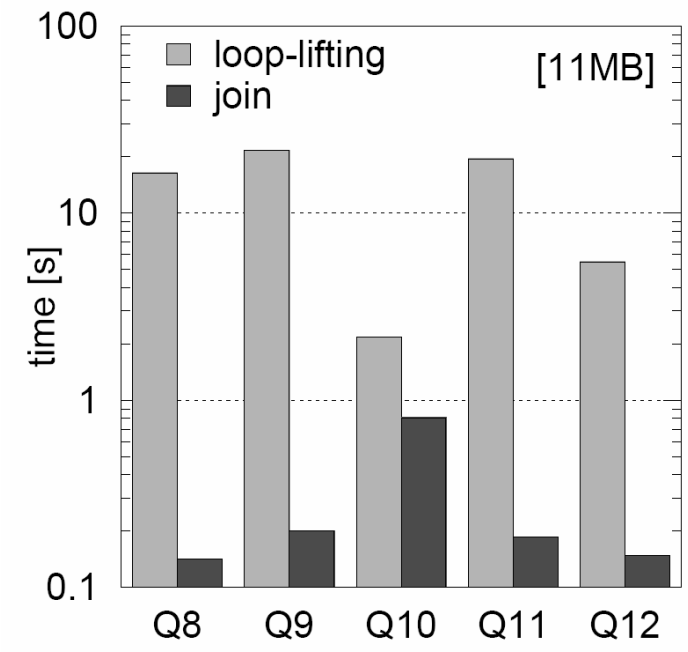
- most queries are 3 to 10 times faster
- queries, where loop lifting is not necessary, pay for additional state keeping information (Q1, Q15)



Experimental results

join recognition:

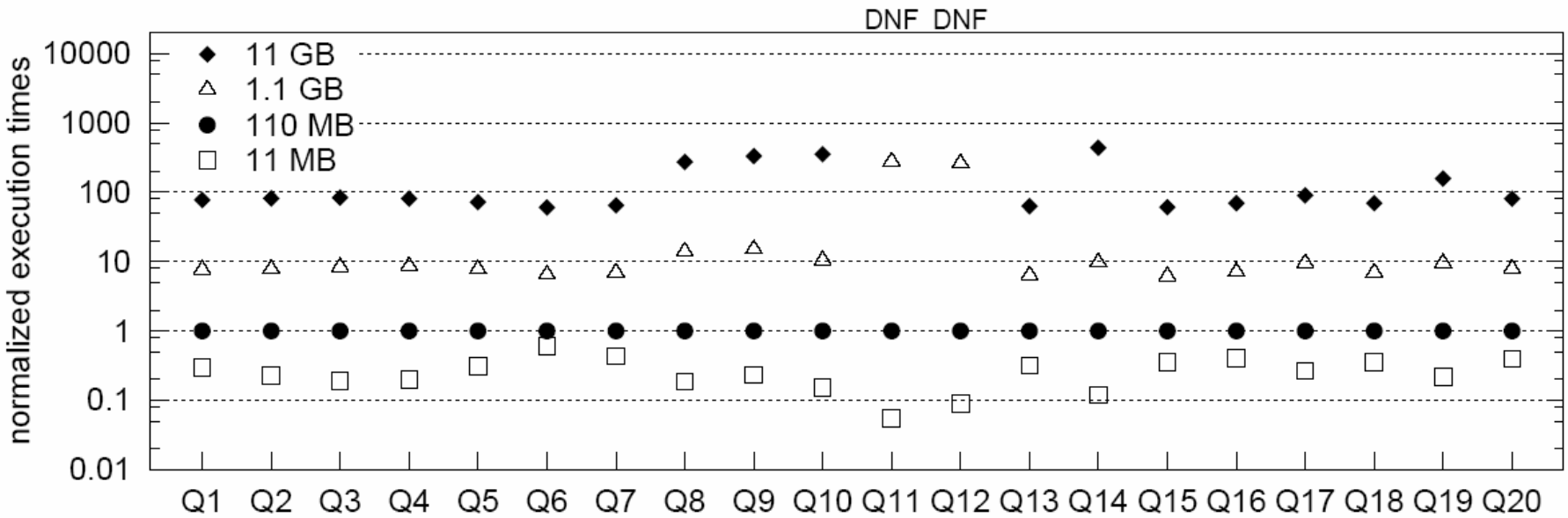
- doesn't finish 100MB and above without join detection
- speedup shows that join recognition is required for reasonable execution times



Experimental results

scaling (normalized to 100 MB):

- system scales (almost) perfectly linear
- Q11, Q12 scale quadratically: intermediate results grows with the size of the Cartesian product ('<'-comparison)



System Comparison

		10 MB			100 MB			1 GB		10 GB
operation:	Q	Galax	X-Hive	PF/M	Galax	X-Hive	PF/M	X-Hive	PF/M	PF/M
selection	1	0.07	0.37	0.05	0.72	1.29	0.17	9.9	1.29	13.15
predicates	2	0.03	0.45	0.07	0.31	1.75	0.31	33.0	2.40	25.27
comparison filter	3	0.14	0.65	0.29	1.76	5.66	1.51	25.1	12.54	126.18
document order	4	0.23	0.10	0.09	2.91	1.00	0.45	18.1	3.83	36.34
casting	5	0.05	0.13	0.05	0.63	0.90	0.16	20.7	1.23	11.53
descendant steps, aggregation (count), +	6	1.30	1.07	0.03	13.30	10.17	0.05	178.1	0.32	3.02
	7	2.69	1.57	0.03	30.01	24.84	0.07	278.4	0.48	4.51
join	8	0.17	0.85	0.14	2.12	3.51	0.76	49.1	10.44	208.28
nested joins	9	113.24	32.25	0.20	DNF	12280.66	0.87	DNF	12.99	289.45
element construction	10	1.74	5.28	0.81	18.62	442.37	5.32	DNF	55.09	1882.27
large (intermediate) results, join	11	2.63	98.91	0.19	DNF	19927.29	3.49	DNF	961.00	DNF
	12	1.44	23.39	0.15	DNF	5100.19	1.67	DNF	431.39	DNF
reconstruction	13	0.04	0.10	0.07	0.67	1.03	0.22	12.9	1.38	13.88
full text search	14	1.93	0.72	0.26	99.54	11.16	2.20	110.2	21.32	6463.22
long path (child steps)	15	0.02	0.03	0.10	0.21	0.49	0.28	10.6	1.70	16.97
empty	16	0.03	0.03	0.11	0.47	0.52	0.27	10.9	1.90	18.80
not, empty	17	0.06	0.09	0.08	0.82	0.85	0.30	11.8	2.81	26.88
UDF, function call	18	0.08	0.08	0.05	0.74	0.64	0.14	14.8	0.96	9.68
sort by	19	1.17	0.67	0.12	14.74	12.15	0.56	254.5	5.32	88.45
group by	20	0.28	0.11	0.25	2.99	1.40	0.63	24.6	4.90	50.95

Conclusion & future work

created a working system,

- which supports **arbitrary computations over XML input documents**
- which is **scalable** due to optimizations like order awareness, loop lifting, and join recognition

⇨ relational approach does work

⇨ Pathfinder/MonetDB is a promising alternative to other XQuery solutions

next steps:

- release in mid-april (under open source license)
- algebraic optimizations (e.g., join detection)