Reducer Hyperobjects

CSE 549
Recitation 9
Motivation

```cpp
bool has_property(Node *);
std::list<Node *> output_list;
// ...
void walk(Node *x)
{
    if (x) {
        if (has_property(x))
            output_list.push_back(x);
        walk(x->left);
        walk(x->right);
    }
}
```
Motivation

```cpp
bool has_property(Node *);
std::list<Node *> output_list;

// ...

void walk(Node *x)
{
    if (x) {
        if (has_property(x))
            output_list.push_back(x);
        cilk_spawn walk(x->left);
        walk(x->right);
        cilk_sync;
    }
}
```
Solutions

• Locks

```c
8 if (has_property(x)) {
9     L.lock();
10    output_list.push_back(x);
11    L.unlock();
12 }
```

– Can create a bottleneck during run time, or possibly cause deadlock

• Restructuring
bool has_property(Node *);

std::list<Node *> output_list;

// ...

void walk(Node *x)
{
    if (x) {
        if (has_property(x))
            output_list.push_back(x);
        cilk_spawn walk(x->left);
        walk(x->right);
        cilk_sync;
    }
}
Solutions

• Hyperobjects
  – What are they?

\[ x_1 \quad x_2 \quad x_1 \quad x_2 \]
Reducer Hyperobjects

• Based on an algebraic monoid:
  – A set, $T$ (integers, Nodes, etc.)
  – An associative binary operator, $\otimes$  
    (+, $x$, concatenation, etc.)
  – An identity, $e$, such that $e \otimes a = a \otimes e = a$ 
    (0, 1, empty set, etc.)
Example of Cilk++ reducer

```cpp
1 int compute(const X& v);
2 int cilk_main()
3 {
4     const std::size_t n = 1000000;
5     extern X myArray[n];
6     // ...
7     sum_reducer<int> result(0);
8     cilk_for (std::size_t i = 0; i < n; ++i)
9         result += compute(myArray[i]);
10
11     std::cout << "The result is: "
12         << result.get_value() << std::endl;
13     return 0;
14 }
```
Naïve Implementation

- Create 2 views at a spawn set to e

\[ x_1 = x_1 \otimes x_2 \otimes x_3 \]
Better Way

- Create 1 view at a spawn set to e
- Give the parent view to the child

\[ x_1 = x_1 \otimes x_2 \]
Even Better Way

- Only create a view if a steal occurs
Even Better Way

\[ x_1 = x_1 \otimes x_2 \]

Worker 1

Worker 2
Even Better Way

\[ x_1 = x_1 \otimes x_2 \otimes x_3 \]

Worker 1

Worker 2

Worker 3
Steal Tree

- Left-child, right-sibling binary tree
Steal Tree

- Left-child, right-sibling binary tree
Steal Tree

• Include 2 “placeholder” variables
Steal Tree

• bar2 finishes first
Steal Tree

• Left child finishes
Steal Tree

• bar1 finishes
Steal Tree

• bar1 finishes
Steal Tree

• bar1 finishes
Steal Tree

- Need to use locks
Holders

holder<T> global_variable;

// originally: T global_variable

void proc1() {
    cilk_for (i = 0; i < N; ++i) { //was: for
        global_variable = f(i);
        proc2();
    }
}

void proc2() { proc3(); }
void proc3() { proc4(); }

void proc4() {
    use(global_variable);
}