Cilk Plus Reducers

Albert DeFusco

April 6, 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

Serial Gold Sifting

```
#include <list>
class pan
{
  public:
    pan();          //create array of random integers between 1 and 10,000
    int sift();     //returns frequency of the number 79 (atomic number of gold)
    bool hasGold(); //calls sift; true if sift returns >0
}

int main()
{
  std::list<int> withGold;
  pan* myPans = new pan[nPans];

  for(int i=0; i<nPans; ++i)
  {
    bool gold = myPans[i].hasGold();
    if(gold) {
      withGold.push_back(i);
    }
  }
  std::list<int>::const_iterator iterator;
  for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
    std::cout << *iterator << "  ";
  std::cout << endl;

  return 0;
}
```

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

```cpp
#include <list>
class pan
{
  public:
    pan();           //create array of random integers between 1 and 10,000
    int sift();      //returns frequency of the number 79 (atomic number of gold)
    bool hasGold();  //calls sift; true if sift returns >0
}

int main()
{
  std::list<int> withGold;
  pan* myPans = new pan[nPans];

  cilk_for(int i=0; i<nPans; ++i)
  {
    bool gold = myPans[i].hasGold();
    if(gold) {
      withGold.push_back(i);
    }
  }
  std::list<int>::const_iterator iterator;
  for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
    std::cout << *iterator << "  ";
  std::cout << endl;

  return 0;
}
```

https://github.com/AlbertDeFusco/goldRush

Parallel Gold Sifting

- Need to report which pans have gold
- How do we keep track of which pans have gold?

Parallel Gold Sifting

```
#include <list>
class pan
{
  public:
    pan();            //create array of random integers between 1 and 10,000
    int sift();       //returns frequency of the number 79 (atomic number of gold)
    bool hasGold();   //calls sift; true if sift returns >0
}

int main()
{
  std::list<int> withGold;
  pan* myPans = new pan[nPans];

  cilk_for(int i=0; i<nPans; ++i)
  {
    bool gold = myPans[i].hasGold();
    if(gold) {
      withGold.push_back(i);
    }
  }
  std::list<int>::const_iterator iterator;
  for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
    std::cout << *iterator << "  ";
  std::cout << endl;

  return 0;
}
```

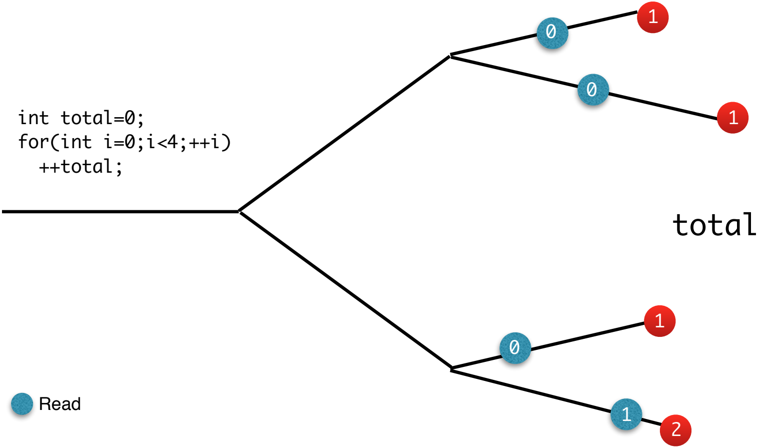
https://github.com/AlbertDeFusco/goldRush

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

```
int total=0;
for(int i=0;i<4;++i)
  ++total;
```



total=1

● Read

● Write

https://github.com/AlbertDeFusco/goldRush

Inefficient solutions

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

```
#include <cilk/cilk.h>

double *sum = new double[N];
cilk_for(int i=0;i<N;++i)
  sum[N] = f(N)
double total=0.0;
for(int i=0;i<N;++i)
  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

$$x\ OP\ y = y\ OP\ x$$

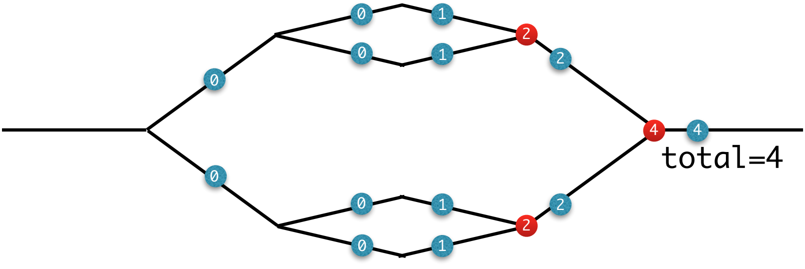
- Small parallel overhead for usage
- Very extensible in C++
- Operations are guaranteed to execute in the same order as in serial

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 strand

Cilk Reducers: views

```
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

Parallel gold sifting

```
#include <cilk/cilk.h>
#include <cilk/reducer_list.h>

  std::list<int> withGold;
  cilk::reducer< cilk::op_list_append<int> > reducer_withGold;
  pan* myPans = new pan[nPans];

  cilk_for(int i=0; i<nPans; ++i)
  {
    bool gold = myPans[i].hasGold();
    if(gold) {
      reducer_withGold->push_back(i);
    }
  }
  withGold = reducer_withGold.get_value();

  list<int>::const_iterator iterator;
  for (iterator = withGold.begin(); iterator != withGold.end(); ++iterator)
    cout << *iterator << "  ";
  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