Cilk Plus Reducers

Albert DeFusco

April 7, 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

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 defined amount of dirt to sift

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 defined 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 between 1 and 10,000
6       int sift();       //returns frequency of the number 79 (atomic number of gold)
7       bool hasGold();   //calls sift; returns true if sift() > 0
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    std::list<int>::const_iterator iterator;
23    for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
24      std::cout << *iterator << "  ";
25    std::cout << endl;
26
27    return 0;
28  }
```

https://github.com/AlbertDeFusco/goldRush

Gold Rush

```
Gold Rush!

  10000 total chunks of dirt
  1000 pans

  Found gold in 15 pans
  Pan IDs: 94  142  265  268  289  440  442  443  569  600  721  781  783  806  818

serial execution took 5.60495 seconds
```

https://github.com/AlbertDeFusco/goldRush

Parallel Gold Sifting

```
1  #include <list>
2  #include <cilk/cilk.h>
3  class pan
4  {
5    public:
6      pan();            //create array of random integers between 1 and 10,000
7      int sift();       //returns frequency of the number 79 (atomic number of gold)
8      bool hasGold();   //calls sift; returns true if sift() > 0
9  }
10
11 int main()
12 {
13   std::list<int> withGold;
14   Pan* myPans = new Pan[nPans];
15
16   cilk_for(int i=0; i<nPans; ++i)
17   {
18     bool gold = myPans[i].hasGold();
19     if(gold) {
20       withGold.push_back(i);
21     }
22   }
23   std::list<int>::const_iterator iterator;
24   for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
25     std::cout << *iterator << "  ";
26   std::cout << endl;
27
28   return 0;
29 }
```

https://github.com/AlbertDeFusco/goldRush

Parallel Gold Sifting

- There is a problem to be solved
  - How do we keep track of which pans have gold?

- Where does the result get stored?
- Who can access the result?

https://github.com/AlbertDeFusco/goldRush

Parallel Gold Sifting

```
 1  #include <list>
 2  #include <cilk/cilk.h>
 3  class pan
 4  {
 5    public:
 6      pan();            //create array of random integers between 1 and 10,000
 7      int sift();       //returns frequency of the number 79 (atomic number of gold)
 8      bool hasGold();   //calls sift; returns true if sift() > 0
 9  }
10
11  int main()
12  {
13    std::list<int> withGold;
14    Pan* myPans = new Pan[nPans];
15
16    cilk_for(int i=0; i<nPans; ++i)
17    {
18      bool gold = myPans[i].hasGold();
19      if(gold) {
20        withGold.push_back(i);
21      }
22    }
23    std::list<int>::const_iterator iterator;
24    for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
25      std::cout << *iterator << "  ";
26    std::cout << endl;
27
28    return 0;
29  }
```

may not be thread-safe

https://github.com/AlbertDeFusco/goldRush

Thread Safety

- Unsafe operations
  - Multiple threads accessing the same address
    - Basic types are not thread safe
    - STL containers *may* be thread safe for some operations

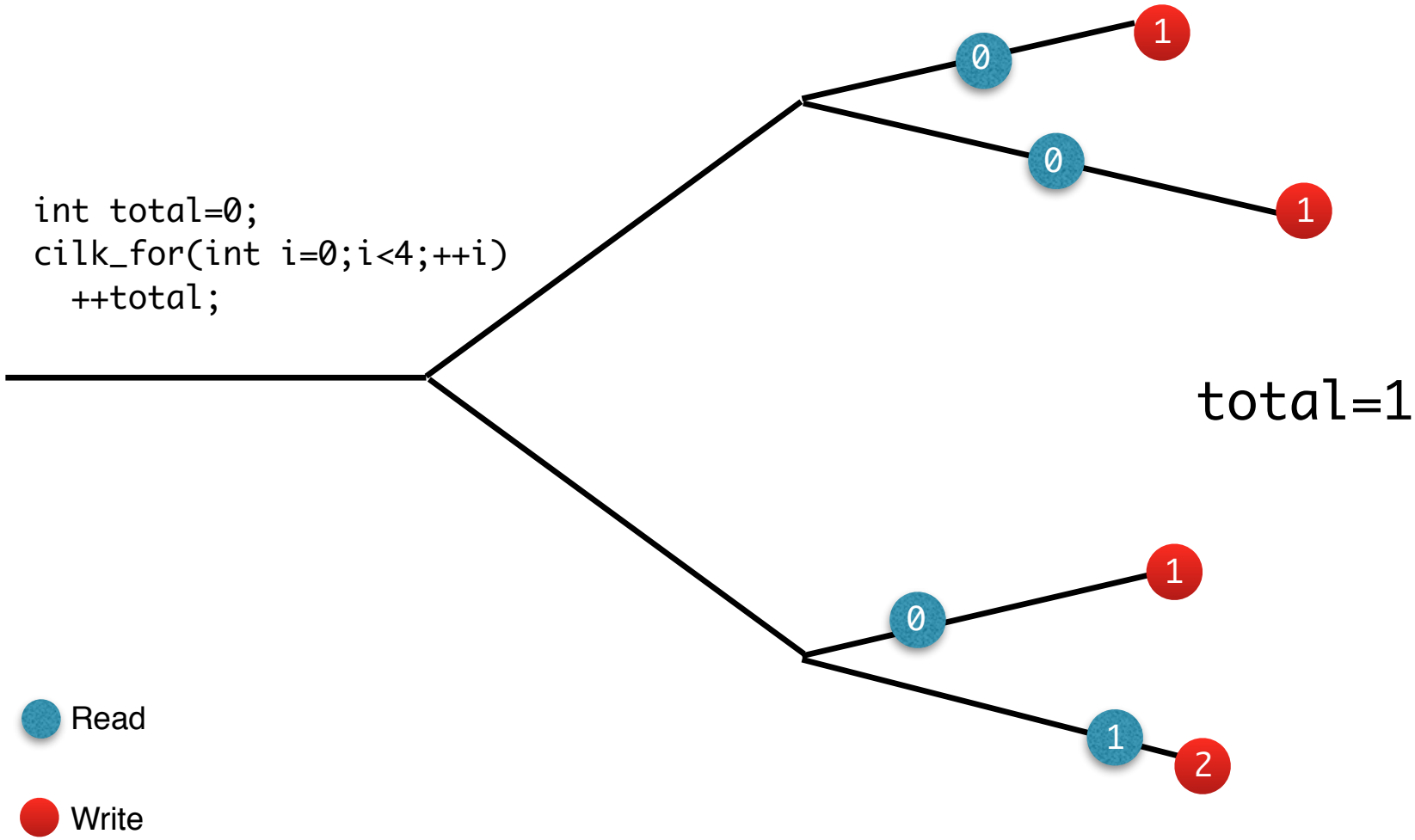
Thread Safety

- Unsafe operations
  - Multiple threads accessing the same address
    - Basic types are not thread safe
    - STL containers *may* be thread safe for some operations
- Threads read and write memory at undetermined times

Thread Safety

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

Race Condition



https://github.com/AlbertDeFusco/goldRush

Inefficient solutions

- Locking
  - Requires careful programming
  - Non deterministic
- cannot use `cilk_sync` in the loop
  - will only sync child threads, not all threads
- Break the loop
  - Requires more storage and management

```
1  #include <cilk/cilk.h>
2
3  double *sum = new double[N];
4  //parallel
5  cilk_for(int i=0;i<N;++i)
6    sum[N] = f(N);
7
8  //serial
9  double total=0.0;
10 for(int i=0;i<N;++i)
11   total+=sum[N];
```

https://github.com/AlbertDeFusco/goldRush

Cilk Reducers

- Provide thread safe access to a “smart pointer”
- Any associative operation is a valid reducer

$$x \; OP \; y \quad = \quad y \; OP \; x$$
$$(x \; OP \; y) \; OP \; (a \; OP \; b) \quad = \quad x \; OP \; y \; OP \; a \; OP \; b$$

- Small performance overhead for usage
- Very extensible in C++
- Operations are guaranteed to execute in the same order as in serial

https://github.com/AlbertDeFusco/goldRush

Cilk Reducers: views

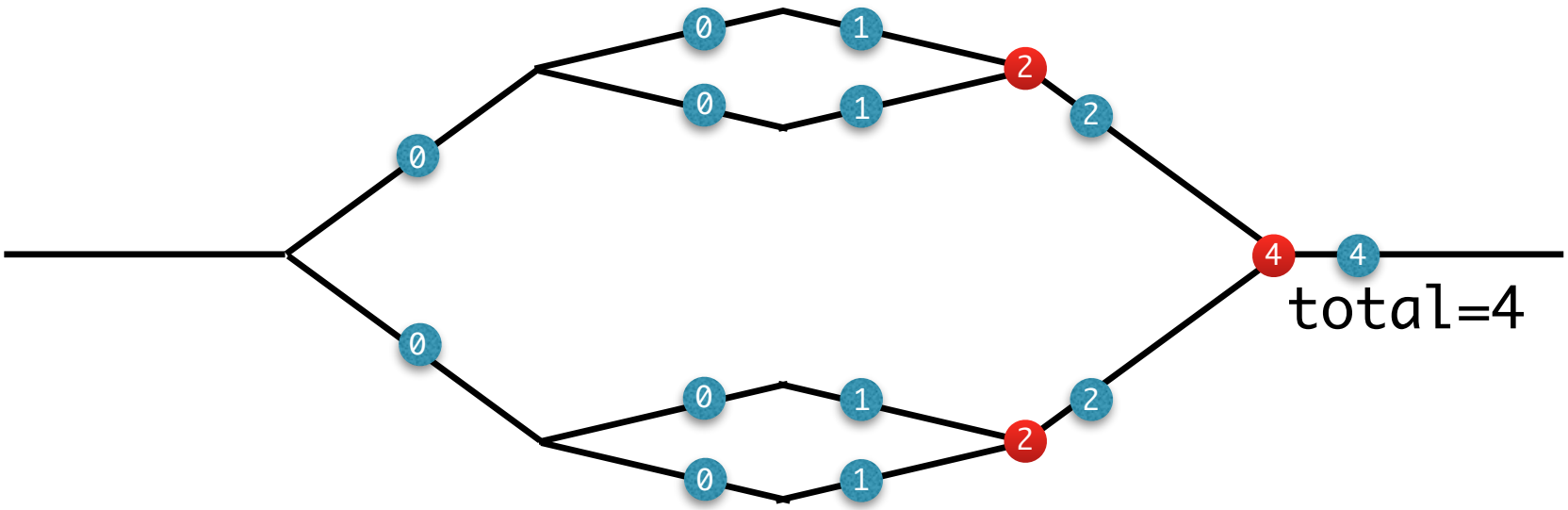
```
1  #include <cilk/cilk.h>
2  #include <cilk/reducer_opadd.h>
3  int total=0;
4  cilk::reducer<cilk::<op_add<int>> reducer_total (0);
5  cilk_for(int i=0;i<4;++i)
6    ++*reducer_total;
7  total = reducer_total.get_value();
```

- At spawn each strand gets a private *view* of the reducer
- Strands must dereference the pointer to operate on its value
- When strands merge
  - *views* are combined by *OP*
  - The combined *view* is given to the exit strand

https://github.com/AlbertDeFusco/goldRush

Cilk Plus Reducers: views

```
#include <cilk/reducer_opadd.h>
int total=0;
cilk::reducer<cilk::<op_add<int>> reducer_total (0);
for(int i=0;i<4;++i)
  ++*reducer_total;
total = reducer_total.get_value();
```



Private View

Merge update

https://github.com/AlbertDeFusco/goldRush

Cilk Plus Reducers: types

usage: `cilk::reducer<cilk::REDUCER_TYPE<<my_type>> my_reducer;`

Reducer Type	Description	Header
op_add	++, --, +=, -=, +, -	#include <cilk/reducer_add.h>
op_vector	Provides push_back()	#include <cilk/reducer_vector.h>
op_list_append	Provides push_back()	#include <cilk/reducer_list.h>
op_list_prepend	Provides push_front()	#include <cilk/reducer_list.h>
op_max	Returns maximum	#include <cilk/reducer_max.h>
op_min	Returns minimum	#include <cilk/reducer_min.h>
op_ostream	Provides <<	#include <cilk/reducer_string.h>

Table: https://software.intel.com/en-us/node/522606

Parallel gold sifting

```
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    std::list<int>::const_iterator iterator;
18    for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
19      std::cout << *iterator << "  ";
20    std::cout << endl;
```

https://github.com/AlbertDeFusco/goldRush

Gold Rush

```
$>cat /proc/cpuinfo | grep Xeon | uniq -c
     16 model name      : Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
$>CILK_NWORKERS=16 ./goldRush
Gold Rush!

  100000 total chunks of dirt
  1000 pans

  Found gold in 15 pans
  Pan IDs: 94  142  265  268  289  440  442  443  569  600  721  781  783  806  818

Cilk identified the correct pans

serial execution took 5.60154 seconds

parallel execution took 0.39801 seconds with 16 workers
parallel speedup 14.0739
paralell efficiency 0.879616
```

https://github.com/AlbertDeFusco/goldRush