RPLSH FACT SHEET

July 2019

Rplsh verbs

show
set
annotate
rewrite
optimize
history
import
gencode
expand
add
load

Rplsh patterns

seq
source
drain
comp
pipe
farm
map
reduce

Rplsh install

- sudo apt install libreadline-dev
- git clone https://github.com/Murray1991/rplsh.git
- cd rplsh
- ./install.sh
- export PATH=`pwd`/build:$PATH
- rplsh

Define pattern expressions

```
rplsh> s1 = seq()
rplsh> annotate s1 with servicetime 10
response: annotated!
rplsh> s2 = seq(20)
rplsh> p = pipe(s1,s2)
rplsh> show p
[0] : pipe(s1,s2)
rplsh> set emitter_time with 1
rplsh> set collector_time with 1
rplsh> rewrite p with farmintro
rplsh> show p
[0] : farm(pipe(s1,s2)) with [ nw: 1]
rplsh> optimize p with farmopt
rplsh> show p
[0] : farm(pipe(s1,s2)) with [ nw: 20]
rplsh> history
0    s1 = seq()
1    annotate s1 with servicetime 10
2    s2 = seq(20)
3    p = pipe(s1,s2)
...
7    rewrite p with farmintro
8    show p
9    optimize p with farmopt
10    show p
rplsh>
```

Apply all known rules

```
rplsh> s1 = seq(10)
rplsh> s2 = seq(20)
rplsh> p  = pipe(s1,s2)
rplsh> show p
[0] : pipe(s1,s2)
rplsh> rewrite p with allrules
rplsh> show p
[0] : pipe(s1,s2)
[1] : pipe(s1,farm(s2) with [ nw: 1])
[2] : pipe(farm(s1) with [ nw: 1],farm(s2) with [ nw: 1])
[3] : comp(s1,s2)
[4] : farm(pipe(s1,s2)) with [ nw: 1]
[5] : pipe(farm(s1) with [ nw: 1],s2)
rplsh> set emitter_time with 1
rplsh> set collector_time with 1
rplsh> set resources with 16
rplsh> optimize p with farmopt
rplsh> show p by resources, servicetime
1    30.000000    [3] : comp(s1,s2)
2    20.000000    [2] : pipe(s1,s2)
13    20.000000    [0] : pipe(farm(s1) with [ nw: 10],s2)
23    10.000000    [1] : pipe(s1,farm(s2) with [ nw: 20])
34    1.000000    [4] : pipe(farm(s1) with [ nw: 10],farm(s2)
with [ nw: 20])
42    1.000000    [5] : farm(pipe(s1,s2)) with [ nw: 20]
rplsh> optimize p with maxresources
rplsh> show p by resources, servicetime
1    30.000000    [4] : comp(s1,s2)
2    20.000000    [2] : pipe(s1,s2)
13    20.000000    [3] : pipe(farm(s1) with [ nw: 10],s2)
16    2.500000    [5] : pipe(farm(s1) with [ nw: 4],farm(s2)
with [ nw: 8])
16    2.857143    [0] : farm(pipe(s1,s2)) with [ nw: 7]
16    10.000000    [1] : pipe(s1,farm(s2) with [ nw: 13])
rplsh>
```

Refactoring rules

farmintro
farmelim
pipeintro
pipeelim
pipeassoc
compassoc
mapofcomp
compofmap
mapofpipe
pipeofmap
mapelim
reduceelim
mapmapelim
farmfarmelim
compdel
pipedel

```
rplsh> p = pipe(s1,s2)
rplsh> rewrite p with farmintro
rplsh> show p
[0] : farm(pipe(s1,s2)) with [ nw: 1]
rplsh> annotate p with pardegree 5
response: annotated!
rplsh> show p
[0] : farm(pipe(s1,s2)) with [ nw: 5]
rplsh>
```

Optimization rules

farmopt
pipeopt
mapopt
reduceopt
maxresources
twotier
farmfarmopt
mapmapopt

```
rplsh> s1 = seq(10)
rplsh> s2 = seq(10)
rplsh> set emitter_time with 1
rplsh> set collector_time with 1
rplsh> p = pipe(s1,s2)
rplsh> m = farm(p)
rplsh> show m
[0] : farm(p) with [ nw: 1]
rplsh> set resources with 16
rplsh> optimize m with farmopt
rplsh> show m
[0] : farm(p) with [ nw: 10]
rplsh> optimize m with maxresources
rplsh> show m
[0] : farm(p) with [ nw: 7]
rplsh>
```

Non functional parameters

servicetime
latency
pardegree
compltime*
resources*
datap**

```
rplsh> annotate pat with latency 100
rplsh> show pat by servicetime, resources

* only with show verb
** only with annotate verb
```

Environmental parameters

emitter_time
collector_time
scatter_time
gather_time
dimension
inputsize
resources
Arch

```
rplsh> set emitter_time with 5
```

Gencode templates

```
template <typename Tout, typename Tin>
class seq_wrapper {
public:
  virtual Tout compute(Tin& input) = 0;
  // user should implement also the following methods
  // in order to generate data paralel skeletons
  // for map:    type_out op(type_in t)
  // for reduce: type op(type t1, type t2)
  // otherwise it will not compile
};

template <typename Tout>
class source {
public:
  virtual bool has_next() = 0;
  virtual Tout* next() = 0;
};

template <typename Tin>
class drain {
public:
  virtual void process(Tin * x) = 0;
};

/*template <typename T>
class composable {
public:
    virtual T&& operator[](size_t idx);
    virtual T& operator[](size_t idx);
}*/

// class type representing a set of values of type T
// that could be splitted in several sub-composable sets
// -> similiar concept to the TBB Range
template <typename T>
class composable {
public:
    virtual bool is_empty() const       = 0;
    virtual std::size_t size() const    = 0;
    virtual bool is_splittable() const  = 0;
};
```

Code generation

```
rplsh> import "business.hpp"
importing from:
/usr/local/rplsh/business/simple/busi
ness.hpp
importing vec_source
importing float_drain
importing inc
importing square
importing redplus
rplsh> kernel = pipe(inc, square)
rplsh> main =
pipe(vec_source,kernel,float_drain)
rplsh> gencode main
-- ff1.cpp
rplsh>
```