Speeding up Array Query Processing by Just-In-Time Compilation

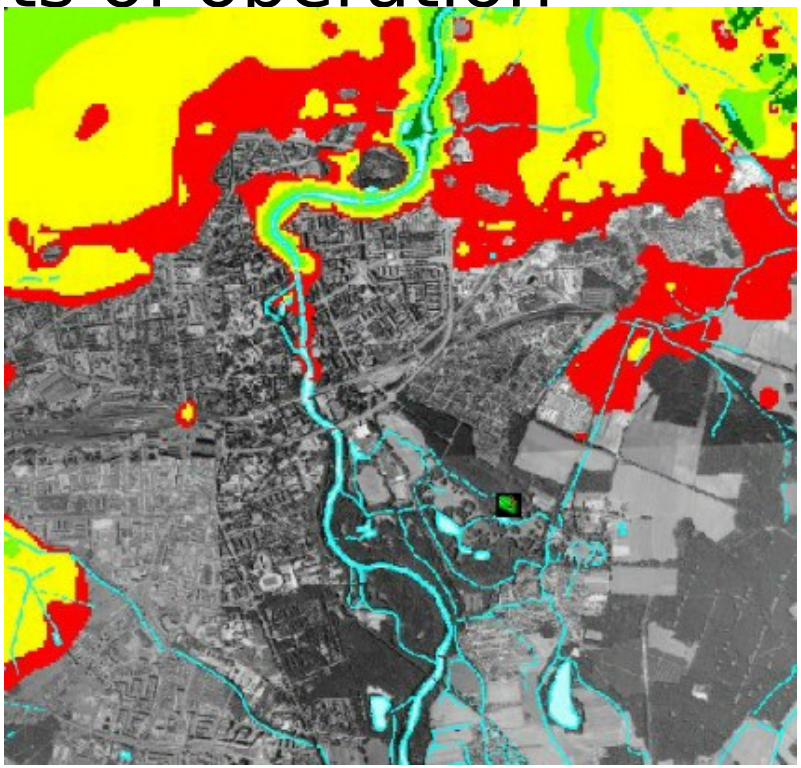
Constantin Jucovschi Peter Baumann Sorin Stancu-Mara

Big Picture

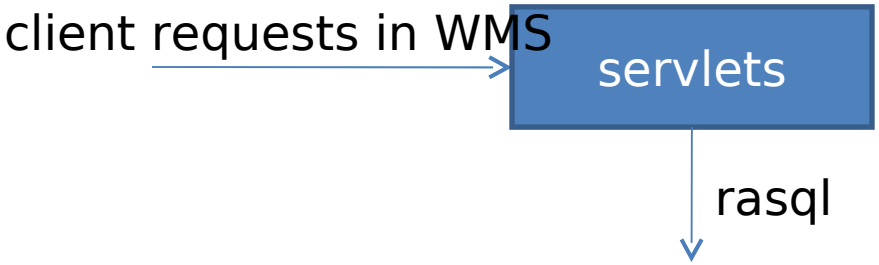
- Interpreted languages
  - Slow for complex computation
- Array DBMS
  - multidimensional array modeling and query
    - Many operations being applied to many arrays
- JIT compilation to opt interpreted array query
  - Group query nodes into complex operation nodes

scenario

- Array DB similar to interpreted language
  - ad-hoc queries with lots of operation steps
    - each operation applied elements in eval
  - Web map navigation



rasdaman




```
select png(
          scale(ap[...],[0:350,0:350])
                    * {1, 1, 1}
  overlay
          bit(scale(tm[...],[0:350,0:350]),2)
                    * {255, 0, 0}
                              )
from      AirbornePhoto as ap,
                    ThematicMap as tm
```

Bottleneck

- simultaneously requested multi-layers →higher overhead, array DBMS CPU bound

Solution for complex query evluation

- hand-crafted code optimization in C/C++, Java
  - high performance, little interpreting overhead
  - user to implement & use stored procedures to identify potential part vs. optimizer responsiblity
- heuristic rewriting to reduce operations
  - Semantically equivalent sequence of op
  - reorder/replace/pre-calculate/join query tree nodes
- Streaming intermediate results: less query evaluation and mem
- interpreted paradigm → exploit JIT to group ops

2 “new” techs for array query

- merges atomic op nodes in the query tree into a complex op node
  - reduced management: less node switching,1 iterator instead of 1 per op
  - Not for op sequences constitutings an infinite set
- JIT node compilation:
  - Caching generated library compiled from C codes
    - Omit compile: great for massive uniform query loads
      - Map navigation

Opportunity

- Frequent highly predefined query pattern
  - surf the map
  - fetches several mosaic elements to achieve a smooth zoom and pan experience

Loop fusion

```
select avg_cells( 1.8*A + 32 ) - B
from A, B
```

Loop fusion

```
select avg_cells( 1.8*A + 32 ) - B
from A, B
```

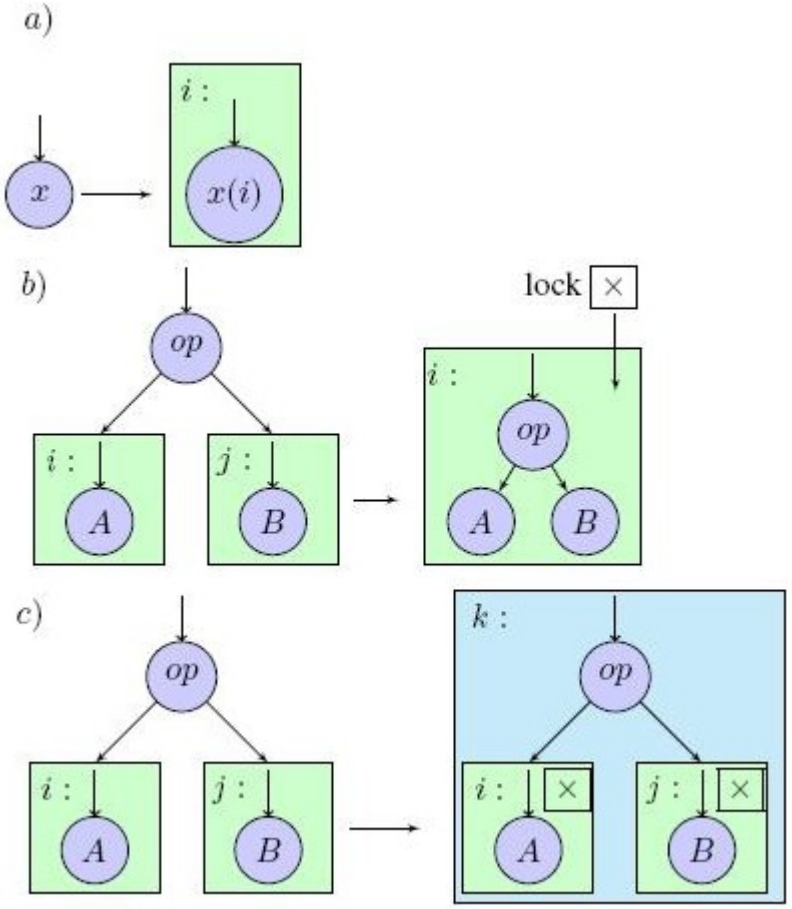
$$T = 3T_{alloc} + 4T_{iter} + 4nm\,T_{op}$$

Loop fusion

```
1. avg = 0;
2. for i in Dom(A)
3. avg += A[i]*1.8 + 32;
4. end
5. avg /= size(A);
6. for i in Dom(B)
7. result[i] = avg - B[i];
8. end
```

$$T = T_{alloc} + 2T_{iter} + 4nm\,T_{op}$$

Loop fusion



GroupIterator generation algorithm

Query fragments' dynamic compilation

- Cache compiled query part
  - WMS's fixed query structure → high hit

Query fragments' dynamic compilation

- Cache compiled query part
  - WMS's fixed query structure → high hit

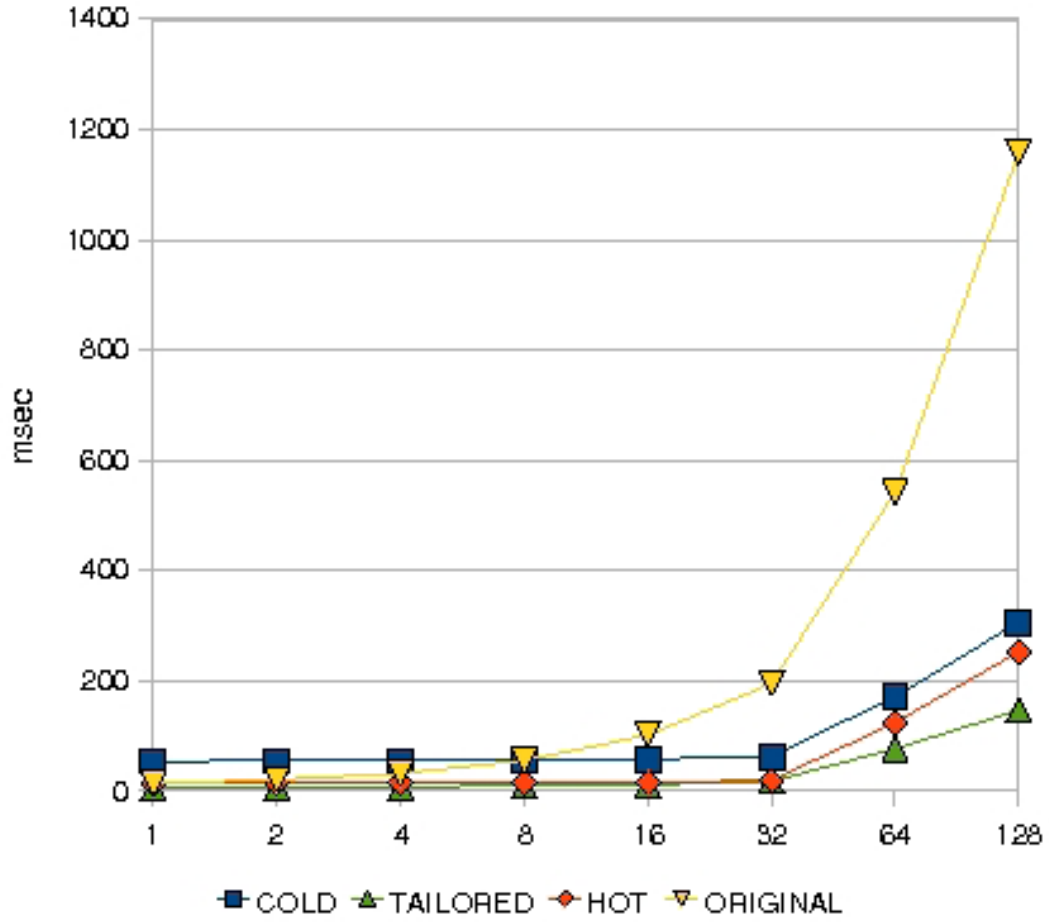
```
function genCCode(node)
{
          if (node.type == MULTIPLICATION)
          {
                    (code1, var1) = genCCode(node.child(0));
                    (code2, var2) = genCCode(node.child(1));
                    res_var = genNewVariableName;
                    code = code1 + code2;
                    code += getResultType() + " " + res_var
                              + "=" + var1 + "*" + var2 + ";";
                    return (code, res_var);
          }
          else if ...
}
```

Query fragments' dynamic compilation

- Cache compiled query part
  - WMS's fixed query structure → high hit

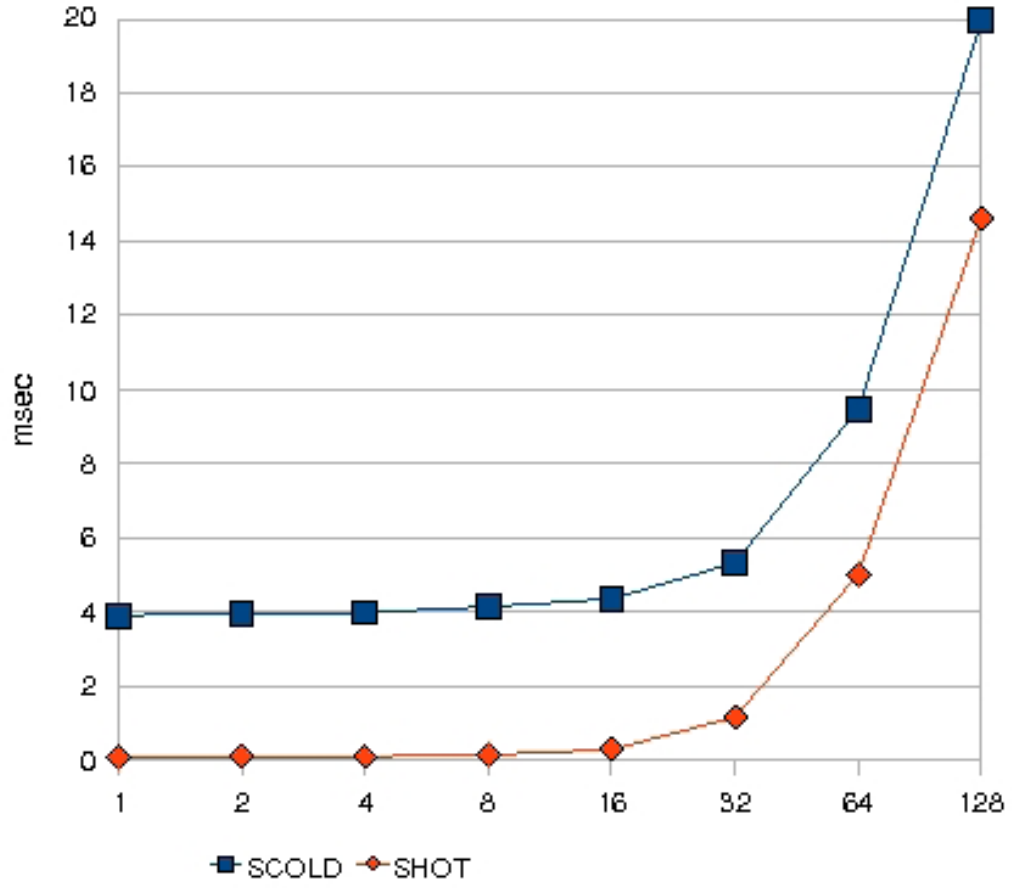
```
void process(int units, void *data,void *result)
{
          void* dataIter = data;
          void* resIter = result;
          for (int iter=0; iter<units;++iter, dataIter+=4, resIter +=12)
          {
                    float var0 = *(float*)dataIter;
                    bool c = (var0>-15) && (var0<0);
                    *((int*)resIter) = 10*c;
                    *((int*)resIter+4) = 40*c;
                    *((int*)resIter+8) = 100*c;
          }
}
```

Performance



cold: C program need to be generated & compiled prior query eval
hot: shared lib for compiled code ready for loading and execution

Standalone Performance



Standalone: query tree and data loaded. only processing time

Conclusion

- loop fusion opt
  - Less memory usage, better locality
  - Interpreter overhead for each unit cell op
- dynamic compilation for unit operations
  - Compilation and library loading overheads

Future work

- graphic card support in query evaluation
- memory operation: a piece of cake?
- Fig 3/4-->disk and network latency