Query processing for parallel languages

Brandon Myers, Mark Oskin, Bill Howe
bdmyers@cs.washington.edu
DB Day 2015
How to turn astrophysics simulation output into scientific knowledge

Using 300 processors: (circa 1995)

Step 1: Run simulation

Step 2: Analyze simulation on workstation

Step 3: Extract meaningful scientific knowledge

(happy scientist)

slide src: Jeff Gardner
How to turn astrophysics simulation output into scientific knowledge

Using 500,000 cores?: (circa 2012)

Step 1: Run simulation
(Single snapshot: 200TB)

By 2012, we will have machines that will have millions of processor cores!

Step 2: Analyze simulation on ???

slide src: Jeff Gardner
GASOLINE (a cosmology N-body code) required 10 person-years of development but code is reused for many years by many people

Data analysis code less reuse, ad hoc queries
global List[int] histogram[N] block;
forall p in Particles {
    atomic {
        if not histogram[h(p.z)] {
            histogram[h(p.z)] = new List[int];
        }
        histogram[h(p.z)].find(p.z) += 1;
    }
}

val histogram = select count(*) , z from Particles group by z
Our solution

Using 500,000 cores?:
(circa 2012)

queries

HPC stack

memory

Step 1: Run simulation

(Single snapshot: 200TB)

By 2012, we will have machines that will have millions of processor cores!
Our solution

**Using 500,000 cores?:**
_(circa 2012)_

- queries
- PGAS
- memory

---

**Step 1:** Run simulation

(Single snapshot: 200TB)

**By 2012, we will have machines that will have millions of processor cores!**
Integrate queries into PGAS languages
Integrate queries into PGAS languages
Partitioned global address space (PGAS) languages
Example PGAS program

global Particle P[N] block;
global int histogram[N] block;
forall pj in P {
    atomic histogram[pj.z] += 1;
}

```
[Image: Diagram of PGAS program]
```
Integrate queries into PGAS languages
Integrate queries into PGAS languages
Radish: queries in PGAS

SQL

Relational Algebra++

PGAS physical algebra

mapping?

PGAS languages
Grappa
Chapel
X10
Approaches for query processing in PGAS

**demand-driven data flow (iterators)**

- Scan(Order)
- Select [o_totalprice > 20]
- Shuffle[o_custkey]
- HashTable
- AggregateScan
- HashTableJoinProbe [o_custkey=c_custkey]
- Project [cnt, c_name]
- HashTableJoinProbe [o_custkey=c_custkey]

**query compilation**

- Scan(Customer)
- Project [c_custkey, c_name]
- Shuffle[c_custkey]
- HashTableJoin
- Build[c_custkey]
- Shuffle[c_custkey]
- Project [c_custkey, c_name]
- HashTableJoin
- Build[c_custkey]

```python
forall c in Customer
    T1 t1
    t1.id = t0.custkey
    t1.c_name = t0.c_name
    on partition(join_table[h(t1.c_custkey)])
    atomic {
        join_table[h(t1.c_custkey)].append(t1)
    }
```

- codegen
- optimize/compile
- binary
Generation of tuple-centric parallel code

global Tuple Customer block;

forall c in Customer
  T1 t1
  t1.id = t0.custkey
  t1.c_name = t0.c_name
Generation of tuple-centric parallel code

- **Scan(Customer)**
- **Project**
  
  `[c_custkey, c_name]`
- **Shuffle[c_custkey]**
- **HashTableJoin**
  
  **Build[c_custkey]**
- **Join**
  
  ...local join
- **Hash partitions**
  
  P0
- **Join**
  
  ...local join
- **Hash partitions**
  
  P1
- **Join**
  
  ...local join
- **Hash partitions**
  
  P2
Generation of tuple-centric parallel code

```python
global Tuple Customer block;
global Cell jointable block;
forall c in Customer
    T1 t1
    t1.id = t0.custkey
    t1.c_name = t0.c_name
    on partition(jointable[h(t1.c_custkey)])
    atomic {
        jointable[h(t1.c_custkey)].append(t1)
    }
```
Generation algorithm

Runtime interpretation

```c++
class Operator {
    // get the next tuple
    next(T&): bool
}
```

Code generation

```c++
class Operator {
    // emit code for producing
    // a tuple
    produce(state&): unit

    // emit code for consuming
    // a tuple
    consume(state&, inputTuple): string
}
```
Generation algorithm

Code generation

```c
class Operator {
    // emit code for producing
    // a tuple
    produce(state&) : unit
    // emit code for consuming
    // a tuple
    consume(state&, inputTuple) : string
}
```
Integrate queries into PGAS languages
Integrate queries into PGAS languages
Experiment #1: conventional approach vs. Radish

![Diagram showing speedup over iterators for queries (TPC-H) with two methods: Radish and Radish-sym.]
Experiment #1: conventional approach vs. Radish

![Bar chart showing speedup over iterators for queries in TPC-H. The chart compares conventional approach to Radish and Radish-sym.](chart.png)
Experiment #2: vs Impala

Query (TPC-H)

Runtime (s)

0 20 40 60 80 100 120 140 160

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

- Radish
- Impala (codegen)
- Impala (no codegen)
Experiment #2: vs Impala

TPC-H
100GB database

Runtime (s)

Query (TPC-H)

Radish  Impala (codegen)  Impala (no codegen)
Compilation time is high
Conclusion

get the code!

RACO+RADISH: github.com/uwescience/raco
GRAPPA: grappa.io
Attribution

Created by Creative Stall from the Noun Project

Created by mantisshrimpdesign from Noun Project

Created by Ruud Smit from Noun Project