Disentanglement in Nested-Parallel Programs

Sam Westrick
Carnegie Mellon University

Joint work with
Rohan Yadav, Umut Acar, and Matthew Fluet
Parallel Programming

**imperative**
- mutability
- manual memory management
- non-determinism

**functional**
- immutability
- automatic memory management
- determinism

---

can parallel functional programming be fast and scalable?
Sequential

Parallel
Sequential

Parallel

Is there a better way?
Nested Parallelism (Fork-Join)

- classic and popular

- MultiLisp, OpenMP, Cilk, Intel TBB, TPL (.NET), Rayon (Rust), Java Fork/Join, Habanero Java, X10, NESL, parallel Haskell, Futhark, Manticore, parallel ML, …
Nested Parallelism (Fork-Join)

```haskell
map f A =
  let
    B = newArray (length A)
    map' i j =
      case j-i of
        | 0  => ()
        | 1  => B[i] := f (A[i])
        | n  =>
          let m = i + n/2
          in (map' i m || map' m j);
      end
  in
    map' 0 (length A);
  B
end
```
Nested Parallelism (Fork-Join)

map f A =
let
  B = newArray (length A)

  map' i j =
    case j-i of
    | 0 => ()
    | 1 => B[i] := f (A[i])
    | n =>
      let m = i + n/2
      in (map' i m || map' m j);
    ()
  end
in
  map' 0 (length A);
B
end
Nested Parallelism (Fork-Join)

map f A =
  let
    B = newArray (length A)
  in
    map' i j =
      case j-i of
        | 0 => ()
        | 1 => B[i] := f (A[i])
        | n =>
          let m = i + n/2
          in (map' i m || map' m j);
      ()
    in
      map' 0 (length A); B
  end
Nested Parallelism (Fork-Join)

map f A =
  let
    B = newArray (length A)

    map' i j =
      case j-i of
        | 0 => ()
        | 1 => B[i] := f (A[i])
        | n =>
          let m = i + n/2
          in (map' i m || map' m j);
      ()
  in
    map' 0 (length A);
    B
end
Nested Parallelism (Fork-Join)

map f A = 
  let 
    B = newArray (length A) 
  in 
    map' i j = 
      case j-i of 
        | 0 => () 
        | 1 => B[i] := f (A[i]) 
        | n => 
          let m = i + n/2 
          in (map' i m || map' m j); 
        end 
      in 
        map' 0 (length A); 
    B 
  end
Nested Parallelism (Fork-Join)

map f A =
let
  B = newArray (length A)

map’ i j =
  case j-i of
    | 0 => ()
    | 1 => B[i] := f (A[i])
    | n =>
      let m = i + n/2
      in (map’ i m || map’ m j);
    end
  in
    map’ 0 (length A);
B
end
Disentanglement

**Definition**
Throughout execution, each thread may only use data allocated by itself or **ancestors**.
Disentanglement

**Definition**
Throughout execution, each thread may only use data allocated by itself or ancestors.
Disentanglement

**Definition**
Throughout execution, each thread may only use data allocated by itself or ancestors.
Disentanglement

**definition** throughout execution, each thread may only use data allocated by itself or ancestors
**theorem**
all race-free programs are disentangled

Proof technique:

- use computation graphs for definitions
- identify single-step invariant:
  - if location $X$ accessible without a race, then $neighbors(X)$ are in root-to-leaf path
- carry invariant through race-free execution
Disentanglement in Practice

**Ligra**
- BFS
- betweenness centrality
- Bellman-Ford
- k-Core
- Page Rank
- maximal independent set
- eccentricity estimation

**PBBS**
- quickhull
deduplication
- sorting
minimum spanning forest
- suffix array
- Barnes-Hut
- nearest neighbors
- ray casting

**all disentangled**
(and likely others too)

**many benign data races**
fully general

disentangled

race-free

mutation-free
Is there a better way?
Hierarchical Memory Management

fork

join
Hierarchical Memory Management

fork

merge heaps into parent

join

fresh empty heaps
Hierarchical Memory Management

• disentanglement: *no cross pointers*
Hierarchical Memory Management

- disentanglement: *no cross pointers*

- *subtree collection*

  reorganize, compact, etc. inside subtree

  naturally parallel
MaPLe

• full ML language, extended with fork-join library
  
  ```ocsaml
  val par: (unit -> 'a) * (unit -> 'b) -> 'a * 'b
  ```

• used by 500+ students at Carnegie Mellon University each year

• implementation details:
  
  • extends MLton
  
  • completely new runtime system
  
  • subtree collection integrated with scheduling
    
    • Cheney-style copying/compacting

github.com/mpillang/mpl
Experiments: Scalability

benchmarks ported to Parallel ML

Speedups relative to MLton
## Experiments: Sorting Shootout

<table>
<thead>
<tr>
<th>Method</th>
<th>$T_1$</th>
<th>$T_{72}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ std::sort</td>
<td>8.8</td>
<td>–</td>
</tr>
<tr>
<td>Cilk samplesort</td>
<td>7.9</td>
<td>0.16</td>
</tr>
<tr>
<td>Cilk mergesort</td>
<td>12.7</td>
<td>0.24</td>
</tr>
<tr>
<td><strong>MPL (Ours) mergesort</strong></td>
<td>18.8</td>
<td>0.37</td>
</tr>
<tr>
<td>Go samplesort</td>
<td>27.2</td>
<td>0.52</td>
</tr>
<tr>
<td>Java mergesort</td>
<td>11.0</td>
<td>0.63</td>
</tr>
<tr>
<td>Haskell/C mergesort</td>
<td>10.6</td>
<td>1.3</td>
</tr>
</tbody>
</table>

2nd fastest, only behind Cilk
Summary

- disentanglement
  - natural and widespread
- question
  can disentanglement be treated as a correctness condition?
- future work
  - static and dynamic checking
- hierarchical memory management
  parallel collection
- MaPLe (MPL)
  real, practical implementation