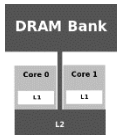
CACHE LINE AWARE OPTIMIZATIONS FOR CCNUMA SYSTEMS

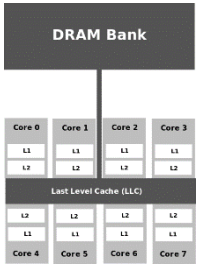
Sabela Ramos (sramos@udc.es)
GAC, Universidade da Coruña (Spain)
Torsten Hoefler (htor@inf.ethz.ch)
SPCL, ETH Zurich (Switzerland)

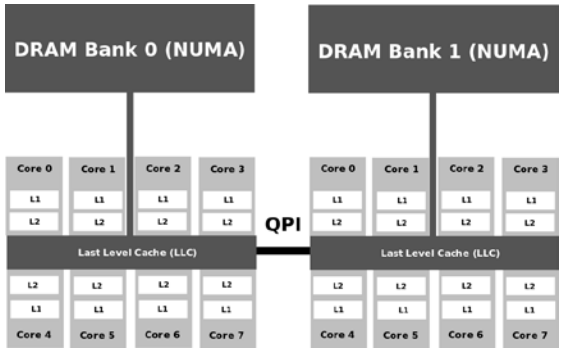
24th ACM International Symposium on High-Performance Parallel and Distributed Computing
HPDC'15, Portland, 2015

WHAT IS THE PROBLEM?

- The increase in
  - Number of cores per processor
  - Complexity of memory hierarchies







- Programmability is maintained through cache coherence

- Which hides peformance characteristics.

OUR PROPOSAL: CLA DESIGN

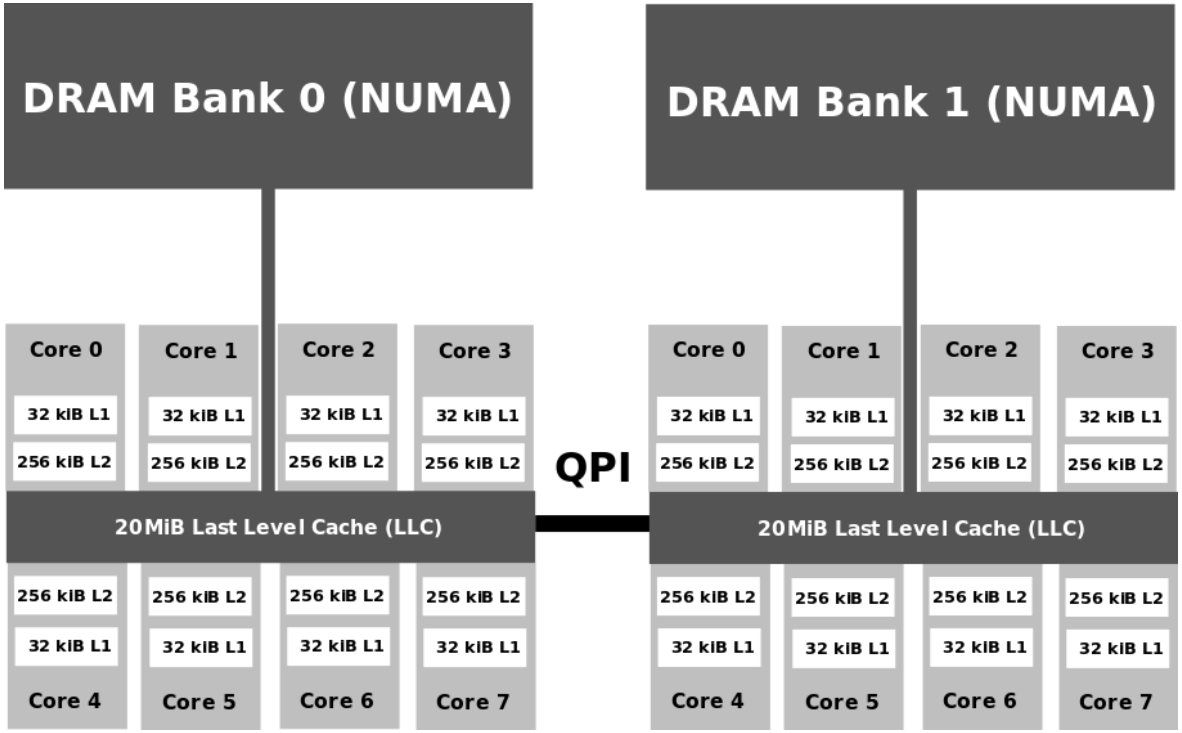
- GOAL: help programmers to be Cache-Aware

- HOW? 

1. Detailed (but simple) performance model of the CC protocol
2. Methodology to translate algorithms into models
3. Select/Optimize/Design algorithms

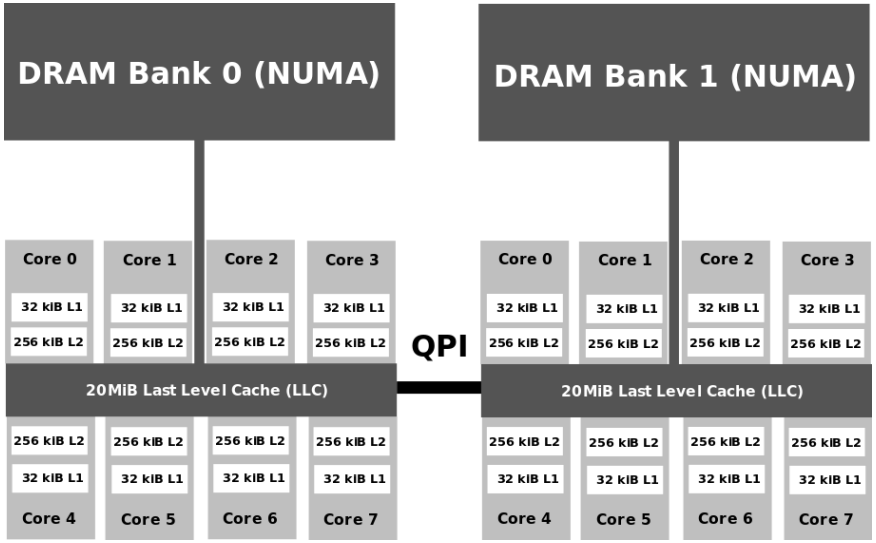
OUR TESTBED

- Dual socket Intel Xeon Sandy Bridge E5-2660
- CC protocol: MESIF



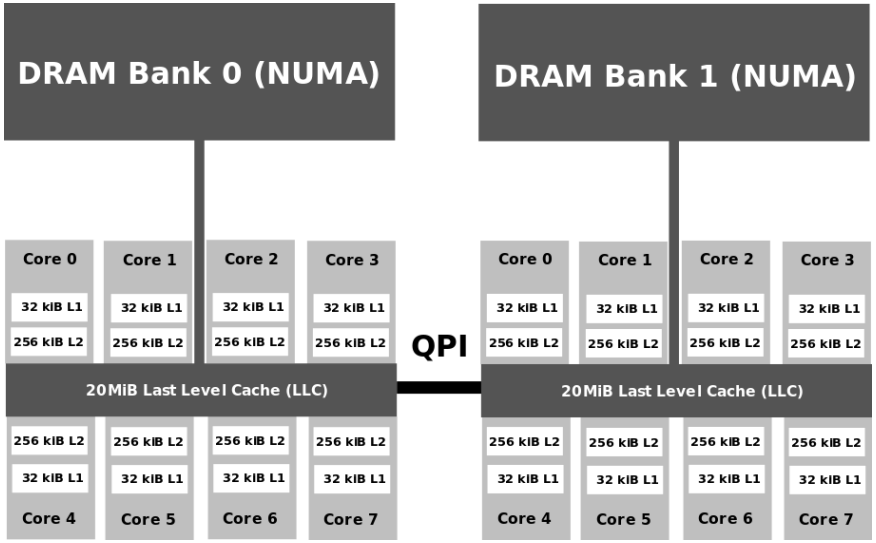
1. PERFORMANCE MODEL
Building Blocks (I)

- Single-Line Transfers
- Multi-Line Transfers



1. PERFORMANCE MODEL
Building Blocks (I)

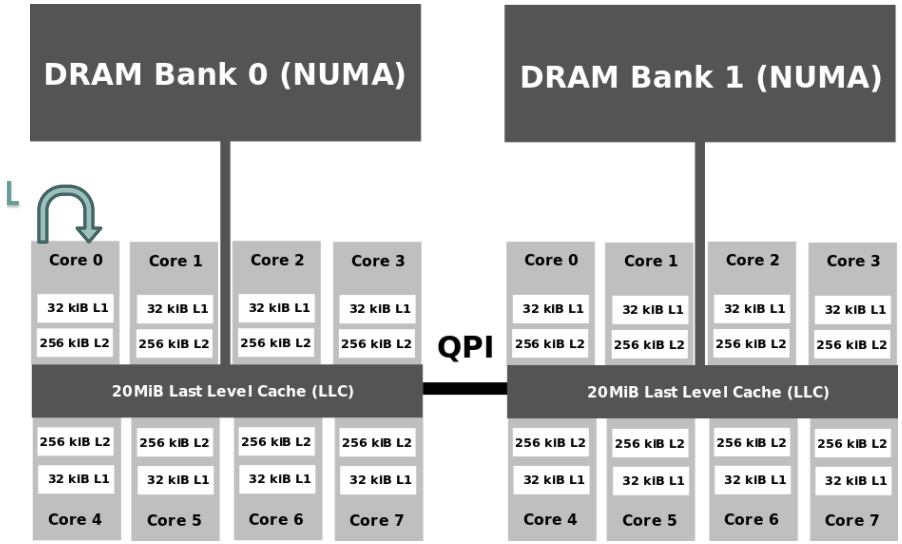
- Single-Line Transfers
- Multi-Line Transfers



1. PERFORMANCE MODEL
Building Blocks (I)

- Single-Line Transfers
- Multi-Line Transfers

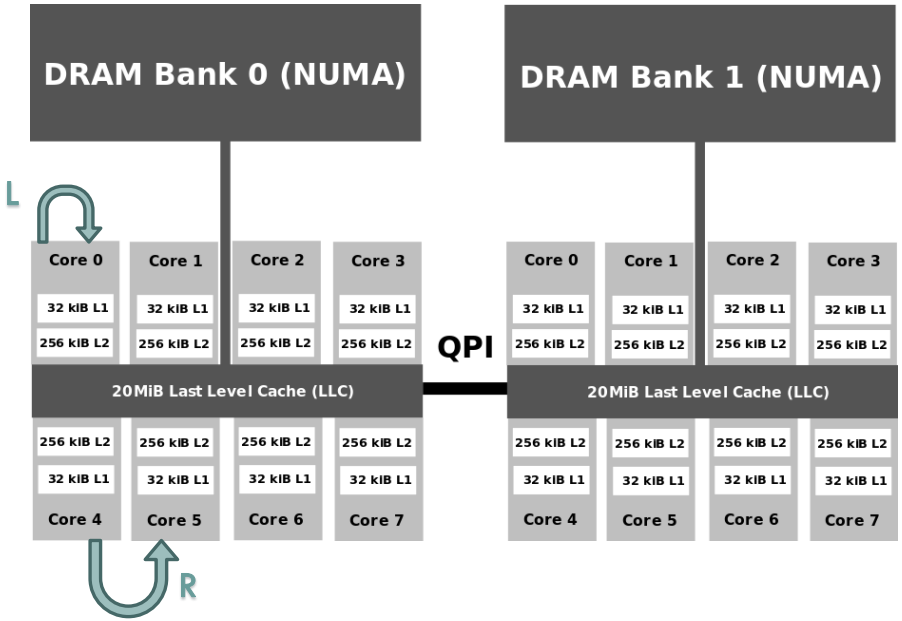
- L: Local



1. PERFORMANCE MODEL
Building Blocks (I)

- Single-Line Transfers
- Multi-Line Transfers

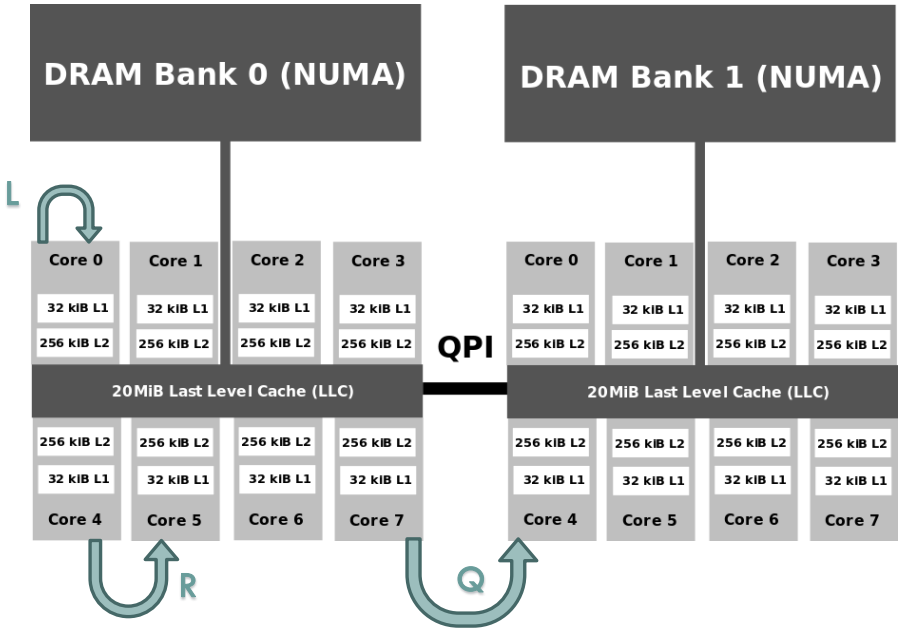
- L: Local
- R: Remote – same socket



1. PERFORMANCE MODEL
Building Blocks (I)

- Single-Line Transfers
- Multi-Line Transfers

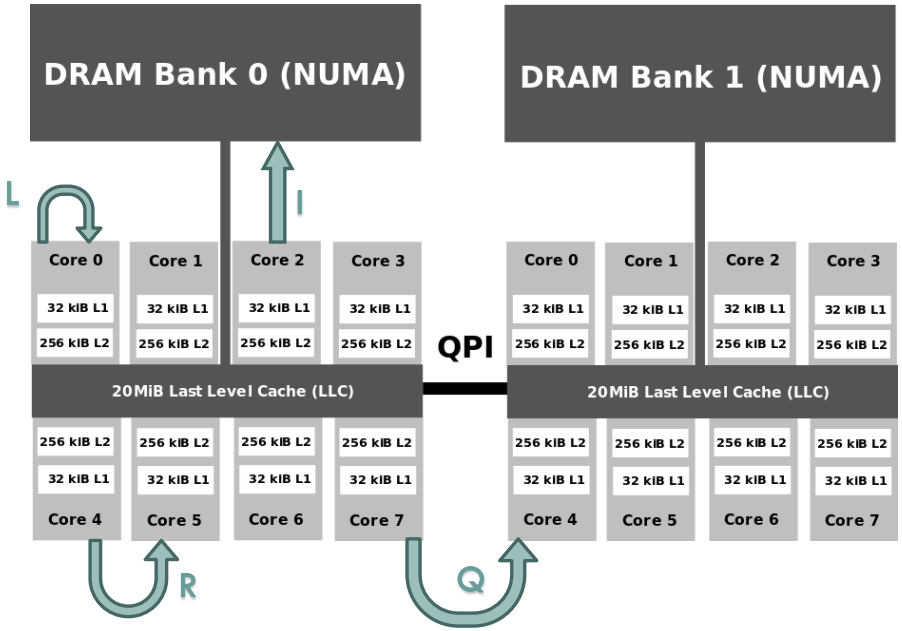
- L: Local
- R: Remote – same socket
- Q: Remote – different sockets



1. PERFORMANCE MODEL
Building Blocks (I)

- Single-Line Transfers
- Multi-Line Transfers

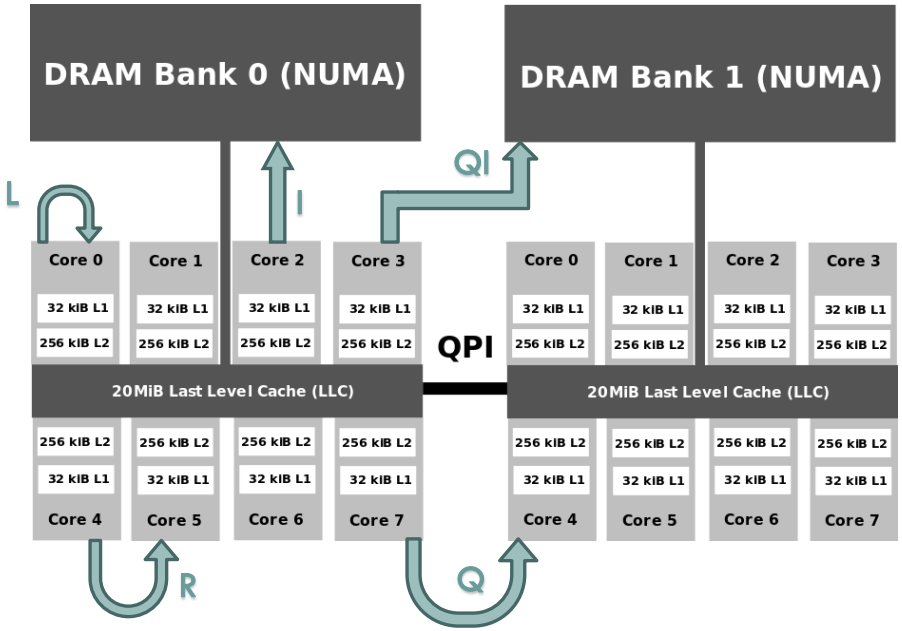
- L: Local
- R: Remote – same socket
- Q: Remote – different sockets
- I: From memory – same socket



1. PERFORMANCE MODEL
Building Blocks (I)

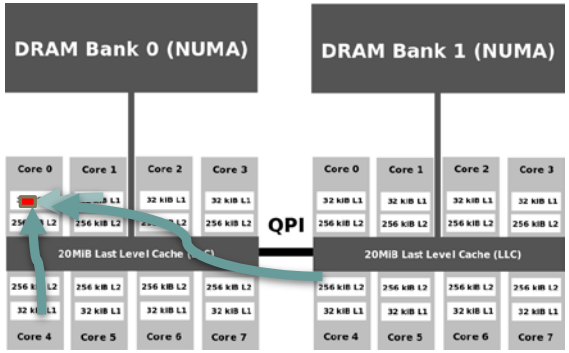
- Single-Line Transfers
- Multi-Line Transfers

- L: Local
- R: Remote – same socket
- Q: Remote – different sockets
- I: From memory – same socket
- QI: From memory – different sockets

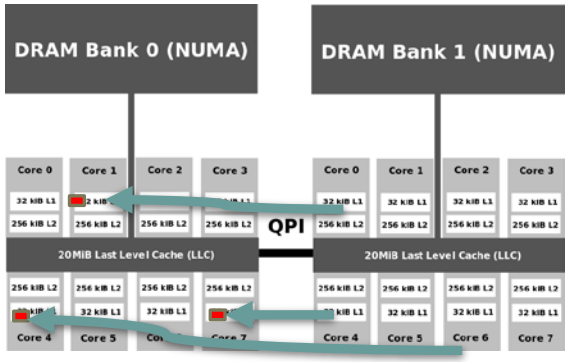


1. PERFORMANCE MODEL
Building Blocks (II)

- Contention
  - Several threads accessing the same line simultaneously
  - Sandy Bridge does not suffer from contention



- Congestion
  - Several threads accessing different lines simultaneously
  - The QPI link suffers from congestion → Regression model



1. PERFORMANCE MODEL
Invalidation and Cache-line Stealing

- RFO of a shared line
- Cache-line stealing
  - Caused by
    - Polling
    - False-sharing

Source of Variability

- Solution? → MIN MAX MODELS

2. CLA
Cla Pseudo-code

- Copy N lines: `cl_copy (cl_t* src, cl_t* dest, int N)`
- Wait (poll): `cl_wait (cl_t* line, clv_t val, op_t comp=eq)`
- Write: `cl_write (cl_t* line, clv_t val)`
- Add: `cl_add (cl_t* line, clv_t val)`

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 1: within the same thread

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 1: within the same thread

Thread 0:
S1: cl_write(a,5)
S2: cl_write(b,6)

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 1: within the same thread

S1

Thread 0:
S1: cl_write(a,5)
S2: cl_write(b,6)

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 1: within the same thread

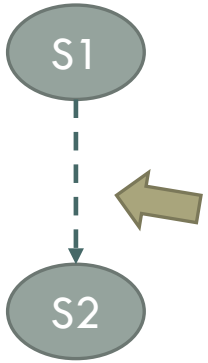
Thread 0:
S1: cl_write(a,5)
S2: cl_write(b,6)

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 1: within the same thread

Thread 0:
S1: cl_write(a,5)
S2: cl_write(b,6)



2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 2: dependency between threads

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 2: dependency between threads

Thread 0:
S01: cl_write(a,5)

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 2: dependency between threads

Thread 0:
S01: cl_write(a,5)

S01

Thread 1:
S11: cl_wait(a,5)

S11

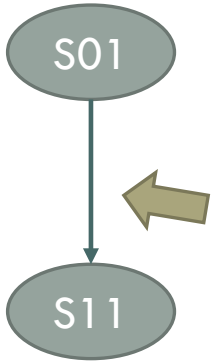
2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 2: dependency between threads

Thread 0:
S01: cl_write(a,5)

Thread 1:
S11: cl_wait(a,5)



2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 3: sequential restriction between threads

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 3: sequential restriction between threads

Thread 0:
S01: cl_add(a,1)

S01

2. CLA
Cla Graph

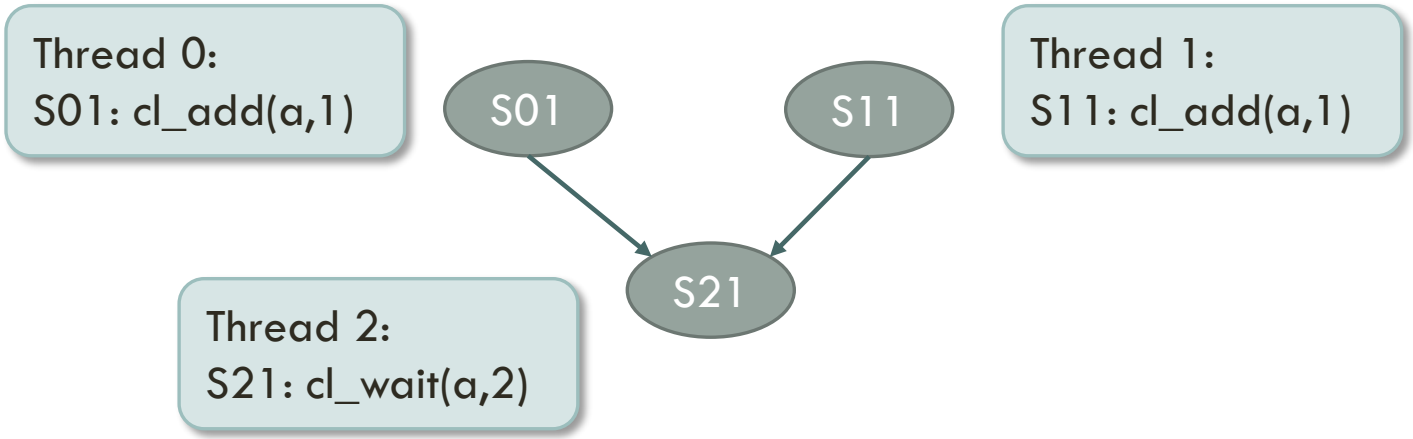
- Nodes: CLa operations
- Edges:

Edge 3: sequential restriction between threads

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

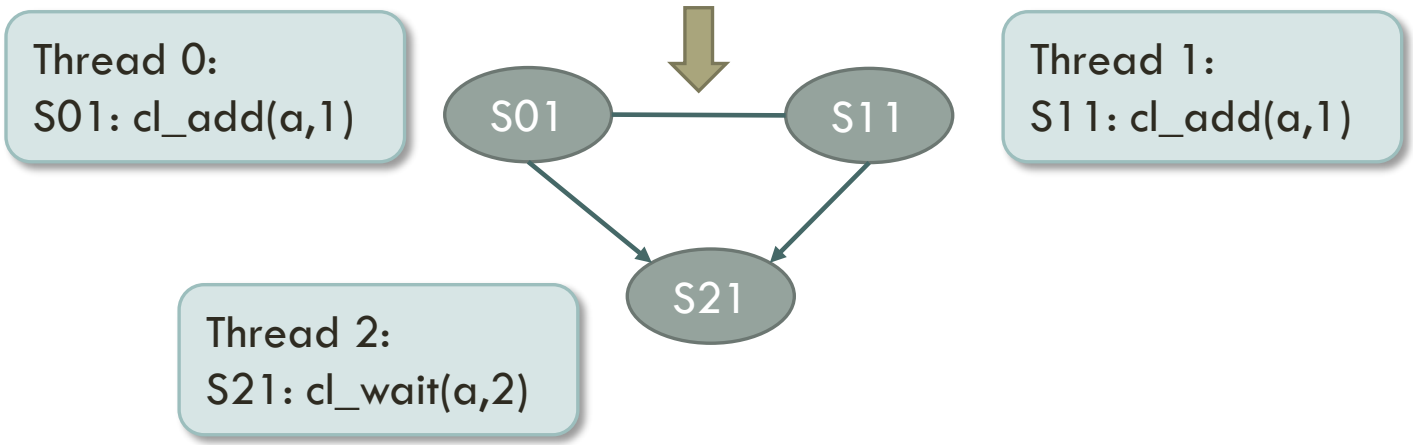
Edge 3: sequential restriction between threads



2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 3: sequential restriction between threads



2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 4: line-stealing caused by non-related operations

2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:

Edge 4: line-stealing caused by non-related operations

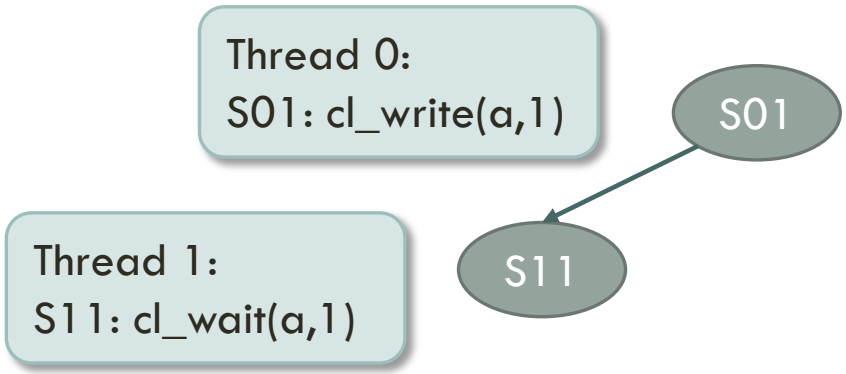
Thread 0:
S01: cl_write(a,1)

S01

2. CLA
Cla Graph

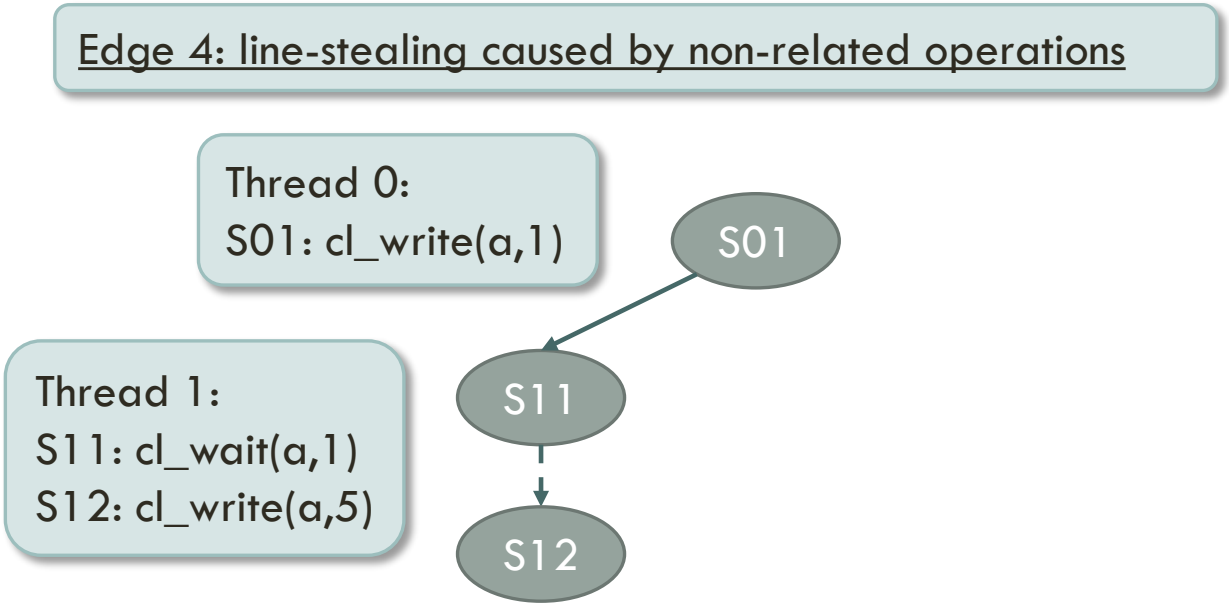
- Nodes: CLa operations
- Edges:

Edge 4: line-stealing caused by non-related operations



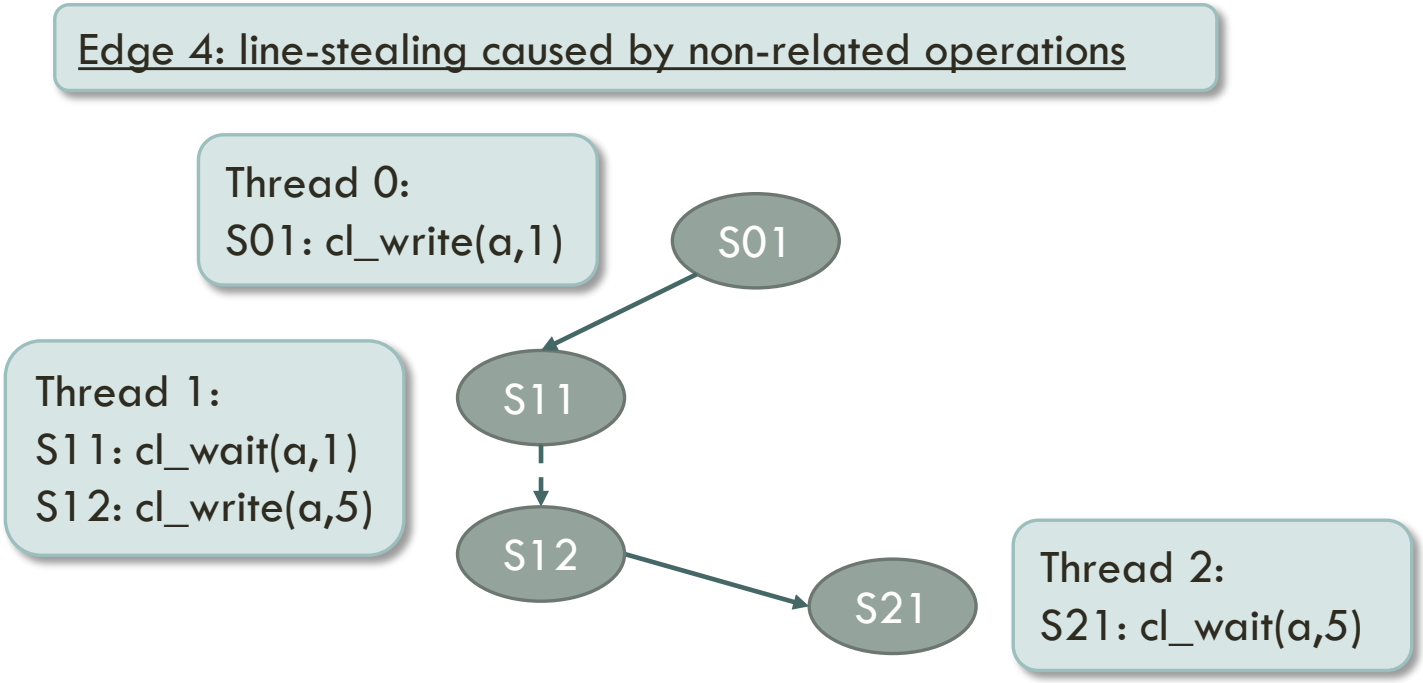
2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:



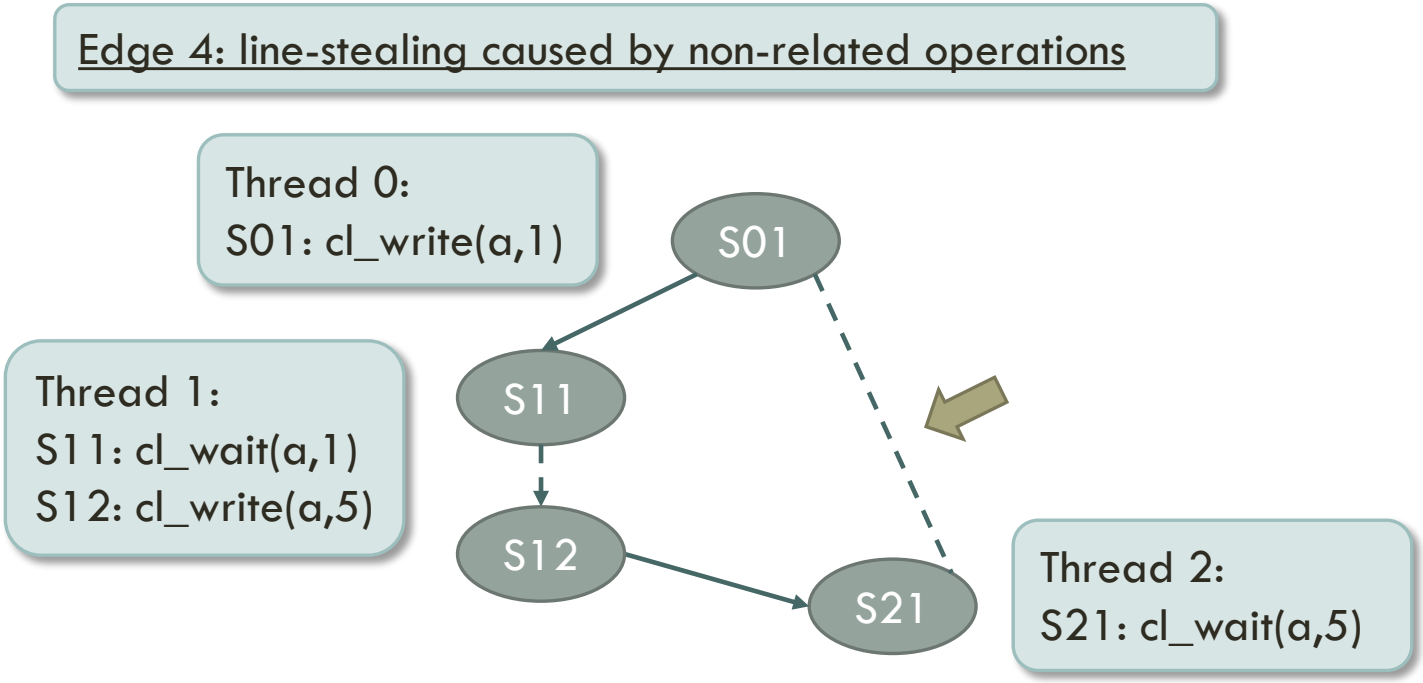
2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:



2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:



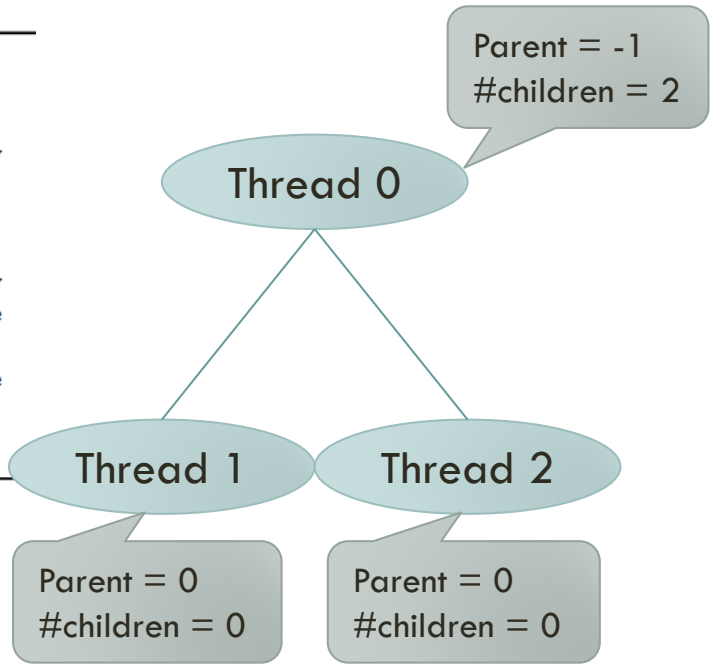
2. CLA
Cla Graph

- Nodes: CLa operations
- Edges:
  - Edge 1: within the same thread
  - Edge 2: dependency between threads
  - Edge 3: sequential restriction between threads
  - Edge 4: line-stealing caused by non-related operations
- Set of rules to obtain the T_{min}

3. ALGORITHM DESIGN

Example: Broadcast

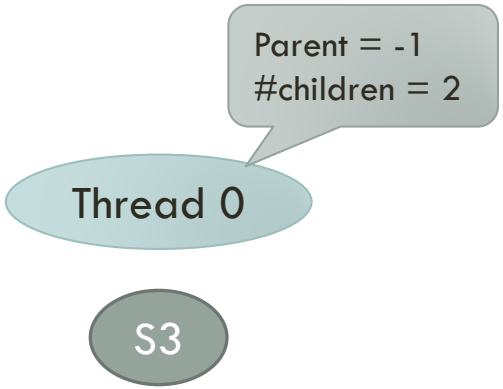
```
     Function OneLineBroadcast(int me, cl_t * mydata, tree_t
     tree)
         if tree.parent != -1 then
[S1]       cl_wait(tree.pflag[tree.parent],1);      //one-to-many
[S2]       cl_copy(tree.data[tree.parent],mydata,1);
         if tree.children > 0 then
[S3]       cl_copy(mydata,tree.data[me],1);
[S4]       cl_write(tree.pflag[me],1);              //one-to-many
[S5]       cl_wait(tree.sflag[me],tree.children); //many-to-one
         if tree.parent != -1 then
[S6]       cl_add(tree.sflag[tree.parent],1);       //many-to-one
         end
     end
```



3. ALGORITHM DESIGN

Example: Broadcast

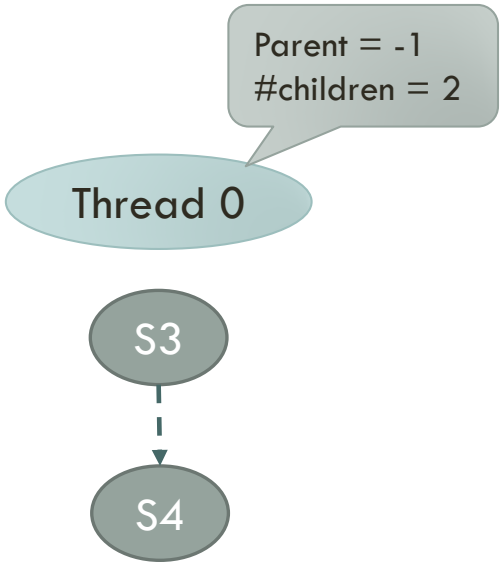
```
     Function OneLineBroadcast(int me, cl_t * mydata, tree_t
     tree)
         if tree.parent != -1 then
[S1]       cl_wait(tree.pflag[tree.parent],1);      //one-to-many
[S2]       cl_copy(tree.data[tree.parent],mydata,1);
         if tree.children > 0 then
[S3]       cl_copy(mydata,tree.data[me],1);
[S4]       cl_write(tree.pflag[me],1);              //one-to-many
[S5]       cl_wait(tree.sflag[me],tree.children); //many-to-one
         if tree.parent != -1 then
[S6]       cl_add(tree.sflag[tree.parent],1);       //many-to-one
         end
     end
```



3. ALGORITHM DESIGN

Example: Broadcast

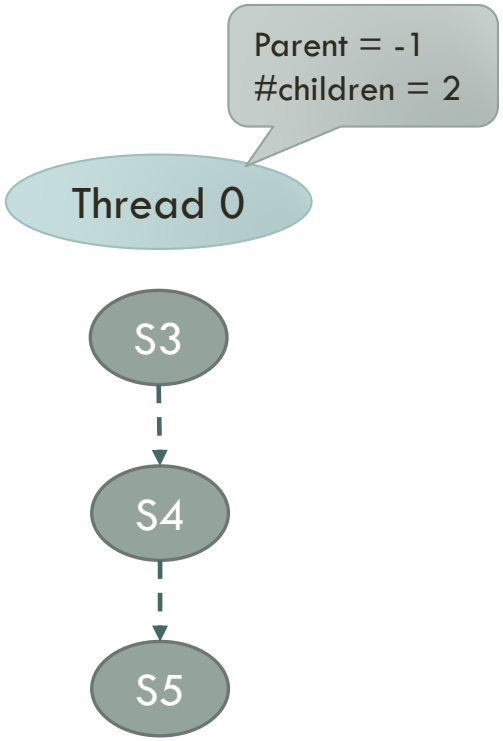
```
Function OneLineBroadcast(int me, cl_t * mydata, tree_t tree)
    if tree.parent != -1 then
[S1]    cl_wait(tree.pflag[tree.parent],1);      //one-to-many
[S2]    cl_copy(tree.data[tree.parent],mydata,1);
    if tree.children > 0 then
[S3]    cl_copy(mydata,tree.data[me],1);
[S4]    cl_write(tree.pflag[me],1);              //one-to-many
[S5]    cl_wait(tree.sflag[me],tree.children); //many-to-one
    if tree.parent != -1 then
[S6]    cl_add(tree.sflag[tree.parent],1);       //many-to-one
    end
end
```



3. ALGORITHM DESIGN

Example: Broadcast

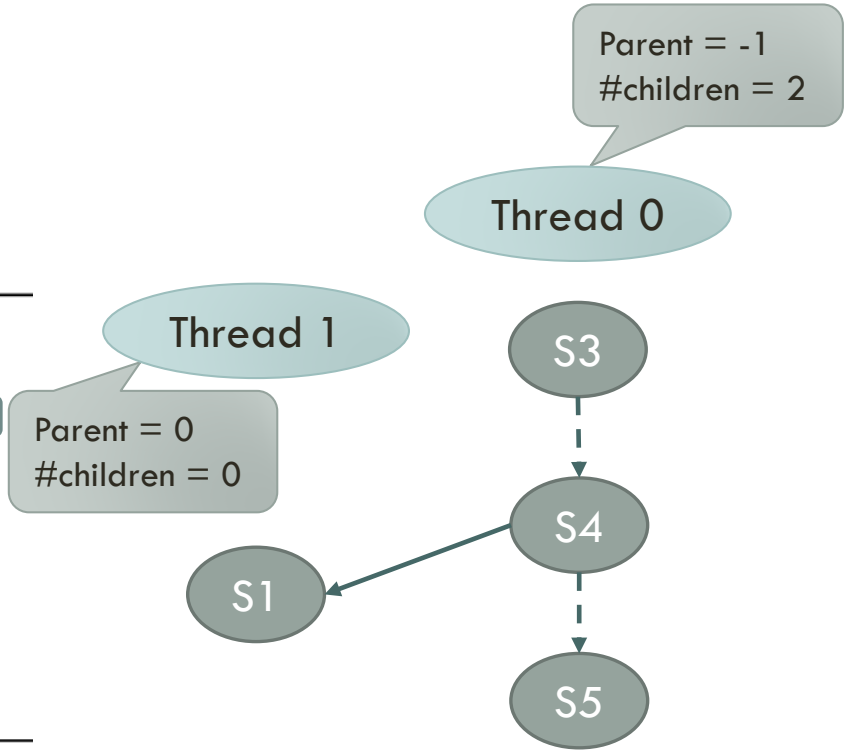
```
Function OneLineBroadcast(int me, cl_t * mydata, tree_t tree)
    if tree.parent != -1 then
[S1]    cl_wait(tree.pflag[tree.parent],1);      //one-to-many
[S2]    cl_copy(tree.data[tree.parent],mydata,1);
    if tree.children > 0 then
[S3]    cl_copy(mydata,tree.data[me],1);
[S4]    cl_write(tree.pflag[me],1);              //one-to-many
[S5]    cl_wait(tree.sflag[me],tree.children); //many-to-one
    if tree.parent != -1 then
[S6]    cl_add(tree.sflag[tree.parent],1);       //many-to-one
    end
end
```



3. ALGORITHM DESIGN

Example: Broadcast

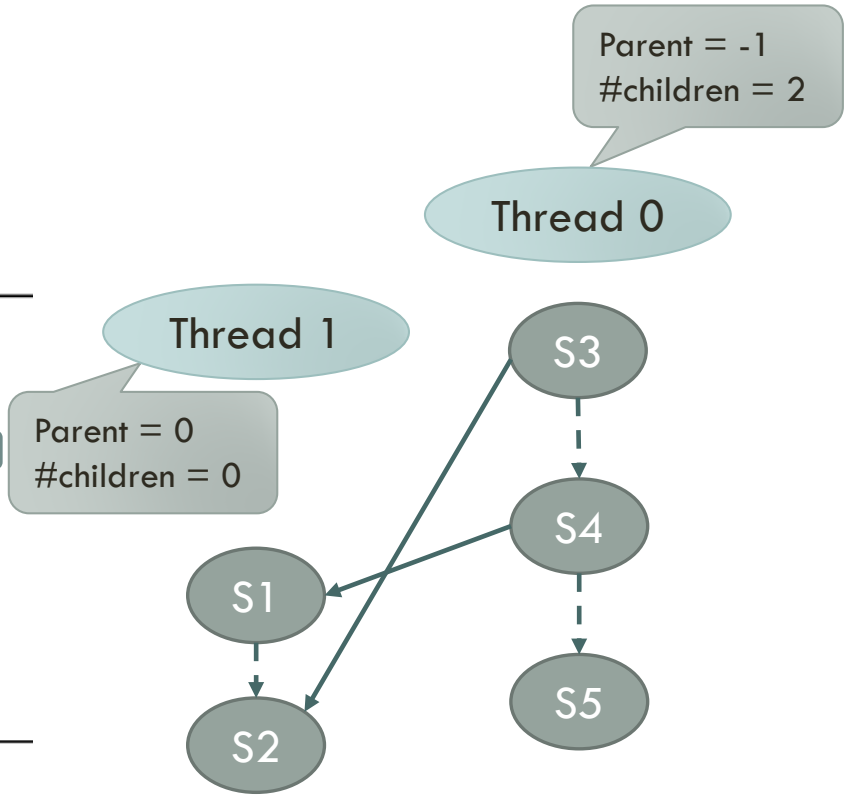
```
Function OneLineBroadcast(int me, cl_t * mydata, tree_t
tree)
    if tree.parent != -1 then
[S1]    cl_wait(tree.pflag[tree.parent],1);      //one-to-many
[S2]    cl_copy(tree.data[tree.parent],mydata,1);
    if tree.children > 0 then
[S3]    cl_copy(mydata,tree.data[me],1);
[S4]    cl_write(tree.pflag[me],1);              //one-to-many
[S5]    cl_wait(tree.sflag[me],tree.children); //many-to-one
    if tree.parent != -1 then
[S6]    cl_add(tree.sflag[tree.parent],1);       //many-to-one
    end
end
```



3. ALGORITHM DESIGN

Example: Broadcast

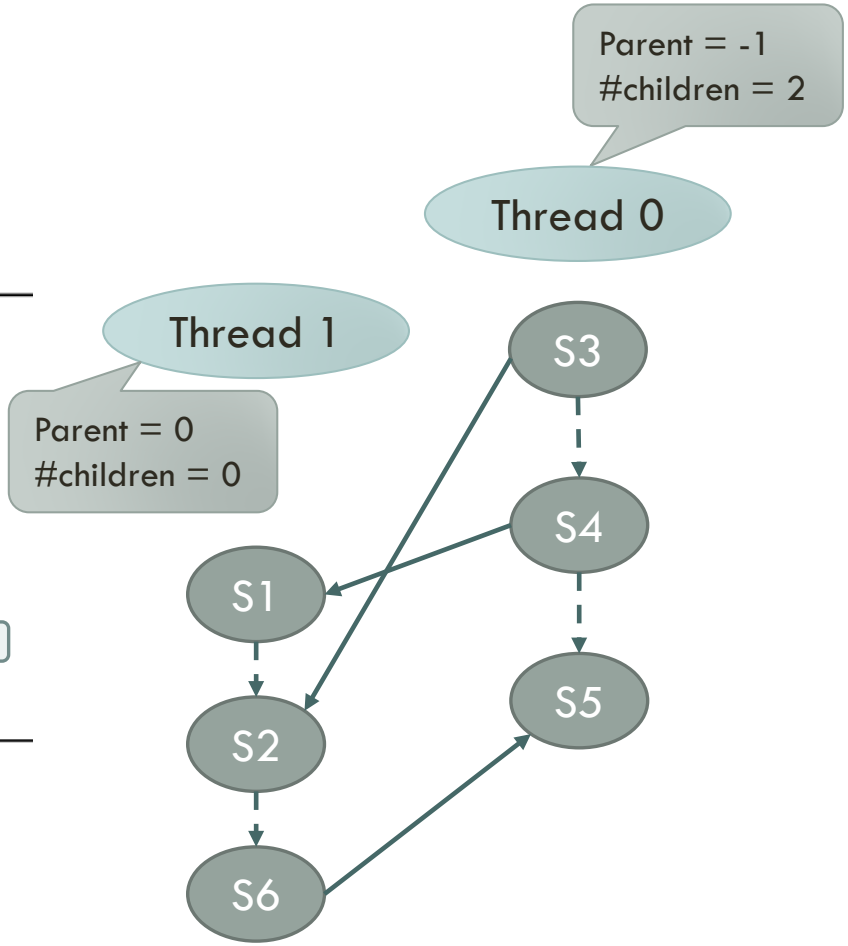
```
Function OneLineBroadcast(int me, cl_t * mydata, tree_t tree)
    if tree.parent != -1 then
[S1]    cl_wait(tree.pflag[tree.parent],1);      //one-to-many
[S2]    cl_copy(tree.data[tree.parent],mydata,1);
    if tree.children > 0 then
[S3]    cl_copy(mydata,tree.data[me],1);
[S4]    cl_write(tree.pflag[me],1);              //one-to-many
[S5]    cl_wait(tree.sflag[me],tree.children); //many-to-one
    if tree.parent != -1 then
[S6]    cl_add(tree.sflag[tree.parent],1);       //many-to-one
    end
end
```



3. ALGORITHM DESIGN

Example: Broadcast

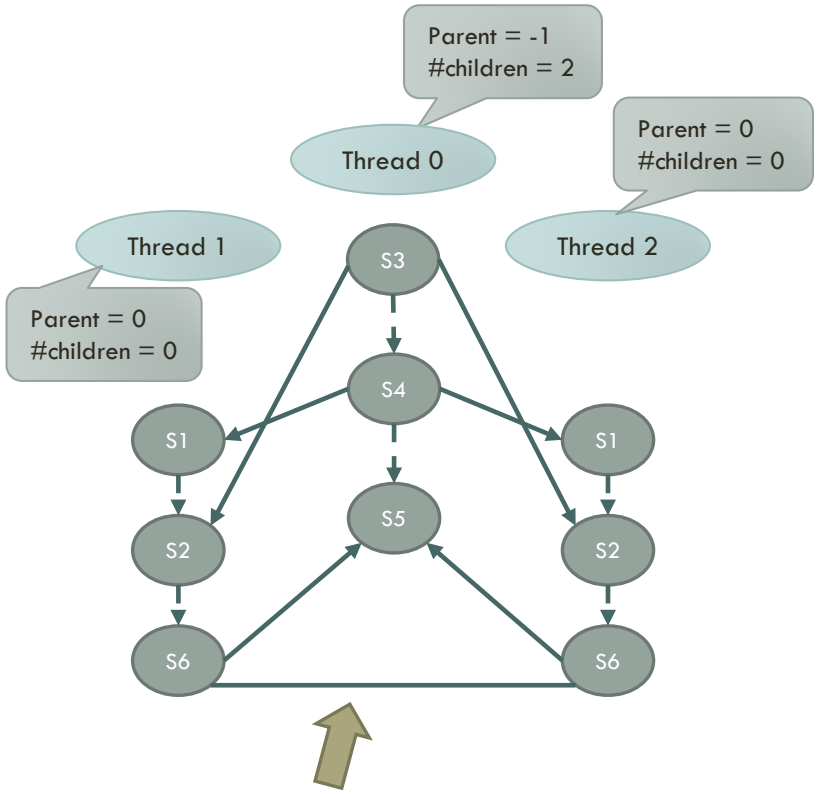
```
Function OneLineBroadcast(int me, cl_t * mydata, tree_t
tree)
    if tree.parent != -1 then
[S1]    cl_wait(tree.pflag[tree.parent],1);      //one-to-many
[S2]    cl_copy(tree.data[tree.parent],mydata,1);
    if tree.children > 0 then
[S3]    cl_copy(mydata,tree.data[me],1);
[S4]    cl_write(tree.pflag[me],1);              //one-to-many
[S5]    cl_wait(tree.sflag[me],tree.children); //many-to-one
    if tree.parent != -1 then
[S6]    cl_add(tree.sflag[tree.parent],1);       //many-to-one
    end
end
```



3. ALGORITHM DESIGN

Example: Broadcast

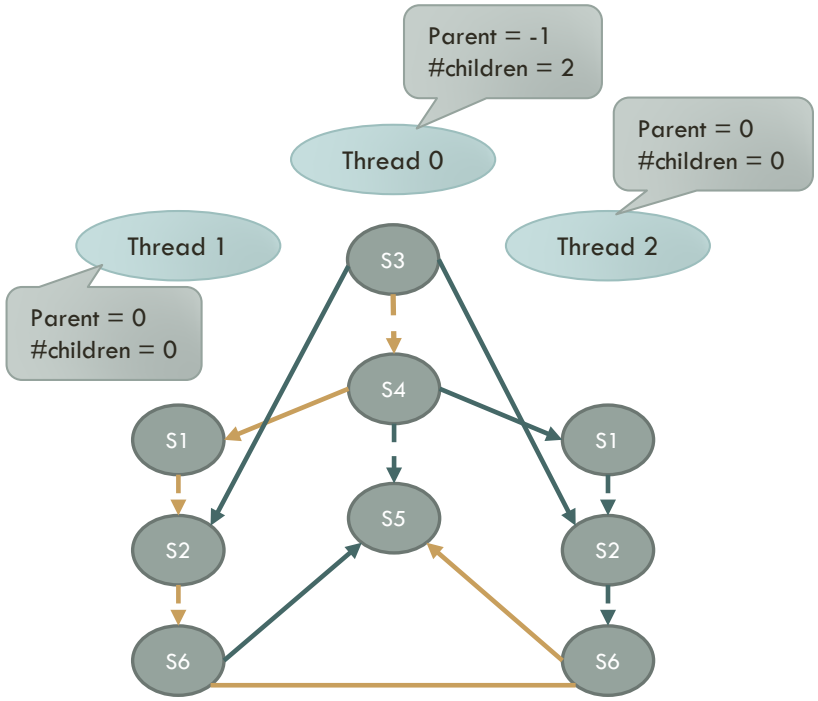
```
Function OneLineBroadcast(int me, cl_t * mydata, tree_t tree)
    if tree.parent != -1 then
[S1]    cl_wait(tree.pflag[tree.parent],1);      //one-to-many
[S2]    cl_copy(tree.data[tree.parent],mydata,1);
    if tree.children > 0 then
[S3]    cl_copy(mydata,tree.data[me],1);
[S4]    cl_write(tree.pflag[me],1);              //one-to-many
[S5]    cl_wait(tree.sflag[me],tree.children); //many-to-one
    if tree.parent != -1 then
[S6]    cl_add(tree.sflag[tree.parent],1);       //many-to-one
    end
end
```



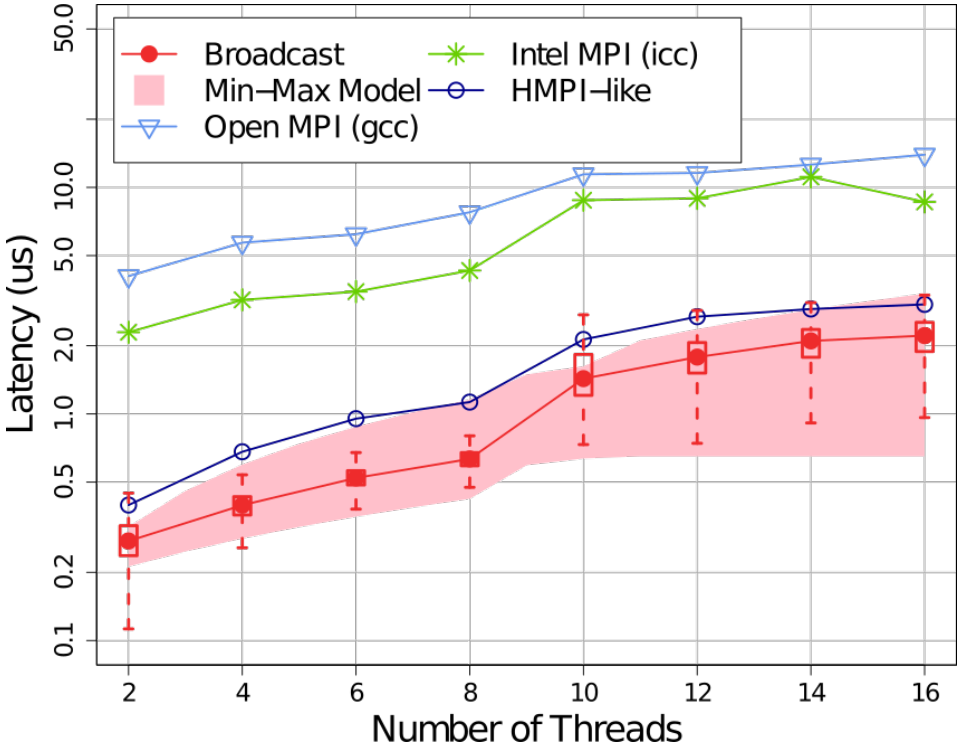
3. ALGORITHM DESIGN

Example: Broadcast

```
     Function OneLineBroadcast(int me, cl_t * mydata, tree_t
     tree)
         if tree.parent != -1 then
[S1]         cl_wait(tree.pflag[tree.parent],1);      //one-to-many
[S2]         cl_copy(tree.data[tree.parent],mydata,1);
         if tree.children > 0 then
[S3]         cl_copy(mydata,tree.data[me],1);
[S4]         cl_write(tree.pflag[me],1);              //one-to-many
[S5]         cl_wait(tree.sflag[me],tree.children); //many-to-one
         if tree.parent != -1 then
[S6]         cl_add(tree.sflag[tree.parent],1);       //many-to-one
         end
     end
```



PERFORMANCE RESULTS



- Speedup of 14x vs. MPI
- Speedup of 1.8x vs. HMPI

CONCLUSIONS AND DISCUSSION

- Cache-coherency helps programmability
- BUT it complicates performance-centric programming
- The CLa methodology simplifies the analysis of algorithms under heavy thread interaction conditions that affect performance:
  - Contention and congestion
  - Polling
  - Cache-line stealing
- We compared our algorithms (communication and synchronization) with MPI, OpenMP and HMPI obtaining high speedups.

CACHE LINE AWARE OPTIMIZATIONS FOR CCNUMA SYSTEMS

Sabela Ramos (sramos@udc.es)
GAC, Universidade da Coruña (Spain)
Torsten Hoefler (htor@inf.ethz.ch)
SPCL, ETH Zurich (Switzerland)

24th ACM International Symposium on High-Performance Parallel and Distributed Computing
HPDC'15, Portland, 2015