CilkMR: A Scalable and Composable Map-Reduce System

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

Queen's University Belfast

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:

Map-Reduce Programming Model

Delivers programmability and performance for [distributed memory systems]

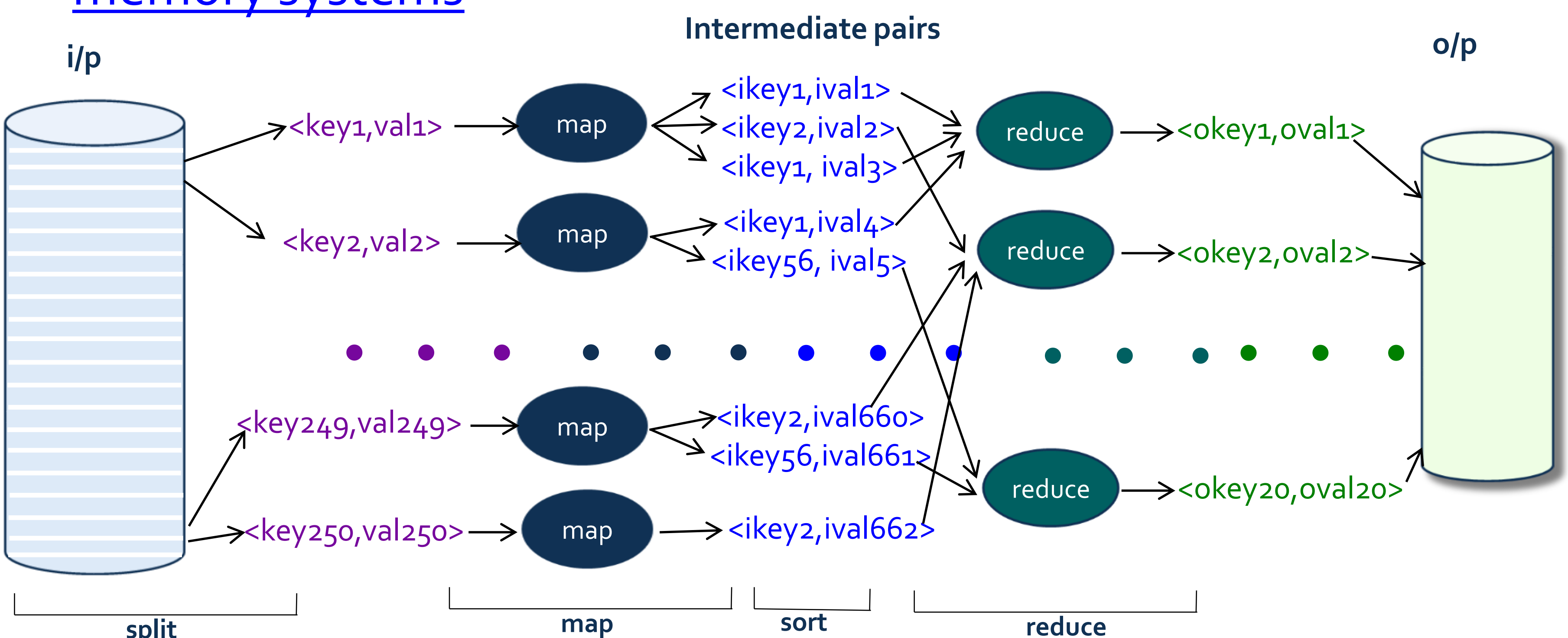
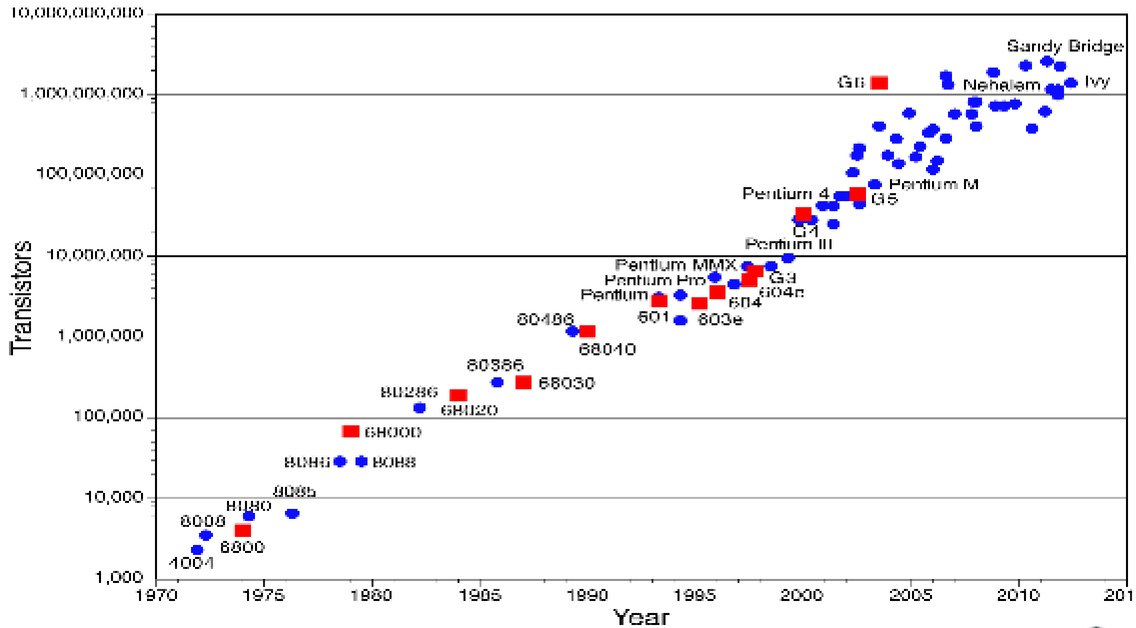


Figure source: http://webmapreduce.sourceforge.net

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

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

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

parallel

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

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

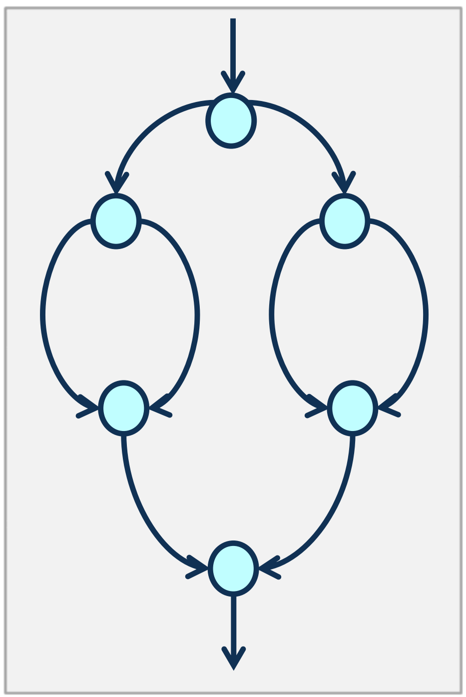
Queen's University Belfast

CilkMR- mapreduce API

Templates for **balanced** and **unbalanced** spawn trees

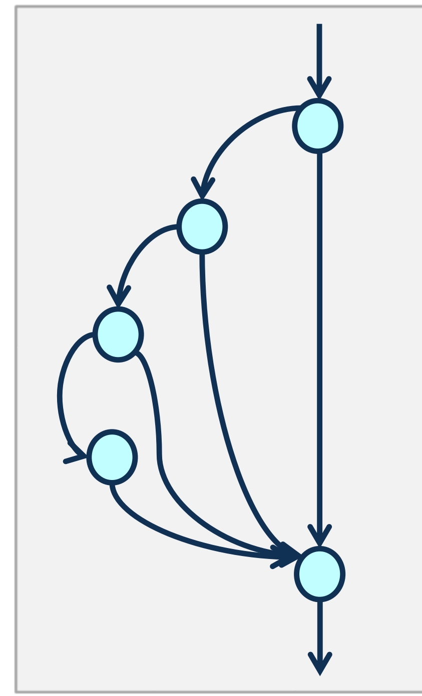
balanced

- `cilk_for`
- choose when iteration range known
- work-stealing minimum
- O(logn) 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

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

Example use-case: histogram

```
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) **α** 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 tedio
    computation, s

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

Queen's University Belfast

Performance Evaluation

Benchmarks

- 7 map-reduce benchmarks from Phoenix++

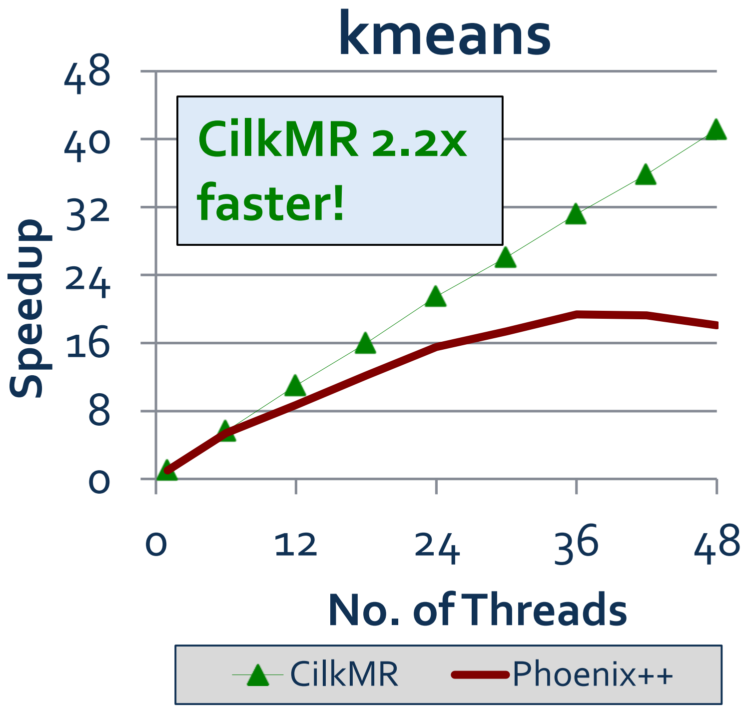
Platform

- 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

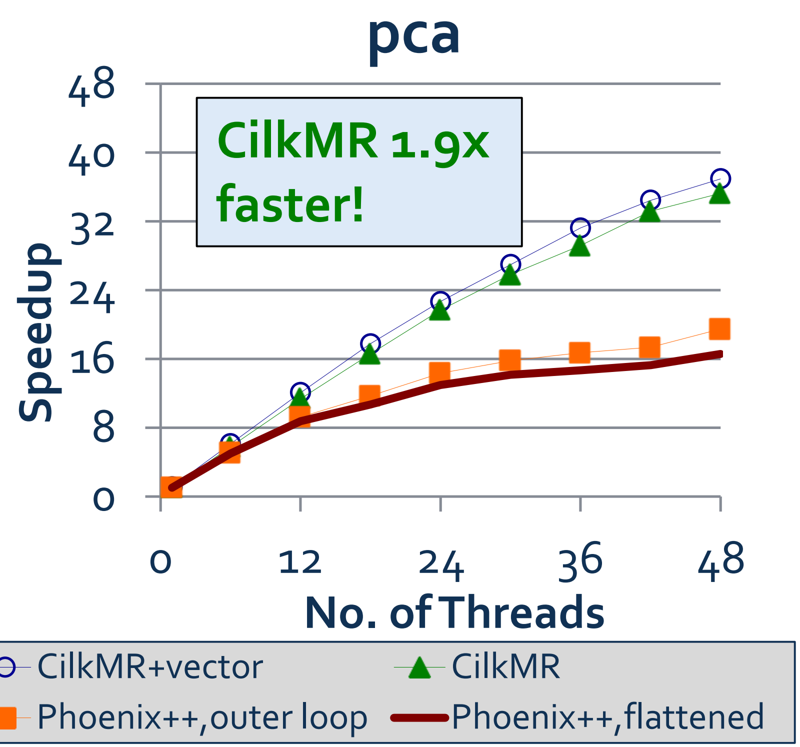
- CilkMR: balanced template
- Each iteration in Phoenix++ is a map-reduce algorithm
- Repeated (de)-serialization of the key-value pairs



Performance Evaluation

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

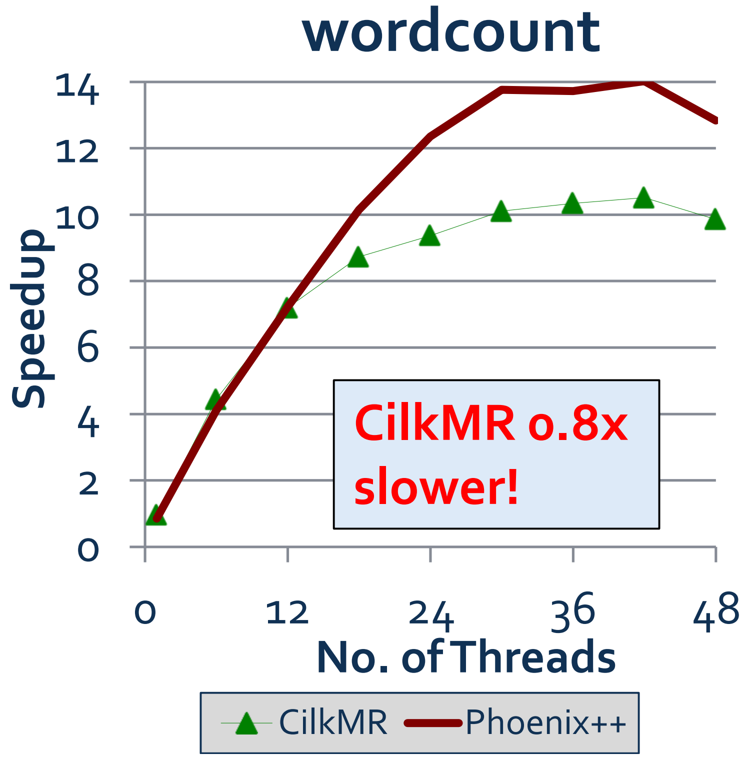
- CilkMR: implemented as general-purpose parallel code
- Covariance calculation code with nested for-loop
- Load-imbalance in the inner loop



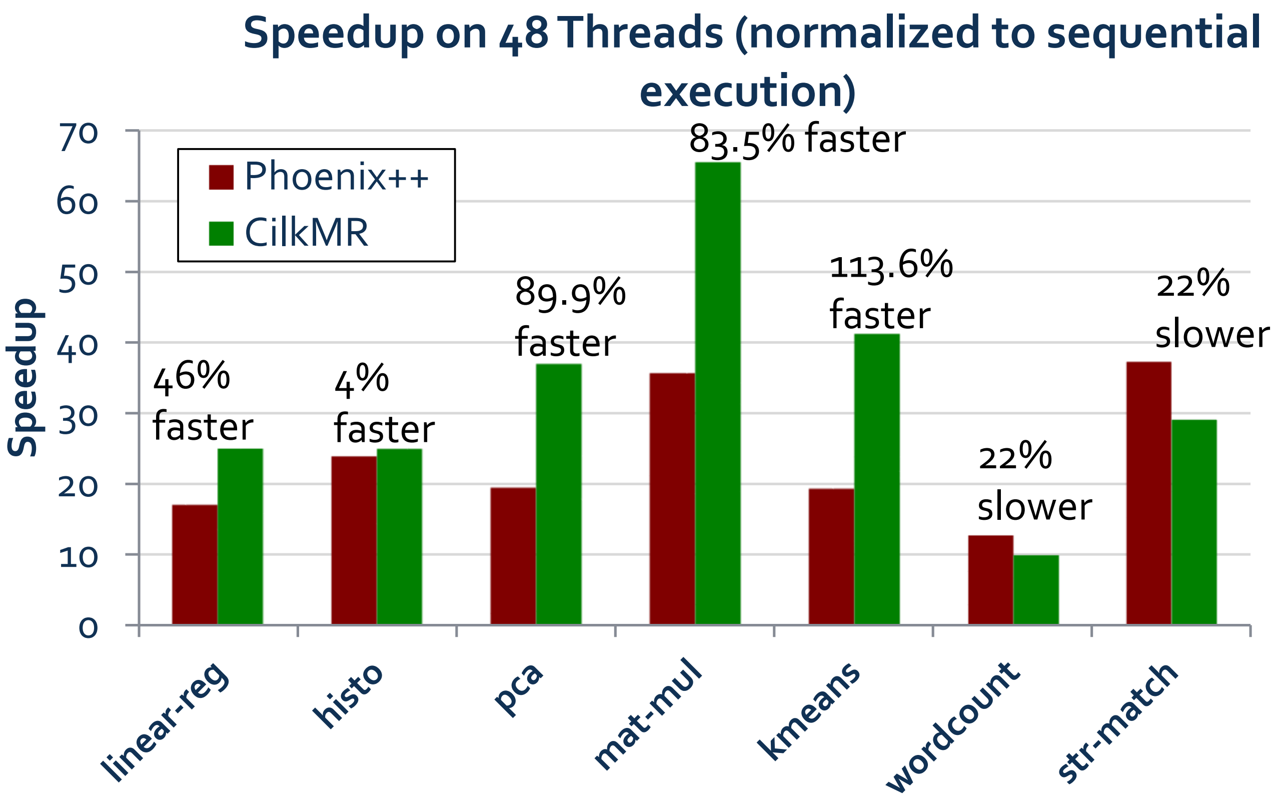
Performance Evaluation

wordcount: Counting occurrence of different words in a file

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



Performance Evaluation



Memory Consumption

		Memory usage (MB) for thread count			
		1	16	32	48
histogram	CilkMR	0.06	0.95	1.67	2.50
	Phoenix++	0.04	0.43	0.86	1.23
lreg	CilkMR	0.06	0.69	1.39	2.08
	Phoenix++	<0.01	0.06	0.11	0.17
wc	CilkMR	11.7	28.10	34.30	34.0
	Phoenix++	15.1	60.60	98.30	117.0
pca	CilkMR	25.82	26.44	27.13	27.82
	Phoenix++	159.90	161.5	160.0	160.10
kmeans	CilkMR	39.81	41.66	42.98	44.55
	Phoenix++	68.62	502.00	963.2	1423.4
strmatch	CilkMR	0.06	0.69	1.38	2.07
	Phoenix++	0.56	0.58	0.61	0.73
matmul	CilkMR	4.06	4.69	5.39	5.35
	Phoenix++	4.06	4.16	4.27	4.39

Memory Consumption

		Memory usage (MB) for thread count			
		1	16	32	48
histogram	CilkMR	0.06	0.95	1.67	2.50
	Phoenix++	0.04	0.43	0.86	1.23
lreg	CilkMR	0.06	0.69	1.39	2.08
	Phoenix++	<0.01	0.06	0.11	0.17
wc	CilkMR	11.7	28.10	34.30	34.0
	Phoenix++	15.1	60.60	98.30	117.0
pca	CilkMR	25.82	26.44	27.13	27.82
	Phoenix++	159.90	161.5	160.0	160.10
kmeans	CilkMR	39.81	41.66	42.98	44.55
	Phoenix++	68.62	502.00	963.2	1423.4
strmatch	CilkMR	0.06	0.69	1.38	2.07
	Phoenix++	0.56	0.58	0.61	0.73
matmul	CilkMR	4.06	4.69	5.39	5.35
	Phoenix++	4.06	4.16	4.27	4.39

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?