Cilk Plus Reducers

Albert DeFusco

April 5, 2015

https://github.com/AlbertDeFusco/goldRush

Parallel Gold Sifting

- each pan can sift a constant amount of dirt/day
- more pans means more dirt sifted

- each pan sifts independently
- each pan has a definite amount of dirt to sift

- sifting is parallelizable

https://github.com/AlbertDeFusco/goldRush

Serial Gold Sifting

```
1   #include <list>
2   class pan
3   {
4     public:
5       pan();            //create array of random integers
6       bool hasGold();   //calls sift; true if sift returns >0
7       int sift();       //returns frequency of 79 (gold)
8   }
9
10  int main()
11  {
12    std::list<int> withGold;
13    pan* myPans = new pan[nPans];
14
15    for(int i=0; i<nPans; ++i)
16    {
17      bool gold = myPans[i].hasGold();
18      if(gold) {
19        withGold.push_back(i);
20      }
21    }
22    list<int>::const_iterator iterator;
23    for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
24      cout << *iterator << "  ";
25    cout << endl;
26
27    return 0;
28  }
```

https://github.com/AlbertDeFusco/goldRush

Parallel Gold Sifting

```
1   #include <list>
2   class pan
3   {
4     public:
5       pan();            //create array of random integers
6       bool hasGold();   //calls sift; true if sift returns >0
7       int sift();       //returns frequency of 79 (gold)
8   }
9
10  int main()
11  {
12    std::list<int> withGold;
13    pan* myPans = new pan[nPans];
14
15    cilk_for(int i=0; i<nPans; ++i)
16    {
17      bool gold = myPans[i].hasGold();
18      if(gold) {
19        withGold.push_back(i);
20      }
21    }
22    list<int>::const_iterator iterator;
23    for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
24      cout << *iterator << "  ";
25    cout << endl;
26
27    return 0;
28  }
```

https://github.com/AlbertDeFusco/goldRush

Parallel Gold Sifting

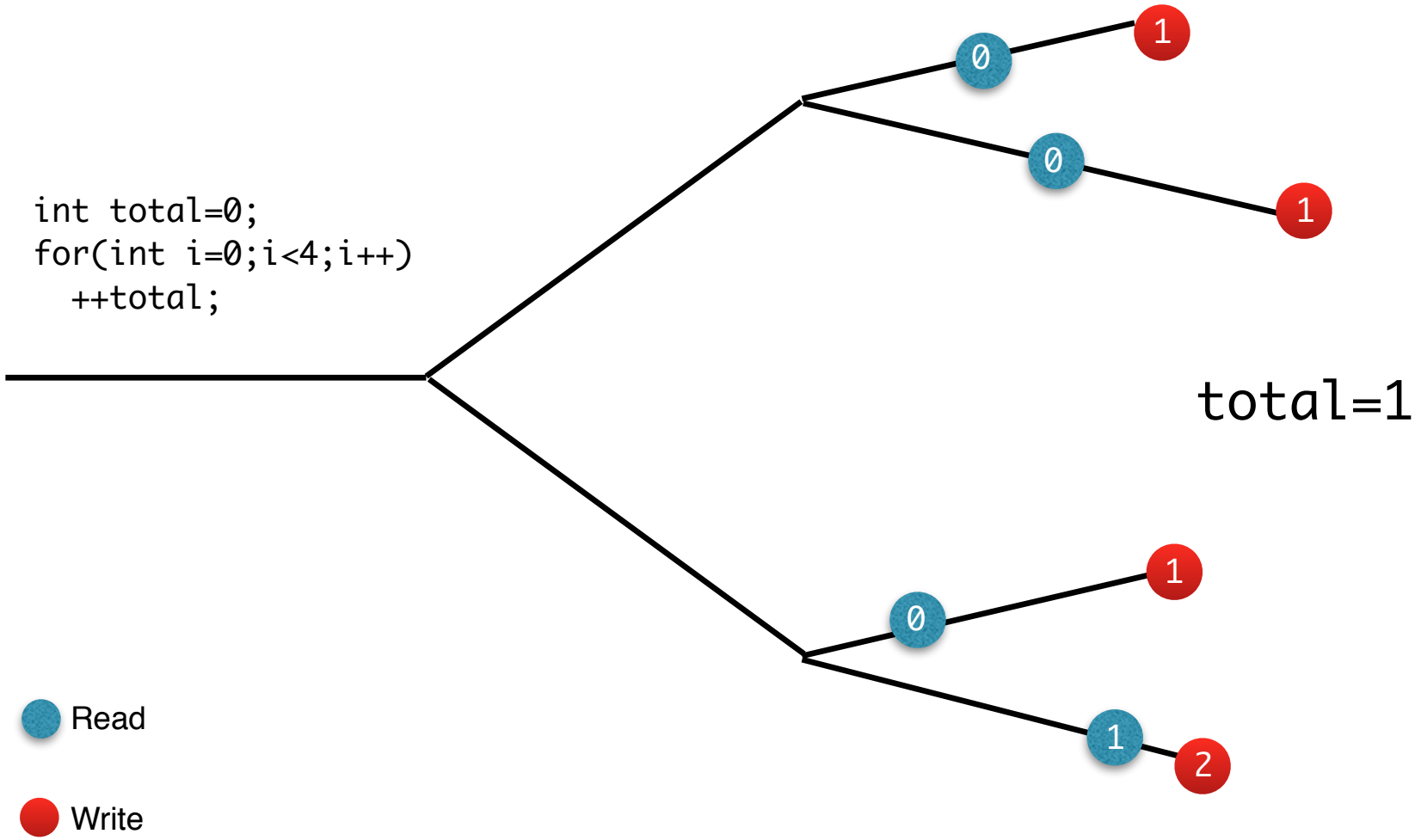
```
1   #include <list>
2   class pan
3   {
4     public:
5       pan();            //create array of random integers
6       bool hasGold();   //calls sift; true if sift returns >0
7       int sift();       //returns frequency of 79 (gold)
8   }
9
10  int main()
11  {
12    std::list<int> withGold;
13    pan* myPans = new pan[nPans];
14
15    cilk_for(int i=0; i<nPans; ++i)
16    {
17      bool gold = myPans[i].hasGold();
18      if(gold) {
19        withGold.push_back(i);
20      }
21    }
22    list<int>::const_iterator iterator;
23    for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
24      cout << *iterator << "  ";
25    cout << endl;
26
27    return 0;
28  }
```

https://github.com/AlbertDeFusco/goldRush

Thread Safety

- Unsafe operations
  - Multiple threads accessing the same address
    - Basic types are not thread safe
    - STL types are not thread safe
  - Threads read and write memory at undetermined times
  - Leads to a race condition

Race Condition



https://github.com/AlbertDeFusco/goldRush

Inefficient solutions

- lock
- mutex
- cannot use `cilk_sync` in the loop
  - will only sync child threads, not all threads
- break the loop; requiring more storage

```
1  double *sum = new double[N];
2  cilk_for(int i=0;i<N;++i)
3    sum[N] = f(N)
4  double total=0.0;
5  for(int i=0;i<N;++i)
6    total+=sum[N];
```

https://github.com/AlbertDeFusco/goldRush

Cilk Reducers

- Any associative operation is a valid reducer

$$x \; OP \; y = y \; OP \; x \tag{1}$$

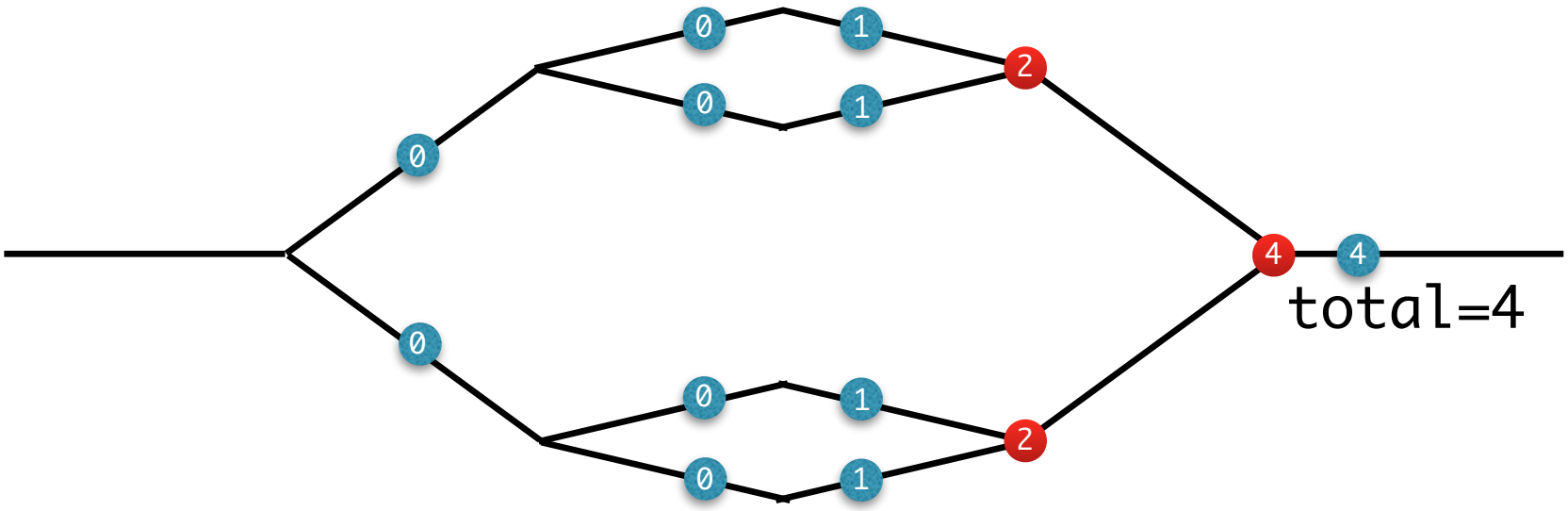
- Provide thread safe access to a “smart pointer”
- Small parallel overhead for usage
- Very extensible
- Operations are guaranteed to execute in the same order as in serial

Cilk Reducers: views

- At spawn each strand gets a private “view” of the reducer
- When strands merge
  - “views” are combined by *OP*
  - The combined “view” is given to the exit thread

Cilk Reducers: views

```
int total=0;
cilk::reducer<cilk::<op_add> red_total (0);
for(int i=0;i<4;++i)
  *red_total += 1;
total = red_total.get_value();
```



Private View

Merge update

https://github.com/AlbertDeFusco/goldRush

Find pans with gold

```
1   #include <cilk/cilk.h>
2   #include <cilk/reducer_list.h>
3
4     std::list<int> withGold;
5     cilk::reducer< cilk::op_list_append<int> > reducer_withGold;
6     pan* myPans = new pan[nPans];
7
8     cilk_for(int i=0; i<nPans; ++i)
9     {
10      bool gold = myPans[i].hasGold();
11      if(gold) {
12        reducer_withGold->push_back(i);
13      }
14    }
15    withGold = reducer_withGold.get_value();
16
17    list<int>::const_iterator iterator;
18    for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
19      cout << *iterator << "  ";
20    cout << endl;
```

https://github.com/AlbertDeFusco/goldRush