EECS
Electrical Engineering and
Computer Sciences

BERKELEY PAR LAB

PARALLEL COMPUTING LABORATORY

Hierarchical Teams in a Single-Program, Multiple-Data Execution Model

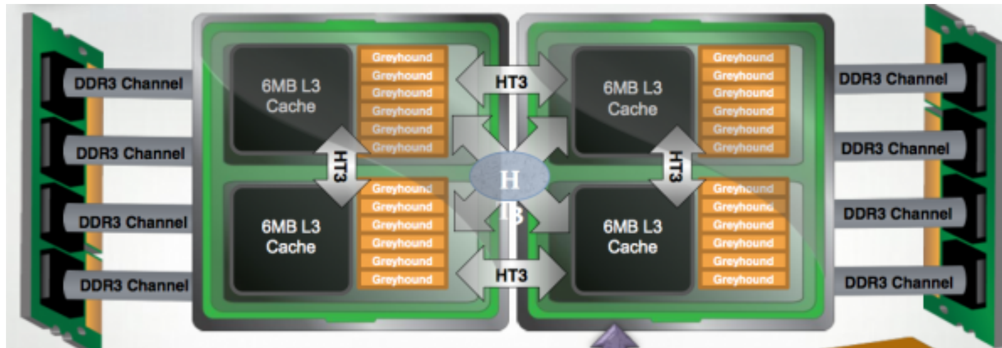
Amir Kamil and Katherine Yelick

DEGAS Retreat

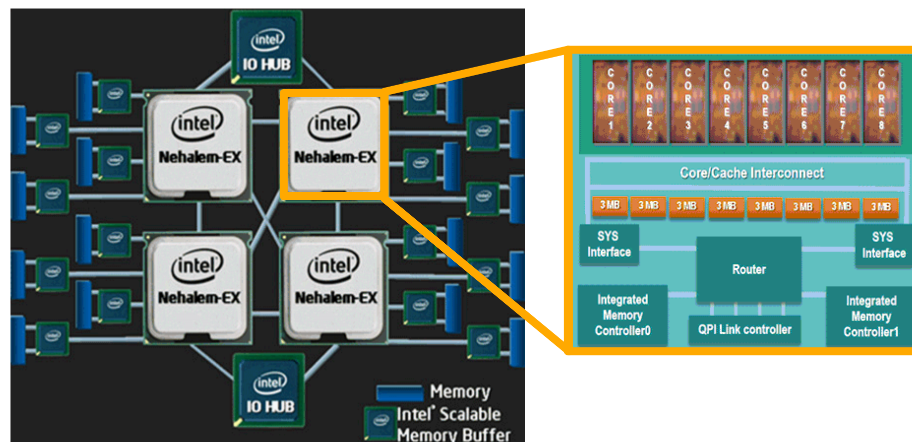
June 4, 2013

Hierarchical Machines

- Parallel machines have hierarchical structure



Dual Socket AMD MagnyCours

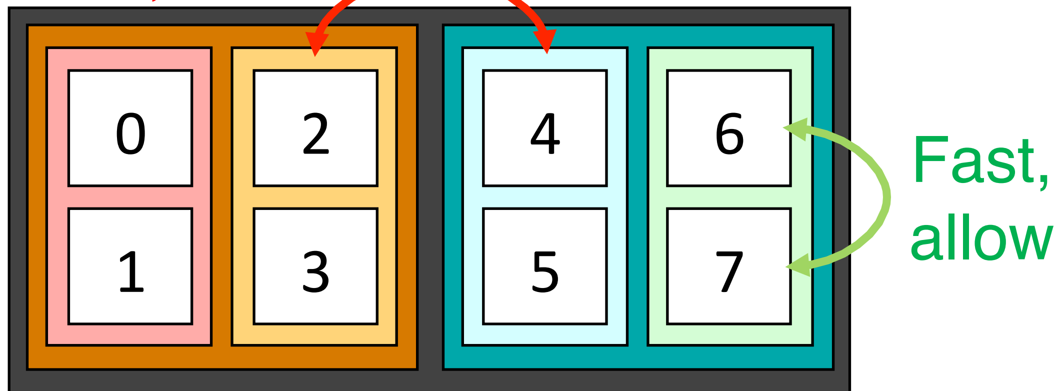


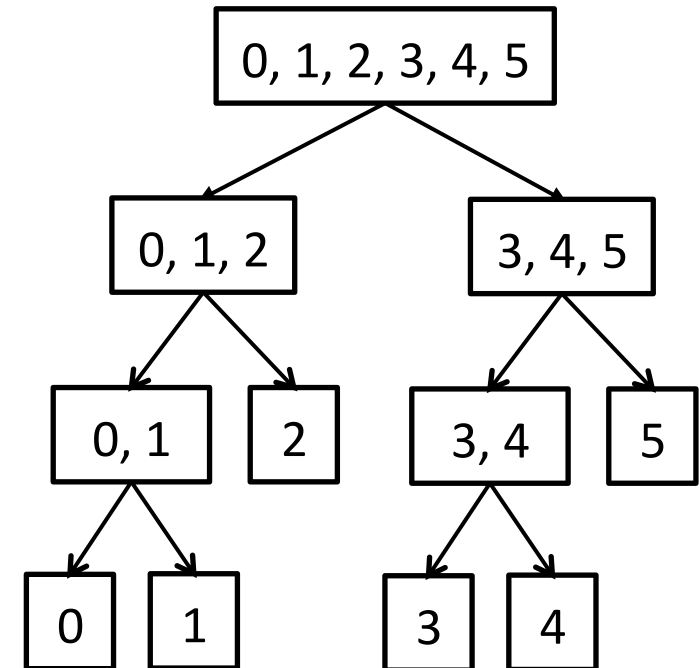
Quad Socket Intel Nehalem EX

- Expect this hierarchical trend to continue with manycore

Application Hierarchy

- Applications can reduce communication costs by adapting to machine hierarchy



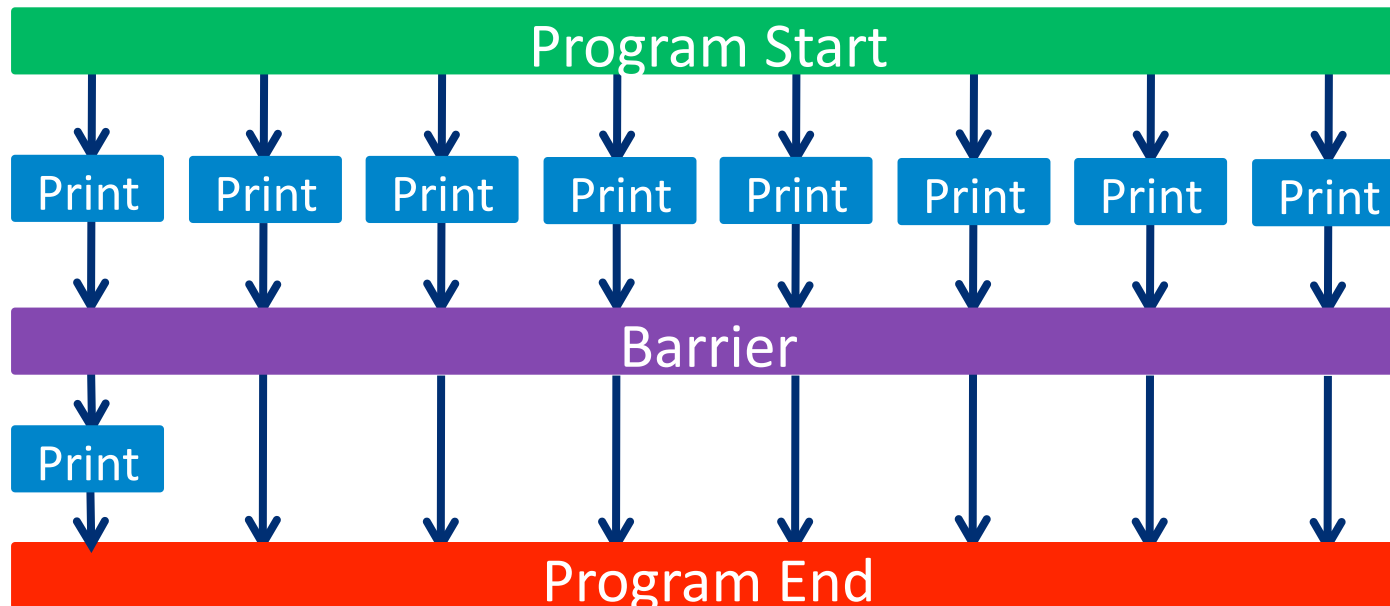


- Applications may also have inherent, algorithmic hierarchy
  - Recursive algorithms
  - Composition of multiple algorithms
  - Hierarchical division of data

Single Program, Multiple Data

- Single program, multiple data (SPMD): fixed set of threads execute the same program image

```
public static void main(String[] args) {
  System.out.println("Hello from " + Ti.thisProc());
  Ti.barrier();
  if (Ti.thisProc() == 0)
    System.out.println("Done.");
}
```



Algorithm Example: Merge Sort

- Task parallel

```
int[] mergeSort(int[] data) {
  int len = data.length;
  if (len < threshold)
    return sequentialSort(data);
  d1 = fork mergeSort(data[0:len/2-1]);
  d2 = mergeSort(data[len/2:len-1]);
  join d1;
  return merge(d1, d2);
}
```

- Cannot fork threads in SPMD
  - Must rewrite to execute over fixed set of threads

Algorithm Example: Merge Sort

EECS Electrical Engineering and Computer Sciences

BERKELEY PAR LAB

- SPMD

```
int[] mergeSort(int[] data, int[] ids) {
  int len = data.length;
  int threads = ids.length;
  if (threads == 1) return sequentialSort(data);
  if (myId in ids[0:threads/2-1])
    d1 = mergeSort(data[0:len/2-1],
                   ids[0:threads/2-1]);
  else
    d2 = mergeSort(data[len/2:len-1],
                   ids              threads-1]);
  barrier(ids);
  if (myId == ids[0]) return merge(d1, d2);
}
```

Team ← `int[] ids`

Team Collective ← `barrier(ids)`

EECS
Electrical Engineering and Computer Sciences

Thread Teams

BERKELEY PAR LAB

- Thread *teams* are basic units of cooperation
  - Groups of threads that cooperatively execute code
  - Collective operations over teams
- Other languages have teams
  - MPI communicators, UPC teams
- However, those teams are flat
  - Do not match hierarchical structure of algorithms, machines
  - Misuse of teams can result in deadlock

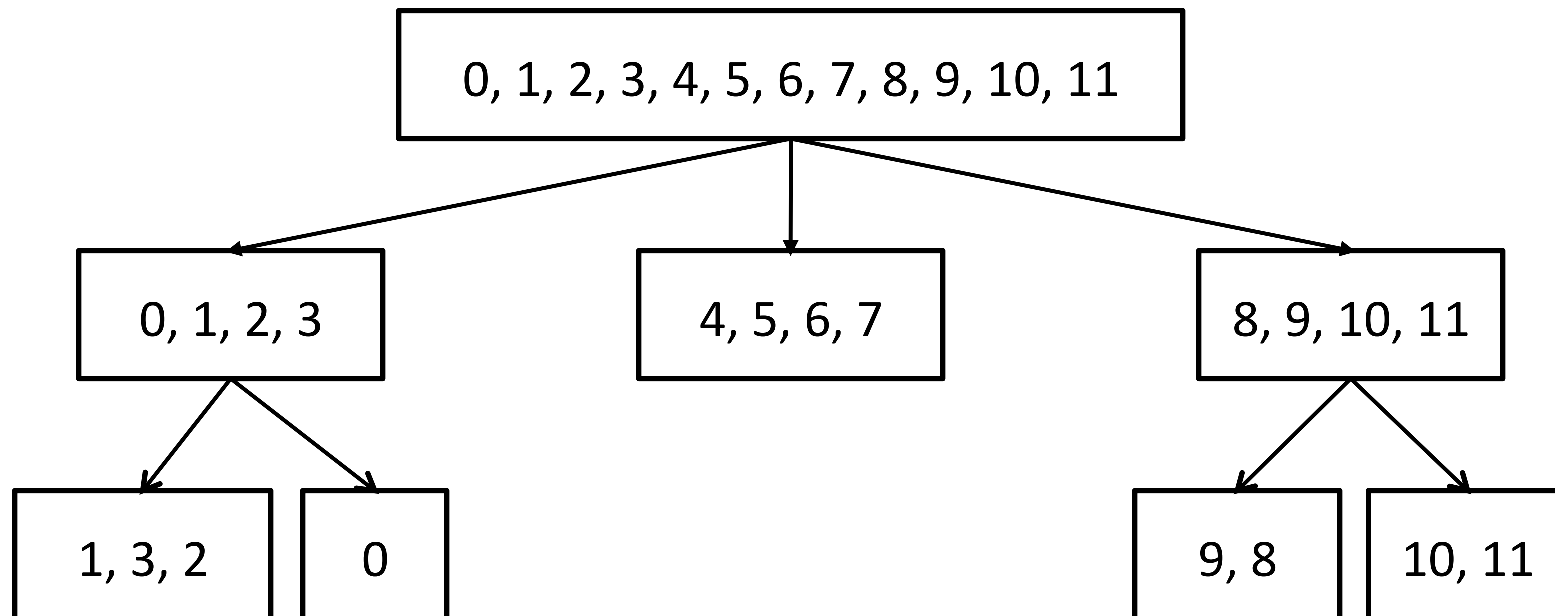
```
Team t1 = new Team(0:7);
Team t2 = new Team(0:3);
if (myId == 0) barrier(t1);
else barrier(t2);
```

Structured Teams

- Structured, hierarchical teams are the solution
  - Expressive: match structure of algorithms, machines
  - Safe: eliminate many sources of deadlock
  - Analyzable: enable simple program analysis
  - Efficient: allow users to take advantage of machine structure, resulting in performance gains

Team Data Structure

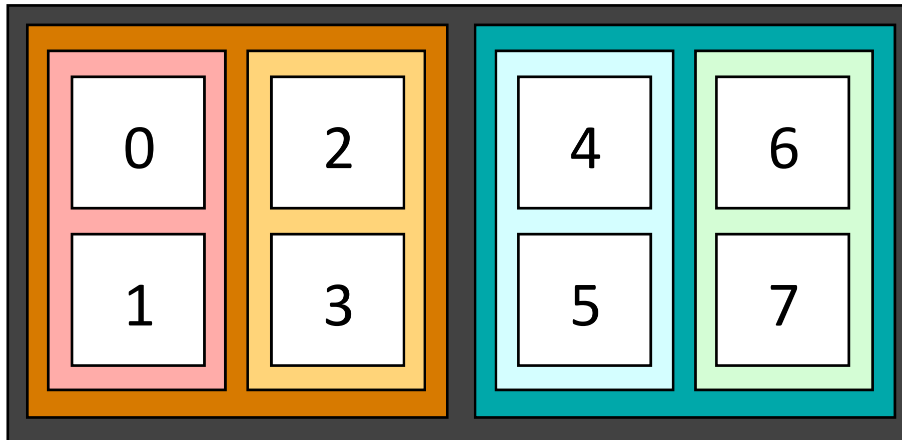
- Threads comprise teams in tree-like structure
- First-class object to allow easy creation and manipulation

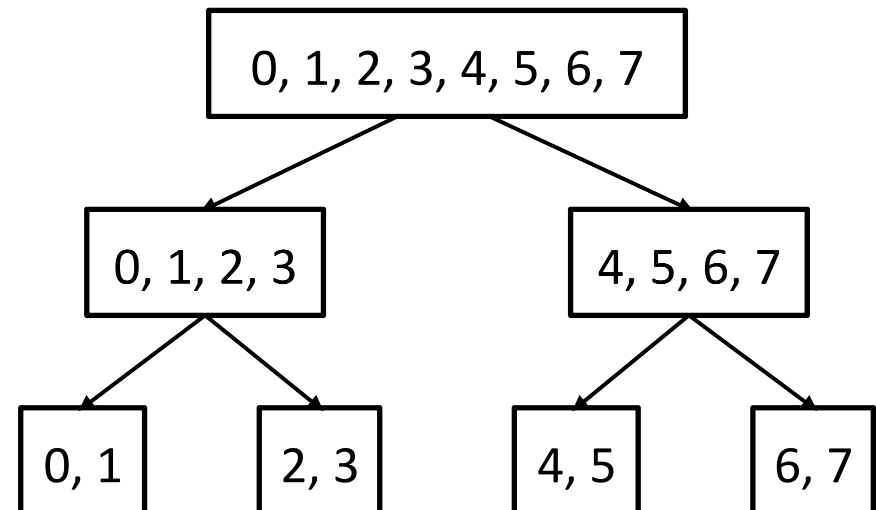


Machine Structure

- Provide mechanism for querying machine structure and thread mapping at runtime

```
Team T = Ti.defaultTeam();
```





Language Constructs

- Thread teams may execute distinct tasks

```
partition(T) {
  { model_fluid(); }
  { model_muscles(); }
  { model_electrical(); }
}
```

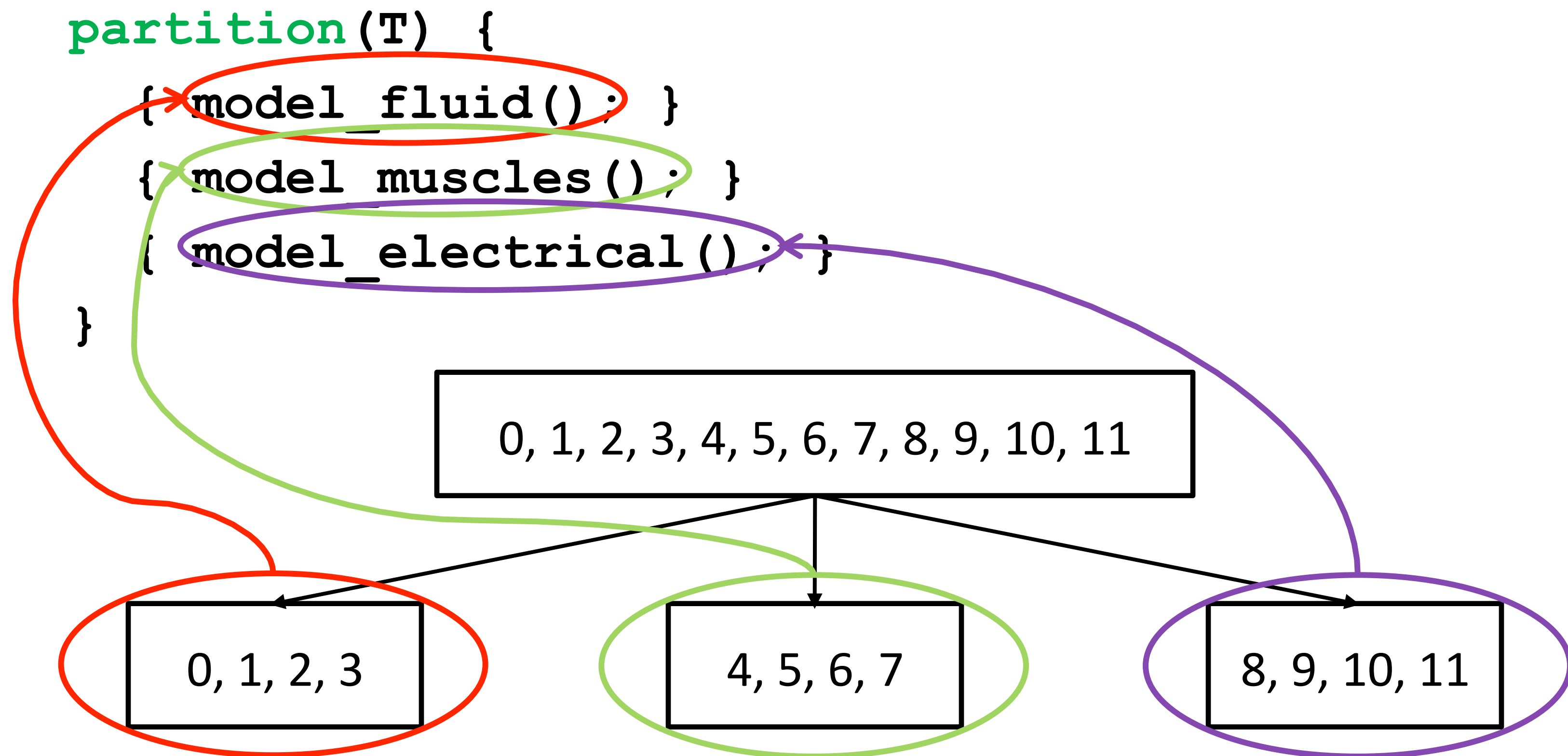
- Threads may execute the same code on different sets of data as part of different teams

```
teamsplit(T) {
  row_reduce();
}
```

- Scoping rules prevent some types of deadlock
  - Execution team determined by enclosing construct

Partition Semantics

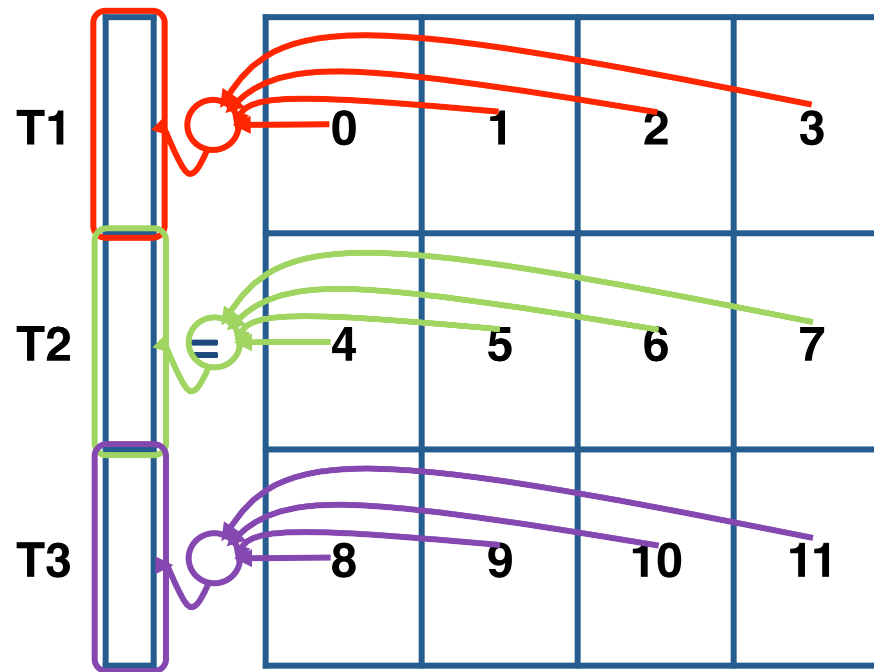
- Different subteams of **`T`** execute each of the branches



Teamsplit Semantics

- Each subteam of **rowTeam** executes the reduction on its own

```
teamsplit(rowTeam) {
  Reduce.add(mtmp, myResults0, rpivot);
}
```



EECS
Electrical Engineering and Computer Sciences

Multiple Hierarchy Levels

BERKELEY PAR LAB

- Constructs can be nested

```
teamsplit(T) {
  teamsplit(T.myChildTeam()) {
    level1_work();
  }
  level2_work();
}
```

- Program can use multiple teams

```
teamsplit(columnTeam) {
  myOut.vbroadcast(cpivot);
}
teamsplit(rowTeam) {
  Reduce.add(mtmp, myResults0, rpivot);
}
```

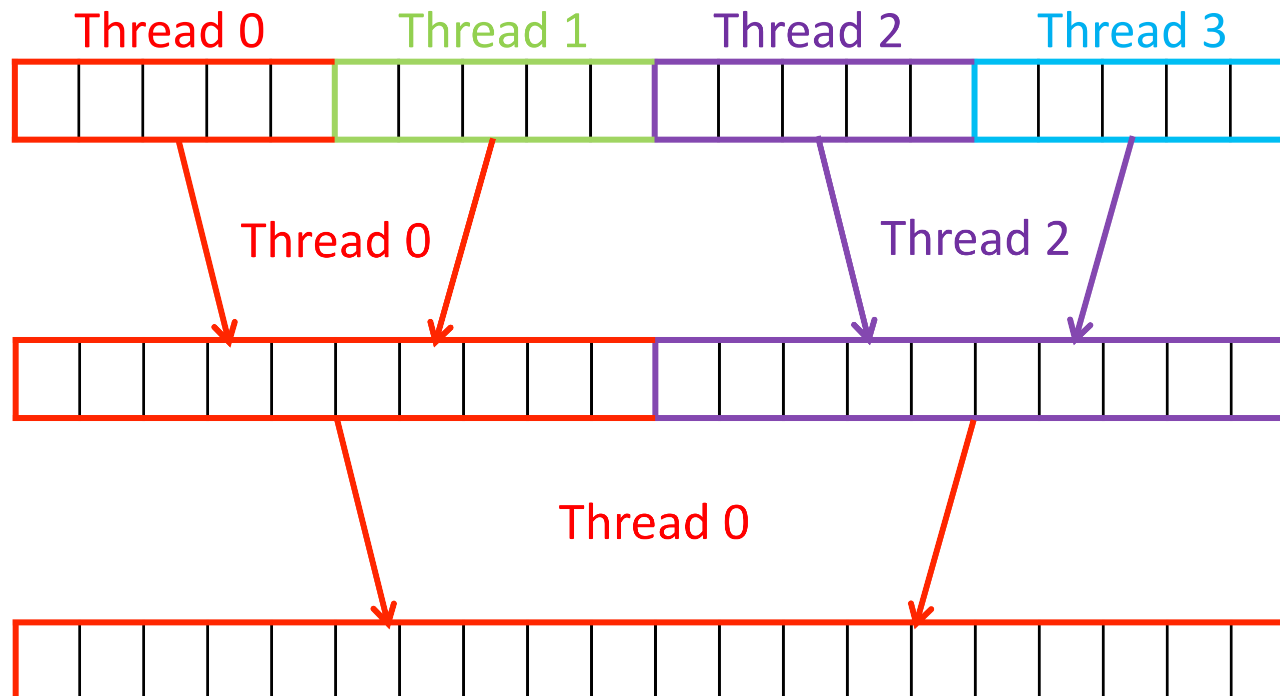
Sorting

- Distributed sorting application using new hierarchical constructs

- Three pieces: sequential, shared memory, and distributed
  - Sequential quick sort from Java 1.4 library
  - Shared memory merge sort
  - Distributed memory sample sort

Shared Memory Sort

- Divide elements equally among threads
  - Each thread processes its elements sequentially
- Merge in parallel
  - Number of threads halved in each iteration

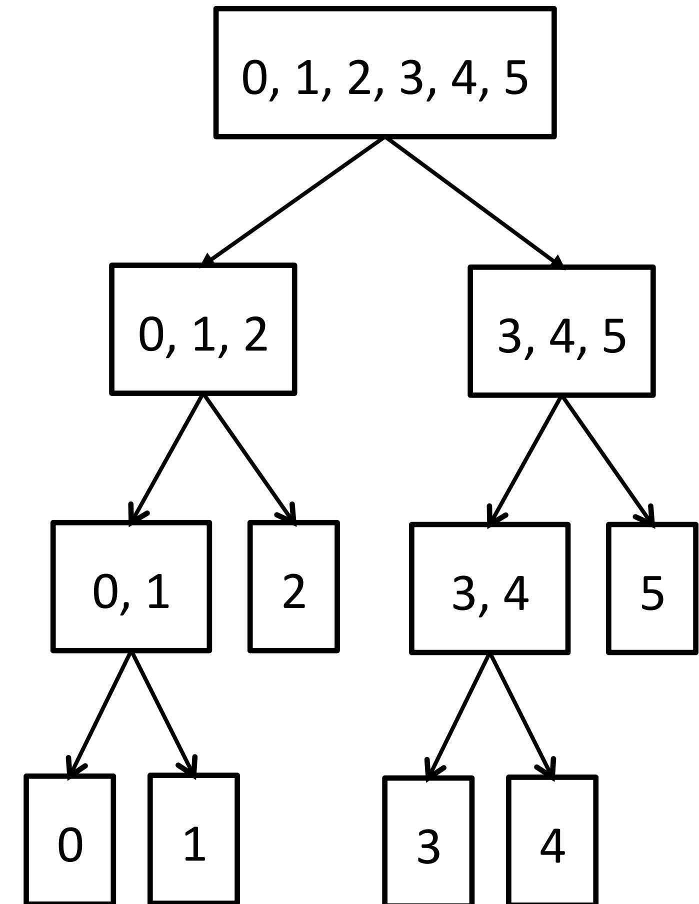


Shared Memory Hierarchy

- Team hierarchy is binary tree
- Trivial construction

```
static void divideTeam(Team t) {
  if (t.size() > 1) {
    t.splitTeam(2);
    divideTeam(t.child(0));
    divideTeam(t.child(1));
  }
}
```

- Threads walk down to bottom of hierarchy, sort, then walk back up, merging along the way



SMP Sort and Merge Logic

- Control logic for sorting and merging

```
static single void sortAndMerge(Team t) {
  if (Ti.numProcs() == 1) {
    allRes[myProc] = sequentialSort(myData);
  } else {
    teamsplit(t) {
      sortAndMerge(t.myChildTeam());
    }
    Ti.barrier();
    if (Ti.thisProc() == 0) {
      int otherProc = myProc + t.child(0).size();
      int[1d] myRes = allRes[myProc];
      int[1d] otherRes = allRes[otherProc];
      int[1d] newRes = target(t.depth(), myRes, otherRes);
      allRes[myProc] = merge(myRes, otherRes, newRes);
    }
  }
}
```

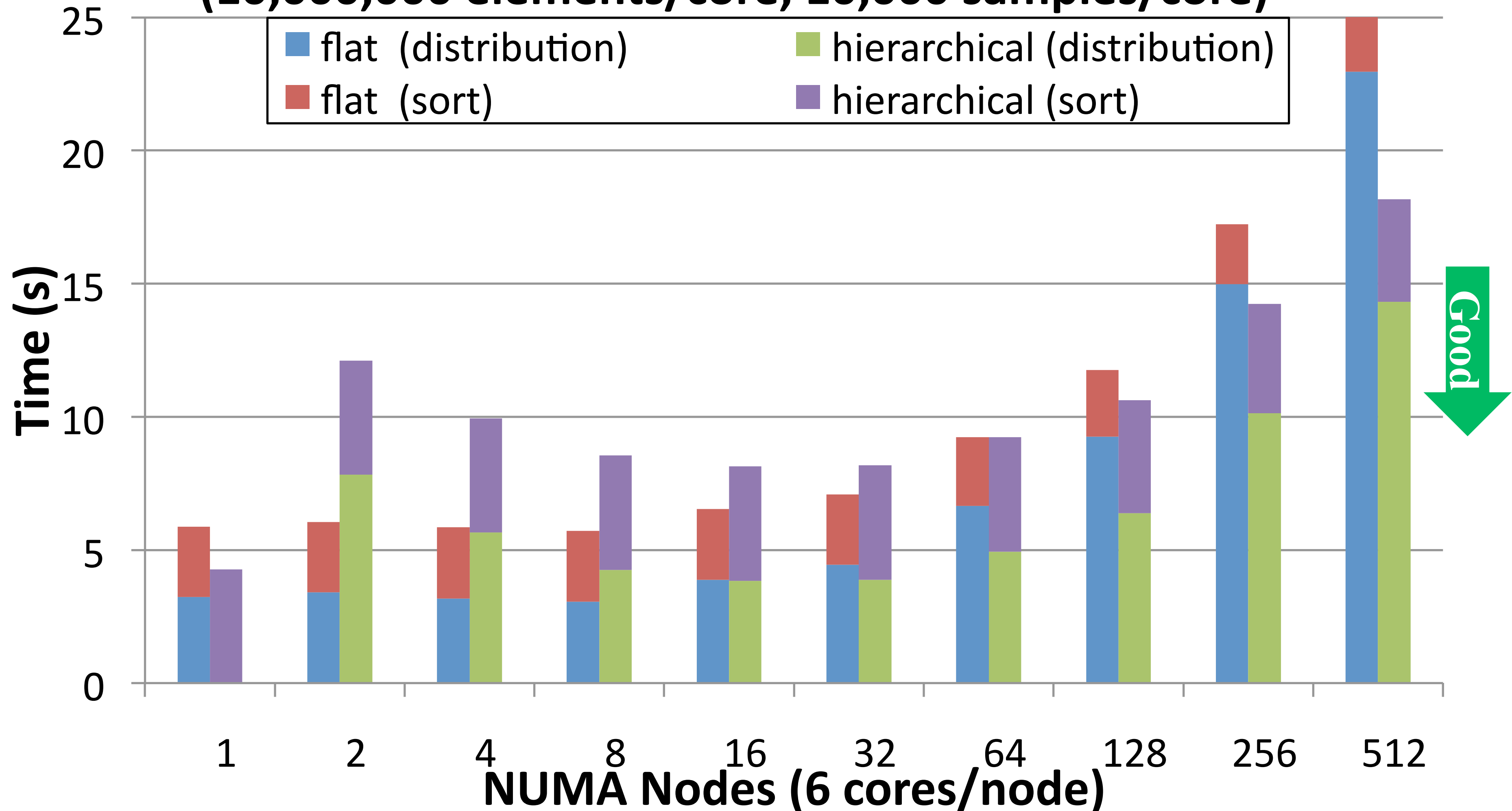
EECS
Electrical Engineering and Computer Sciences

Algorithms for Hierarchical Machines

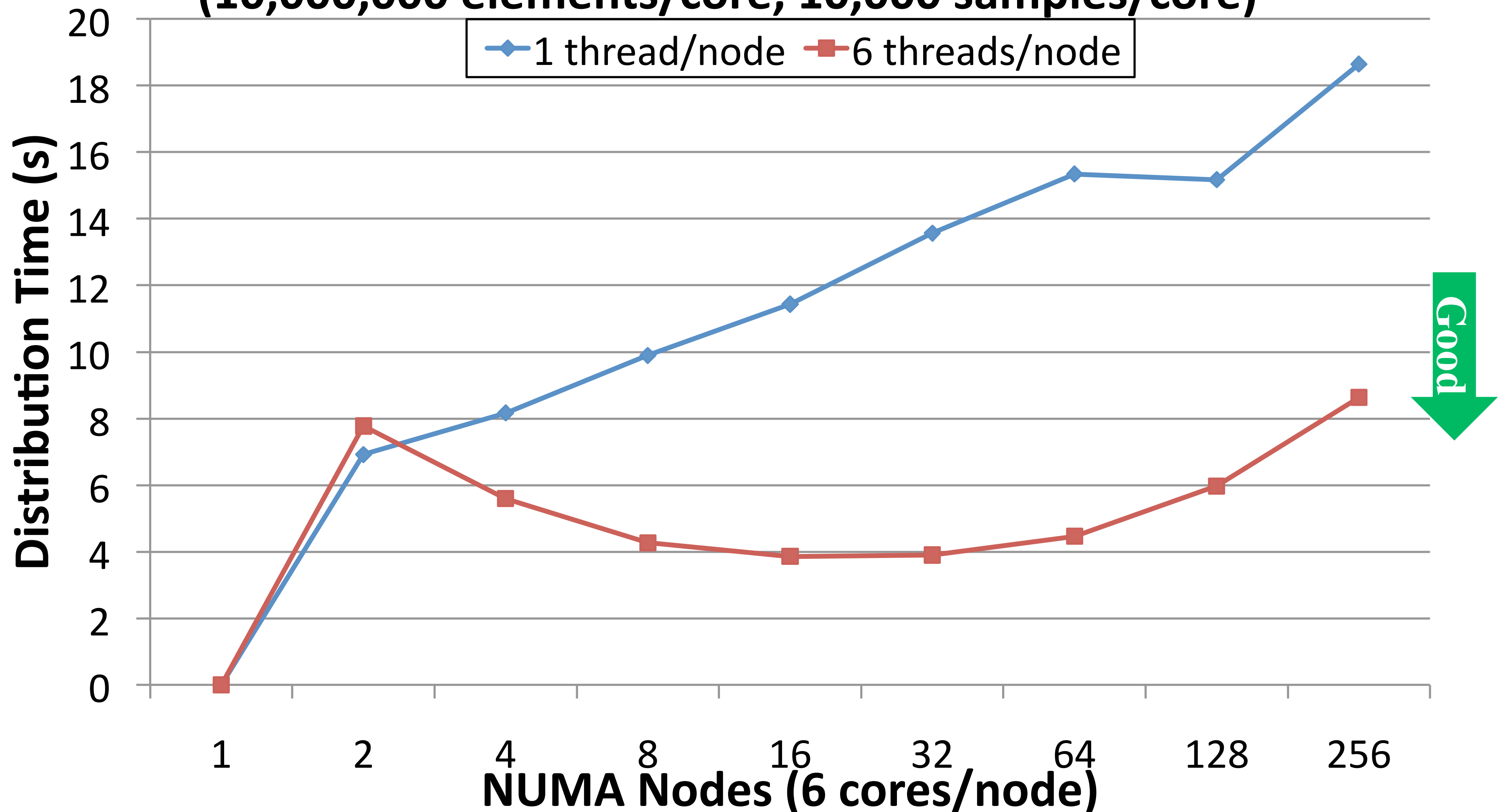
BERKELEY PAR LAB

- Three strategies for hierarchical machines (e.g. clusters of SMPs):
  - Treat the machine as a flat collection of processors that don't share memory
  - Compose a distributed communication library (e.g. MPI) with a shared memory library (e.g. Pthreads)
  - Implement a hierarchical algorithm that takes advantage of both shared memory and all available concurrency
- Sort example:
  - Pure sample sort treats the machine as flat
  - Hierarchical sort uses sampling/distribution between shared-memory domains, SMP sort in a node

Flat vs. Hierarchical Sort



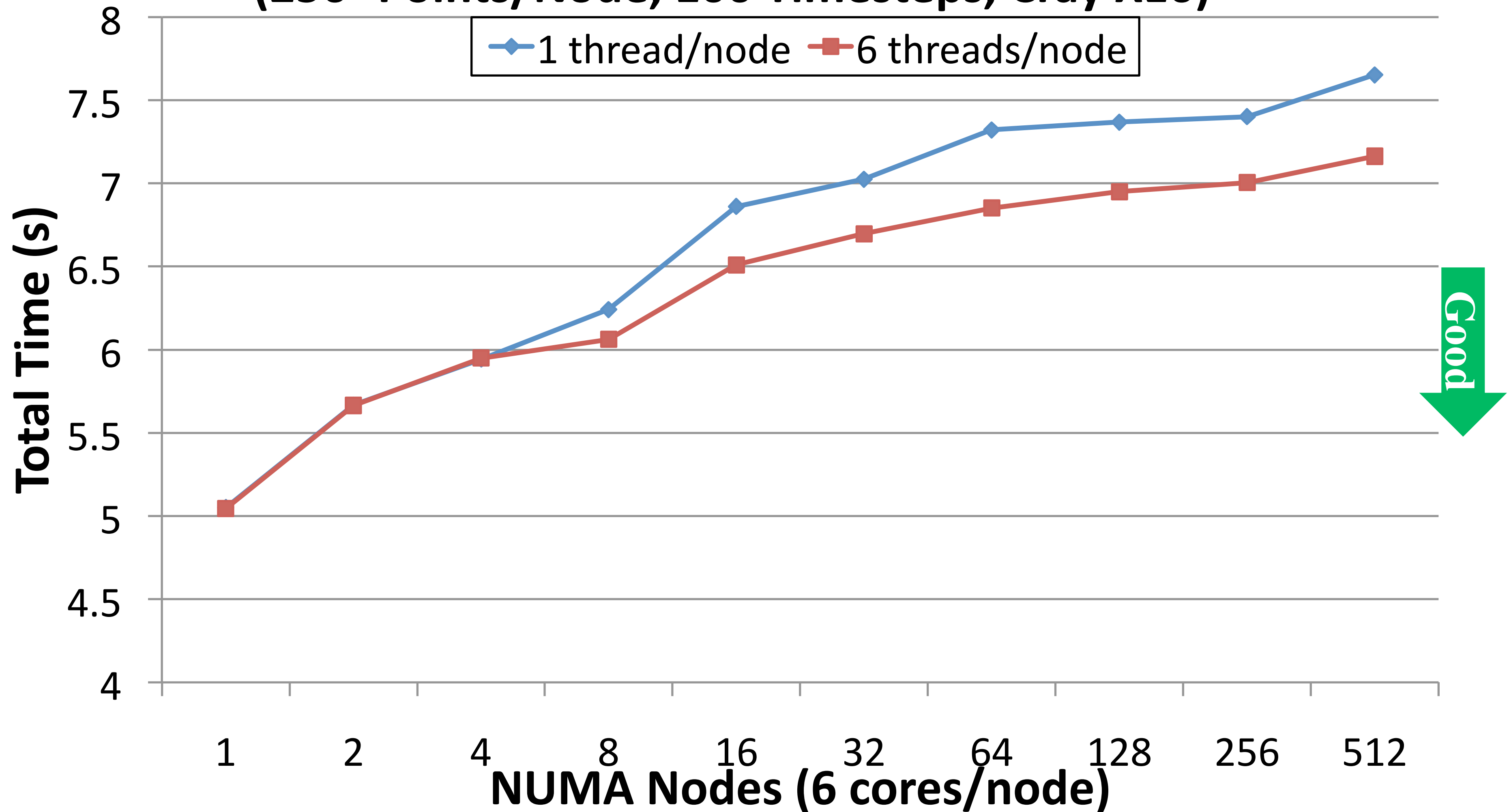
Communication Concurrency



Communication Concurrency

EECS Electrical Engineering and Computer Sciences

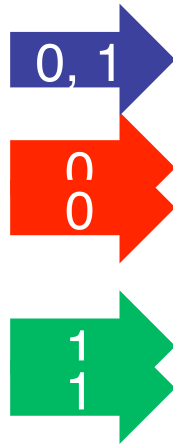
BERKELEY PAR LAB



Dynamic Alignment of Collectives

- Misaligned collective operations can result in deadlock
- Enforcing textual alignment of collectives at runtime can provide safety and analyzability while minimizing programmer burden
- Basic idea:
  - Track control flow on all threads
  - Check that preceding control flow matches when:
    - Performing a team collective
    - Changing team contexts
- Compiler instruments source code to perform tracking and checking

Checking Example



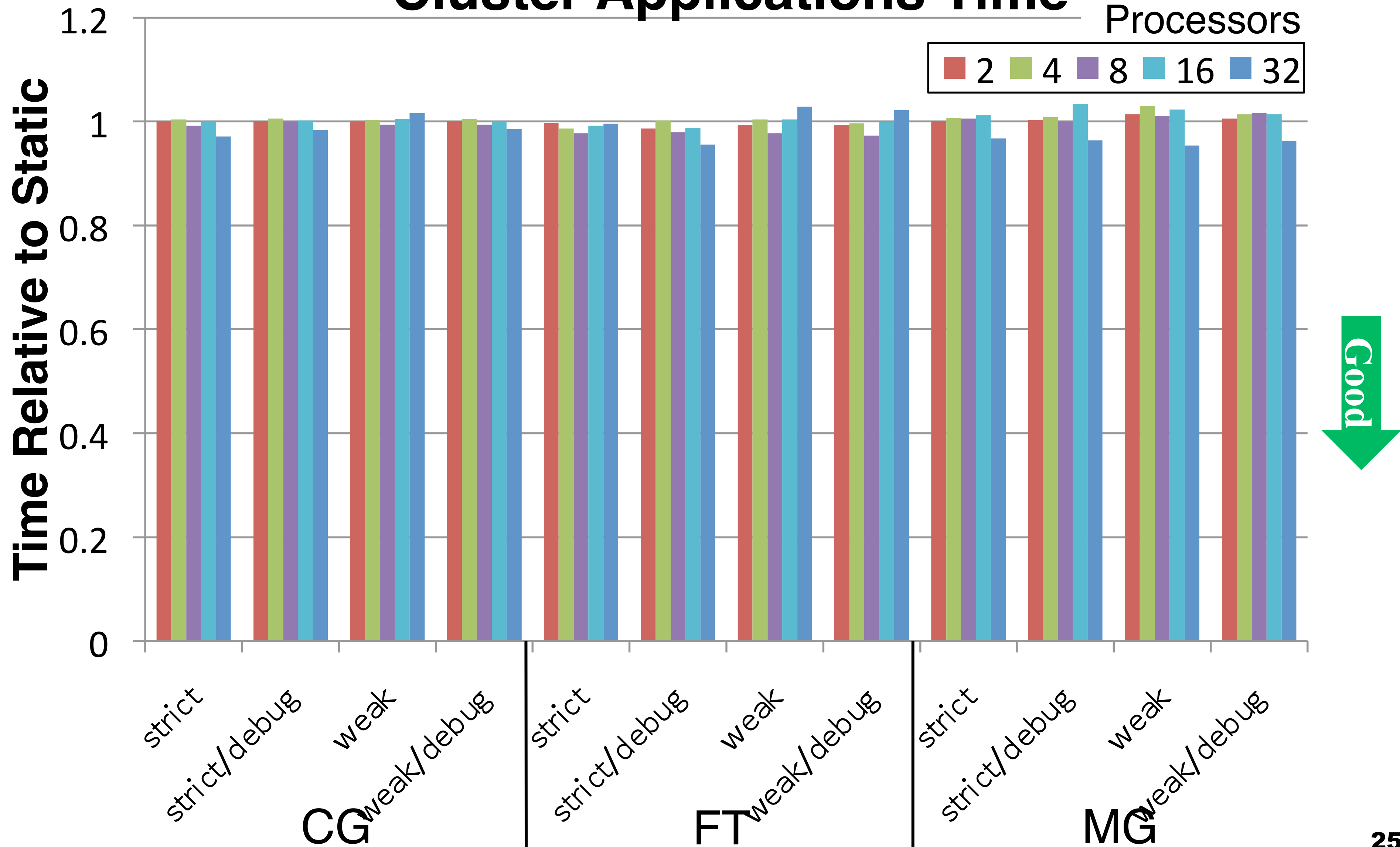

```
5 if (Ti.thisProc() == 0)
6   Ti.barrier();
7 else
8   Ti.barrier();
```

Meaningful error generated local hash

Thread	Hash	Hash from 0	MISALIGNMENT	
0	0x7e8 / 0x0d ERROR	3a6fa0	…*	(5, then)
1	0x2d27693a	0x7e8a6fa0	…*	(5, else)

* Entries prior to line 5

Overhead of Dynamic Alignment is Minimal



Conclusions

- Hierarchical language extensions simplify job of programmer
  - Can organize application around machine characteristics
  - Easier to specify algorithmic hierarchy
  - Seamless code composition
  - Better productivity, performance with team collectives
    - See poster for details
- Language extensions are safe to use
  - Safety provided by lexical scoping and dynamic alignment checking

This slide intentionally left blank.