CilkMR: A Scalable and Composable Map-Reduce System

M. Arif, H. Vandierendonck, D.S. Nikolopoulos, B. R. de. Supinski
Agenda

- Introduction and Background
- Contribution
- Evaluation
- Conclusion
Agenda

- Introduction and Background
- Contribution
- Evaluation
- Conclusion
Introduction

Data analytics has
+ Increased importance for businesses
+ Growing dataset

Design goals:

Performance

Programmability
Map-Reduce Programming Model

Delivers programmability and performance for distributed memory systems

Figure source: http://webmapreduce.sourceforge.net
Moore’s law continues..

+ Shared-memory machines with higher core count and terabytes of memory now feasible for data analytics.
Agenda

- Introduction and Background
- Contribution
- Evaluation
- Conclusion
CilkMR

A C++ template-based library to provide map-reduce functionality for shared memory systems

- aims to provide programmability and performance

- built on top of Cilk, a task-parallel programming model with work-stealing based scheduler

- expression of map (task) and reduce operations derived from Cilk
CilkMR

Cilk provides simple keywords to express parallelism

- **cilk_for, cilk_spawn and cilk_sync**

### Sequential

```c
int fib(int n){
    int op=0;
    op+=fib(n-1);
    op+=fib(n-2);
    return op; }
```

```c
for(int i=0; i<n;i++)
a[i]= b[i]+c[i]
```

### Parallel

```c
int fib(int n){
    if(n<2) return n;
    else{
        int op=0;
        op+=cilk_spawn fib(n-1);
        op+=cilk_spawn fib(n-2);
        cilk_sync;
        return op; }
```

```c
cilk_for(int i=0; i<n;i++)
a[i]= b[i]+c[i]
```
CilkMR- mapreduce API

Templates for balanced and unbalanced spawn trees

**balanced**
- **cilk_for**
- choose when iteration range known
- work-stealing minimum
- $O(\log n)$ steals

**unbalanced**
- **cilk_spawn, cilk_sync**
- choose when iteration range not known
- more work-stealing
- $O(n)$ steals
CilkMR- mapreduce API

CilkMR template for balanced spawn tree

```cpp
1 template<class Monoid, class InputIterator, class MapFunctor>
2 map_reduce(InputIterator ibegin, InputIterator iend, MapFunctor mapfn, typename Monoid::value type & output ) {
3       cilk::reducer<Monoid> imp_;  
4       cilk_for(InputIterator I=ibegin, E=iend; I != E; ++I )
5           mapfn(*I, imp_.view());
6       std::swap( output, imp_.view());
}
```

Example use-case: histogram

```cpp
histo_map(){
    histogram[pix[0]]++;
    histogram[256+pix[1]]++;
    histogram[512+pix[2]]++;
}
map_reduce(img_array, img_array_length/3, histo_map(), result);
```
CilkMR - Reducers

Reduction defined through monoid \((T, x, e)\) where \(T\) is type, \(x\) is reduction operation and \(e\) is identity

hyper-objects: the view may not be the same for each observer

+ avoids reductions unless necessary. new views created only after a steal
+ reduction operations (and overall cost) \(\alpha\) number of steals
+ binary reduction operations required to hold associative property.
+ operate independently of the control structure. managed only at spawn and sync’s.
Programming Style

**CilkMR:**

+ does not require fitting the problem in map-reduce model.
+ Follows the structure of general purpose code

**Specialized map-reduce frameworks** (such as Phoenix++)

- Requires effort to fit the problem in map-reduce model
  - **inefficient** for iterative algorithms such as Kmeans
  - Long and tedious codes for defining mechanics of computation, such as PCA

Lines of code for covariance calculation for PCA: **18** for CilkMR, **50** for Phoenix++
Choice of intermediate data structures

**CilkMR**

- allows arbitrary intermediate data structures
- appropriate data structures can be chosen for a given problem.

**Specialized map-reduce frameworks (such as Phoenix++)**

- require representation of intermediate data structures as key-value pairs
- **costs performance** for restructuring-sorting of keys.
Reduction operations

CilkMR

+ generalized reduction operations on data containers
+ overlap of map and reduce phases. Better load-balancing

Specialized map-reduce frameworks (such as Phoenix++)

- reductions over key-value pairs.
- reduction phase starts only after the completion of map phase
Memory Consumption

CilkMR

+ Cilk runtime does not delay all reductions, and thus avoids large excessive memory usage for storing unreduced views

Specialized map-reduce frameworks (such as Phoenix++)

- delayed reductions require storing large volumes of intermediate data structures
Additional feature support

+ **CilkMR** allows use of additional features supported by Cilk such as nested parallelism and vectorization.
Agenda

- Introduction and Background
- Contribution
- Evaluation
- Conclusion
Performance Evaluation

Benchmarks
- 7 map-reduce benchmarks from Phoenix++

Platform
- Quad-socket 12(x2)-core Intel Xeon E7-4860-v2@2.6GHz
- No hyper-threading used
- 30MB L3 cache / 12 physical cores
- CentOS 6.5, ICC compiler v14.0.1
- Comparison to Phoenix++ 1.0, specialized shared-memory map-reduce system
**Performance Evaluation**

**kmeans**: Unsupervised clustering algorithm: iteratively groups input data points into \( K \) clusters, based on the nearest mean

- **CilkMR**: balanced template
- Each iteration in Phoenix++ is a map-reduce algorithm
- Repeated (de)-serialization of the key-value pairs

---

**Graph**

- **CilkMR** vs. **Phoenix++**
- **CilkMR 2.2x faster!**

<table>
<thead>
<tr>
<th>No. of Threads</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
</tr>
<tr>
<td>48</td>
<td>32</td>
</tr>
</tbody>
</table>

---

**Legend**

- ▲ CilkMR
- Phoenix++
**Performance Evaluation**

**pca**: row mean and covariance matrix calculation for Principal Component Analysis

- CilkMR: implemented as general-purpose parallel code
- Covariance calculation code with nested for-loop
- Load-imbalance in the inner loop

![Graph showing speedup vs. number of threads for different methods:
- CilkMR+vector
- CilkMR
- Phoenix++, outer loop
- Phoenix++, flattened

CilkMR 1.9x faster!
wordcount: Counting occurrence of different words in a file

- CilkMR: unbalanced template
- Reduce phase: reduction on hash table
- Unbalanced spawn tree
Performance Evaluation

Speedup on 48 Threads (normalized to sequential execution)

- Phoenix++
- CilkMR

<table>
<thead>
<tr>
<th>Task</th>
<th>Phoenix++ Speedup</th>
<th>CilkMR Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear-reg</td>
<td>46% faster</td>
<td></td>
</tr>
<tr>
<td>histo</td>
<td>4% faster</td>
<td></td>
</tr>
<tr>
<td>pca</td>
<td>89.9% faster</td>
<td></td>
</tr>
<tr>
<td>mat-mul</td>
<td>83.5% faster</td>
<td></td>
</tr>
<tr>
<td>kmeans</td>
<td>113.6% faster</td>
<td></td>
</tr>
<tr>
<td>wordcount</td>
<td></td>
<td>22% slower</td>
</tr>
<tr>
<td>str-match</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Memory Consumption

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>0.06</td>
<td>0.04</td>
<td>0.95</td>
<td>0.43</td>
<td>1.67</td>
<td>0.86</td>
<td>2.50</td>
<td>1.23</td>
</tr>
<tr>
<td>lreg</td>
<td>0.06</td>
<td>&lt;0.01</td>
<td>0.69</td>
<td>0.06</td>
<td>1.39</td>
<td>0.11</td>
<td>2.08</td>
<td>0.17</td>
</tr>
<tr>
<td>wc</td>
<td>11.7</td>
<td>15.1</td>
<td>28.10</td>
<td>60.60</td>
<td>34.30</td>
<td>98.30</td>
<td>34.0</td>
<td>117.0</td>
</tr>
<tr>
<td>pca</td>
<td>25.82</td>
<td>159.90</td>
<td>26.44</td>
<td>161.5</td>
<td>27.13</td>
<td>160.0</td>
<td>27.82</td>
<td>160.10</td>
</tr>
<tr>
<td>kmeans</td>
<td>39.81</td>
<td>68.62</td>
<td>41.66</td>
<td>502.00</td>
<td>42.98</td>
<td>963.2</td>
<td>44.55</td>
<td>1423.4</td>
</tr>
<tr>
<td>strmatch</td>
<td>0.06</td>
<td>0.56</td>
<td>0.69</td>
<td>0.58</td>
<td>1.38</td>
<td>0.61</td>
<td>2.07</td>
<td>0.73</td>
</tr>
<tr>
<td>matmul</td>
<td>4.06</td>
<td>4.06</td>
<td>4.69</td>
<td>4.16</td>
<td>5.39</td>
<td>4.27</td>
<td>5.35</td>
<td>4.39</td>
</tr>
</tbody>
</table>
## Memory Consumption

<table>
<thead>
<tr>
<th></th>
<th>Memory usage (MB) for thread count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>histogram</td>
<td></td>
</tr>
<tr>
<td>CilkMR</td>
<td>0.06</td>
</tr>
<tr>
<td>Phoenix++</td>
<td>0.04</td>
</tr>
<tr>
<td>lreg</td>
<td></td>
</tr>
<tr>
<td>CilkMR</td>
<td>0.06</td>
</tr>
<tr>
<td>Phoenix++</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>wc</td>
<td></td>
</tr>
<tr>
<td>CilkMR</td>
<td>11.7</td>
</tr>
<tr>
<td>Phoenix++</td>
<td>15.1</td>
</tr>
<tr>
<td>pca</td>
<td></td>
</tr>
<tr>
<td>CilkMR</td>
<td>25.82</td>
</tr>
<tr>
<td>Phoenix++</td>
<td>159.90</td>
</tr>
<tr>
<td>kmeans</td>
<td></td>
</tr>
<tr>
<td>CilkMR</td>
<td>39.81</td>
</tr>
<tr>
<td>Phoenix++</td>
<td>68.62</td>
</tr>
<tr>
<td>strmatch</td>
<td></td>
</tr>
<tr>
<td>CilkMR</td>
<td>0.06</td>
</tr>
<tr>
<td>Phoenix++</td>
<td>0.56</td>
</tr>
<tr>
<td>matmul</td>
<td></td>
</tr>
<tr>
<td>CilkMR</td>
<td>4.06</td>
</tr>
<tr>
<td>Phoenix++</td>
<td>4.06</td>
</tr>
</tbody>
</table>
Agenda

+ Introduction and Background
+ Contribution
+ Evaluation
+ Conclusion
Conclusion

- CilkMR outperforms Phoenix++ for 5 out of 7 benchmarks.
- Forcing applications into map-reduce model has its inefficiencies.
- CilkMR composable with general purpose code.
- Intuitive selection of containers, intermediate data structures and program structure.
- Reductions over containers instead of key-value pairs.
Thank You

Questions?