Accelerating Computationally Intensive Queries on Massive Earth Science Data

Array Databases 2011
Uppsala, 2011-mar-25

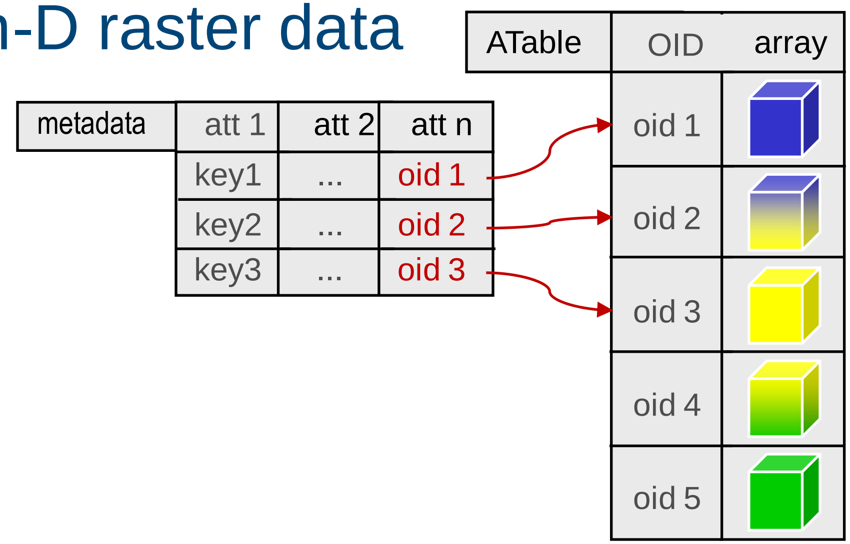
Peter Baumann
Jacobs University Bremen,
rasdaman GmbH

rasdaman

www.rasdaman.org

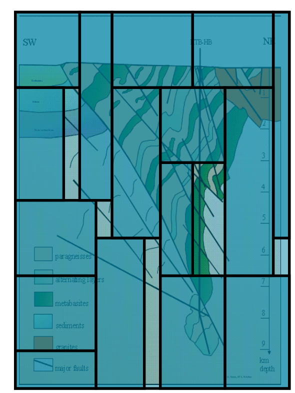
JACOBS UNIVERSITY

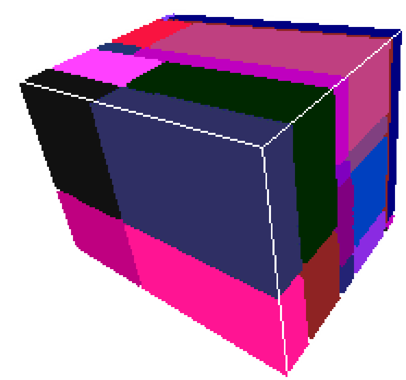
- C/S Array DBMS for massive n-D raster data
  - typed n-D arrays
  - storage & query optimization
  - In operational use



- rasql = declarative array QL
  - ```
    select img.green[x0:x1,y0:y1] > 130
    from   LandsatArchive as img
    ```

- n-D array → set of n-D tiles
  - tiles stored in DBMS blobs
  - arbitrary tiling (layout language)



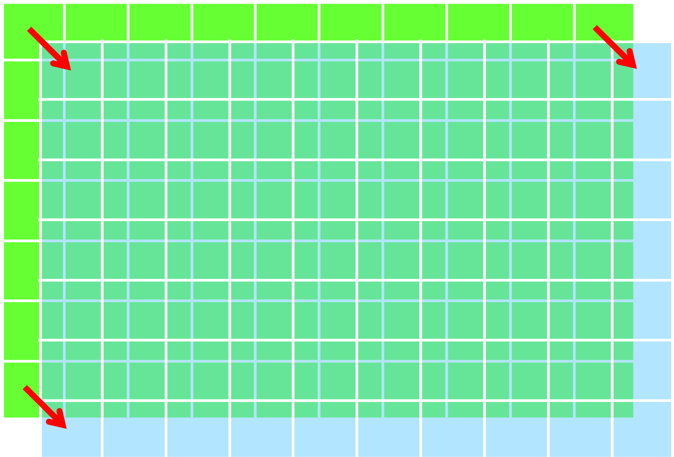

    ```

Array Operations: MARRAY

- Array constructor: $MARRAY_{X,x}(e|_x)) := \{ (x,f): f = e|_x, x \in X \}$
  - for expression $e|_x$
    potentially containing occurrences of x, of result type F
- Example: image addition
  - $a + b := MARRAY_{X,x}(a[x] + b[x]) := \{ (x,f): f = a[x] + b[x], x \in X \}$

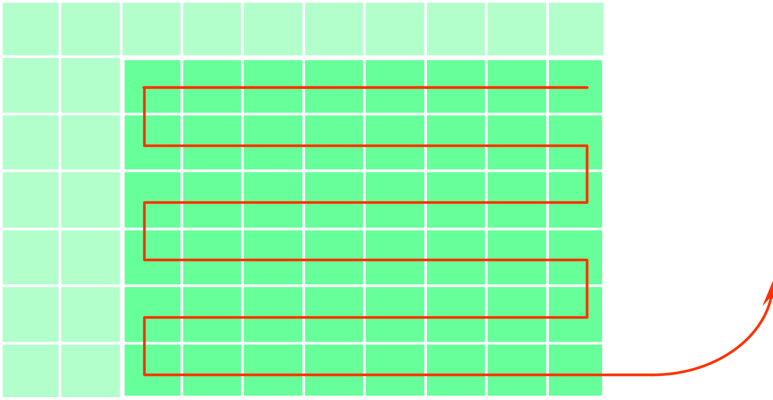
addition of pixels!

- $\rightarrow$ shorthands:
  unary and binary "induced" operations
  - *"whenever I have a pixel operation,*
    I automatically have the corresponding
    image operation"



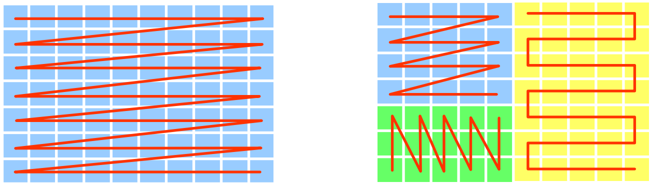
Array Operations: COND

- Condenser: $COND_{o,X,x}(e|_{a,x}) := e|_{a,p1} \;\; o \;\; e|_{a,p2} \;\; o \ldots o \;\; e|_{a,pn}$
  - x visits each coordinate in $X = \{ p_1, \ldots, p_n \}$
  - $e|_{a,pi}$ expression potentially containing a and p_i
  - o commutative: $a \; o \; b = b \; o \; a$
  - o associative: $(a \; o \; b) \; o \; c = a \; o \; (b \; o \; c)$
- Example: "*Sum over all cell values*"
  - add(a) $= COND_{+,sdom(a),x}(a[x])$
    $= a[p_1] + a[p_2] + \ldots + a[p_n]$



Why Commutative & Associative?

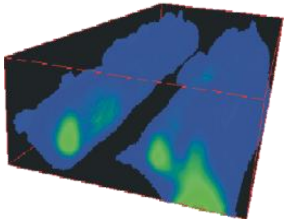
- Goal: declarative query language
  - Declarative = *express what you want, not how you get it*
  - Ex: select id from R where id < 10
    ...nothing about index usage, sequence,...
- Advantages:
  - Database user doesn‘t have to care about details
  - Optimiser gets liberty to (re-) organise query evaluation
- Example: tile-based processing:



The rasql Query Language

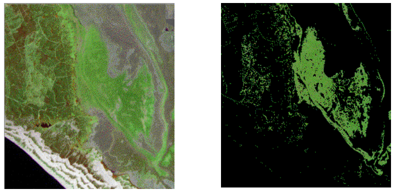
- selection & section

```
select c[ *:*, 100:200, *:*, 42 ]
from   ClimateSimulations as c
```



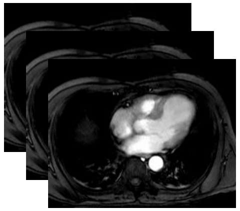
- result processing

```
select img * (img.green > 130)
from   LandsatArchive as img
```



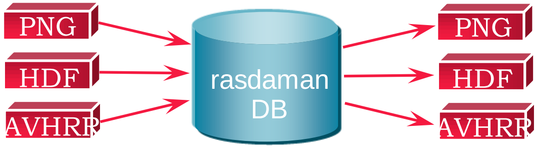
- search & aggregation

```
select mri
from   MRI as img, masks as am
where  some_cells( mri > 250 and m )
```

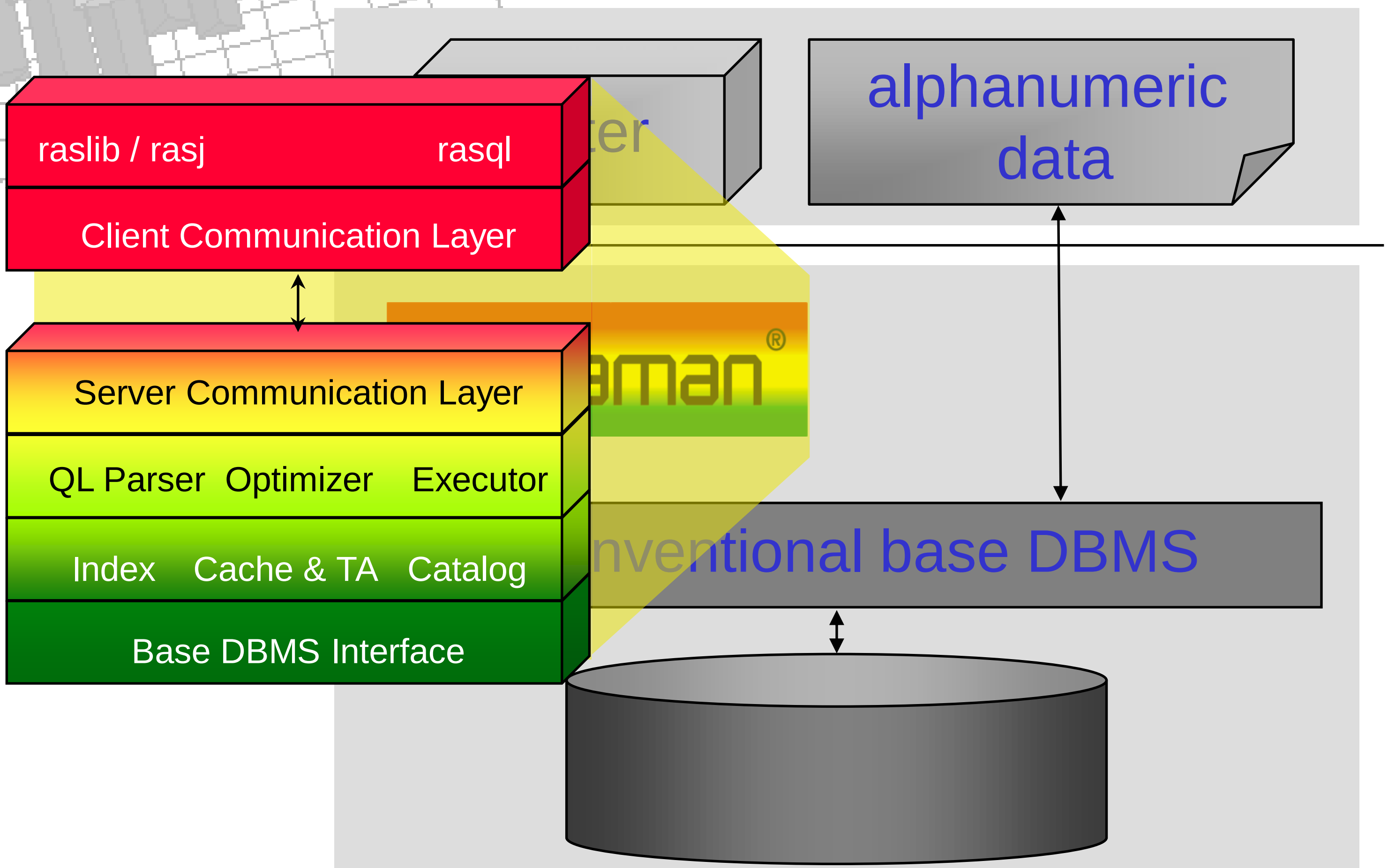


- data format conversion

```
select png( c[ *:*, *:*, 100, 42 ] )
from   ClimateSimulations as c
```



Architecture



Tile-Based Operation Evaluation

- Within tile: iteration over all relevant cells
- Conceptually:

```
for ( i0 = low0; i0 < high0; i0++)
        for ( i1 = low1; i1 < high1; i1++)
                for ( i2 = low2; i2 < high2; i2++)
                        result[i0,i1,i2] = f( left[i0,i1,i2], right[i0,i1,i2] );
```

- ...but infeasible in practice
  - Dimension and extents not known at compile time
  - Array access inefficient
- Several performance bottlenecks
  - Passing arrays to next node; iteration & increment management; operation application

Issue: Complexity

- Per pixel dozens, if not hundreds of operations
  - Query interpreted; handwritten C code 5-181 times faster [Marathe & Salem 2002]
  - Tile streaming → high control flow overhead
- 1 map client mouse click = dozens of queries
- Potentially high number of concurrent users

Issue: Complexity

Ex: 1 background, 1 bathymetry, 3 RGB = 5 layers

- www.earthlook.org


```
SELECT png(
(marray x in [0:399,0:399] values {255c,255c,255c})
overlay
((scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) < -1300)*{0c
+(-1300.000000< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]    ] ) <= -1290)*{219c,0c,172c}
+(-1289.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]    ] ) <= -1282)*{209c,0c,178c}
+(-1281.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]    ] ) <= -1275)*{199c,0c,186c}
+(-1274.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]    ] ) <= -1272.5)*{186c,0c,189c}
+(-1272.499999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]    ] ) <= -1271)*{174c,0c,194c}
+(-1270.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]    ] ) <= -1270.5)*{162c,0c,199c}
+(-1270.499999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]    ] ) <= -1270)*{150c,0c,204c}
+(-1269.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]    ] ) <= -1269.5)*{139c,0c,209c}
+(-1269.499999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]
+(-1268.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]
+(-1268.499999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]
+(-1267.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399]
+(-1267.499999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1266.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1266.499999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1265.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1265.499999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1264.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1264.499999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1263.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1263.499999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1262.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1261.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1260.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1259.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1256.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1249.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1239.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+(-1229.999999< scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ) and scale( extend( img0[269:349,0:65], [269:395,-
+ (-126.5 < scale( extend( img0[269:349,0:65], [269:395,-60:65] ), [0:399,0:399] ))*{255c,255c,255c})
overlay (scale( extend( img2[124:468,0:578], [124:717,-14:578] ), [0:399,0:399] ))
overlay (scale( extend( img3[11375:11578,0:120], [11375:11968,-473:120] ), [0:399,0:399] )) )
FROM Hakon_Bathy AS img0, Hakon_Dive1_8 AS img1, Hakon_Dive2_8 AS img2, Hakon_Dive2b_8 AS img3
```

Issue: Complexity

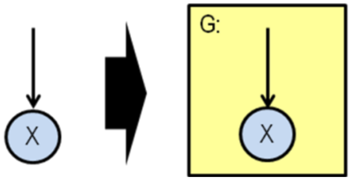
- Per pixel dozens, if not hundreds of operations
  - Query interpreted; handwritten C code 5-181 times faster [Marathe & Salem 2002]
  - Tile streaming → high control flow overhead
- 1 map client mouse click = dozens of queries
- Potentially high number of concurrent users
- ...a case for optimization
- Approach: conflate suitable query fragments, compile

JIT/1: Operator Conflation

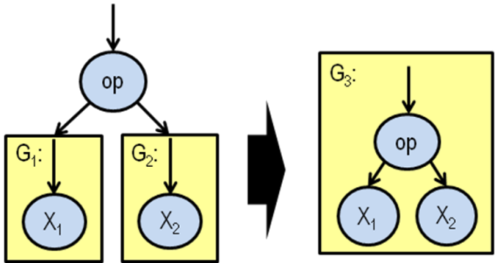
[Jucovschi, Stancu Mara]

Bottom-up recursive conflation:

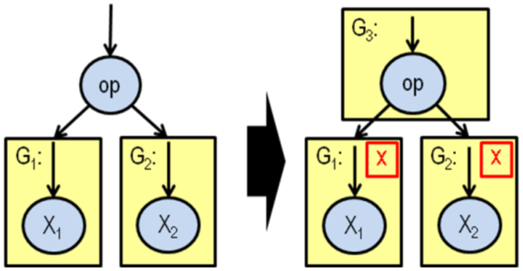
- Create group from leaf



- non-blocking inner node:
  merge parent + child groups



- blocking inner node:
  start new group



JIT/2: Dynamic Compilation

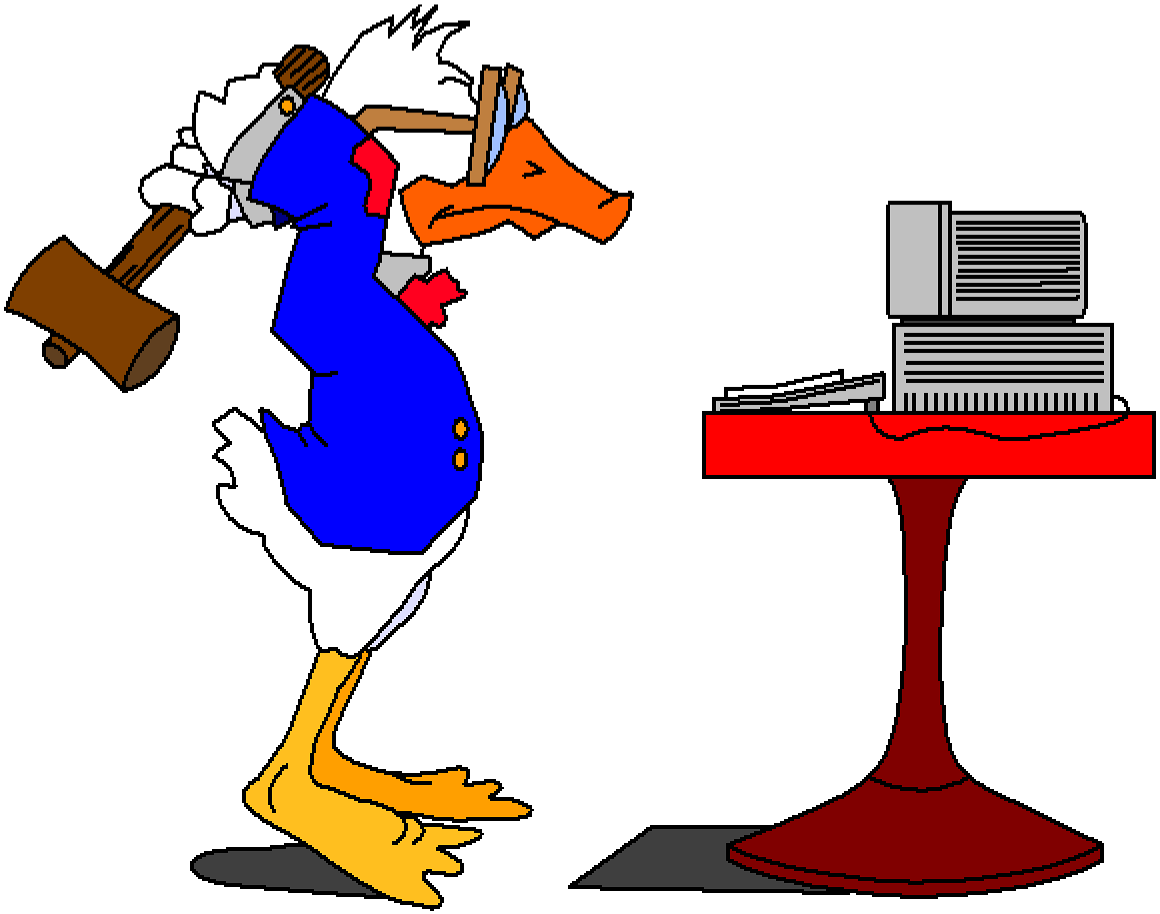
- Approach:
  - Transform conflated subtree(s) into C program
  - Compile into shared library
  - Load shared library
  - run code on tiles
  - Reuse code when similar query fragments occur

```
(T>-15 and T<0) * {10,40,100}
```

```
void process(int units,
                    void *data, void *result)
{ int iter;
  void* dataIter = data;
  void* resIter = result;
  for (iter=0;
       iter<units;
       ++iter, dataIter+=4, resIter +=3)
  { float var0 = *((float*)dataIter);
    bool c = (var0>-15) && (var0<0);
    char res_red = 10*c;
    char res_green = 40*c;
    char res_blue = 100*c;
    *((char*)resIter) = res_red;
    *((char*)resIter+4) = res_green;
    *((char*)resIter+8) = res_blue;
  }
}
```

Demo

JACOBS UNIVERSITY



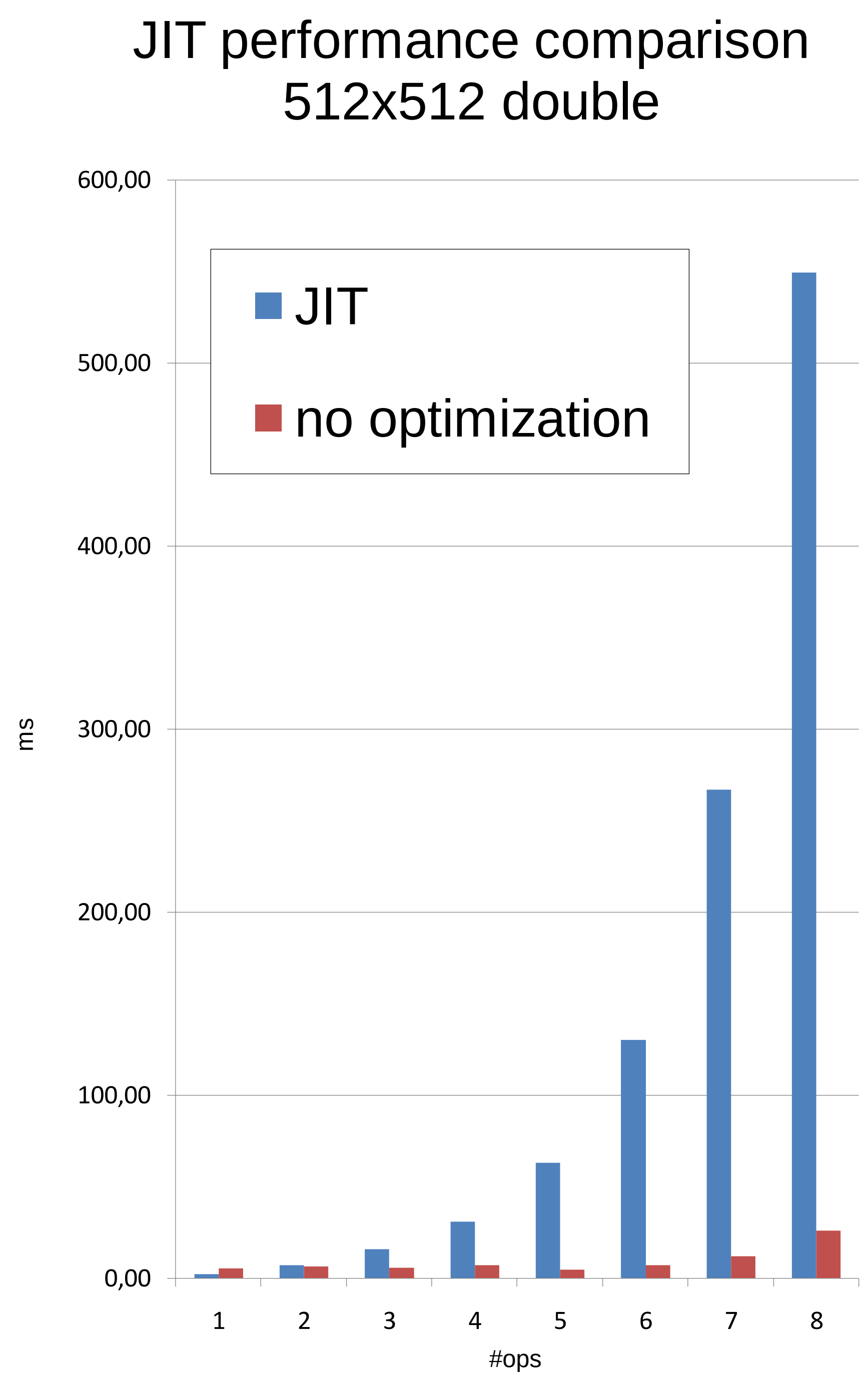
Performance Evaluation Laptop

- Tested on queries with 2^k operations

```
select x*x*...*x
from float_matrix x
```

  - k = 0..7

- Evaluated in 2 scenarios:
  - Unoptimized
  - JIT
- Measured: processing time



State of the Art

- loop fusion in super computing [Gao et al. 1992]

...we do it runtime

- merging of operators common on physical level (DB2, Oracle...)

...more dynamic & flexible

- extensible databases [Ravada & Sharma 1999, Oracle]

…needs expert to write code

- dynamic relational query compilation [Acheson 2004]

…we do it for array query compilation

Summary

JACOBS UNIVERSITY

- Analytics in Array DBMSs typically CPU-bound
- JIT = operator node conflation + dynamic compilation
  - Reduce iteration overheads & other drawbacks of dynamic typing
  - Speed up from native code, can exploit compiler optimization, can adapt to different architectures
- Future work
  - systematic evaluation (industry approach until now: „...but it works“)
  - SMP & other hardware
  - Forthcoming EU project EarthServer: 100 TB databases