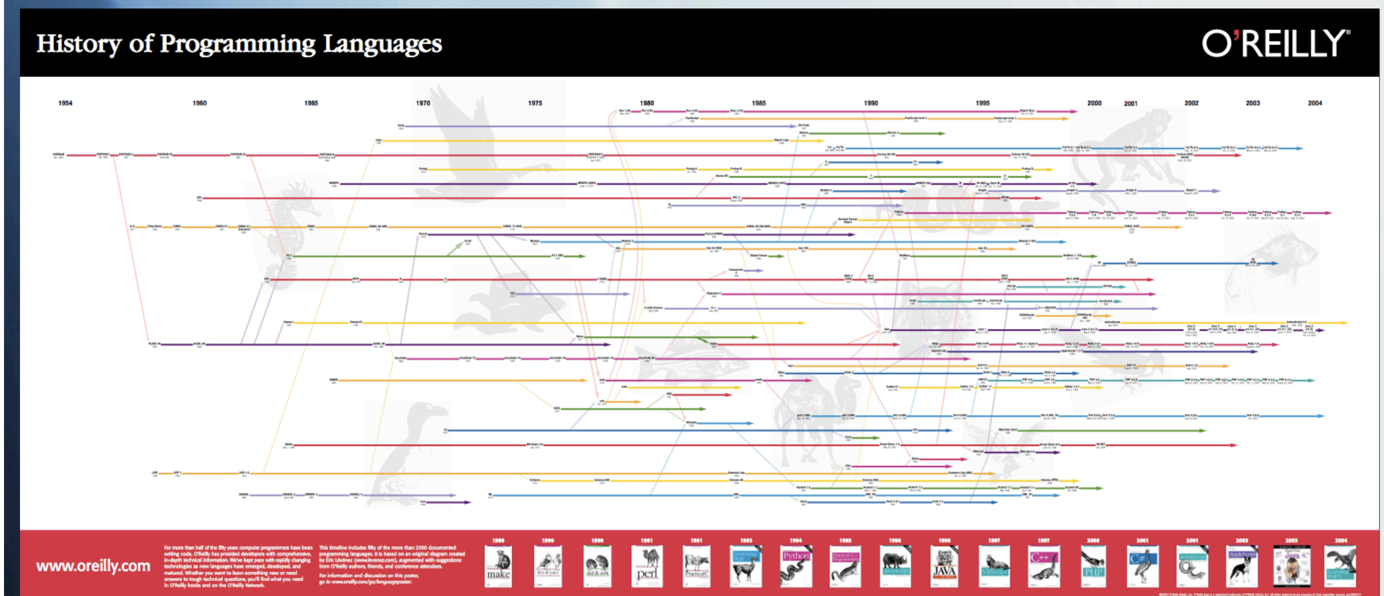
RICE George R. Brown School of Engineering
Computer Science

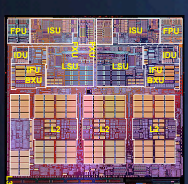
Multicore Programming Models and their Implementation Challenges

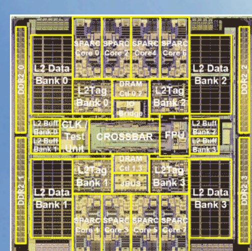


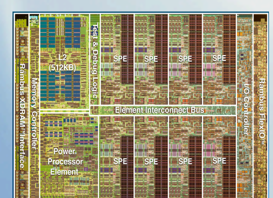
Vivek Sarkar

Rice University

vsarkar@rice.edu

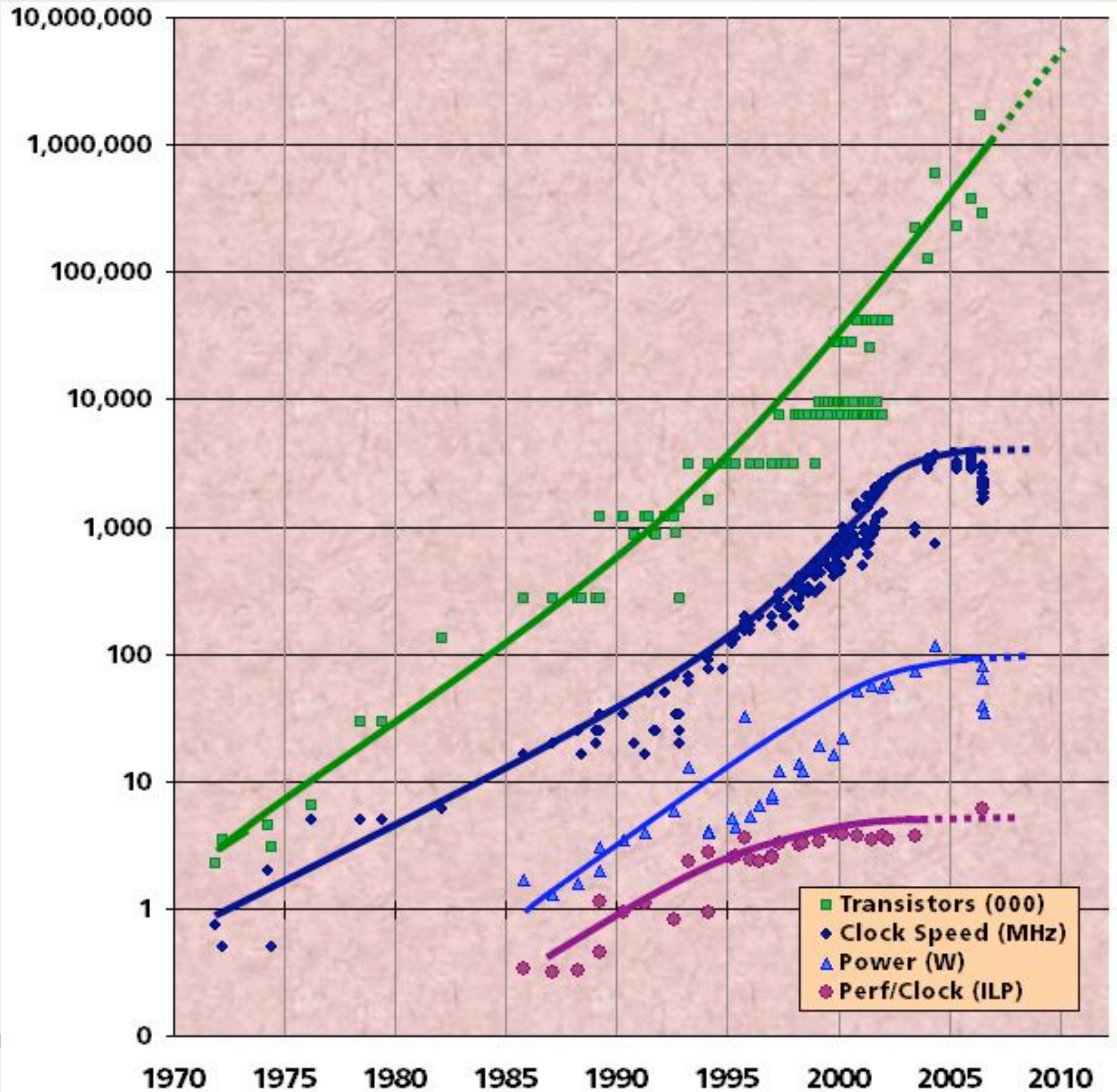






The Multicore Revolution: why Concurrency has become critical for Mainstream Computing

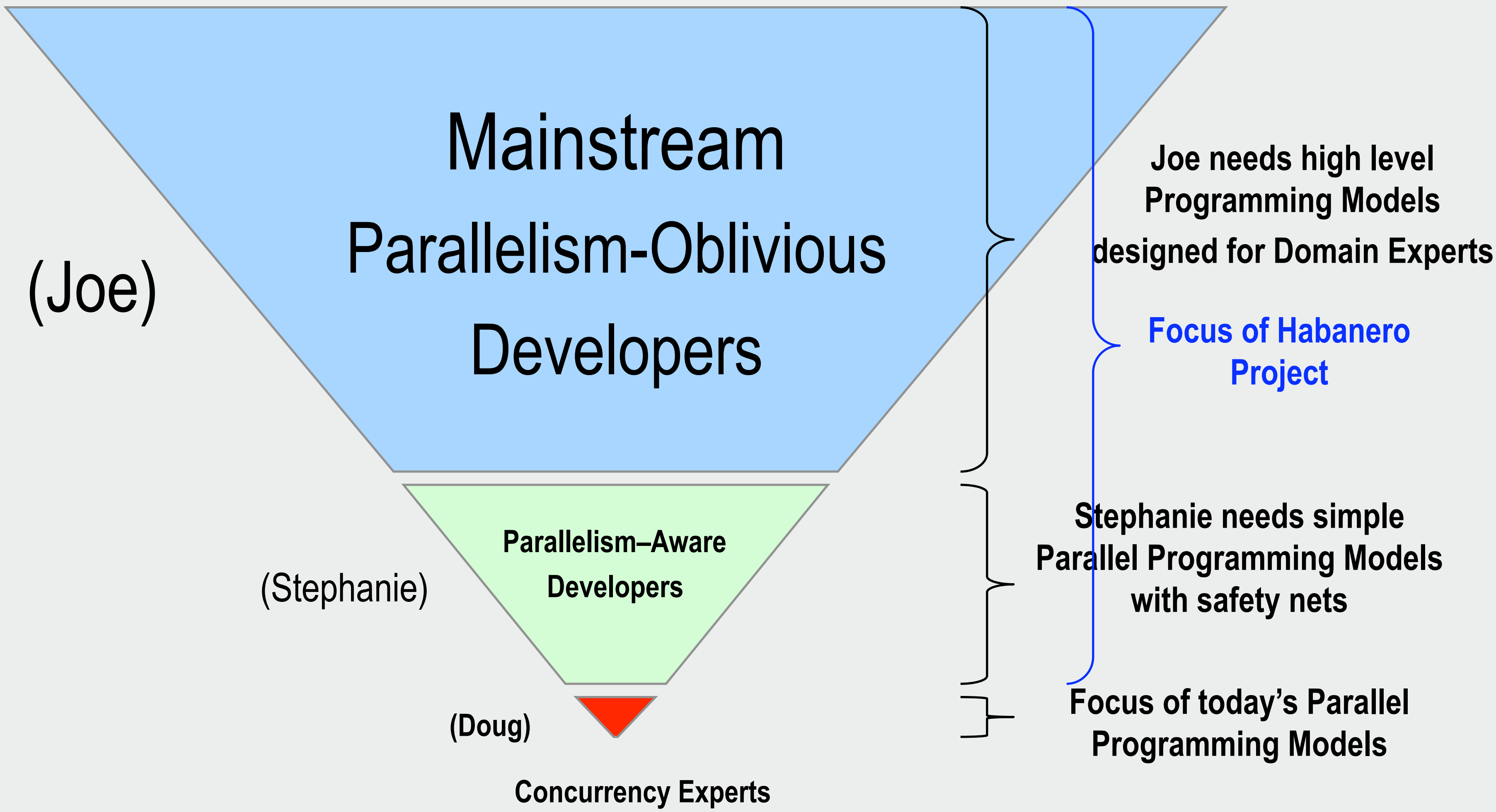
- Chip density is continuing to increase ~2x every 2 years
    - Clock speed is not
    - Number of processor cores is doubling instead
- There is little or no hidden parallelism (ILP) to be found
- ***Parallelism must be exposed to and managed by software***



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)

RICE

Parallel Software Challenge & Inverted Pyramid of Parallel Programming Skills



Habanero Project Overview (habanero.rice.edu)

Parallel Applications

(Seismic analysis, Medical imaging, Finite Element Methods, …)

Challenge: Develop new programming technologies and pedagogical foundations for portable parallelism on future multicore hardware

1) Habanero Programming Languages

2) Habanero Static Compiler & Parallel Intermediate Representation

3) Habanero Runtime & Dynamic Compiler

Two-level programming model

Implicitly Parallel Coordination Language for Joe, CnC (Intel Concurrent Collections)

+

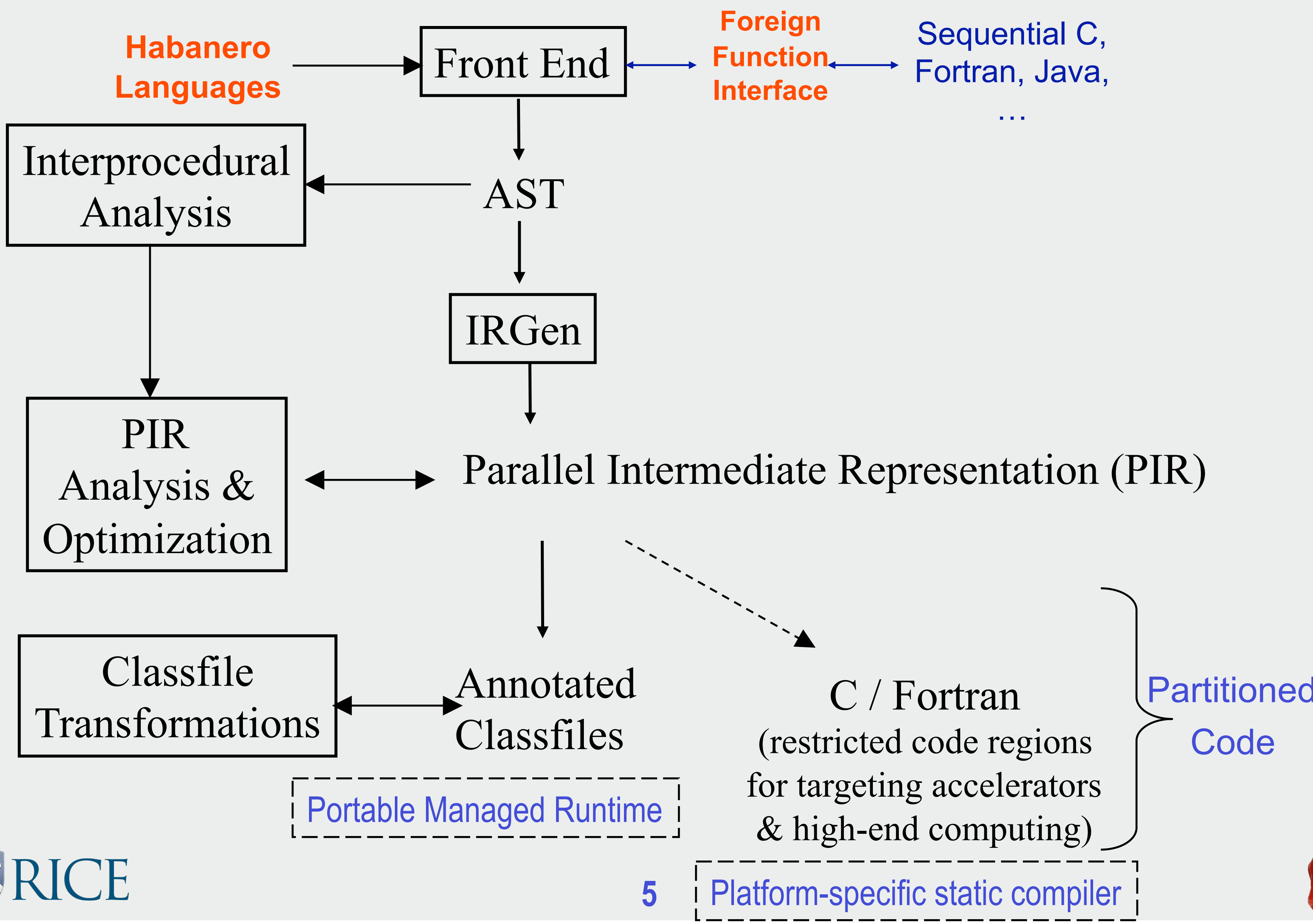
Explicitly Parallel Programming Languages for Stephanie, Habanero-Java (from X10 v1.5) and Habanero-C

Foreign Code (Matlab, Java, C, C++, Fortran, CUDA) ⟷ Foreign Function Interface ⟷ 3) Habanero Runtime & Dynamic Compiler

Multicore Platforms

(Cell, Clearspeed, Cyclops, GeForce, Niagara, Opteron, Power, Xeon, …)

Habanero Static Parallelizing & Optimizing Compiler



Outline

- Intel Concurrent Collections Coordination Language and Implementation Challenges
- X10 + Habanero Execution Model and Implementation Challenges

Acknowledgments

Intel ™ Concurrent Collections Project

http://whatif.intel.com

- **Developer Products Division (DPD)**
  - Aparna Chandramowlishwaran, Nikolay Kurtov, Shin Lee, Bob Monteleone, David Moore, John Parks, Stephen Rose, Frank Schlimbach, Leo Treggiari, Judy Ward, Brian Kazin
- **Software Pathfinding and Innovation (SPI)**
  - Kath Knobe, Geoff Lowney
- **Digital Enterprise Group (DEG)**
  - Steve Lang
- **Ex-colleagues**
  - Alex Nelson (HP, Intel), Carl Offner (HP), Kishore Ramachandran (Georgia Tech), Hasnain Mandviwala (Georgia Tech)

RICE

The problem for Joe

- Most serial languages over-constrain orderings
  - Require arbitrary serialization
  - Allow for overwriting of data
  - The decision of *if* and *when* to execute are bound together
  - This makes parallel programming hard

- Most parallel programming languages are embedded within serial languages
  - Inherit problems of serial languages
  - Too specific w.r.t. type of parallelism in the application and wrt the type target architecture

- Concurrent Collections Approach: introduce a coordination language that
  - Systematically eliminate over-constraints
  - Explicitly specify required constraints

Example of a Coordination Language for Domain Experts: Intel Concurrent Collections (CnC), f.k.a. TStreams

Domain Expert: (person)

Only domain knowledge

No tuning knowledge

"Joe"

The application problem

- Decomposition into **Steps**
- Single-Assignment **Collections** as interfaces between steps
- Inter-step **data flow** = put/get operations on Item Collections
- Inter-step **control flow** = put operations on Tagged Collections to create (prescribe) new steps

Tuning Expert: (person, runtime, compiler)

No domain knowledge

Only tuning knowledge

"Stephanie"

Concurrent Collections Program

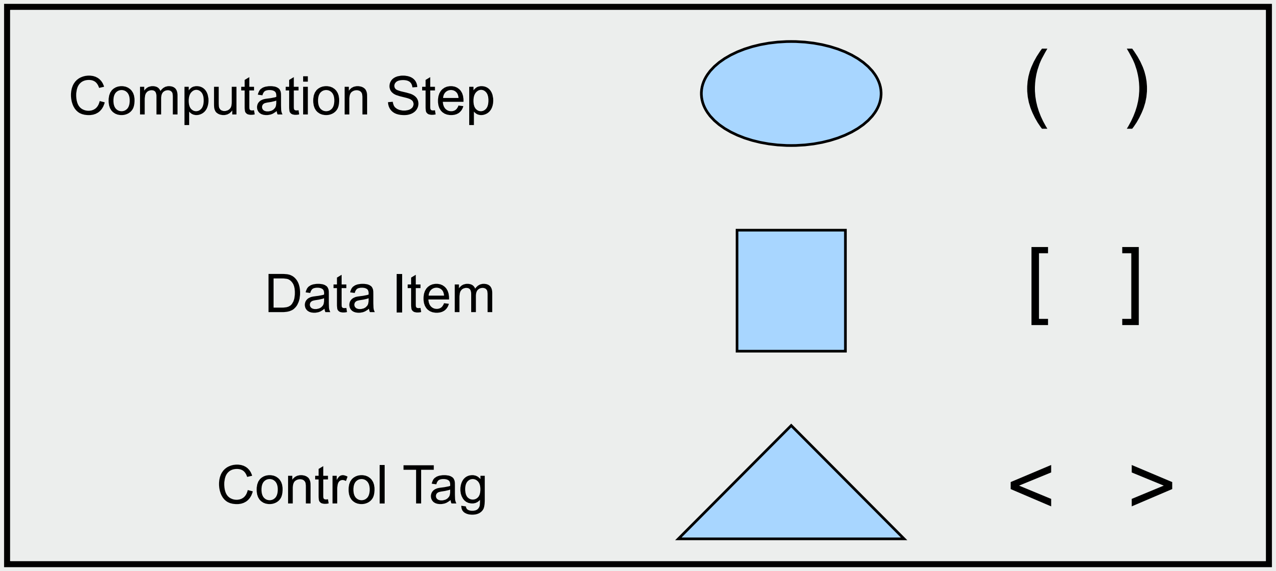
- Exploit parallelism across and within steps
- Locality
- Overhead
- Load balancing
- Distribution among processors
- Scheduling within a processor
- Platform-aware optimizations

Explicit parallel program (Intel TBB or Habanero/X10)

Source: Kathleen Knobe

http://softwarecommunity.intel.com/articles/eng/3862.htm

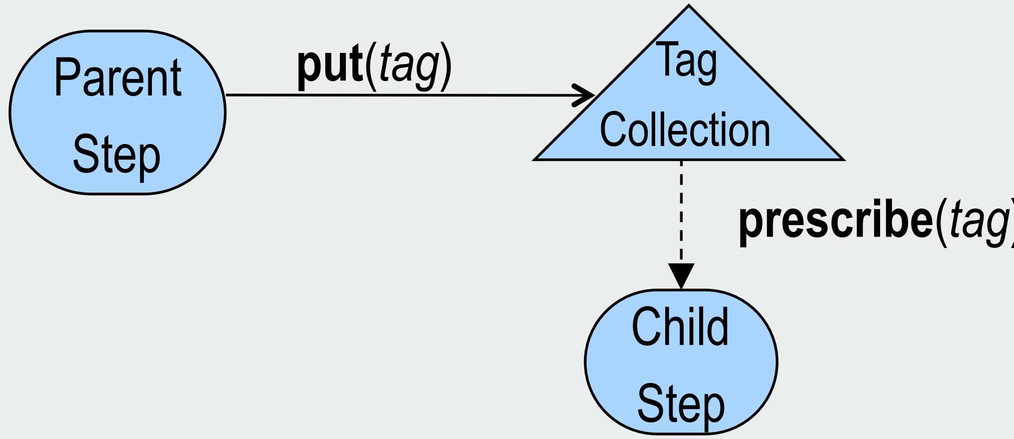
Notation



Producer-Consumer Relationship in CnC

- Tag can be any hashable value (numeric, string, …) that supports equality comparison
  - We will restrict our attention to integer tuple tags in this talk
- Item can be any immutable data structure
  - Two **get**'s with the same tag must return identical items
- Single assignment rule
  - At most one **put** permitted with a given tag value; an exception is thrown if a second **put** is attempted with the same tag value
- Blocking **get**'s
  - A **get** operation blocks if no item is present with the given tag, and is unblocked when a matching **put** is performed

Creating new steps in CnC



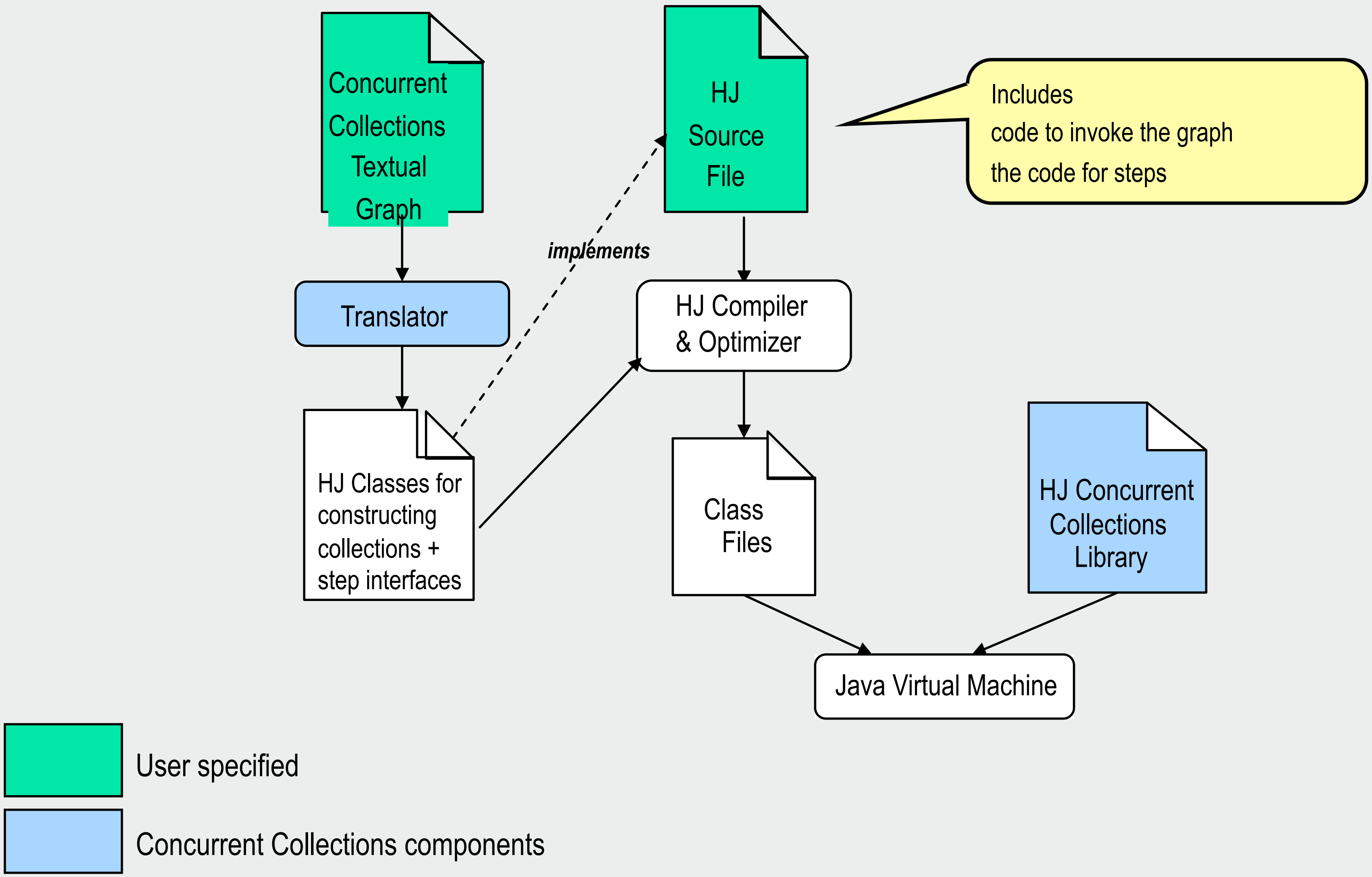
- Tag collection
  - Role of tag collection is to prescribe (create) new steps
- Tag can be any hashable value (numeric, string, …) that supports equality comparison
  - We will restrict our attention to integer tuple tags in this talk
- Single assignment rule
  - At most one **put** permitted with a given tag value; an exception is thrown if a second **put** is attempted with the same tag value
- Step prescription
  - Runtime system guarantees that **prescribe** operation is performed eventually on child step for each tag in tag collection

Domain Expert's view of Concurrent Collections

- No thinking about parallelism
  - Only domain knowledge
- No overwriting
  - Single assignment collections
    - Can be extended with fetch-and-op & reduce operations
- No arbitrary serialization
  - only constraints on ordering via tagged puts and gets
- Result is:
  - Deterministic
  - Race-free
  - Fault-tolerant

RICE

CnC Compile and Execute Model for Habanero-Java

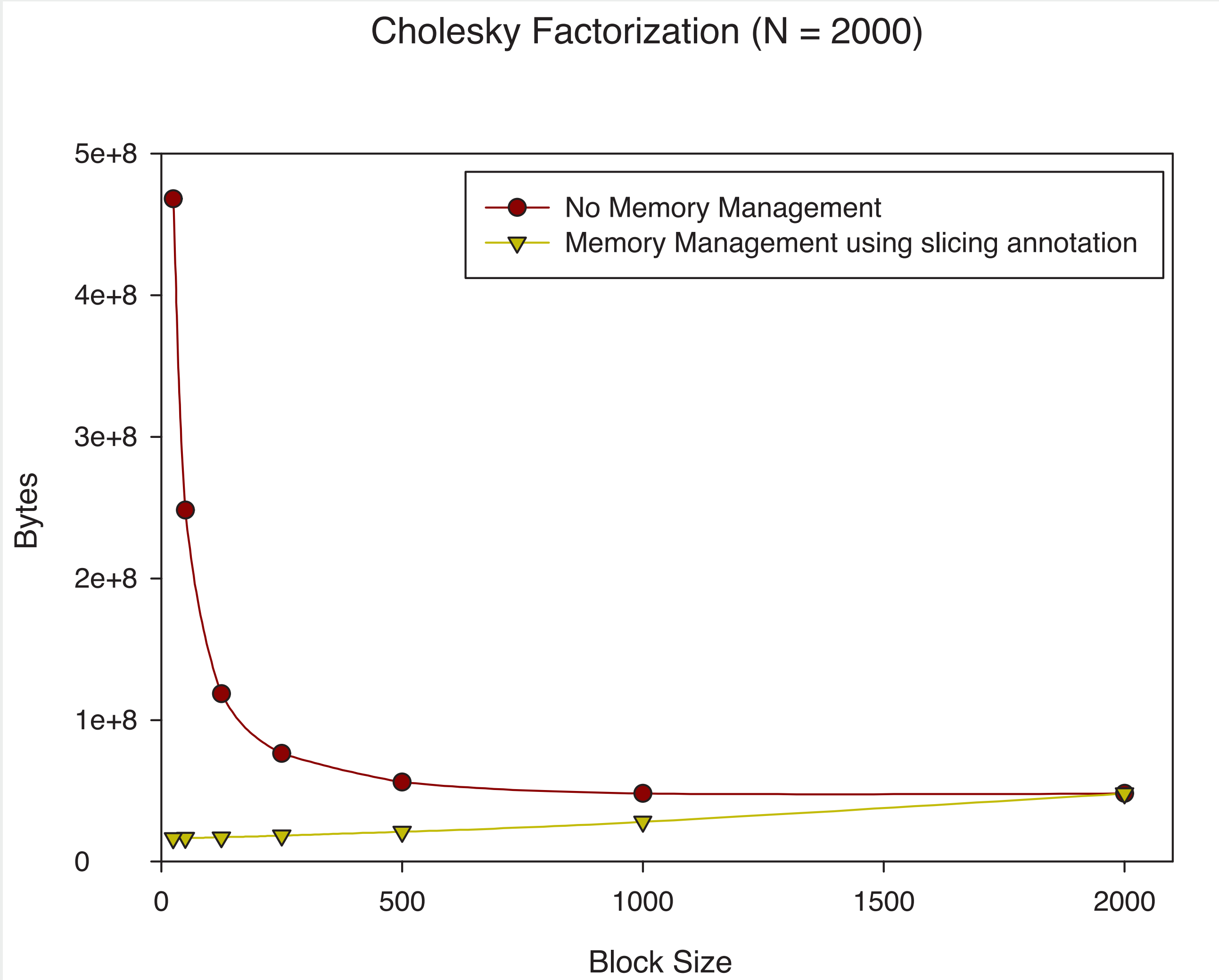


RICE

CnC Implementation Challenges

- Scalable runtime implementation for multicore parallelism
  - "Multicore Implementations of the Concurrent Collections Programming Model", Zoran Budimlic, Aparna Chandramowlishwaran, Kathleen Knobe, Geoff Lowney, Vivek Sarkar, Leo Treggiari, CPC 2009 workshop
- Garbage collection of dead items
  - "Declarative Aspects of Memory Management in the Concurrent Collections Parallel Programming Model", Zoran Budimlic, Aparna Chandramowlishwaran, Kathleen Knobe, Geoff Lowney, Vivek Sarkar, Leo Treggiari, DAMP 2009 workshop
- Extending CnC with hierarchical (modular) structure (in progress)
- Copy avoidance and update-in-place optimizations (in progress)
- Scheduling optimizations for parallelism and locality
- CnC extensions for domain-specific languages and runtimes
- Upcoming Tutorial at PLDI 2009
  - "The Concurrent Collections Parallel Programming Model --- Foundations and Implementation Challenges", K.Knobe, V.Sarkar

Example: Memory Requirements for 2000x2000 Cholesky Factorization w/ and w/o Garbage Collection of Dead Items



"Declarative Aspects of Memory Management in the Concurrent Collections Parallel Programming Model", Zoran Budimlic, Aparna Chandramowlishwaran, Kathleen Knobe, Geoff Lowney, Vivek Sarkar, Leo Treggiari, DAMP 2009 workshop

Outline

- Intel Concurrent Collections Coordination Language and Implementation Challenges
- X10 + Habanero Execution Model and Implementation Challenges

The problem for Stephanie

- Stephanie needs to map & tune Joe's CnC model (graph + steps) onto parallel systems
  - Exploit parallelism across and within steps
  - Optimize Locality, Data Movement, Load balancing, Scheduling, ..

- Most parallel programming languages are tied to specific parallel architecture models

- X10/Habanero Approach: support a portable abstract execution model that supports high performance with high productivity
  1. Lightweight dynamic task creation & termination
  2. Locality control --- task and data distributions
  3. Mutual exclusion and isolation
  4. Collective and point-to-point synchronization

X10 Background

- Developed at IBM since 2004 as part of DARPA HPCS program
  - DARPA's goal: increase development productivity by 10x from 2002 to 2010
- Productivity approach:
  - High Level Language designed for portability and safety
  - Unified abstractions of asynchrony and concurrency for Multi-core & Cluster Parallelism
    - Subsumes threads, shared memory, message-passing, active messages
- Performance transparency – don't lock out the performance expert!
    - Expert programmer should have controls to tune deployments of portable code
- X10 programming model can be used to extend any sequential language
  - X10 v1.5 language is based on a sequential subset of Java
  - Reference: "X10: An Object-Oriented Approach to Non-Uniform Cluster Computing", P.Charles et al, OOPSLA 2005 Onward! Track.
  - Open source SMP reference implementation for X10 v1.5: x10.sf.net
  - X10 v1.7 has adopted Scala-like syntax and richer type system (http://x10-lang.org/)
- Habanero approach: address implementation challenges for X10 v1.5 on multicore, with programming model extensions as needed

RICE

X10 Acknowledgments (as of mid-2008)

- X10 Core Team (IBM)
  - Ganesh Bikshandi, Sreedhar Kodali, Nathaniel Nystrom, Igor Peshansky, Vijay Saraswat, Pradeep Varma, Sayantan Sur, Olivier Tardieu, Krishna Venkat, Tong Wen, Jose Castanos, Ankur Narang, Komondoor Raghavan
- X10 Tools
  - Philippe Charles, Robert Fuhrer
- Emeritus
  - Kemal Ebcioglu, Christian Grothoff, Vincent Cave, Lex Spoon, Christoph von Praun, Rajkishore Barik, Chris Donawa, Allan Kielstra
- Research colleagues
  - Vivek Sarkar, Rice U
  - Satish Chandra,Guojing Cong
  - Ras Bodik, Guang Gao, Radha Jagadeesan, Jens Palsberg, Rodric Rabbah, Jan Vitek
  - Vinod Tipparaju, Jarek Nieplocha (PNNL)
  - Kathy Yelick, Dan Bonachea (Berkeley)
  - Several others at IBM

Publications

1. “Type Inference for Locality Analysis of Distributed Data Structures”, PPoPP 2008.
2. “Deadlock-free scheduling of X10 Computations with bounded resources”, SPAA 2007
3. “A Theory of Memory Models”, PPoPP 2007.
4. “May-Happen-in-Parallel Analysis of X10 Programs”, PPoPP 2007.
5. “An annotation and compiler plug-in system for X10”, IBM Technical Report, Feb 2007.
6. “Experiences with an SMP Implementation for X10 based on the Java Concurrency Utilities” Workshop on Programming Models for Ubiquitous Parallelism (PMUP), September 2006.
7. "An Experiment in Measuring the Productivity of Three Parallel Programming Languages”, P-PHEC workshop, February 2006.
8. "X10: An Object-Oriented Approach to Non-Uniform Cluster Computing", OOPSLA conference, October 2005.
9. "Concurrent Clustered Programming", CONCUR conference, August 2005.
10. "X10: an Experimental Language for High Productivity Programming of Scalable Systems", P-PHEC workshop, February 2005.

Tutorials

- TiC 2006, PACT 2006, OOPSLA 2006, PPoPP 2007, SC 2007
- Graduate course on X10 at U Pisa (07/07)
- Graduate course at Waseda U (Tokyo, 04/08)

RICE

X10 + Habanero Execution Model: Portable Parallelism in Four Dimensions

1. Lightweight dynamic task creation & termination
   - *async*, *finish* (from X10)
2. Locality control --- task and data distributions
   - *places* (from X10)
3. Mutual exclusion
   - *isolated* (from Habanero --- extension of X10 atomic)
4. Collective and point-to-point synchronization
   - *phasers* (from Habanero --- extension of X10 *clocks*)

RICE

Async and Finish

Stmt ::= **async** *Stmt*

async S

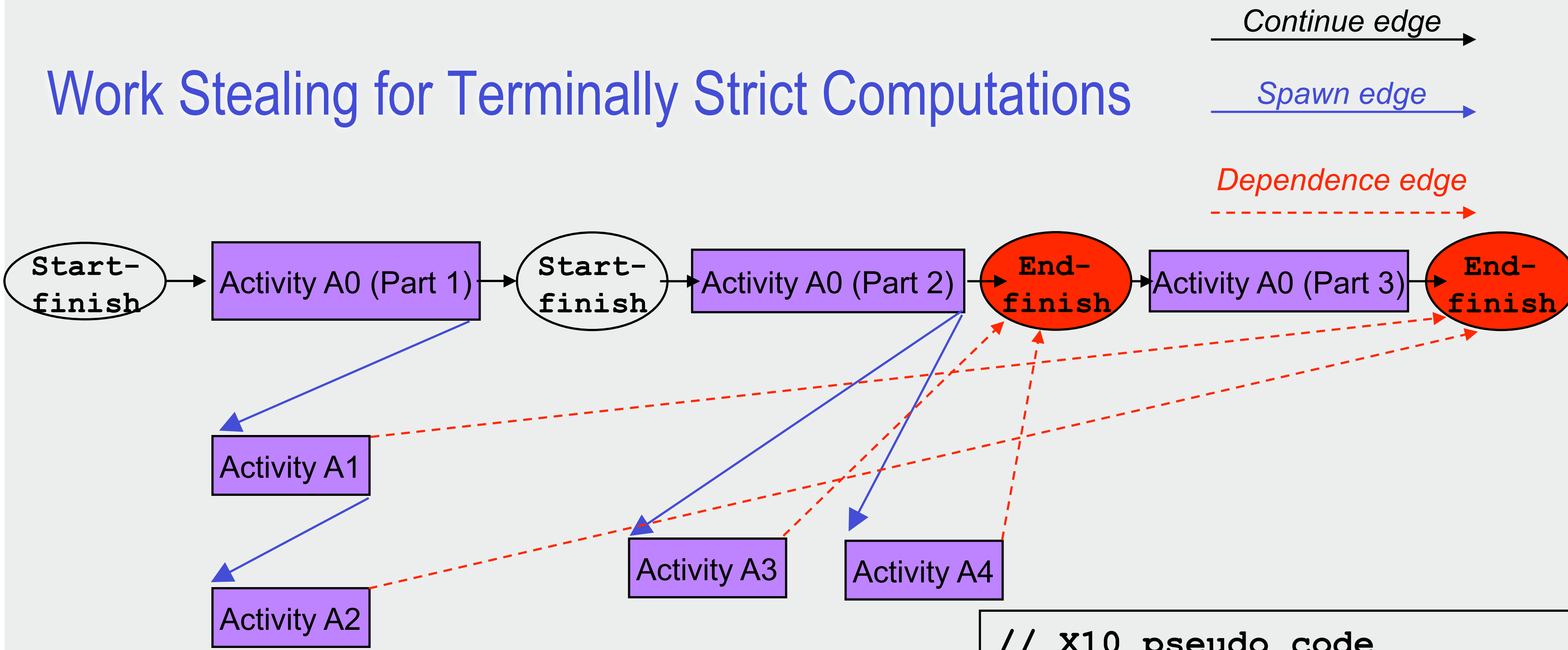
- Creates a new child activity that executes statement S
- Returns immediately
- S may reference final variables in enclosing blocks
- Activities cannot be named
- Activity cannot be aborted or cancelled

Stmt ::= **finish** *Stmt*

finish S

- Execute S, but wait until all (transitively) spawned asyncs have terminated.
- Rooted exception model
  - Trap all exceptions thrown by spawned activities.
  - Throw an (aggregate) exception if any spawned async terminates abruptly.
- implicit finish between start and end of main program

Work Stealing for Terminally Strict Computations



"Deadlock-Free Scheduling of X10 Computations with Bounded Resources", S.Agarwal et al, SPAA 2007.

Theorem 2.6: A work-stealing execution of a *terminally strict* multithreaded computation with finish & async constructs on P processor uses at most S_1*P space in its dequeue's, where S_1 is the maximum stack depth in a sequential execution of the program.

```
// X10 pseudo code
main(){ // implicit finish
  Activity A0 (Part 1);
  async {A1; async A2;}
  try {
    finish {
      Activity A0 (Part 2);
      async A3;
      async A4;
    }
  catch (…) { … }
  Activity A0 (Part 3);
}
```

RICE

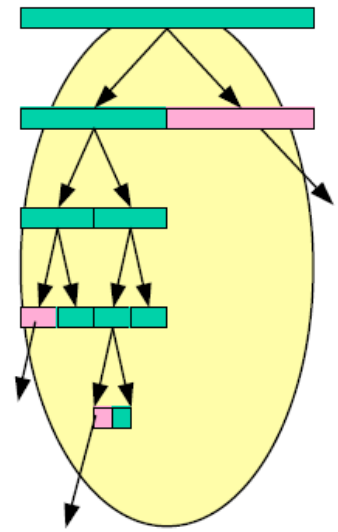
Loop Parallelism with Finish and Async: One-Dimensional Iterative Averaging Example

```
int iters = 0; delta = epsilon+1;
while ( delta > epsilon ) {
  finish {
    for ( jj = 1 ; jj <= n ; jj++ ) {
      final int j = jj;
      async { // for-async can be replaced by foreach
        newA[j] = (oldA[j-1]+oldA[j+1])/2.0f ;
        diff[j] = Math.abs(newA[j]-oldA[j]);
      } // async
    } // for
  } // finish (join)
  delta = diff.sum(); iters++;
  temp = newA; newA = oldA; oldA = temp;
}
System.out.println("Iterations: " + iters);
```

Recursive Parallelism with Finish and Async

```
void refine(final int n, final int l, final int nmax) {
        left = new Tree(this,2.0*l);
        right =new Tree(this, 2.0*l+1);
        final nullable Tree ll = left, rr=right;
        if (n < (nmax-1)) {
                async {ll.refine(n+1,2*l,nmax);}
                async { rr.refine(n+1,2*l+1,nmax);}
        }
        if (n < nmax) data = null;

. . .
// Main program
. . .
finish refine(root, 1, nmax);
```



From "What's in it for the Users? Looking Toward the HPCS Languages and Beyond", D. Bernholdt, W.R. Elwasif, Robert J. Harrison, PGAS 2006

Habanero Framework for Work-Stealing Schedulers

"Work-First and Help-First Scheduling Policies for Terminally Strict Parallel Programs", Yi Guo, Rajkishore Barik, Raghavan Raman, Vivek Sarkar (to appear in IPDPS 2009)

Terminally Strict Parallel Program

Front-end

Transformed Program with Runtime calls for Async/Finish

Work-Stealing Code Gen for Work-First policy

Work-Stealing Code Gen for Help-First policy

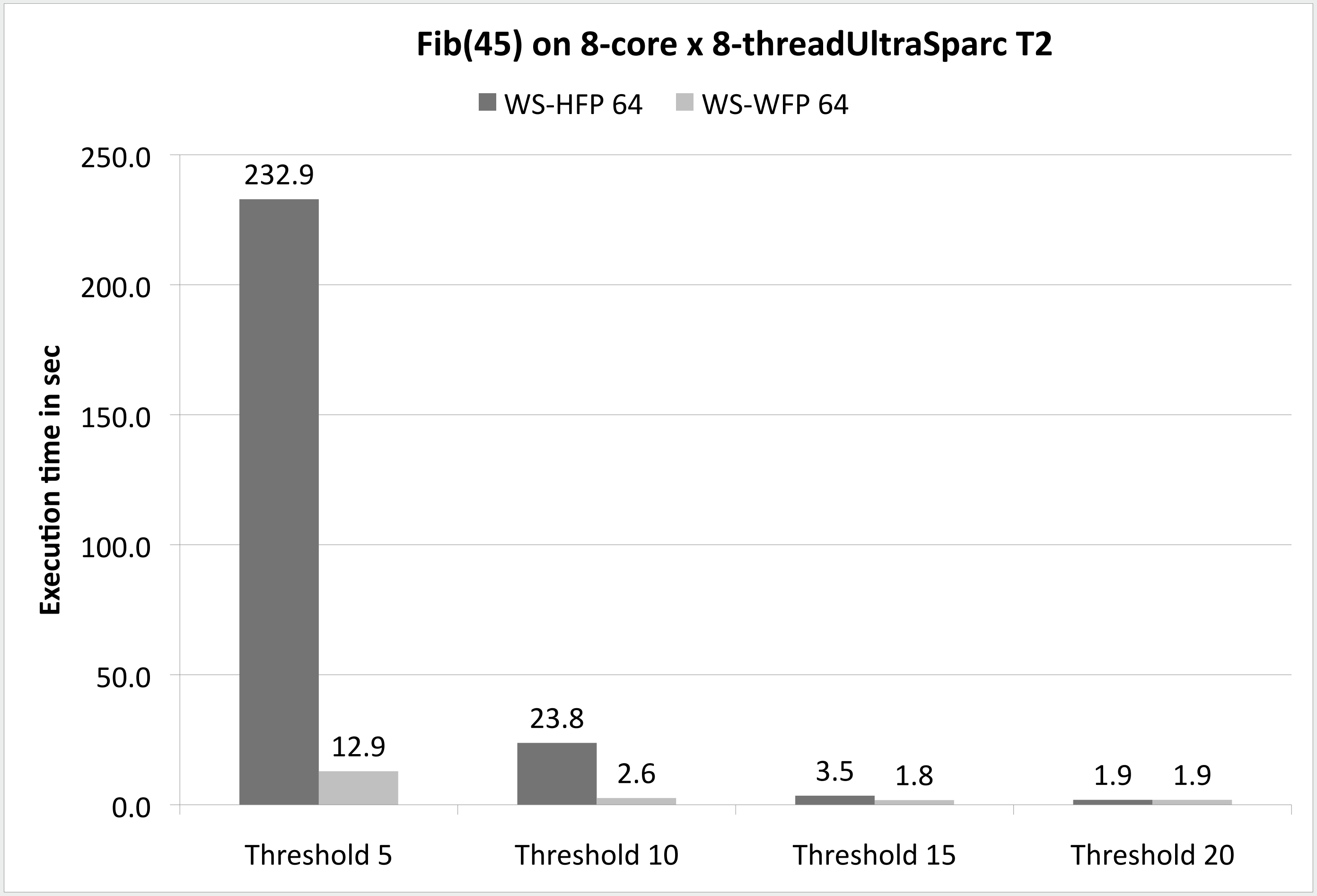
Work-Sharing Runtime with Single Queue (j.u.c. ThreadPoolExecutor)

Habanero Work-Stealing Runtime with Work-First policy

Habanero Work-Stealing Runtime with Help-First policy

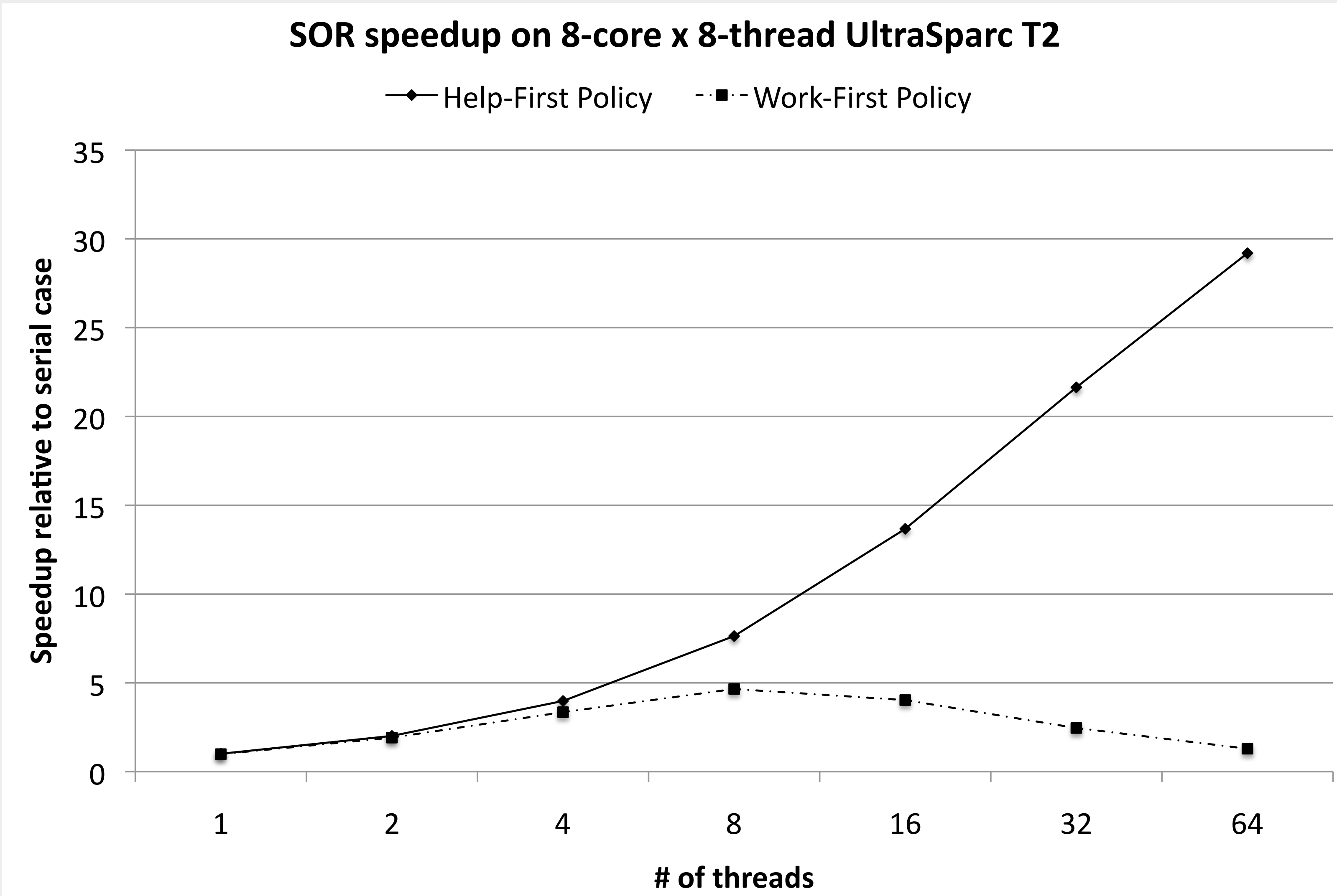
- <u>Work-First Policy:</u> worker executes child task and leaves continuation to be stolen
- <u>Help-First Policy:</u> worker executes continuation and leaves async to be stolen
- Additional X10 requirements
  - Escaping Asyncs
  - Sequential and parallel invocations of same code

Work-First Policy is better than Help-First Policy for Recursive Divide-and-Conquer Parallel Algorithms …

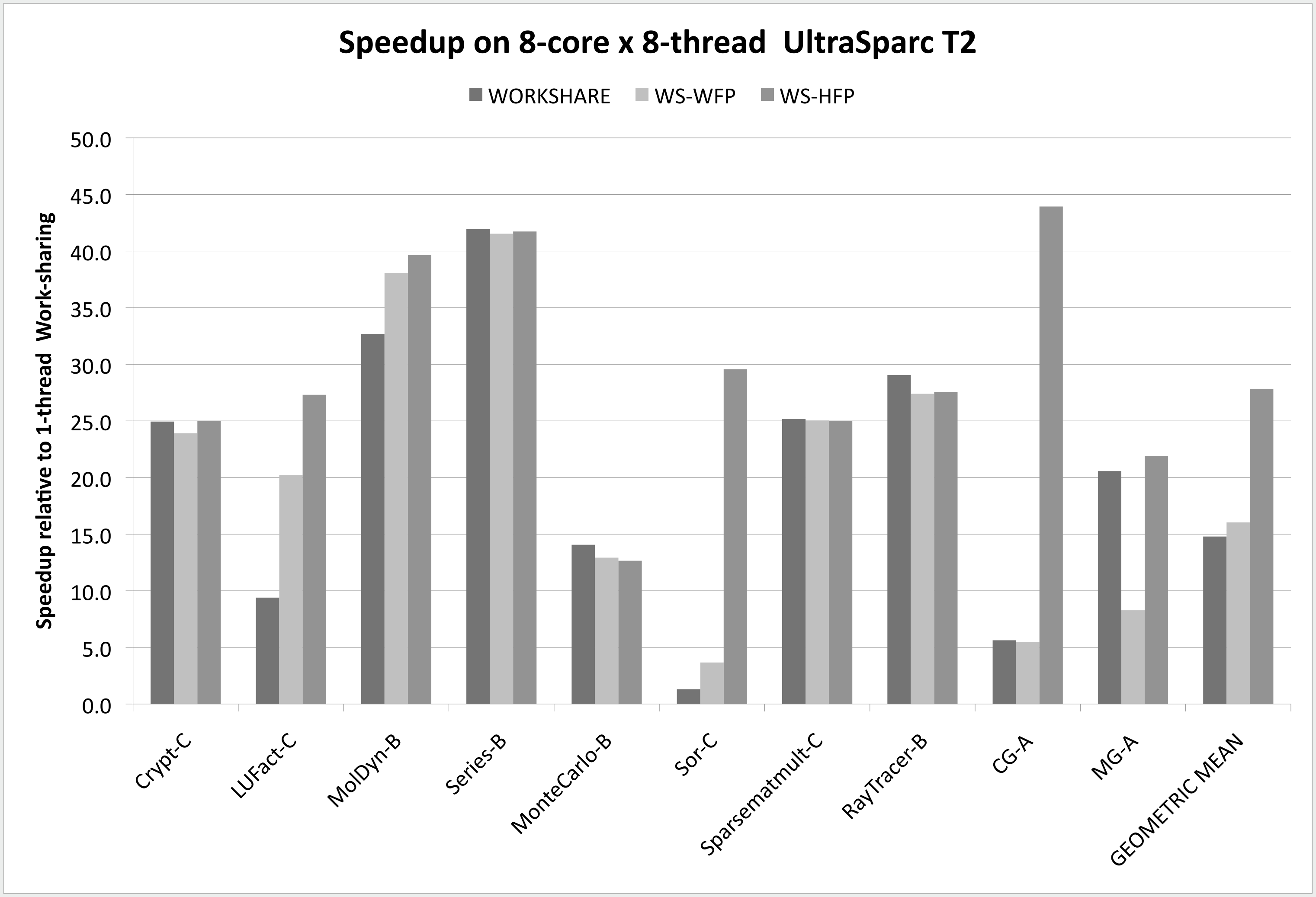


… but the gap between the two decreases as the task granularity increases

Work-First Policy is not always better than Help-First



Additional Results



RICE

Implementation Challenges for Finish & Async

- Extend space-efficient scalable work-stealing schedulers to support terminally strict finish-async programs
- Extend work-stealing algorithms to be locality-conscious (place-aware)
- Extend work-stealing algorithms to support directed point-to-point and barrier synchronizations (phasers)
- Reduce footprint impact of inflated blocked activities
  - Delayed asyncs

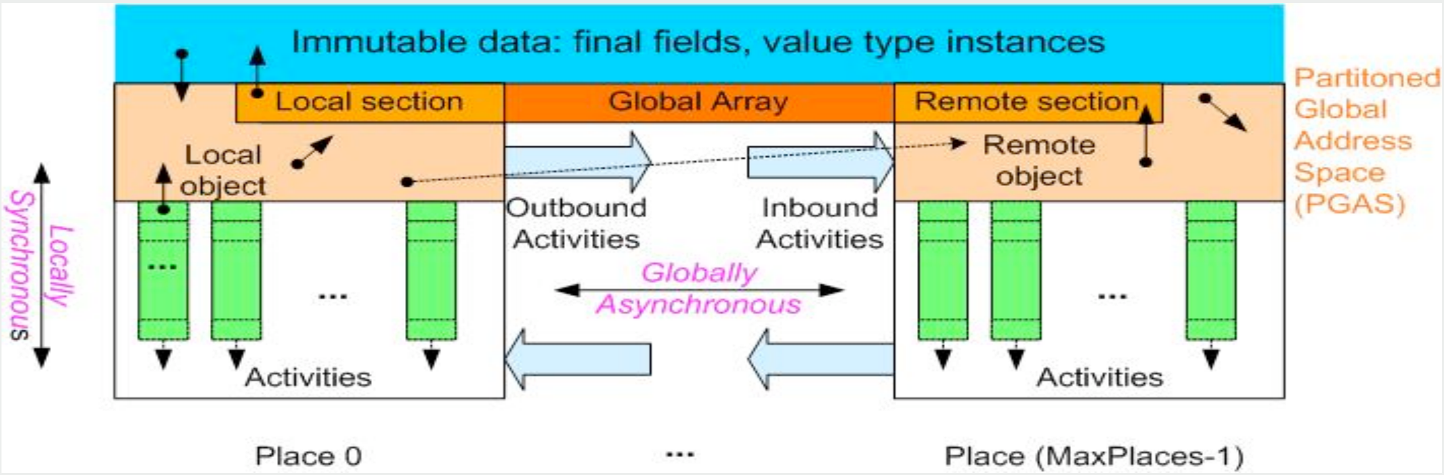
RICE

X10 + Habanero Execution Model: Portable Parallelism in Four Dimensions

1. Lightweight dynamic task creation & termination
   - *async*, *finish* (from X10)
2. Locality control --- task and data distributions
   - *places* (from X10)
3. Mutual exclusion
   - *isolated* (from Habanero --- extension of X10 atomic)
4. Collective and point-to-point synchronization
   - *phasers* (from Habanero --- extension of X10 *clocks*)

RICE

Task and Data Distributions with Places



Storage classes:

- Activity-local
- Place-local
- Partitioned global
- Immutable

- Dynamic parallelism with a *Partitioned Global Address Space*
- *Places* encapsulate binding of activities and globally addressable mutable data
  - Number of places currently fixed at launch time
- Each *datum* has a designated place specified by its distribution
- Each *async* has a designated place specified by its distribution --- subsumes threads, structured parallelism, messaging, DMA transfers, etc.
  - Keyword *here* evaluates to place where current activity is executing
- *Immutable* data (value types, value arrays) is place-independent and offers opportunity for functional-style parallelism
- Type system for places --- “Type Inference for Locality Analysis of Distributed Data Structures”, S.Chandra et al, PPoPP 2008.

Extension of Async with Places

Examples

```
1) finish { // Inter-place parallelism
       final int x = … , y = … ;
       print here; // Print current activity's place
       async (a) { // Execute at a's place
         a.foo(x);
         print here; // Print a's place
       }
       async (b[i]) b[i].bar(y); // Execute at b[i]'s place
   }

2) // Implicit and explicit versions of remote fetch-and-op
   a) a.x = foo(a.x, b.y) ;
   b) async (b) {
         final double v = b.y; // Can be any value type
         async (a) isolated (a) a.x = foo(a.x, v);
       }
```

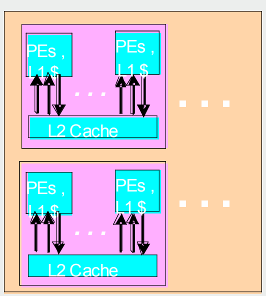
Portable Parallel Programming via X10 Places

X10 language defines mapping from X10 objects & activities to X10 places

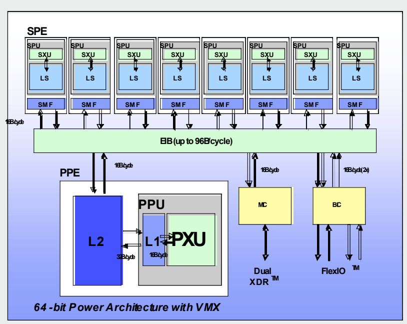
X10 deployment defines mapping from virtual X10 places to physical processing elements

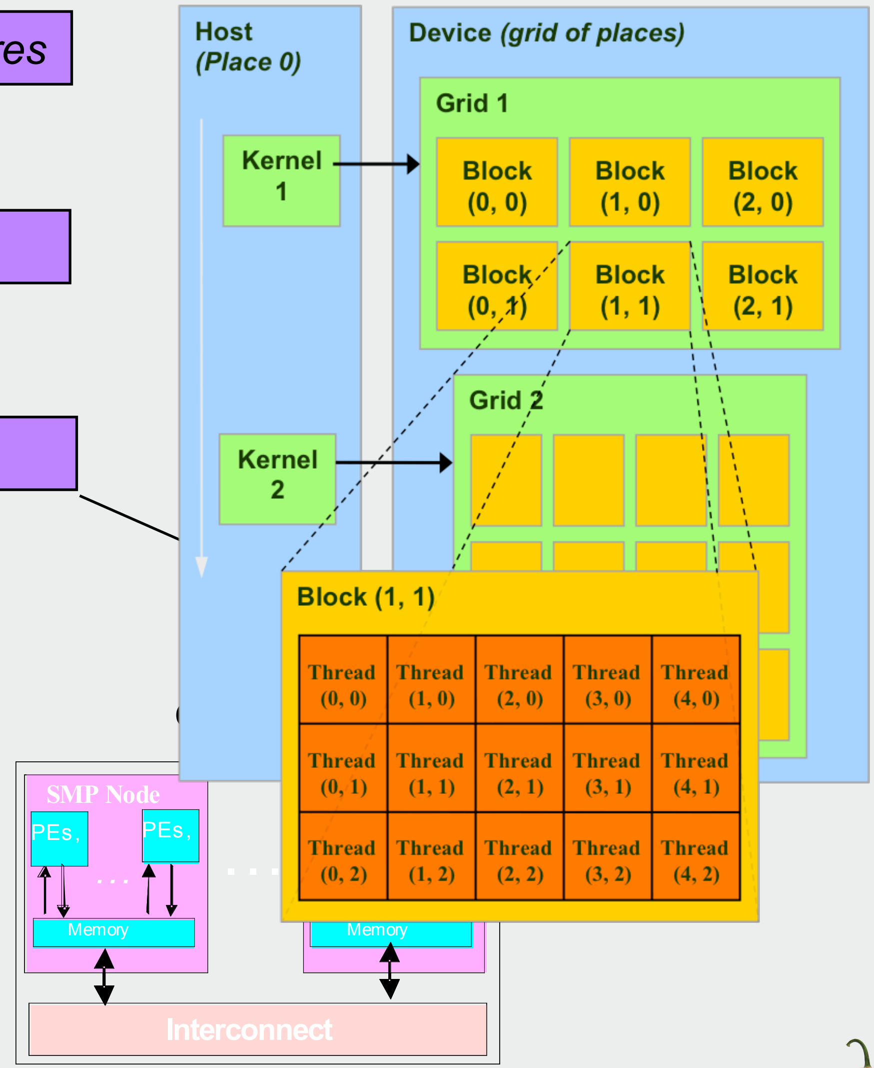
X10 Data Structures → *X10 Places* → *Physical PEs*

Homogeneous Multi-core



Heterogeneous Accelerators

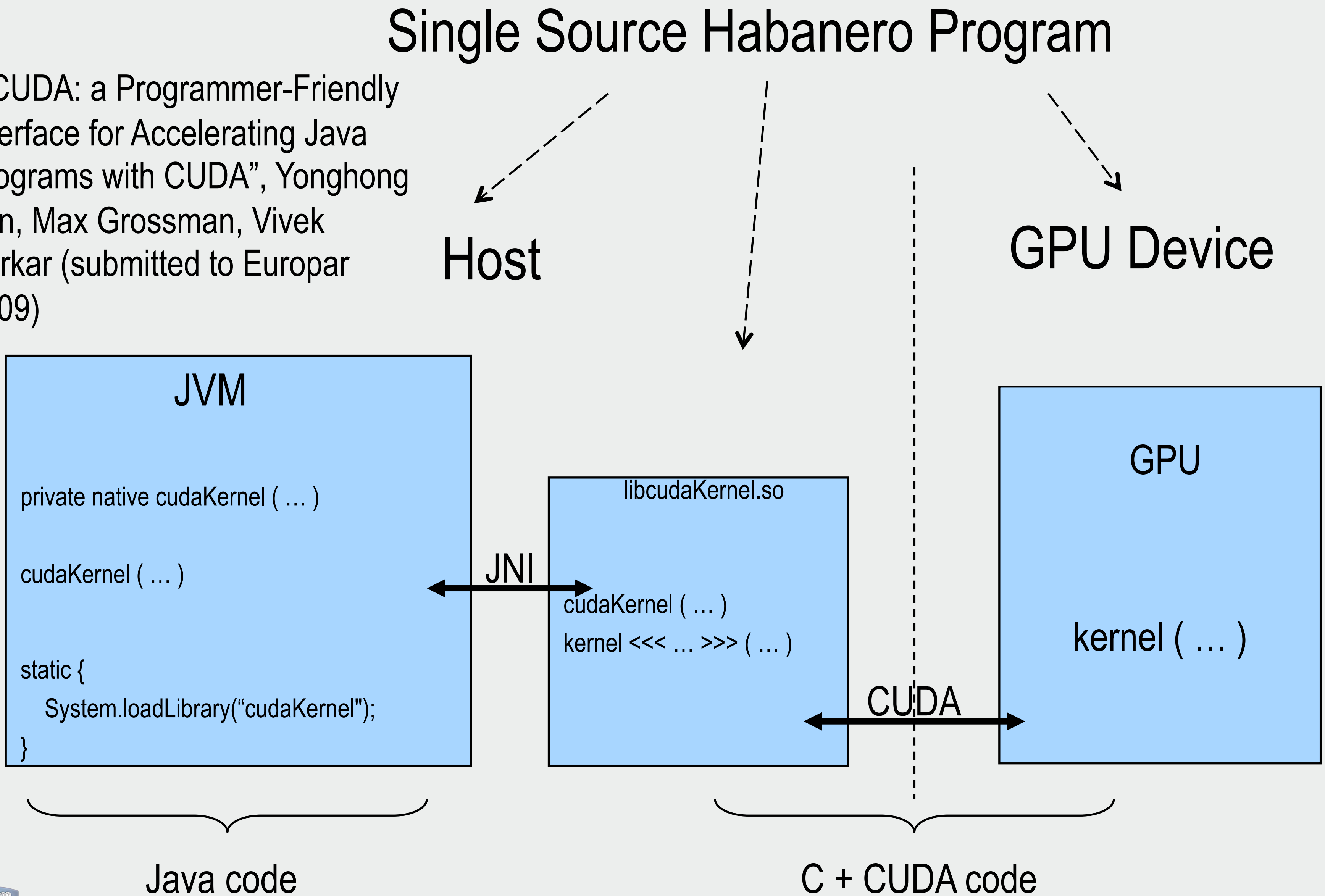




RICE

Hybrid Java+CUDA code generation for CPU+GPU

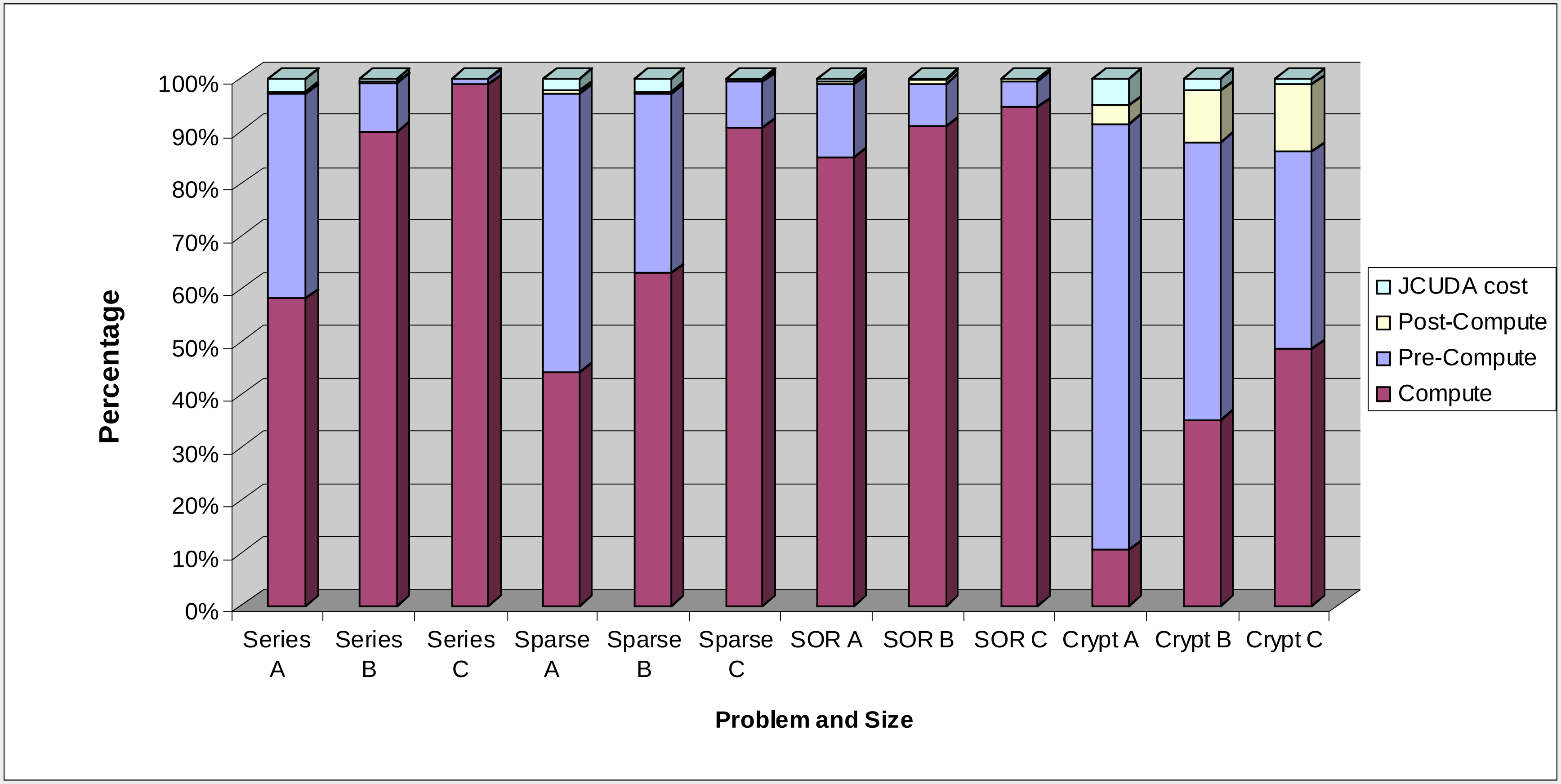
"JCUDA: a Programmer-Friendly Interface for Accelerating Java Programs with CUDA", Yonghong Yan, Max Grossman, Vivek Sarkar (submitted to Europar 2009)



Speedups using Nvidia GTX 280 GPU

Benchmark	**Series**			**Sparse**			**SOR**			**Crypt**		
Data Size	A	B	C	A	B	C	A	B	C	A	B	C
Java-1-thread execution time (s)	7.62	77.42	1219.40	0.50	1.17	19.87	0.62	1.60	2.82	0.51	3.26	8.16
Java-2-thread execution time (s)	3.84	39.21	755.05	0.26	0.54	8.68	0.26	1.32	2.59	0.27	1.65	4.10
Java-4-thread execution time (s)	2.03	19.82	390.98	0.25	0.39	5.32	0.16	1.37	2.70	0.11	0.21	2.16
JCUDA execution time (s)	0.23	0.98	8.54	0.17	0.27	1.22	0.68	1.19	2.12	0.11	0.21	0.37
JCUDA Speedup w.r.t. Java-1-thread	**32.55**	**78.68**	**142.87**	**2.90**	**4.29**	**16.26**	**0.92**	**1.34**	**1.33**	**4.54**	**15.76**	**21.87**

Breakdown of Kernel Execution Time



Implementation Challenges for Places

- Extend work-stealing algorithms to be locality-conscious (place-aware)
- Efficient implementations of data distributions
- Multi-place memory management and garbage collection
- Efficient translation of inter-place communication to multicore communication primitives
- Memory consistency for shared data accessed at multiple places

X10 + Habanero Execution Model: Portable Parallelism in Four Dimensions

1. Lightweight dynamic task creation & termination
   - *async*, *finish* (from X10)
2. Locality control --- task and data distributions
   - *places* (from X10)
3. Mutual exclusion
   - *isolated* (from Habanero --- extension of X10 atomic)
4. Collective and point-to-point synchronization
   - *phasers* (from Habanero --- extension of X10 *clocks*)

RICE

Multi-Place Isolation

- **X10 atomic:** An atomic block ...
  - must be **nonblocking**
  - must be **sequential**
  - must not access remote data (**single-place locality**)
- **Habanero isolated:** An isolated block
  - must be **nonblocking** (finish is okay, but blocking wait operations are not)
  - may create child activities --- **nested parallelism** with implicit finish for isolated
  - can be **multi-place**
    - isolated (*) --- isolated at all places
    - isolated (<place-list>) --- isolated at designated places
    - Default: isolated = isolated (*)

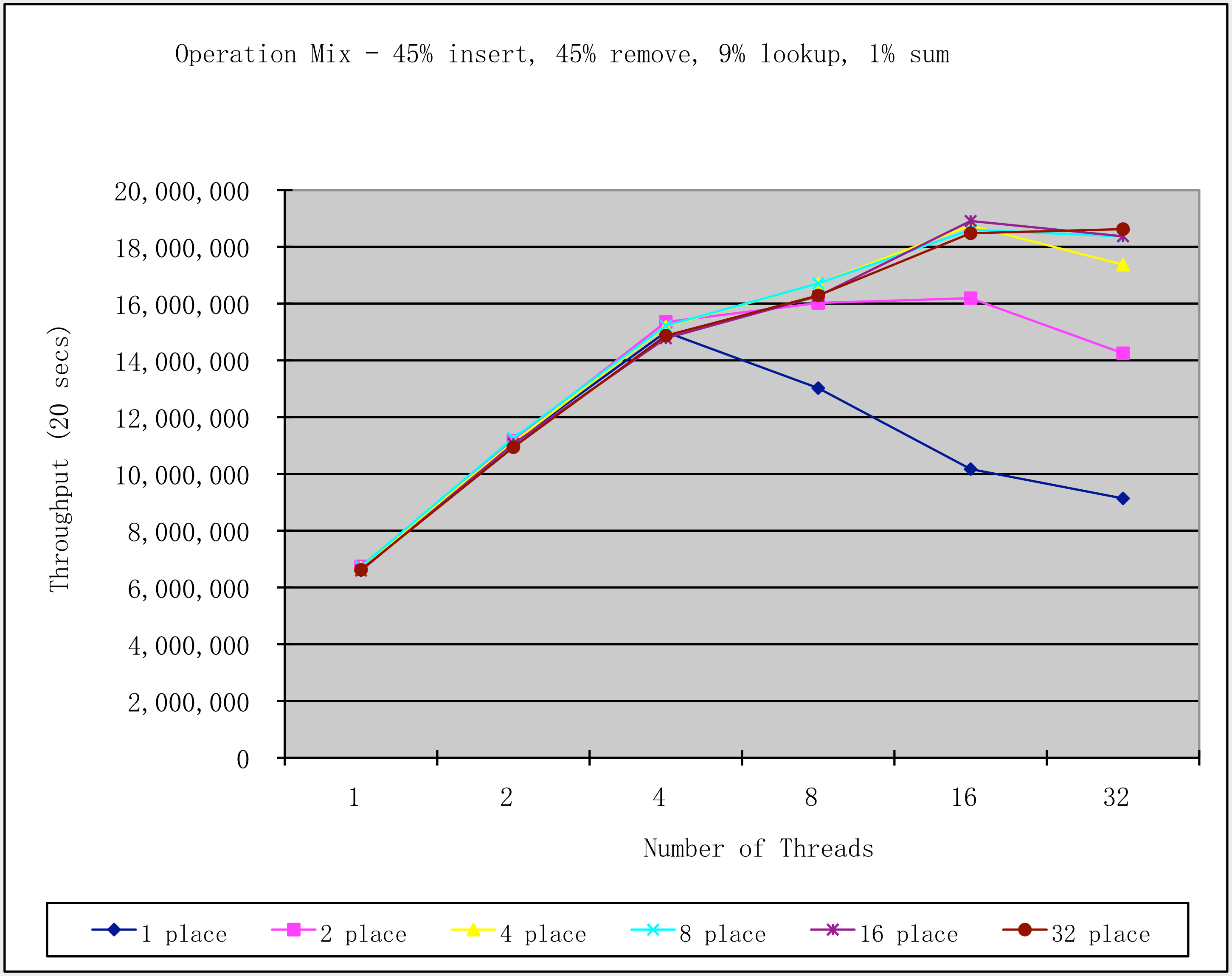
```
// X10 example w/ single-place atomic:
// insert in middle of list
Node node = new Node(data);
atomic {
  // Throw BadPlace Exception if
  // node.place or cur.place != here
  node.next = cur.next;
  cur.next = node;
}
```

```
// Habanero example w/ multi-place
// isolated: insert in middle of list
Node node = new Node(data);
isolated (cur, node) {
  // No BadPlaceException in this
  // example
  node.next = cur.next;
  async cur.next = node;
} // implicit finish at end of isolated
```

Lock-Based Implementation of Multi-Place Isolation

- Two levels of locks
  - Level 1: global read-write lock, G
  - Level 2: array of locks L, one per place
- isolated (<place-list>) implemented as follows
  - Obtain read lock on G
  - Obtain place lock L[p], for each place p in place-list (in sorted order to avoid deadlock)
- isolated (*) implemented as follows
  - Obtain write lock on G
- "Improved Scalability of Lock-Based Atomicity through Places", R.Zhang, Z.Budimlic, V.Sarkar, W.Scherer

Preliminary Evaluation of Multi-Place Isolation: Sorted Linked List on UltraSPARC T1



RICE

Implementation Challenges for Multi-Place Isolation

- Extend two-level locking approach to hierarchical places
- Extend transactional memory implementations for multi-place isolation
- Use compiler techniques to further refine locking granularity e.g.,
    - "Minimum lock assignment: A method for exploiting concurrency among critical sections", Yuan Zhang, Vugranam Sreedhar, Weirong Zhu, Vivek Sarkar, Guang Gao, LCPC 2008

X10 + Habanero Execution Model: Portable Parallelism in Four Dimensions

1. Lightweight dynamic task creation & termination
   - *async*, *finish* (from X10)
2. Locality control --- task and data distributions
   - *places* (from X10)
3. Mutual exclusion
   - *isolated* (from Habanero --- extension of X10 atomic)
4. <u>Collective and point-to-point synchronization</u>
   - <u>*phasers*</u> (from Habanero --- extension of X10 *clocks*)

Overview of Phasers

- **Designed to handle multiple communication patterns**
  - Collective Barrier
  - Point-to-point synchronization
- **Dynamic parallelism**
  - # activities synchronized on phaser can vary dynamically
- **Support for “single” statements**
- **Phase ordering property**
- **Deadlock freedom in absence of explicit wait operations**
- **Amenable to efficient implementation**
  - Lightweight local-spin multicore implementation in Habanero project
- **Extension of X10 clocks**

- “Phasers: a Unified Deadlock-Free Construct for Collective and Point-to-point Synchronization”, J.Shirako, D.Peixotto, V.Sarkar, W.Scherer, ICS 2008

- “Phaser Accumulators: a New Reduction Construct for Dynamic Parallelism”, J.Shirako, D.Peixotto, V.Sarkar, W.Scherer, to appear in IPDPS 2009

RICE

Collective and Point-to-point Synchronization with Phasers

phaser ph = new phaser(MODE);

- **Allocate a phaser, register current activity with it according to MODE. Phase 0 of ph starts.**
- **MODE can be SIGNAL_ONLY, WAIT_ONLY, SIGNAL_WAIT (default) or SINGLE**
- ***Finish Scope rule:* phaser ph cannot be used outside the scope of its immediately enclosing finish operation**

async phased (MODE1(ph1), MODE2(ph2), …) S

- **Spawn S as an asynchronous (parallel) activity that is registered on phasers ph1, ph2, … according to MODE1, MODE2, …**
- ***Capability rule:* parent activity can only transmit phaser capabilities to child activity that are a subset of the parent's capabilities, according to the lattice:**

SINGLE

SIGNAL_WAIT

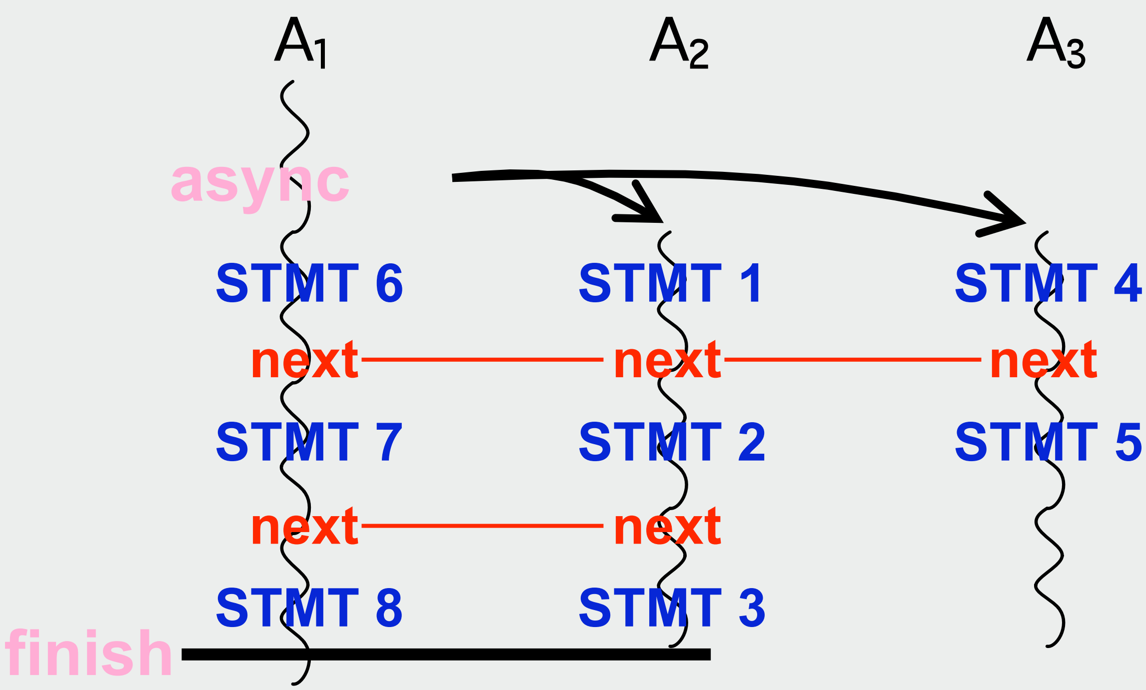
SIGNAL_ONLY WAIT_ONLY

next;

- **Advance *each* phaser that activity is registered on to its next phase; semantics depends on registration mode**

Using Phasers as Barriers with Dynamic Parallelism

```
finish {
  phaser ph = new phaser(); //A1
  async phased(ph){ STMT1; next; STMT2; next; STMT3; } //A2
  async phased(ph){ STMT4: next; STMT5; } //A3
                    STMT6; next; STMT7; next; STMT8; //A1
}
```



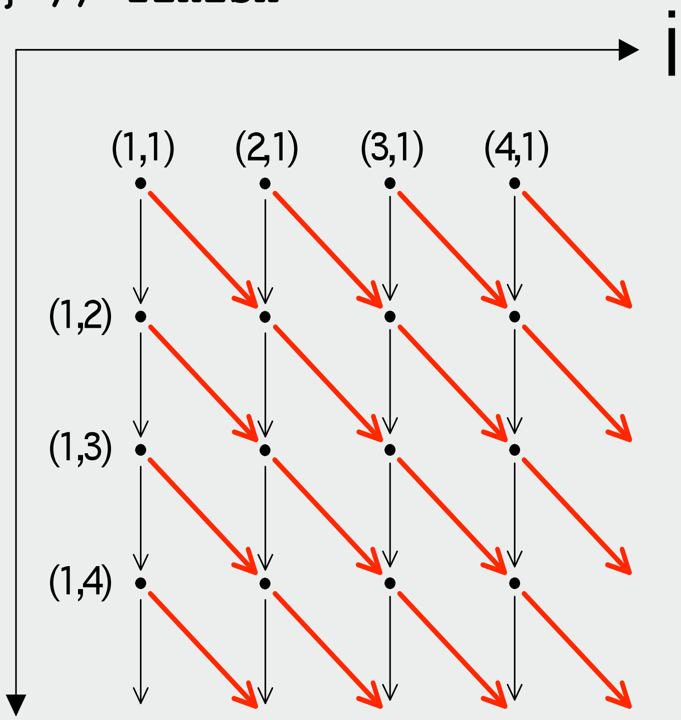
A_1 , A_2 , A_3 are registered on phaser ph

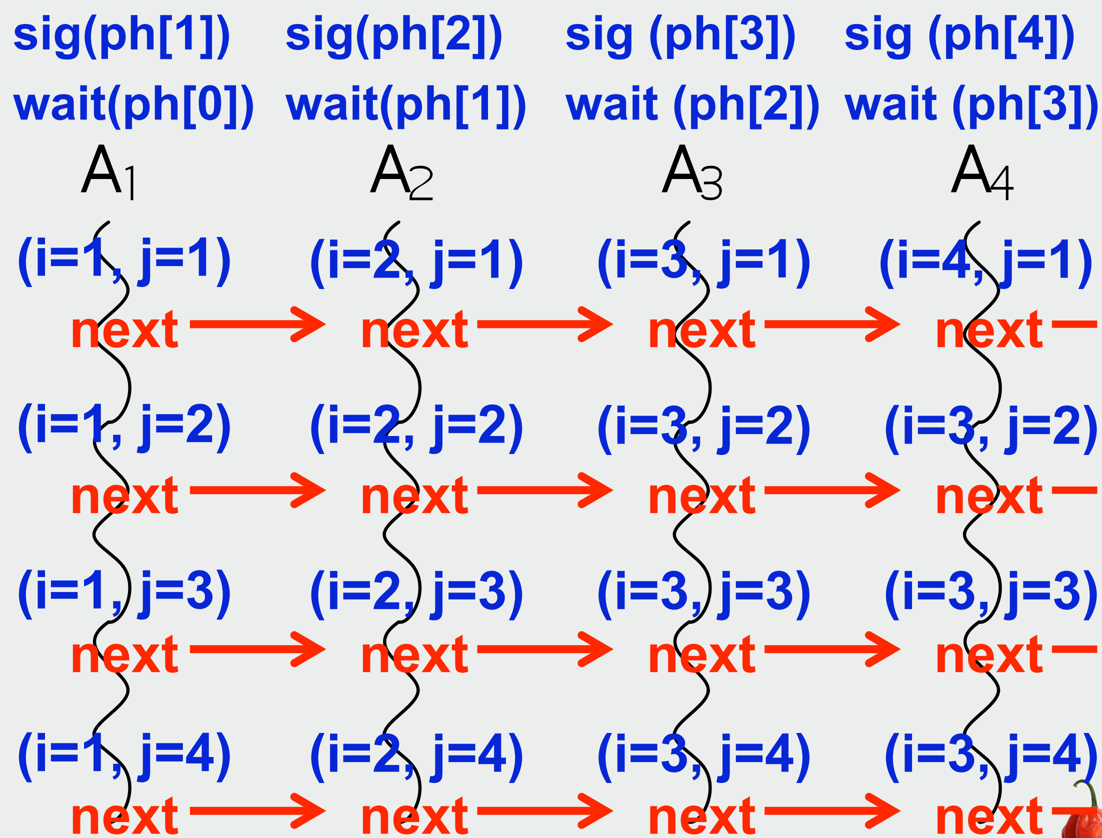
Dynamic parallelism

activities registered on phaser can vary

Example of Pipeline Parallelism with Phasers

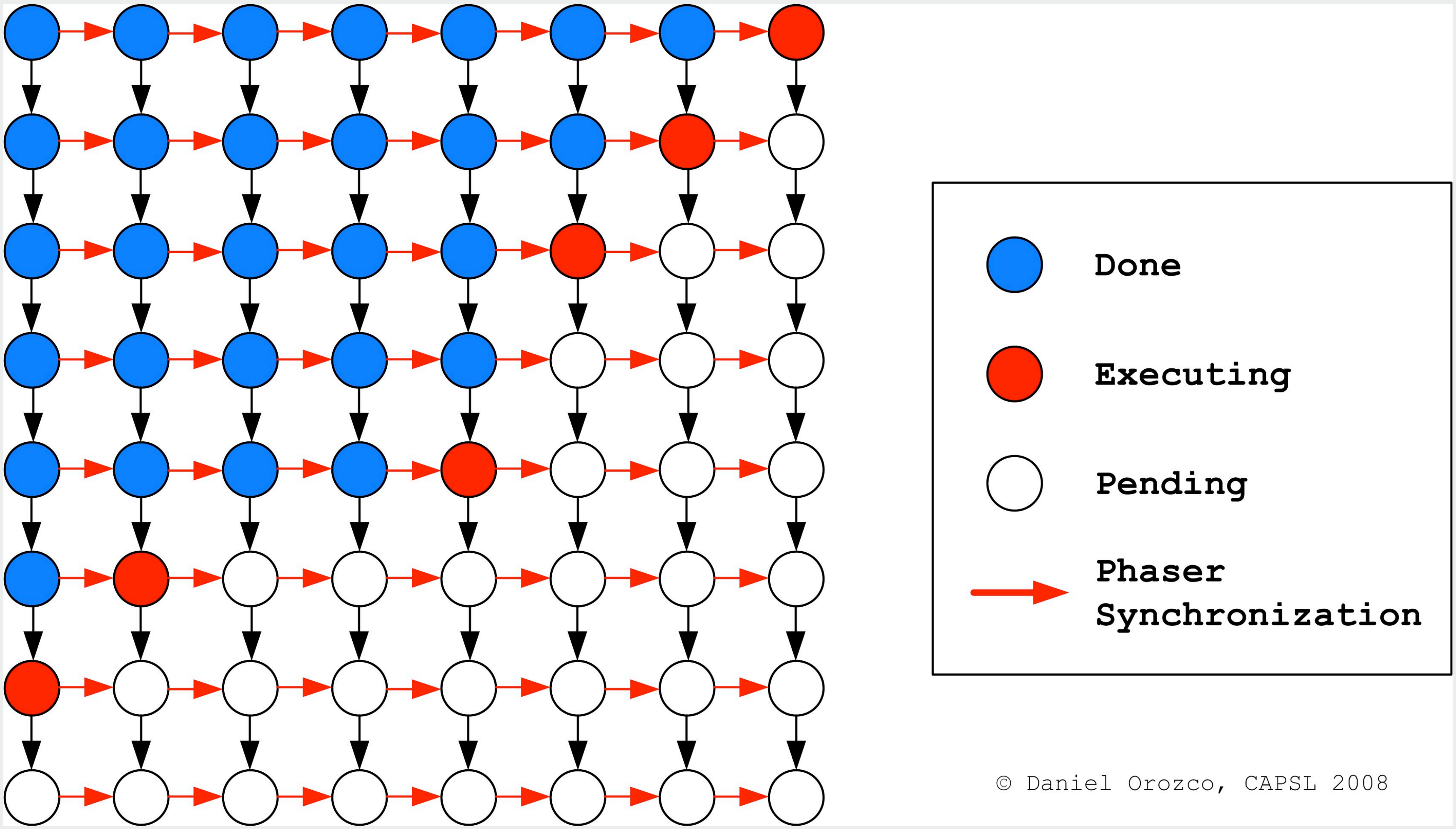
```
finish {
  phaser [] ph = new phaser[m+1];
  for (int i = 1; i < m; i++)
    async phased (ph[i]<SIG>, ph[i-1]<WAIT>){
      for (int j = 1; j < n; j++) {
        a[i][j] = foo(a[i][j], a[i][j-1], a[i-1][j-1]);
        next;
      } // for
} // finish
```





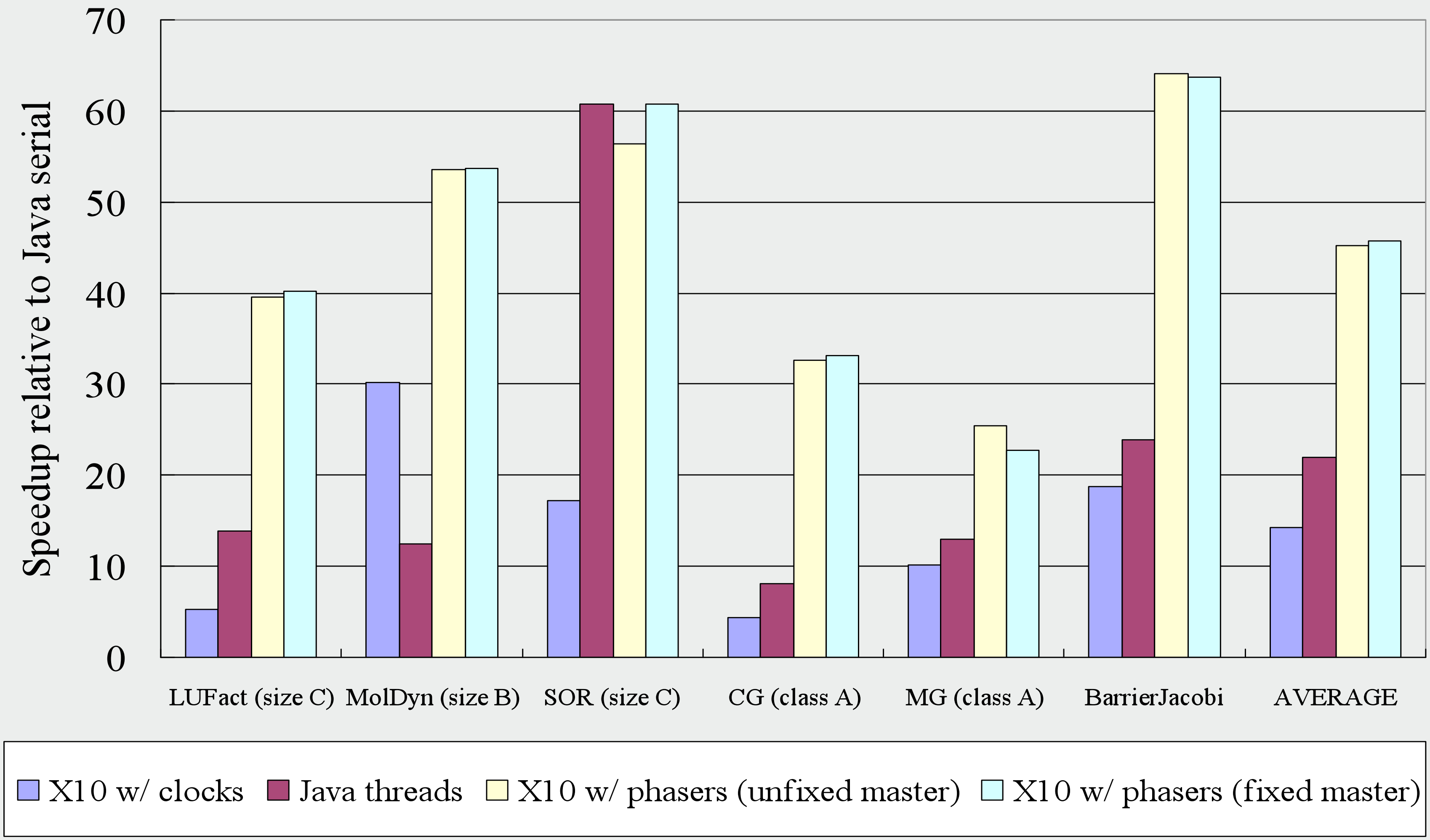
→ : Loop carried dependence

Example of Pipeline Parallelism with Phasers (contd)



RICE

Speedup on 64-way Power5+ SMP:
Java Grande Benchmarks & NAS Parallel Benchmarks



Average speedup with phasers (fixed master)

3.19x faster than X10 clocks, **2.08x** faster than Java threads

RICE

Implementation Challenges for Phasers

- Efficient performance (especially in context of JVM's and managed runtimes)
- Support for dynamic parallelism
- Support for single statements
- Support for split-phase barriers
- Extension to reductions (in progress)
- Extension to streaming parallelism (in progress)

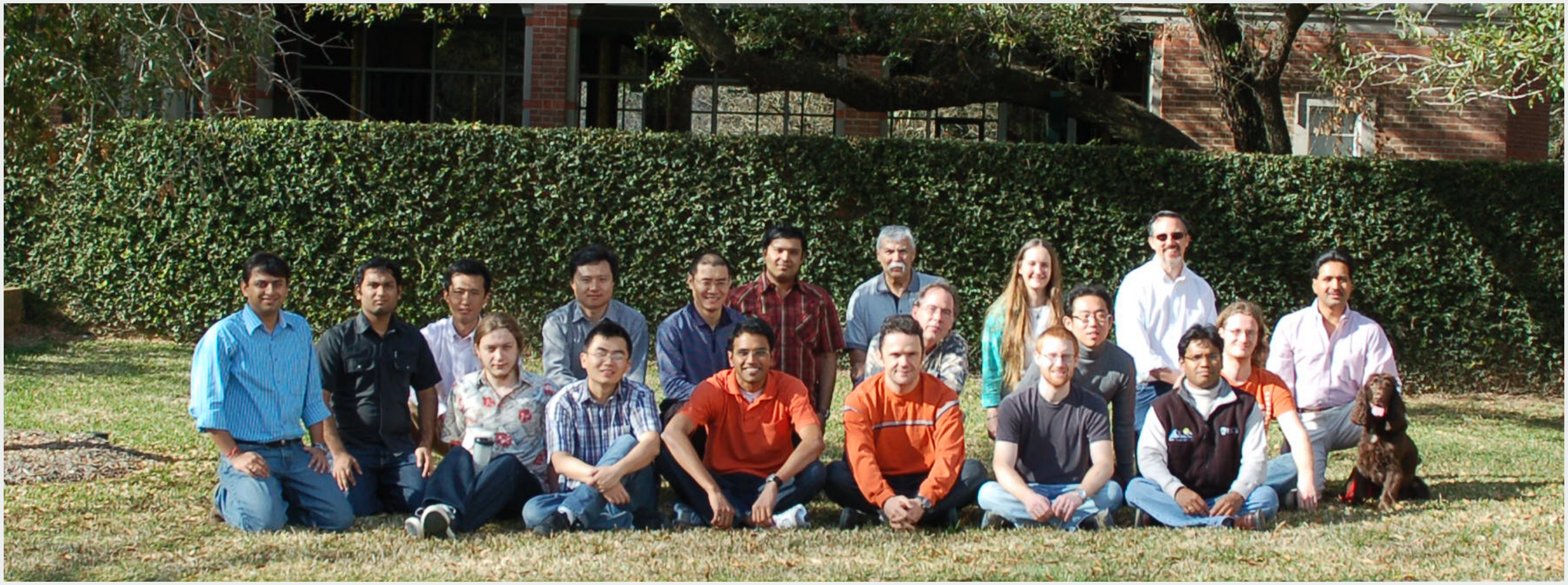
Comparison of Multicore Programming Models along Selected Dimensions

	Dynamic Parallelism	Locality Control	Mutual Exclusion	Collective & Point-to-point Synchronization	Data Parallelism
Cilk	Spawn, sync	None	Locks	None	None
Java Concurrency	Executors, Task Queues	None	Locks, monitors, atomic classes	Synchronizers	Concurrent collections
Intel TBB	Generic algs, tasks	None	Locks, atomic classes	None	Concurrent containers
.Net Parallel Extensions	Generic algs, tasks	None	Locks, monitors	Futures	PLINQ
OpenMP	SPMD (v2.5), Tasks (v3.0)	None	Locks, critical, atomic	Barriers	None
CUDA v1.0	None	Device, grid, block, threads	None	Barriers	SIMD
Intel Concurrent Collections	Tagged prescription of steps	None	None	Tagged put & get operations on Item Collections	None
X10 + *Habanero extensions* (builds on Java Concurrency)	Async, finish	Places	Isolated blocks, Java atomic classes	*Phasers, delayed async*	SIMD/MIMD array operations, Java concurrent collections

Acknowledgments: Rice Habanero Multicore Software Project

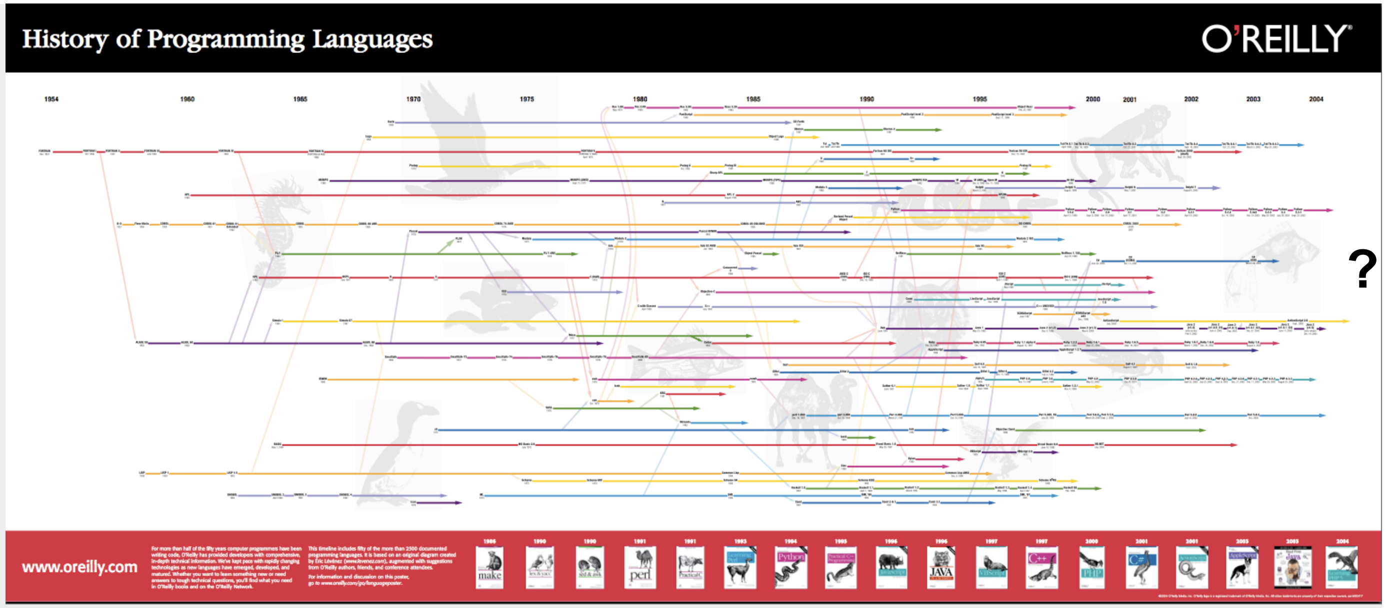
- Faculty
  - Vivek Sarkar, Bill Scherer
- Research Scientists
  - Zoran Budimlic, Chuck Koelbel
- Research Programmer
  - Vincent Cavé
- Postdocs
  - Jun Shirako, Yonghong Yan, Jisheng Zhao
- PhD Students
  - Current: Rajkishore Barik, Yi Guo, David Peixotto, Raghavan Raman, Sagnak Tasirlar
  - Graduated: Mack Joyner (9/08)
- Other collaborators at Rice
  - Laksono Adhianto, Keith Cooper, Tim Harvey, John Mellor-Crummey, Krishna Palem, Walid Taha, Linda Torczon, Anna Youssefi, Rui Zhang, Ryan Zhang, Fengmei Zhao
- Sponsors and Donors
  - AMD, BHP Billiton, DARPA, IBM, Intel, Microsoft, NSF, NVIDIA, Sun

Habanero Team Pictures



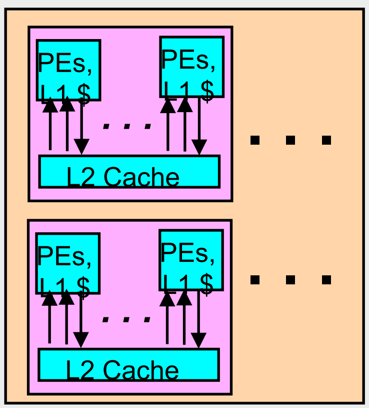
Send email to Vivek Sarkar (vsarkar@rice.edu) if you are interested in a PhD, postdoc, research scientist, or programmer position in the Habanero project, or in collaborating with us!

Conclusion

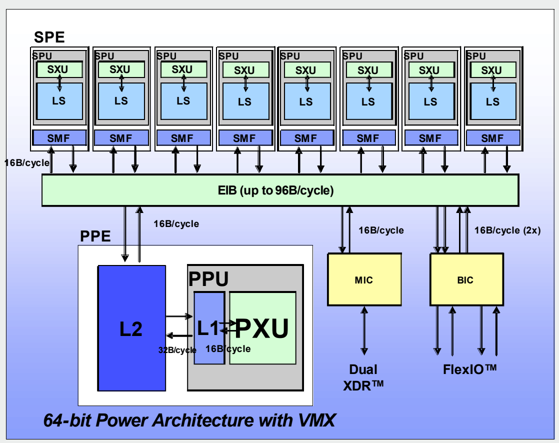


?

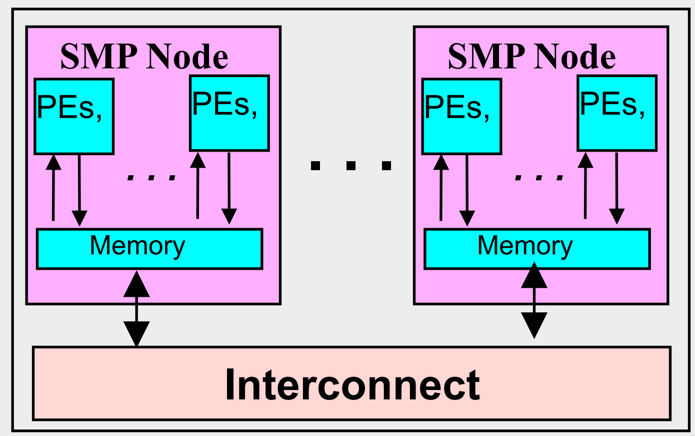
Homogeneous Multi-core



Heterogeneous Accelerators



High Performance Clusters



Advances in parallel languages, compilers, and runtimes are necessary to address the implementation challenges of multicore programming

RICE