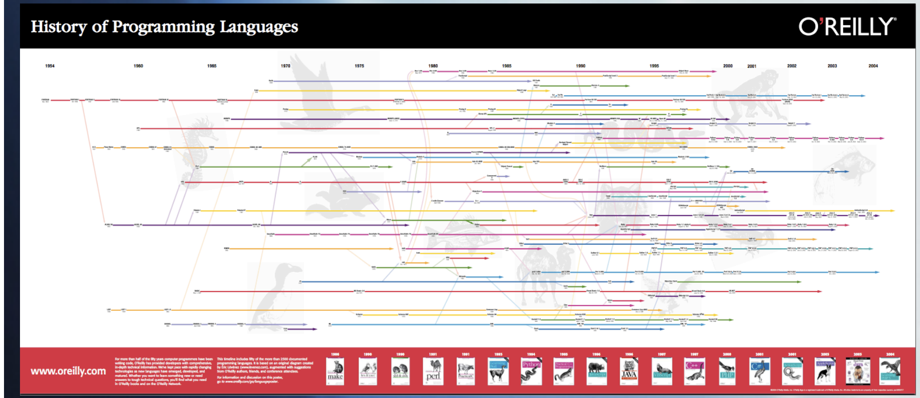
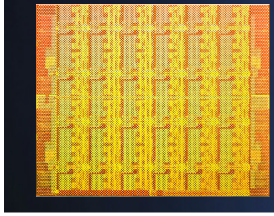
RICE George R. Brown School of Engineering
Computer Science

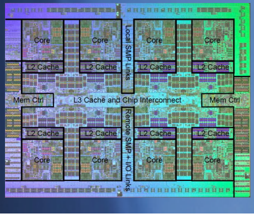
Programming Constructs for Exascale Systems and their Implementation Challenges

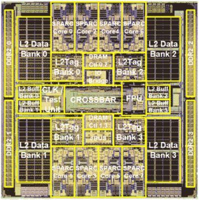
Vivek Sarkar
Rice University
vsarkar@rice.edu
http://habanero.rice.edu

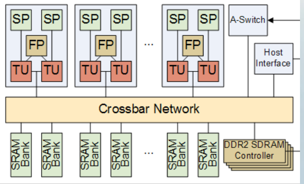


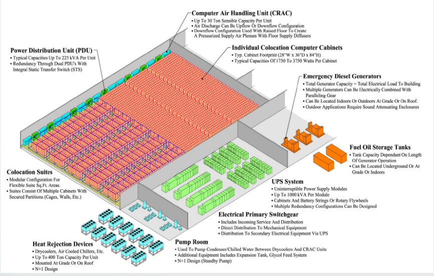
Workshop on Exascale Programming Challenges
July 27 – 29, 2011, Marina del Rey, CA





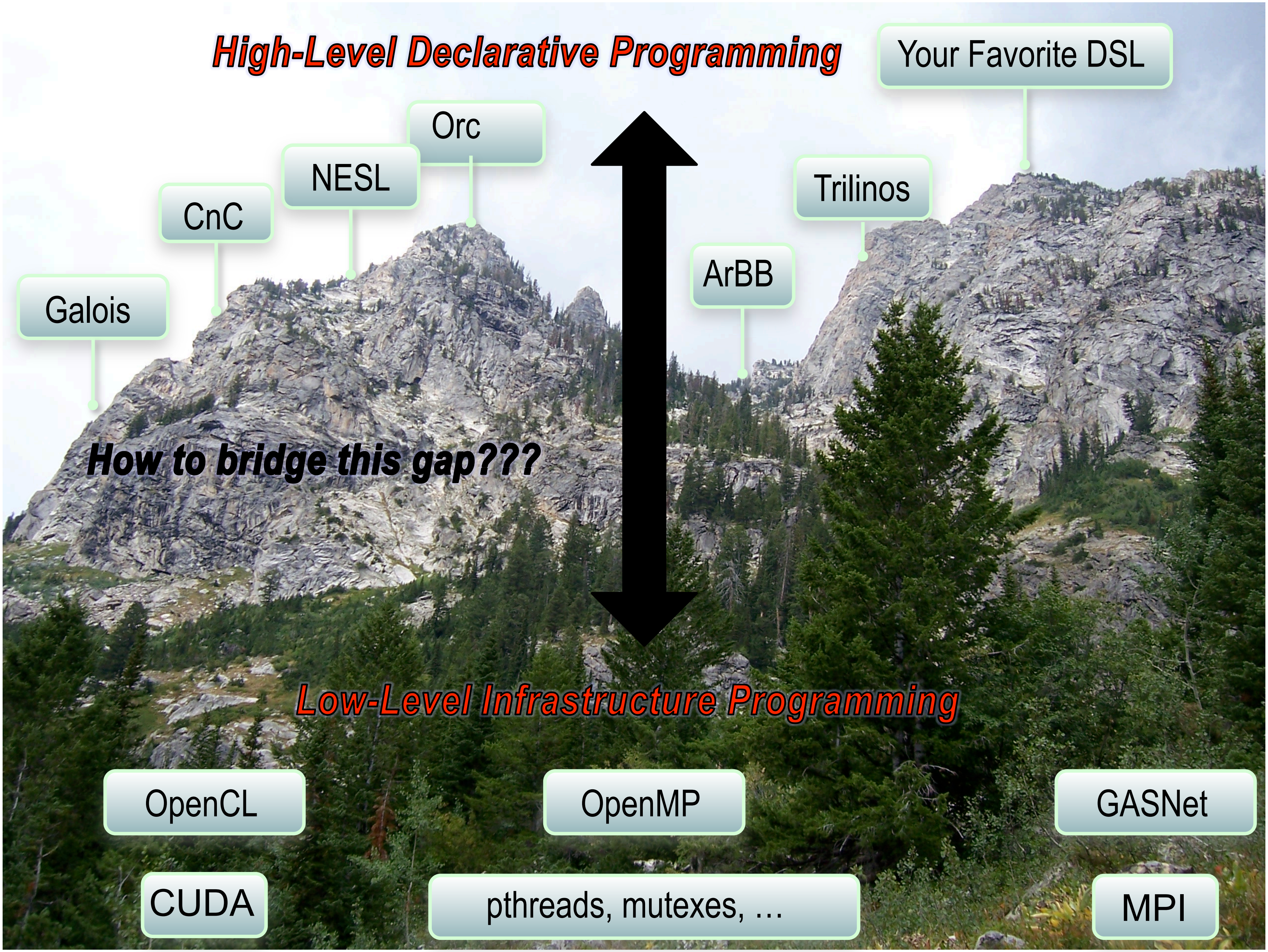






Workshop Goals

1. Define objective criteria for assessing programming models, language features, compilers, and runtime systems and metrics for success.
2. Prioritize **programming model, language, compiler and runtime challenges** for Exascale systems.
3. Prioritize options for (i) evolutionary path, (ii) revolutionary path, and (iii) bridging the gap between evolutionary and revolutionary paths.
4. Lay out a roadmap, with options, timeline, and rough cost estimates for programming Exascale systems that are responsive to the needs of applications and future architectural constraints.



High-Level Declarative Programming

Your Favorite DSL

Orc

NESL

CnC

Trilinos

ArBB

Galois

How to bridge this gap???

Low-Level Infrastructure Programming

OpenCL

OpenMP

GASNet

CUDA

pthreads, mutexes, …

MPI

This Talk

- Identification of four classes of intermediate-level programming constructs that will be necessary for mapping applications onto Exascale hardware
  - Focus is on “performance-aware” parallel constructs
    - Compiler and runtime should devote major effort to optimizing these constructs
- Summary of some recent compiler and runtime implementation experiences with these constructs
- Use of these constructs in bridging the gap between evolutionary and revolutionary solutions

Four classes of Intermediate-Level Programming Constructs

1) Asynchronous tasks and data transfers e.g.,

- **MPI:** *mpi_isend, mpi_irecv, mpi_wait*
- **OpenMP:** *task, taskwait*
- **Cilk:** *spawn, sync*
- **CAF, UPC, Chapel:** *function shipping*
- **X10:** *async, finish, asyncMemcpy, futures, foreach*
- **Habanero:** *async, finish, asyncMemcpy, futures, async-await, forall*

2) Collective and point-to-point synchronization & reductions e.g.,

- **MPI:** *mpi_send, mpi_recv, mpi_barrier, mpi_reduce,*
- **OpenMP:** *barrier, reductions*
- **Cilk:** *reducers*
- **CAF, UPC, Chapel:** *barrier, reductions*
- **X10:** *clocks, finish accumulators, conditional atomic*
- **Habanero:** *phasers, phaser accumulators, finish accumulators*

Four classes of Intermediate-Level Programming Constructs

3) Mutual exclusion e.g.,

- **OpenMP:** ***atomic, critical***
- **X10, Chapel, STM systems:** ***atomic***
- **Galois: operations on unordered sets**
- **Habanero:** ***isolated***

4) Locality control for task and data distribution e.g.,

- **MPI:** ***all-local (shared-nothing)***
- **CAF, Chapel, UPC, X10:** ***PGAS storage model (local vs. remote)***
- **Sequoia:** ***hierarchical storage model w/ static tasks***
- **Habanero:** ***hierarchical place tree w/ dynamic parallelism, heterogeneity***

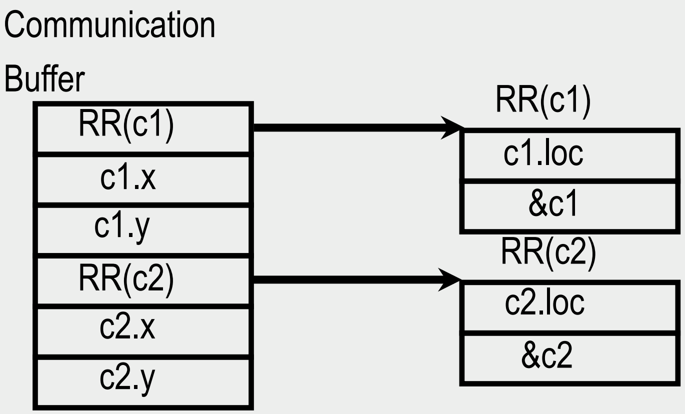
- Scalable implementations of these constructs require first-class compiler and runtime support; evolutionary solutions are possible with only runtime support
- Constructs can be exposed to the programmer by extending current low-level programming models, or can be generated from high-level programming models

Implementation Experiences with these constructs

- **Habanero-Java**
  - Pedagogic language and implementation used to teach sophomore-level class on Fundamentals of Parallel Programming at Rice (COMP 322)
  - Derived from Java-based v1.5 of X10 language from 2007
- **Habanero-C**
  - Habanero-C optimizing compiler builds on Rose and LLVM
    - New Rose→LLVM translator improves communication between high-level & low-level optimizers
  - Habanero-C runtime supports work-stealing and work-sharing schedulers across homogeneous and heterogeneous processors (hybrid task scheduling across CPU computation workers, CPU communication workers, GPU workers, FPGA workers)
- **X10 v2.0.6 and 2.1.1**
  - First-class PGAS support with extensions for dynamic parallelism and heterogeneity (GPUs)
  - Communication optimizations implemented in X10 compiler front-end
- **OpenMP 3.0**
  - Implementation of phasers as library extension to OpenMP
  - Enhancements to OpenMP task scheduling (collaboration with IBM XL compiler team)

Class 1 example (Lightweight asynchronous tasks and data transfers) --- Communication Optimizations for Distributed-Memory X10 Programs

```
// Original Code
class C {
global var x;
global var y;
}
val c1:C = new C(2,3);
val c2:C = new C(3,4);
at (p) async {
... c1.x ...;
... c2.x ...;
... c2.y ...;
}
```

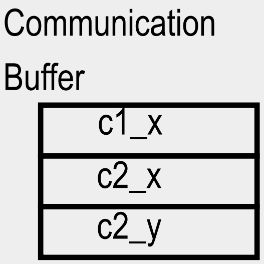


Communication Buffer

RR(c1)
c1.x
c1.y
RR(c2)
c2.x
c2.y

```
// Transformed Code
val c1:C = new C(2,3);
val c2:C = new C(3,4);
val c1_x = c1.x;
val c2_x = c2.x;
val c2_y = c2.y;
at (p) async {
... c1_x ...;
... c2_x ...;
... c2_y ...;
}
```

only three scalar values are communicated & no RR handles are communicated

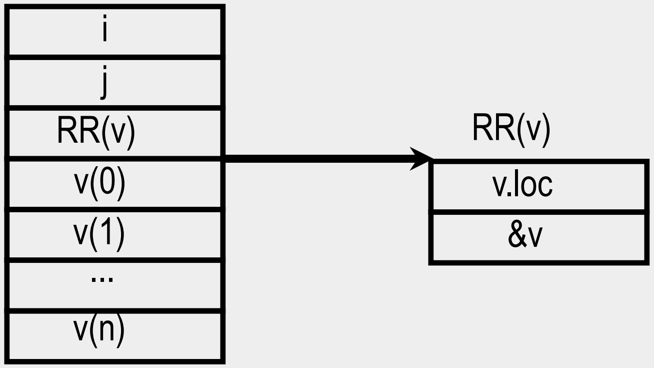


Communication Buffer

c1_x
c2_x
c2_y

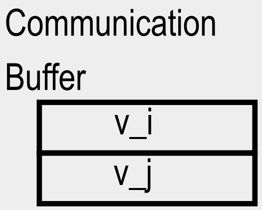
Communication Optimization: Scalar Replacement for Global Arrays

```
// Original Code
val i:int = ...;
val j:int = ...;
val v:Array[int](1) = new Array[int](n);
at (p) async {
... v(i);
... v(j);
}
```

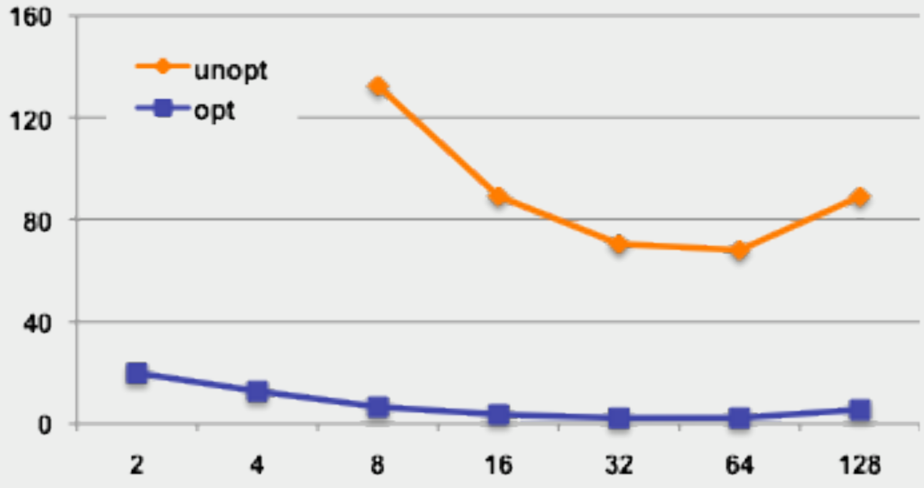


```
// Transformed Code
val i:int = ...;
val j:int = ...;
val v:Array[int](1) = new Array[int](n);
val v_i:int = v(i);
val v_j:int= v(j);
at (p) async {
... v_i;
... v_j;
}
```

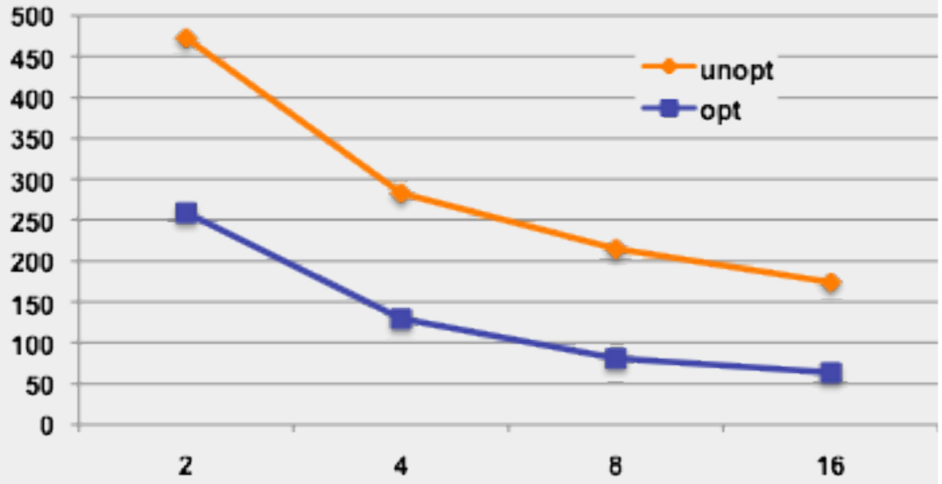
only two scalar values v_i and v_j are communicated



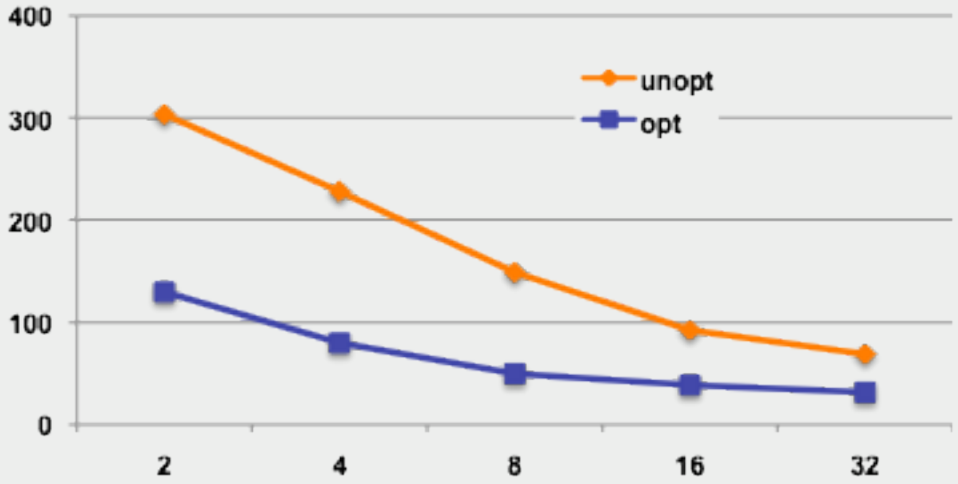
Experimental Results (MolDyn): Execution time in seconds



BlueGene/P Cluster



Power7 Cluster



Nehalam Cluster

Performance improvements due to communication optimization on

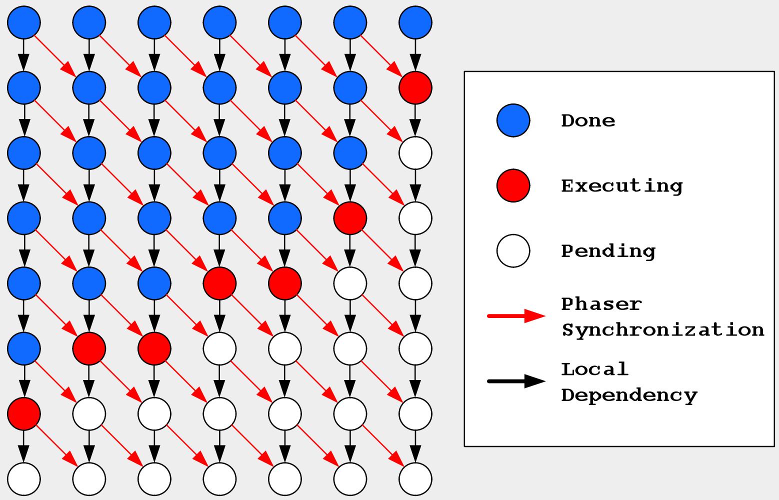
128-node BlueGene/P

32-node Nehalem

16-node Power7

“Communication Optimizations for Distributed-Memory X10 Programs”. Rajkishore Barik, Jisheng Zhao, David Grove, Igor Peshansky, Zoran Budimlić, Vivek Sarkar. IPDPS 2011.

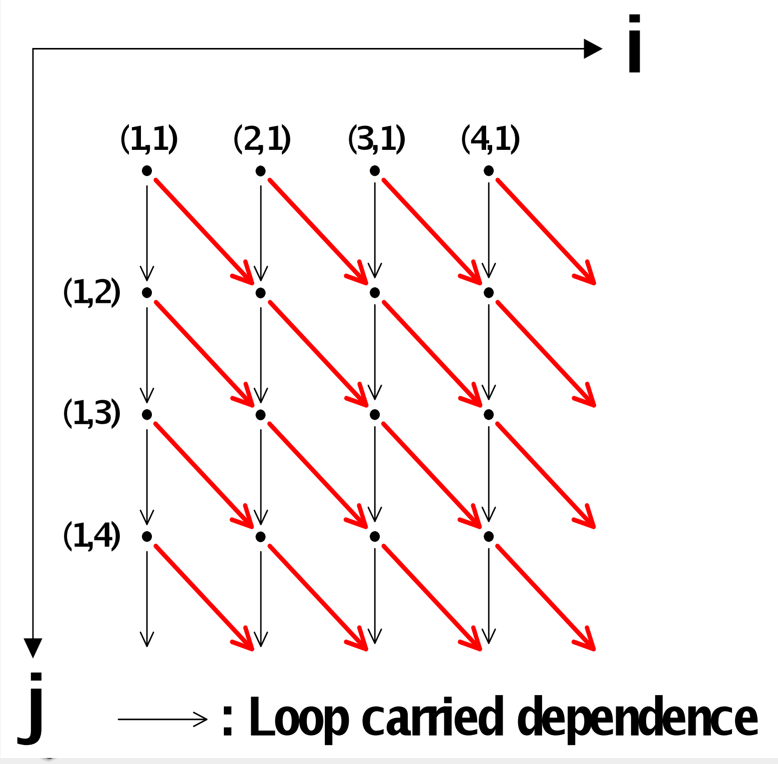
Class 2 example (Collective and point-to-point synchronization & reductions) --- Phasers

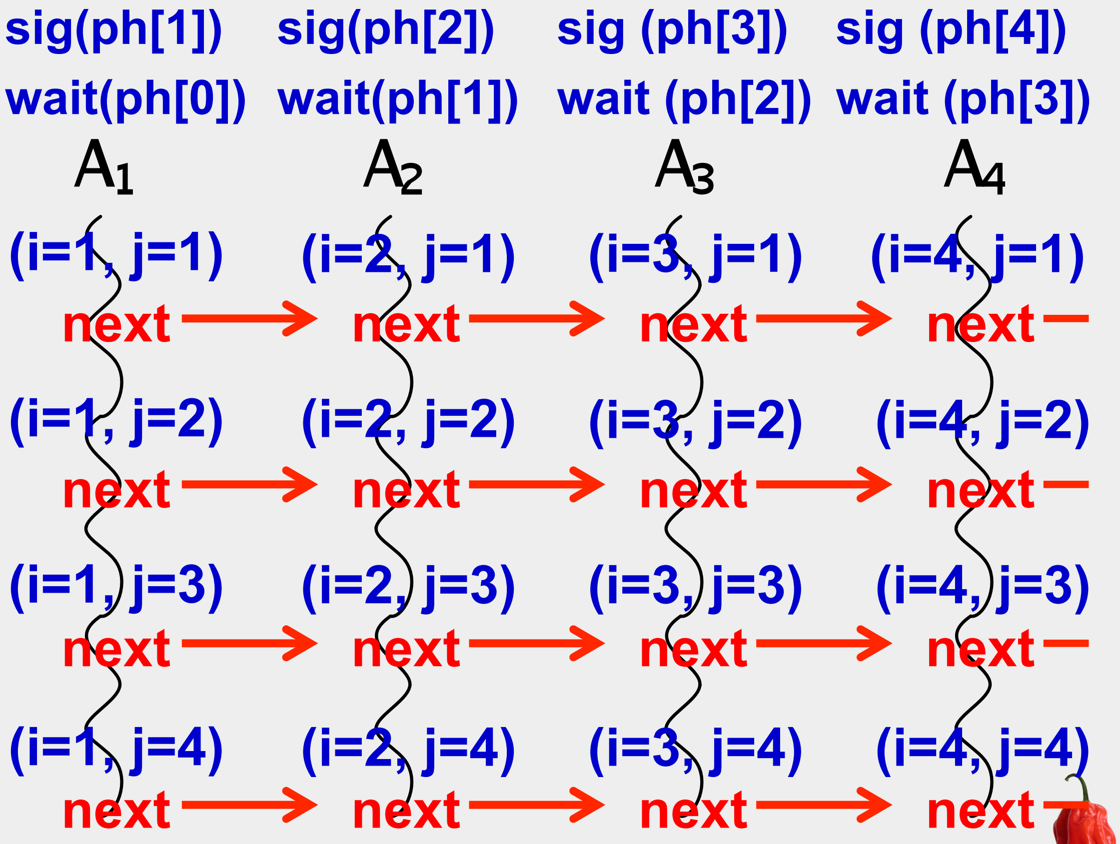


- **New synchronization construct designed to unify**
  - Strict and fuzzy barriers
  - Single statements
  - Asynchronous point-to-point synchronization
  - Asynchronous collectives
  - Streaming computations
  - Dynamic parallelism
- **Semantic guarantees**
  - **Dynamic phase ordering** --- if ∃ a phaser ph s.t. i1's signal phase w.r.t. ph is < i2's wait phase w.r.t. ph, then i1 must have completed before i2 started
  - **Deadlock freedom** --- no deadlock possible with next and finish operations
  - **Determinism** --- a data-race-free program with finish, async, futures, phasers must be deterministic
- **References**
  - "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, IPDPS 2009
  - "Hierarchical Phasers for Scalable Synchronization and Reduction", J.Shirako, V.Sarkar, IPDPS 2010

Example of Point-to-point Synchronization with Phaser Array

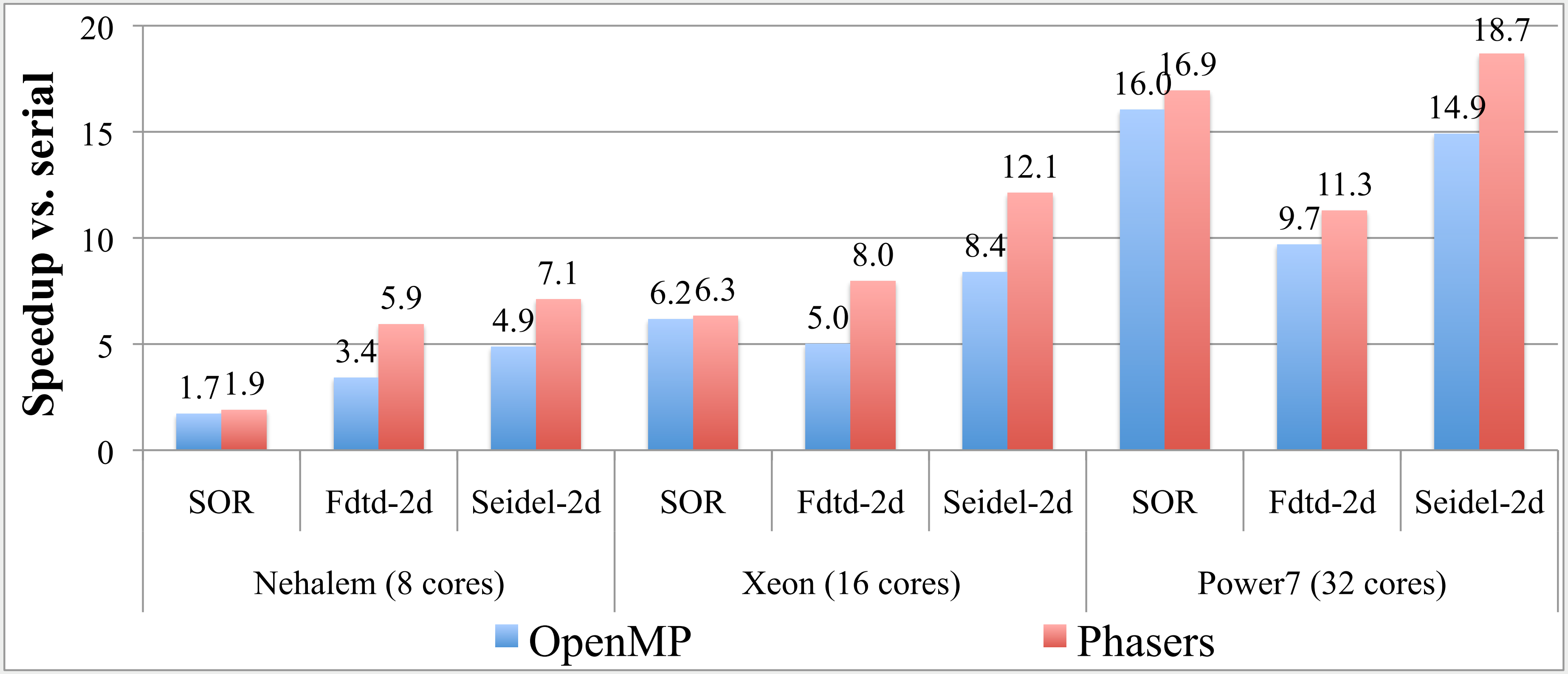
```
finish {
  phaser [] ph = new phaser[m+1];
  for (int i = 1; i < m; i++)
    async phased (ph[i]<SIG>, ph[i-1]<WAIT>){
      for (int j = 1; j < n; j++) {
        a[i][j] = foo(a[i][j], a[i][j-1], a[i-1][j-1]);
        next;
      } // for
} // finish
```





sig(ph[1])	sig(ph[2])	sig (ph[3])	sig (ph[4])
wait(ph[0])	wait(ph[1])	wait (ph[2])	wait (ph[3])
A_1	A_2	A_3	A_4
(i=1, j=1)	(i=2, j=1)	(i=3, j=1)	(i=4, j=1)
next	next	next	next
(i=1, j=2)	(i=2, j=2)	(i=3, j=2)	(i=4, j=2)
next	next	next	next
(i=1, j=3)	(i=2, j=3)	(i=3, j=3)	(i=4, j=3)
next	next	next	next
(i=1, j=4)	(i=2, j=4)	(i=3, j=4)	(i=4, j=4)
next	next	next	next

Comparing OpenMP Barriers with Point-to-Point Synchronization using Phasers Library in OpenMP



"Unifying Barrier and Point-to-Point Synchronization in OpenMP with Phasers", J. Shirako, K. Sharma, V. Sarkar, IWOMP 2011, June 2011.

Example of Asynchronous Reductions with Phaser Accumulators

```
phaser ph = new phaser(signalWait);
accumulator a = new accumulator(ph, accumulator.SUM, int.class);
accumulator b = new accumulator(ph, accumulator.MIN, double.class);
```

Allocation: Specify operator and type of accumulator

```
foreach (point [i] : [0:n-1]) phased (ph<signalWait>) {
   int iv = 2*i + j;
   double dv = -1.5*i + j;
   a.send(iv); b.send(dv);
   // Do other work before next
```

send: Send a value to accumulator

```
   next;
```

next: Barrier operation; advance the phase

```
   int sum = a.result().intValue();
   double min = b.result().doubleValue();
   …
}
```

result: Get the result from previous phase (no race condition)

Data-Driven Futures (DDFs) for Deterministic Task Parallelism --- creating Dynamic Dataflow graphs on the fly

Approach: separation of classical “futures” into data (DDF) and control (async await) parts

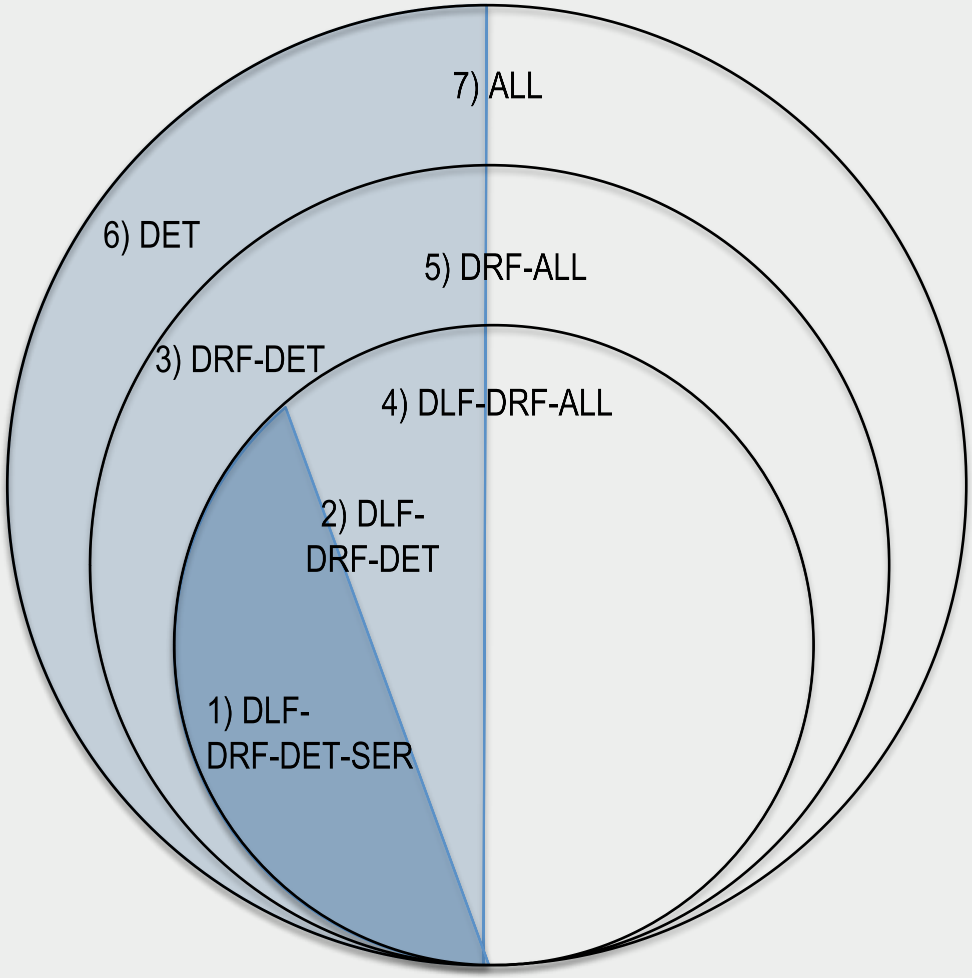
🟦	New	DDF creation e.g., new Data DrivenFuture()
🟩	Put	Fill in DDF and release any waiting async’s
🟥	Await	An await clause on an async ensures that the async is not scheduled until all input DDF’s become available; gets on these input DDF’s will not block as a result (Different from Ivar model, where task may block on each Ivar access)

DDF Example

```
DataDrivenFuture left = new DataDrivenFuture();
DataDrivenFuture right = new DataDrivenFuture();
finish {
    async await ( left ) useLeftChild();
    async await ( right ) useRightChild();
    async await ( left, right ) useBothChildren();
    async left.put(leftChildCreator());
    async right.put(rightChildCreator());
}
```

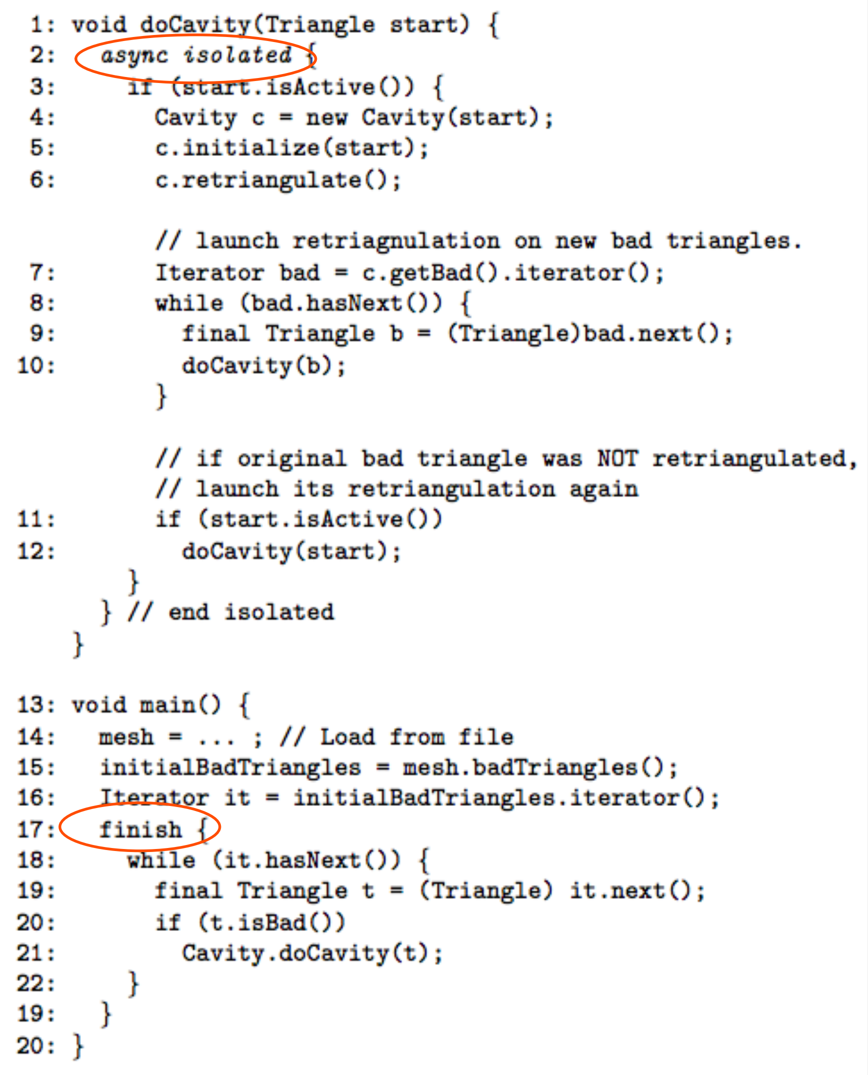
Reference: Sağnak Taşırlar, Vivek Sarkar, “Data-Driven Tasks and their Implementation,” ICPP 2011 (to appear).

Classification and Properties of Parallel Programs



- Legend
  - DET = Deterministic
  - DRF = Data-Race-Free
  - DLF = DeadLock-Free
  - SER = Serializable
- Subsets of task-parallel constructs can be used to guarantee membership in certain classes e.g.,
- *If an HJ program is data-race-free and only uses async, finish, and phaser constructs (no mutual exclusion), then it is guaranteed to belong to the DLF-DRF-DET class*
- *Adding async await yields programs in the DRF-DET class*
- *Adding isolated yields programs in the DRF-ALL class*

Class 3 example (Mutual exclusion) --- Delauney Mesh Refinement in Habanero-Java and Galois-Java



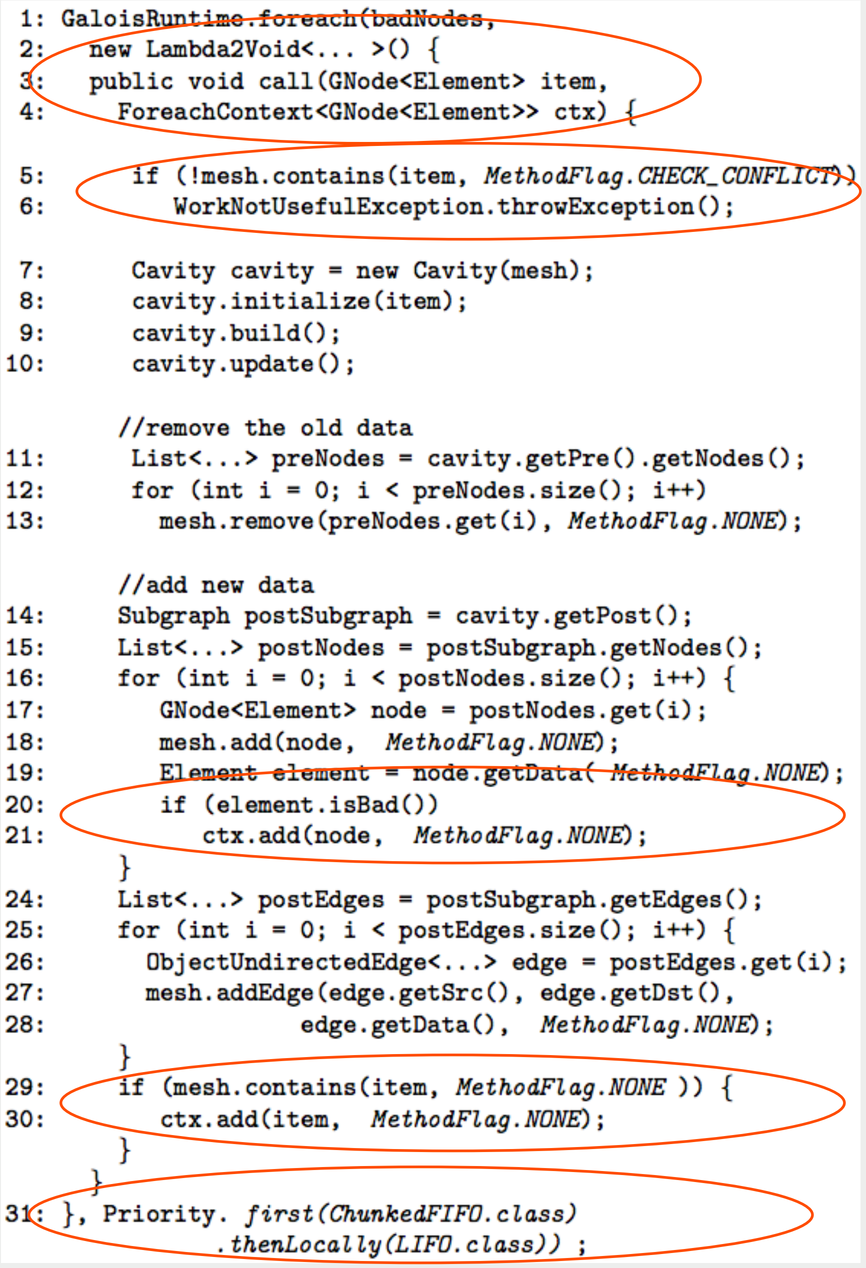
```
void doCavity(Triangle start) {
  async isolated {
    if (start.isActive()) {
      Cavity c = new Cavity(start);
      c.initialize(start);
      c.retriangulate();

      // launch retriagnulation on new bad triangles.
      Iterator bad = c.getBad().iterator();
      while (bad.hasNext()) {
        final Triangle b = (Triangle)bad.next();
        doCavity(b);
      }

      // if original bad triangle was NOT retriangulated,
      // launch its retriangulation again
      if (start.isActive())
        doCavity(start);
    }
  } // end isolated
}

void main() {
  mesh = ... ; // Load from file
  initialBadTriangles = mesh.badTriangles();
  Iterator it = initialBadTriangles.iterator();
  finish {
    while (it.hasNext()) {
      final Triangle t = (Triangle) it.next();
      if (t.isBad())
        Cavity.doCavity(t);
    }
  }
}
```

Habanero-Java



```
GaloisRuntime.foreach(badNodes,
  new Lambda2Void<... >() {
  public void call(GNode<Element> item,
    ForeachContext<GNode<Element>> ctx) {

     if (!mesh.contains(item, MethodFlag.CHECK_CONFLICT))
        WorkNotUsefulException.throwException();

     Cavity cavity = new Cavity(mesh);
     cavity.initialize(item);
     cavity.build();
     cavity.update();

    //remove the old data
     List<...> preNodes = cavity.getPre().getNodes();
     for (int i = 0; i < preNodes.size(); i++)
       mesh.remove(preNodes.get(i), MethodFlag.NONE);

    //add new data
    Subgraph postSubgraph = cavity.getPost();
    List<...> postNodes = postSubgraph.getNodes();
    for (int i = 0; i < postNodes.size(); i++) {
       GNode<Element> node = postNodes.get(i);
       mesh.add(node,  MethodFlag.NONE);
       Element element = node.getData( MethodFlag.NONE);
       if (element.isBad())
          ctx.add(node,  MethodFlag.NONE);
    }
    List<...> postEdges = postSubgraph.getEdges();
    for (int i = 0; i < postEdges.size(); i++) {
      ObjectUndirectedEdge<...> edge = postEdges.get(i);
      mesh.addEdge(edge.getSrc(), edge.getDst(),
                 edge.getData(),  MethodFlag.NONE);
    }
    if (mesh.contains(item, MethodFlag.NONE )) {
       ctx.add(item,  MethodFlag.NONE);
    }
  }
}, Priority. first(ChunkedFIFO.class)
           .thenLocally(LIFO.class)) ;
```

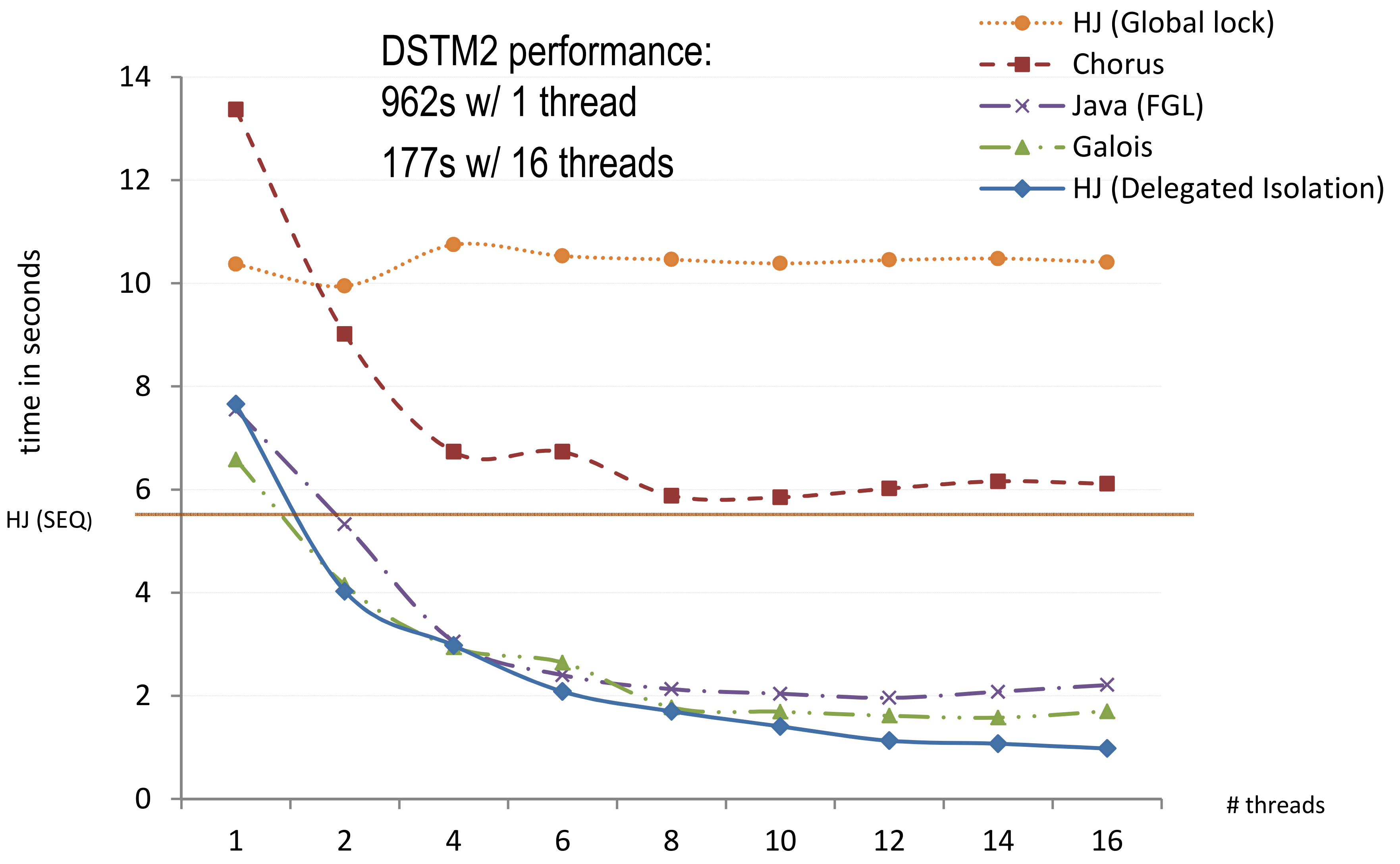
Galois-Java

Delegated Isolation

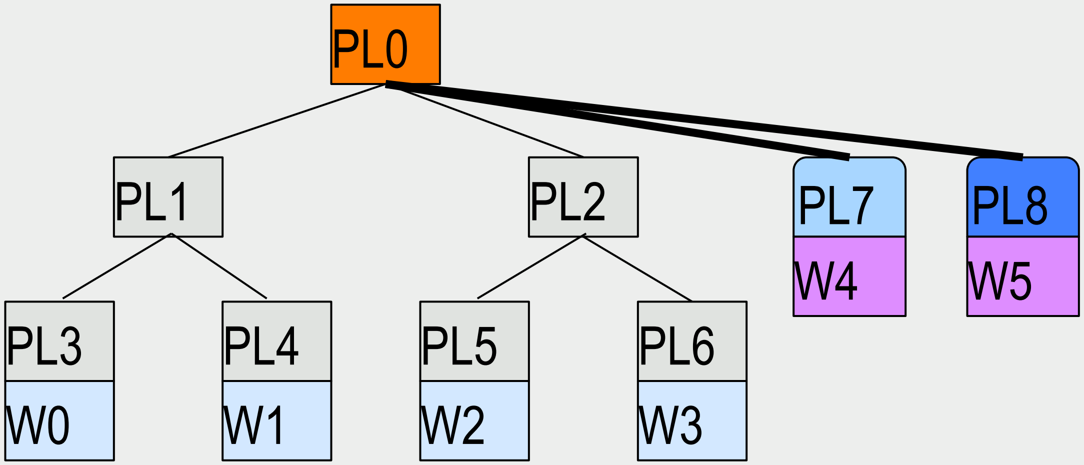
- Challenge: scalable implementation of *isolated* with deadlock safety and livelock safety when object-set is not known on entry to isolated block
- Approach:
  - Restrict attention to "async isolated" case
    - replace non-async "isolated" by "finish async isolated"
  - Task dynamically acquires ownership of each object accessed in isolated block (optimistic parallelism)
  - On conflict, task A transfers all ownerships to conflicting task B and delegates execution of isolated block to B
  - Deadlock-freedom and livelock-freedom guarantees
- "Delegated Isolation", R. Lublinerman, J. Zhao, Z. Budimlic, S. Chaudhuri, V. Sarkar, OOPSLA 2011 (to appear)

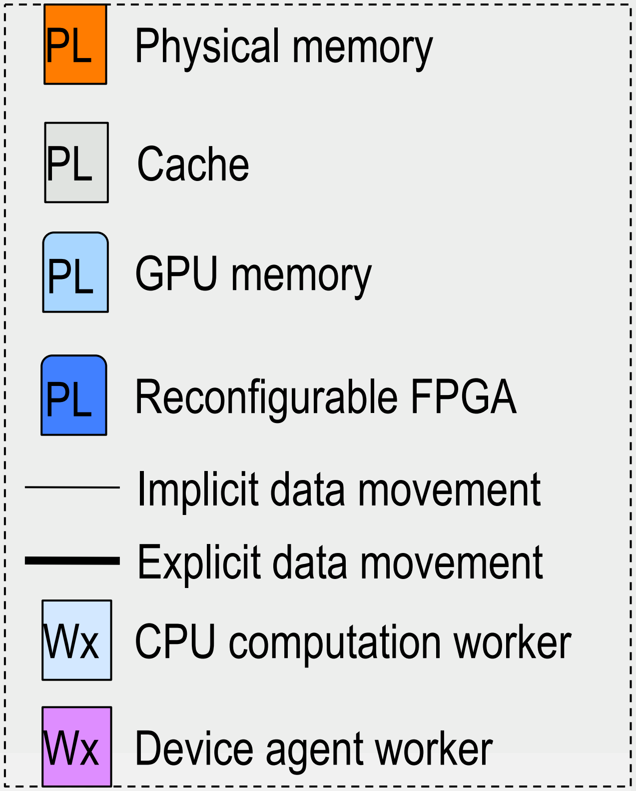
Performance: DMR benchmark on 16-core Xeon SMP

(100,770 initial triangles of which 47,768 are “bad”; average # retriangulations is ~ 130,000)



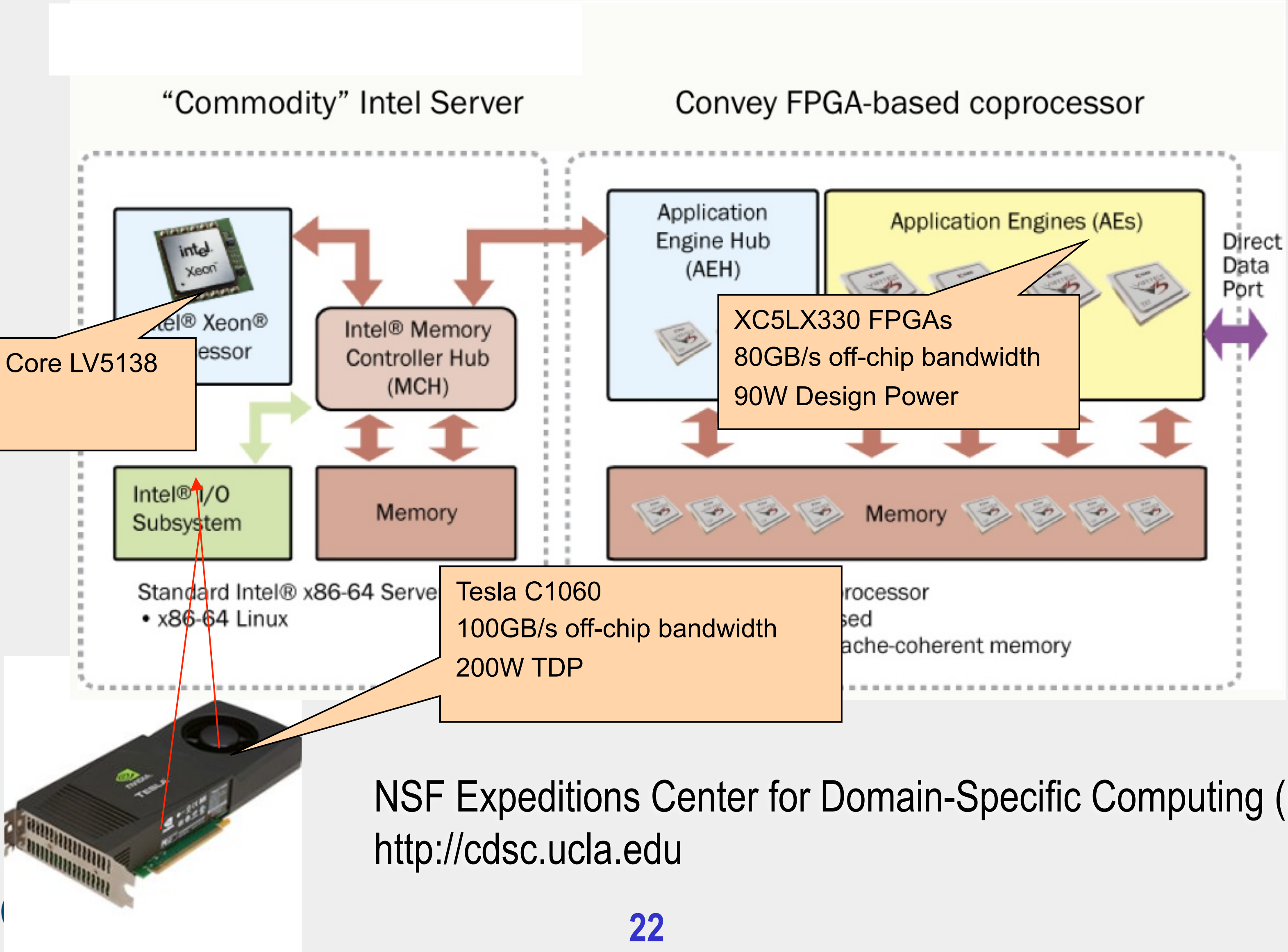
Class 4 example: HC Hierarchical Place Trees for heterogeneous architectures





- **Devices (GPU or FPGA) are represented as memory module places and agent workers**
  - **GPU memory configuration are fixed, while FPGA memory are reconfigurable at runtime**
- **async at(P) S**
  - **Creates new activity to execute statement S at place P (can be CPU, GPU, FPGA)**
- **Physically explicit data transfer between main memory and device memory**
  - **Use of IN and OUT clauses to improve programmability of data transfers**
- **Device agent workers**
  - **Perform asynchronous data copy and task launching for device**

Heterogeneous testbed: Convey HC-1 + GPU



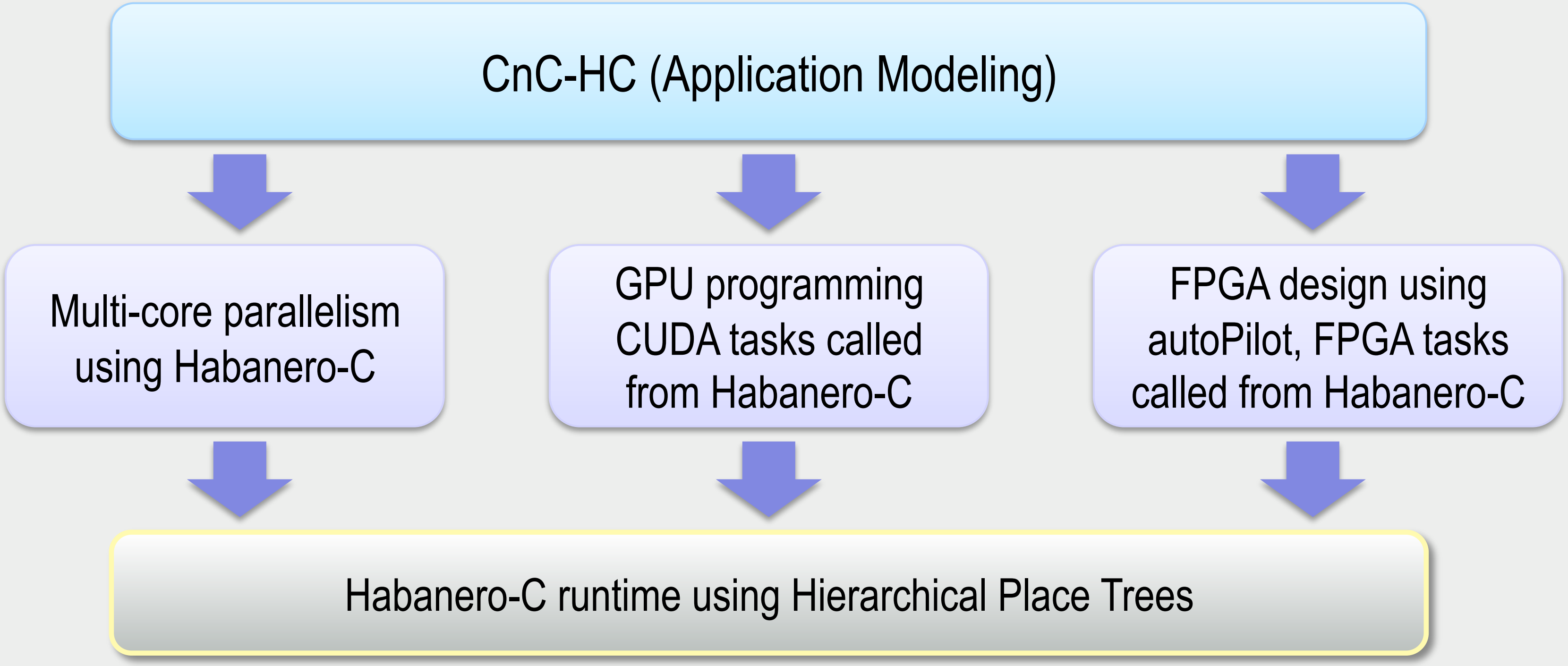
Xeon Dual Core LV5138
35W TDP

XC5LX330 FPGAs
80GB/s off-chip bandwidth
90W Design Power

Tesla C1060
100GB/s off-chip bandwidth
200W TDP

NSF Expeditions Center for Domain-Specific Computing (CDSC)
http://cdsc.ucla.edu

Toolchain for Server-class CHP testbed



CnC-HC (Application Modeling)

Multi-core parallelism using Habanero-C

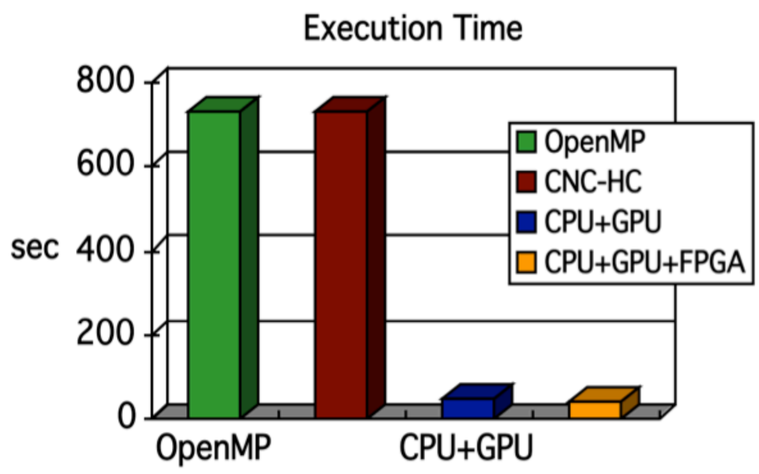
GPU programming CUDA tasks called from Habanero-C

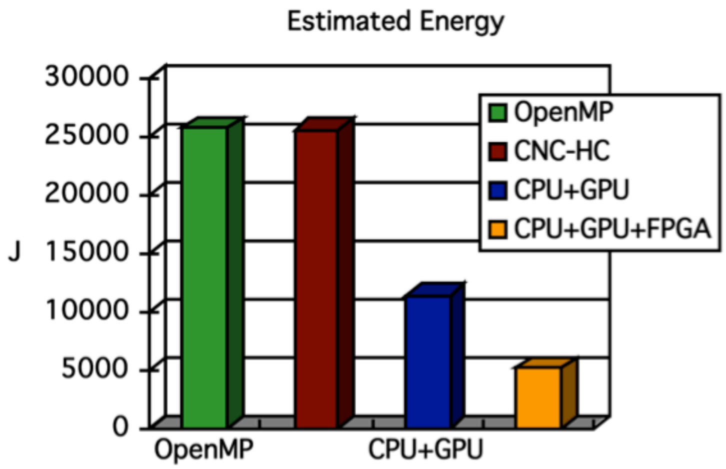
FPGA design using autoPilot, FPGA tasks called from Habanero-C

Habanero-C runtime using Hierarchical Place Trees

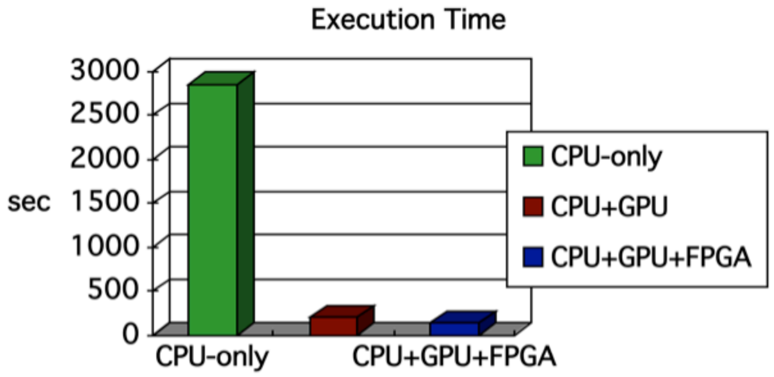
Experimental Results on Convey HC-1 testbed

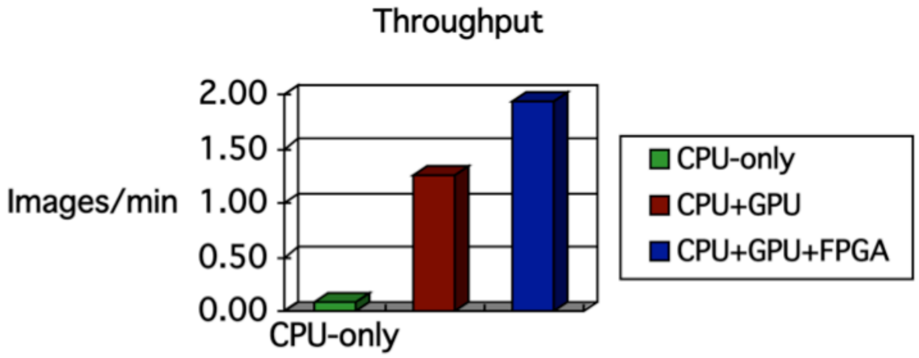
- **Pipeline: Denoise-->Registration (200 iterations)-->Segmentation (100 iterations)**





- **Multi-images (4 images)**

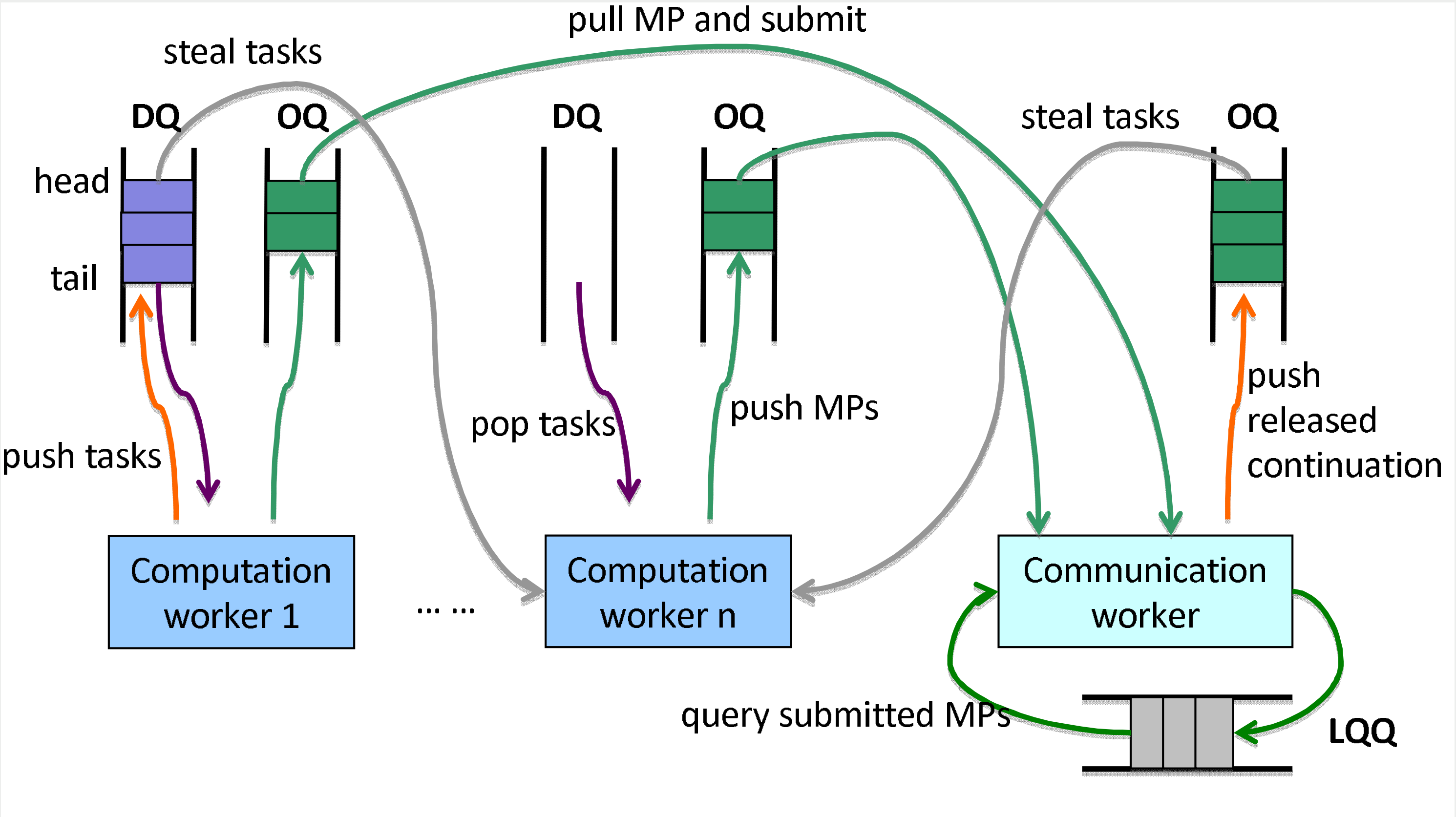




Bridging the Gap between Evolutionary and Revolutionary Solutions

- Exascale will cause a larger disruption at the intra-node level than at the inter-node level
- Ideal solution would be to design an integrated programming model from scratch … but can we also bridge between revolutionary intra-node programming models and evolutionary inter-node programming models?
- One possible approach
  - Intra-node runtime system with computation and communication workers
    - Communication subsystem only interacts with communication workers
  - Computation tasks execute on computation workers to support intra-node programming constructs
  - Computation tasks dispatch communication tasks on communication workers
  - Termination of communication tasks may unblock some computation tasks
  - This approach can also be extended for reductions/collectives

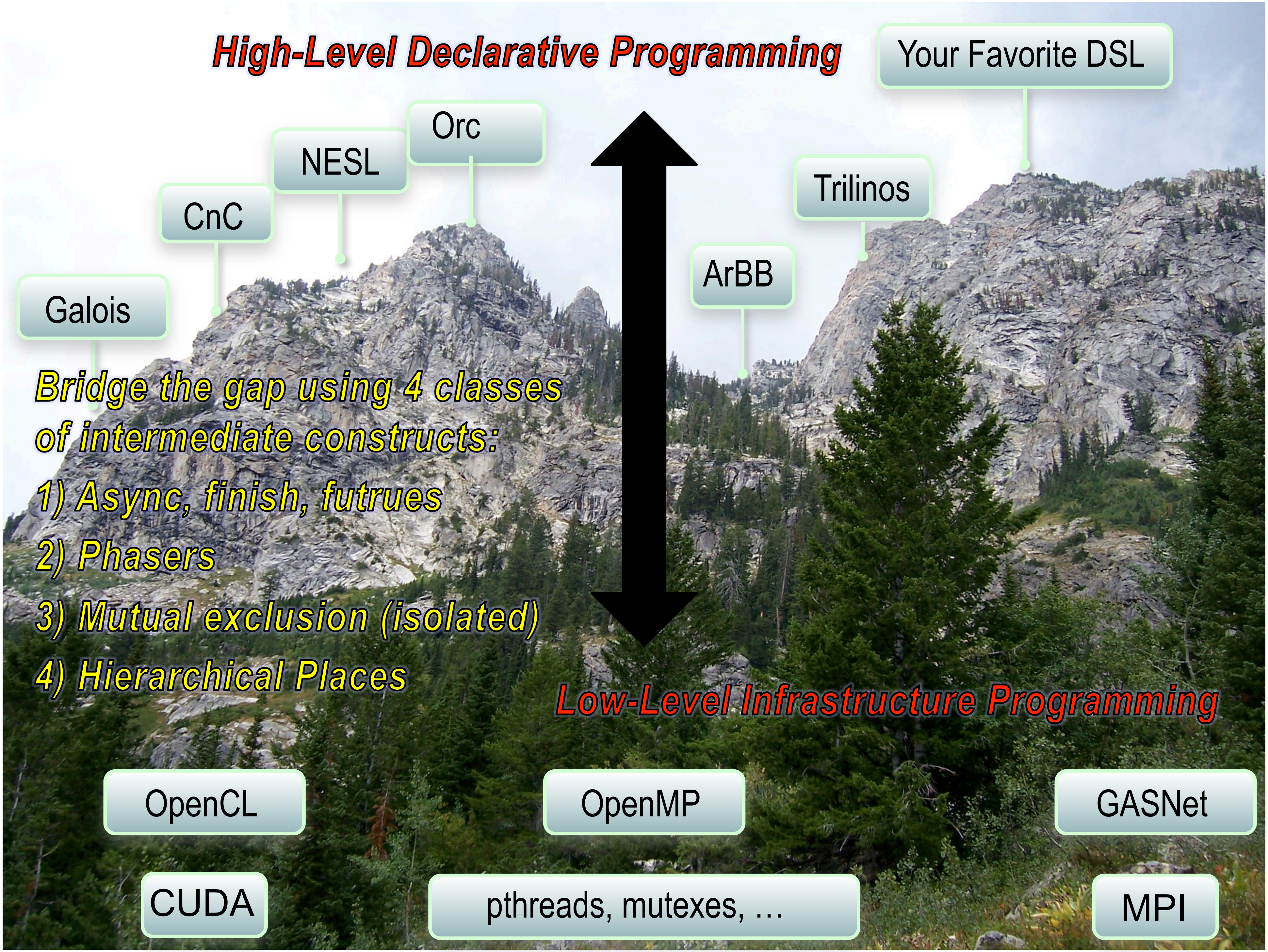
Prototype Integration of Habanero-C Computation Workers with MPI Communication Workers



DQ: double-ended queue (deque), **OQ**: outbox queue, **LQQ**: local query queue

Similar prototypes also in progress for shmem and GASNet

"Integrating MPI with Asynchronous Task Parallelism", Poster abstract, EuroMPI 2011.
Yonghong Yan, Sanjay Chatterjee, Zoran Budimlic, and Vivek Sarkar.



High-Level Declarative Programming

Your Favorite DSL

Orc

NESL

CnC

Galois

Trilinos

ArBB

Bridge the gap using 4 classes of intermediate constructs:

1) Async, finish, futrues

2) Phasers

3) Mutual exclusion (isolated)

4) Hierarchical Places

Low-Level Infrastructure Programming

OpenCL

OpenMP

GASNet

CUDA

pthreads, mutexes, …

MPI