George R. Brown School of Engineering
Computer Science

# Hybrid Programming Challenges for Extreme Scale Software

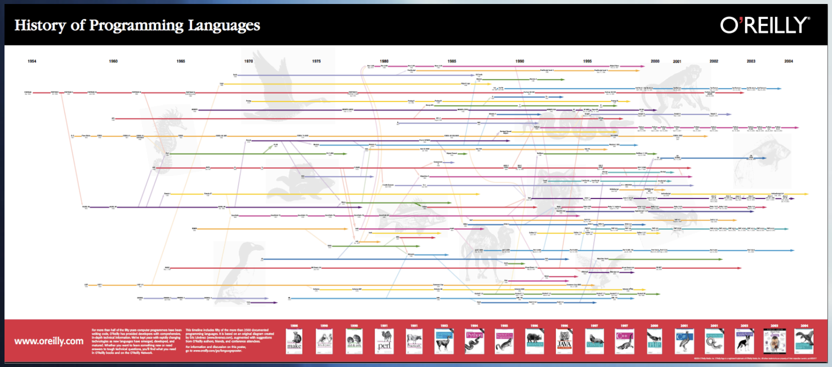

**Vivek Sarkar**

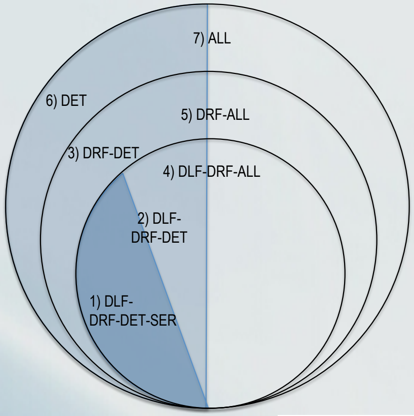
E.D. Butcher Chair in Engineering

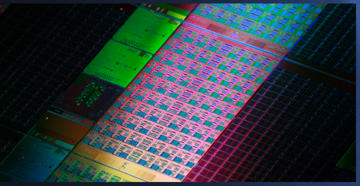
Professor of Computer Science

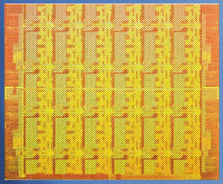
Rice University

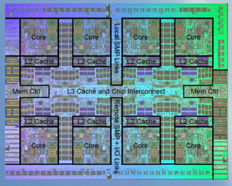
vsarkar@rice.edu

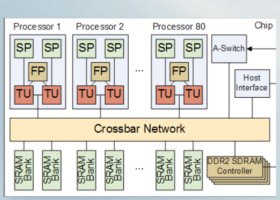



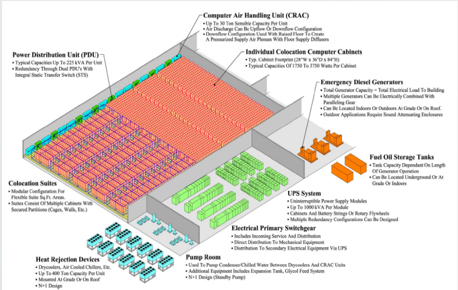









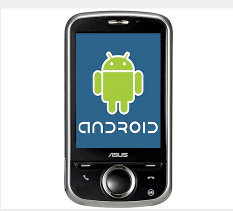
# Acknowledgments --- Habanero Extreme Scale Software Group

- Faculty
  - Vivek Sarkar
- Senior Research Scientist
  - Michael Burke
- Research Scientists
  - Zoran Budimlić, Philippe Charles, Vivek Kumar, Jun Shirako, Jisheng Zhao
- Research Programmer
  - Vincent Cavé
- Postdoctoral Researchers
  - Akihiro Hayashi
- PhD Students
  - Kumud Bhandari, Shams Imam, Deepak Majeti, Alina Sbîrlea, Dragoș Sbîrlea, Kamal Sharma, Rishi Surendran, Sağnak Taşırlar, Nick Vrvilo, Yunming Zhang
- Undergraduate Students
  - Kyle Kurihara, Bing Xue
- Supported in part by the National Science Foundation, DoD Advanced Computing Initiative (ACI) program, and by the X-Stack program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR)

RICE

# Multicore Processors and Extreme Scale Systems

- Characteristics of Extreme Scale systems in the next decade
  - *Massively multi-core --- 1000+ homogeneous/heterogeneous cores per node*
  - *Performance driven by parallelism, constrained by energy*
  - *Subject to frequent faults and failures*
- Many Classes of Extreme Scale Systems



***Mobile, < 10 Watts, $O(10^1)$ concurrency***

***Embedded, 100's of Watts, $O(10^3)$ concurrency***

***Departmental, 100's of KW, $O(10^6)$ concurrency***

***Data Center > 1 MW, $O(10^9)$ concurrency***

Key Challenges

- ***Energy Efficiency***
- ***Concurrency***
- ***Resiliency***

**References:**


- **DARPA Exascale Software study, V. Sarkar et al, Sep 2009**
- **"Software Challenges in Extreme Scale Systems". V. Sarkar, W. Harrod, A.E. Snavely. SciDAC Review, January 2010.**

# What is “Hybrid Programming”?

Zonkey

Liger

Jaglion

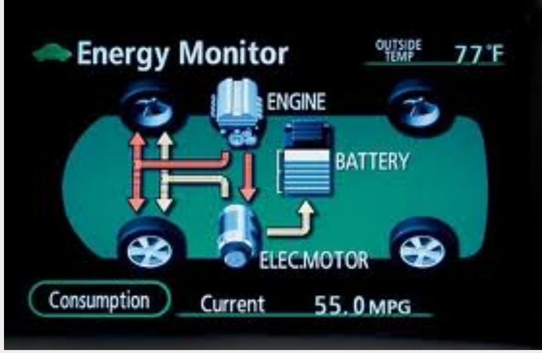


Image source: http://en.wikipedia.org/wiki/Hybrid_(biology)

# Observation: definition of "Hybrid" depends on your starting point

- If your starting point is a bulk-synchronous SPMD program with one thread per rank, then "hybridizations" have to be implemented as special-case extensions, e.g.,
  - Asynchronous data movements across ranks
  - Task parallelism within a rank
  - Accelerator parallelism
  - Task/process cancellation and migration
  - . . .

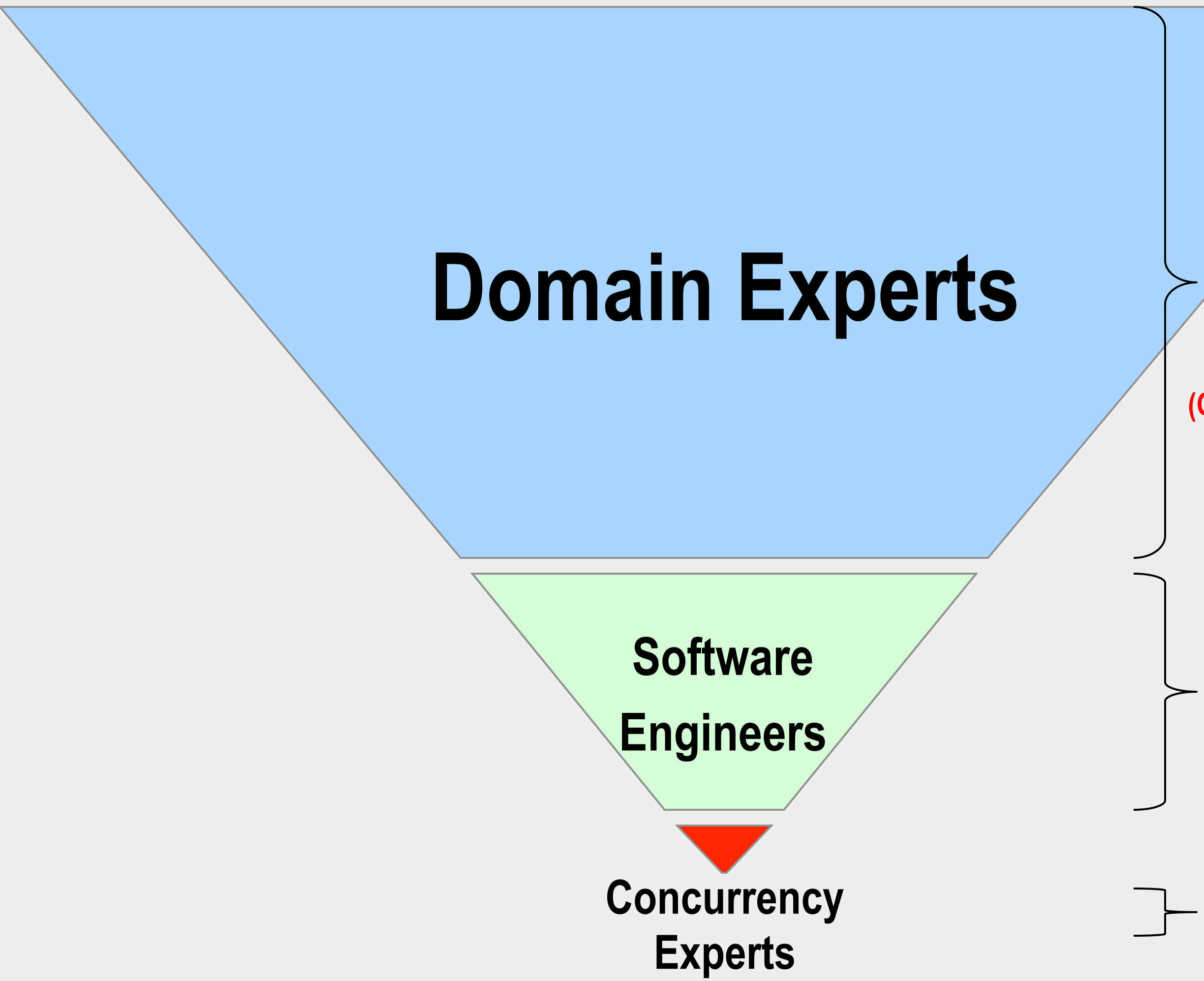
# Alternate Approach: Hybrid by Design




- If your starting point is a general unified execution model and runtime system for extreme scale computing, then "hybridizations" are simply combinations of features, e.g.,
  - Integration of task parallelism and message passing
  - Integration of fork-join and point-to-point synchronization
  - Integration of actors and collectives
  - . . .

RICE

# Programmability Challenge --- Bridging the Expertise Gap between Domain Experts and Concurrency Experts




**Domain Experts need high level parallelism-*oblivious* Problem Solving Environments**

**Concurrent Collection dialects (CnC-C, CnC-Matlab, CnC-Python, CnC-Java, CnC-Scala)**

***Focus of Rice Habanero Project***

***Software Engineers need mid-level Parallel Programming Models with safety nets***

**Habanero Execution Model in C, C++, Java, Scala (HC, HJ, HS)**

**Concurrency Experts use low-level Parallel Programming Models**

# Rice Habanero Multicore Software Project: Enabling Technologies for Extreme Scale

Parallel Applications

**Portable execution model**

1) Lightweight asynchronous tasks and data transfers

- Creation: *async tasks, future tasks, data-driven tasks*
- Termination: *finish, future get, await*
- Data Transfers: *asyncPut, asyncGet*

2) Locality control for task and data distribution

- Computation and Data Distributions: *hierarchical places, global name space*

3) Inter-task synchronization operations

- Mutual exclusion: *isolated, actors*
- Collective and point-to-point operations: *phasers, accumulators*

Habanero Programming Languages

Habanero Static Compiler & Parallel Intermediate Representation

Habanero Runtime System

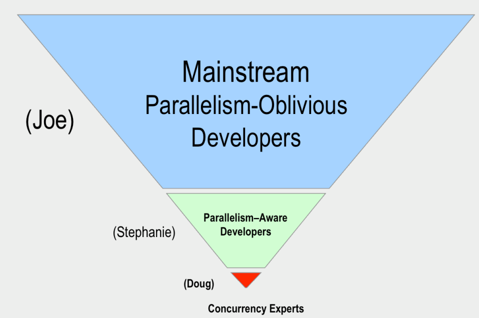
**Two-level programming model**

Declarative Coordination Language for Domain Experts:

CnC-HC, CnC-Java, CnC-Python, CnC-Matlab, … +

Task-Parallel Languages for Parallelism-aware Developers:

Habanero-C, Habanero-Java, Habanero-Scala




Extreme Scale Platforms

# Target Platforms

Habanero programs have been executed on a wide range of production and experimental systems

- Multicore SMPs (IBM, Intel)
- Discrete GPUs (AMD, NVIDIA)
- Integrated GPUs (AMD, Intel)
- FPGA (Convey, w/ GPU added)
- HPC Clusters
- Hadoop Clusters
- Experimental processors: IBM Cyclops, Intel SCC
- . . .

# Elements of Habanero Execution Model

**1) Lightweight asynchronous tasks and data transfers**

- **Creation:** ***async tasks, future tasks, data-driven tasks***
- **Termination:** ***finish, future get, await***
- **Data Transfers:** ***asyncPut, asyncGet***

**2) Locality control for control and data distribution**

- **Computation and Data Distributions:** ***hierarchical places, global name space***

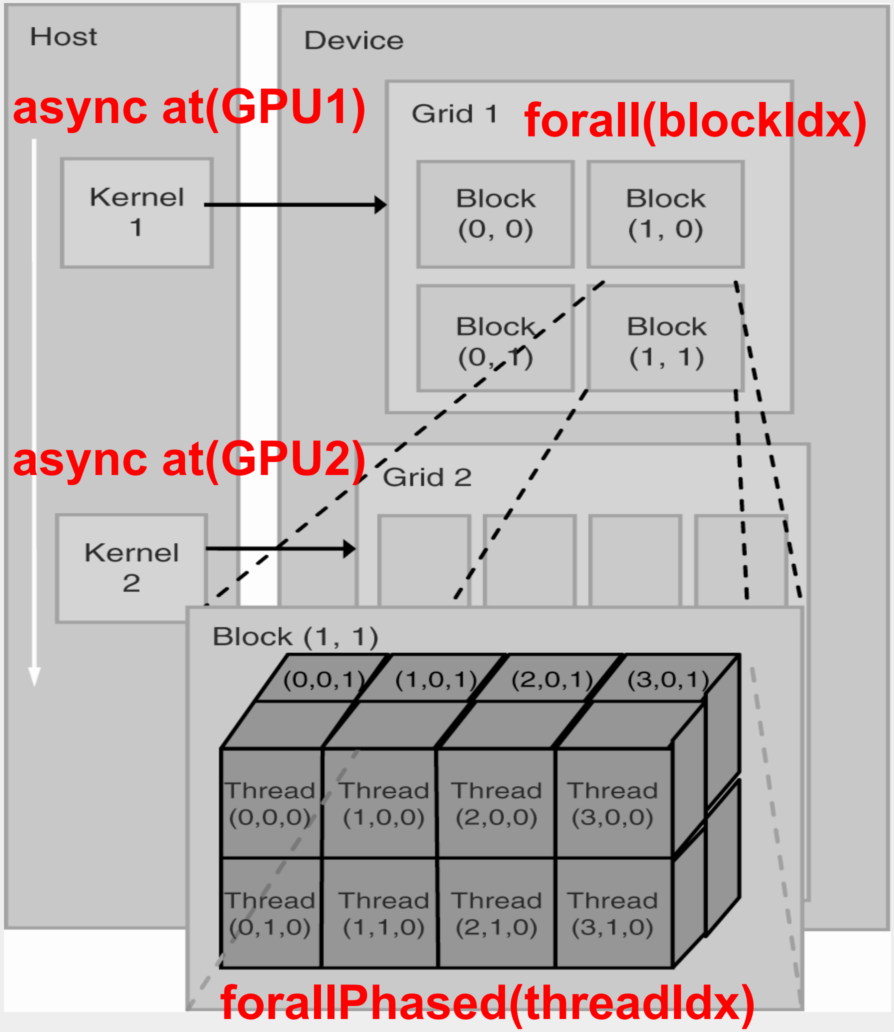
**3) Inter-task synchronization operations**

- **Mutual exclusion:** ***global/object-based isolation, actors***
- **Collective and point-to-point operations:** ***phasers, accumulators***

***Goal: unified model of parallelism that spans a wide range of extreme scale platforms***

RICE

# Example: Habanero abstraction of a CUDA kernel invocation

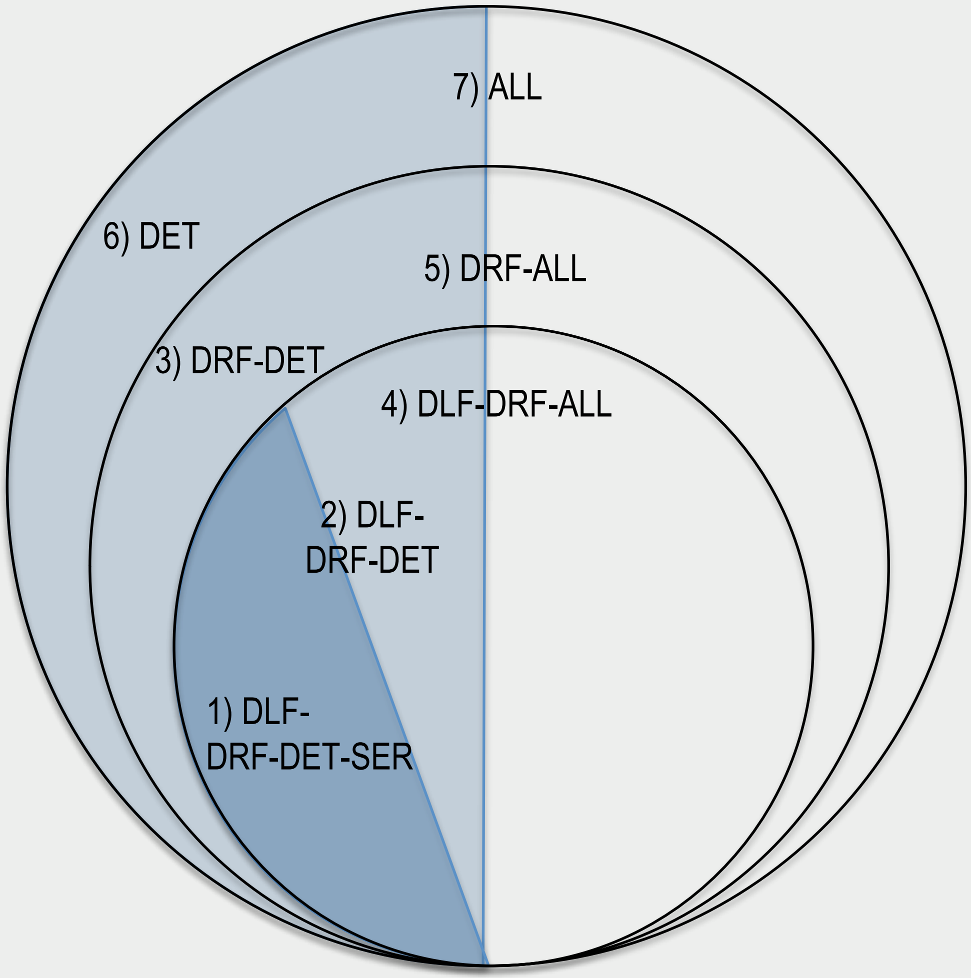



RICE

# Properties of Habanero Execution Model

- Deadlock freedom guarantee for large subset of operations
  - All operations except explicit wait in phasers, accumulators and explicit await clause in async
- Data-race freedom guarantee for subset of data accesses
  - Future values, accumulator values
  - Read-write permission regions
  - Isolated accesses, actors
- Determinacy guarantee for subset of programs
  - Data-race freedom implies determinacy for all programs that do not use mutual exclusion constructs (isolated, actors)
- Amenable to efficient asynchronous and portable implementations
  - Locality-aware work-stealing
  - Hierarchical places with support for heterogeneous processors
  - Integration with cluster-level communication runtime systems
  - Scalable synchronization with phasers, accumulators and delegated isolation
  - Compiler optimizations for structured parallelism

# Semantic Classification of Habanero Parallel Programs




- Legend
  - DLF = DeadLock-Free
  - DRF = Data-Race-Free
  - DET = Determinate
  - DRF➔DET = DRF implies DET
  - SER = Serializable
- *If a Habanero program only uses async, finish, and future constructs (no mutual exclusion), then it is guaranteed to belong to the DLF + DRF➔DET + SER class*
- *Adding phasers yields programs in the DLF + DRF➔DET class*
- *Adding async await yields programs in the DRF➔DET class*
- *Restricting shared data accesses to futures, isolated, actors yields programs in the DRF-ALL class*

RICE

"Habanero-Java: the New Adventures of Old X10." Vincent Cave, Jisheng Zhao, Jun Shirako, Vivek Sarkar PPPJ 2011.

# Pedagogy using Habanero execution model, COMP 322: Fundamentals of Parallel Programming

- **Sophomore-level CS Course at Rice**
  - https://wiki.rice.edu/confluence/display/PARPROG/COMP322
- **Approach – mid-level parallel programming model**
  - **"Simple things should be simple, complex things should be possible"**
  - Introduce students to fundamentals of parallel programming
    - Primitive constructs for task creation & termination, collective & point-to-point synchronization, task and data distribution, and data parallelism
    - Abstract models of parallel computations and computation graphs
    - Parallel algorithms & data structures including lists, trees, graphs, matrices
    - Common parallel programming patterns
  - Use Habanero-Java (HJ) library for Java 8 as pedagogic programming model to understand fundamentals in two-thirds of course, and then introduce students to lower-level parallel programming models (Java threads, MPI, CUDA) using HJ principles
  - Video lectures and demos are available as well

RICE

# 1) Primitives for Lightweight Asynchronous Tasks

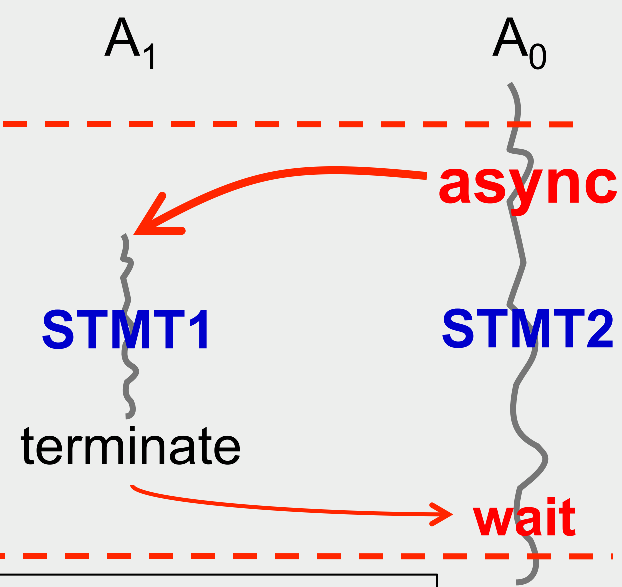
**async S**

- Creates a new child task that executes statement S
  - Like OpenMP's task pragma
- Parent task moves on to statement following the async
- *async can be a computation or a communication task*

**finish S**

- Execute S, but wait until *all* (transitively) spawned asyncs in S's scope have terminated
  - Like OpenMP's taskwait
- Implicit finish between start and end of main program
- Use of finish cannot create a deadlock cycle

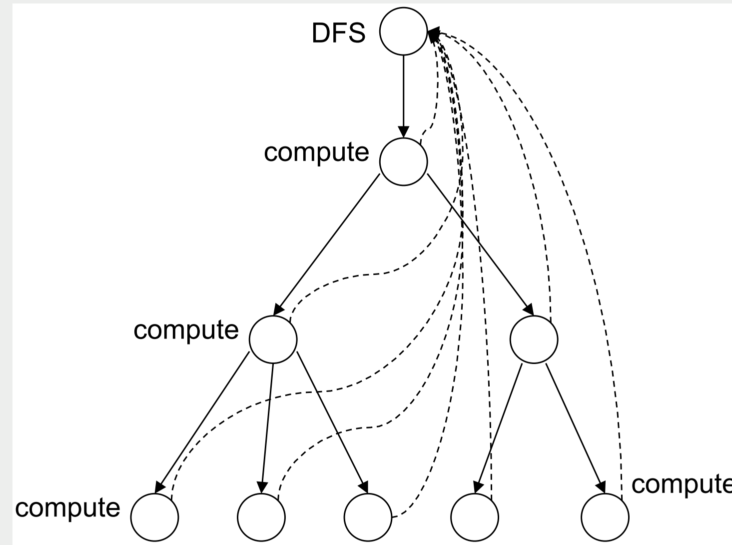
```
//A0 (Parent)
finish {    //Begin finish
  async {
    STMT1; //A1 (Child)
  }
  STMT2;    //A0
}           //End finish
```

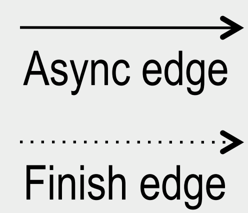



"X10: An Object-oriented approach to non-uniform Clustered Computing", P.Charles et al. OOPSLA 2005.

# Parallel Spanning Tree Algorithm using Habanero-Java Library

```
class V  {
  V [] neighbors; // Input adjacency list
  V parent; // Output spanning tree
  . . .
  boolean tryLabeling(final V n) {
    return isolatedWithReturn(this, () -> {
      if (parent == null) parent = n;
      return parent == n;
    });
  } // tryLabeling
  void compute() {
    for (int i=0; i<neighbors.length; i++) {
      final V child = neighbors[i];
      if (child.tryLabeling(this))
          async(()->{child.compute()}); //escaping async
     }
  } // compute
} // class V
   . . .
root.parent = root; //Use self-cycle to identify root
finish(()->{root.compute()});
```



# Data-Driven Futures (DDFs) and Data-Driven Tasks (DDTs) in Habanero-C language

**`DDF_t* ddfA = DDF_CREATE();`**

- Allocate an instance of a data-driven-future object (container)

**`async AWAIT(ddfA, ddfB, …) <Stmt>`**

- Create a new data-driven-task to start executing Stmt after all of ddfA, ddfB, … become available (i.e., after task becomes "enabled")

**`DDF_PUT(ddfA, V);`**

- Store object V in ddfA, thereby making ddfA available
- Single-assignment rule: at most one put is permitted on a given DDF

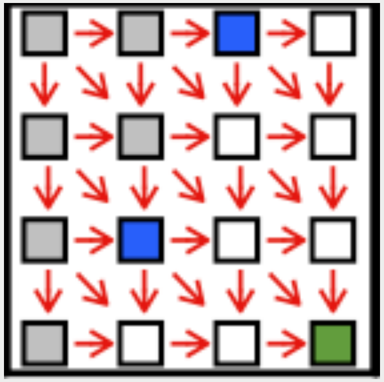
**`DDF_GET (ddfA)`**

- Return value stored in ddfA
- No blocking needed --- should only be performed by tasks that contain ddfA in their AWAIT clause, or when some other synchronization (e.g., finish) guarantees that DDF_PUT must have been performed.

DDFs and DDTs can be more efficient than OpenMP regions and barriers

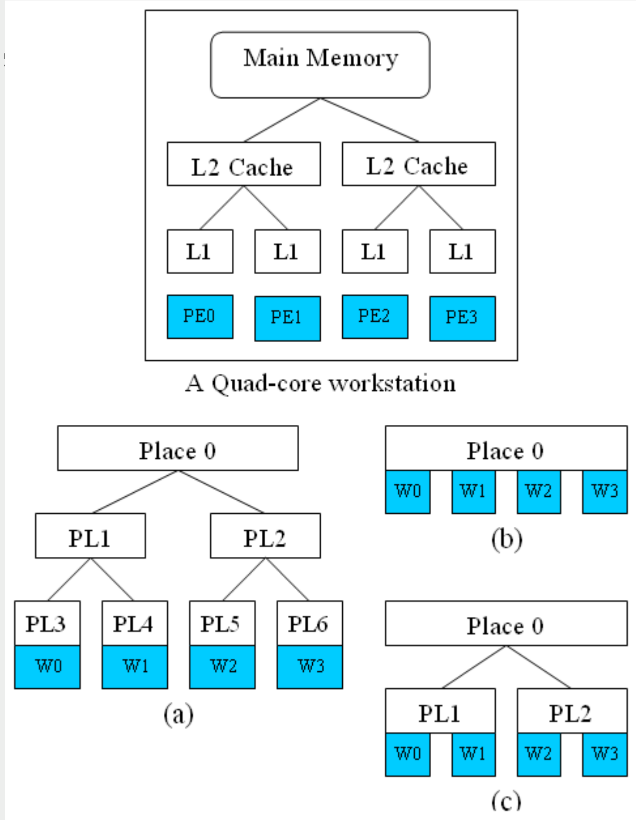
# Smith Waterman example with DDFs (Habanero-C)

```
finish { // matrix is a 2-D array of DDFs
  for (i=0,i<H;++i) {
    for (j=0,j<W;++j) {
      DDF_t* curr = matrix[i][j];
      DDF_t* above = matrix[i-1][j];
      DDF_t* left = matrix[i][j-1];
      DDF_t* uLeft = matrix[i-1][j-1];
      async AWAIT (above, left, uLeft){
          Elem* currElem =
            init(DDF_GET(above),DDF_GET(left), DDF_GET(uLeft));
          compute(currElem);
          DDF_PUT(curr, currElem);
      }/*async*/
    }/*for-j*/
  }/*for-i*/
}/*finish*/
```



# 2) Locality control for task and data distribution: Hierarchical Place Trees (HPT) abstraction

- HPT approach
  - Hierarchical memory + Dynamic parallelism
- Place denotes affinity group at memory hierarchy level
  - L1 cache, L2 cache, CPU memory, GPU memory, …
- Leaf places include worker threads
  - e.g., W0, W1, W2, W3
- Explore multiple HPT configurations
  - For same hardware and application
  - Trade-off between locality and load-balance

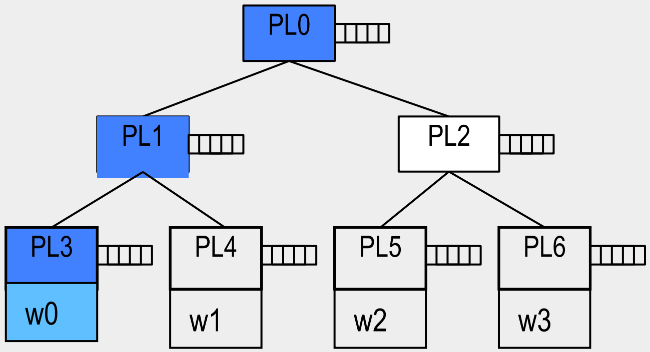



“Hierarchical Place Trees: A Portable Abstraction for Task Parallelism and Data Movement”, Y.Yan et al, LCPC 2009

RICE

# Locality-aware Scheduling using the HPT

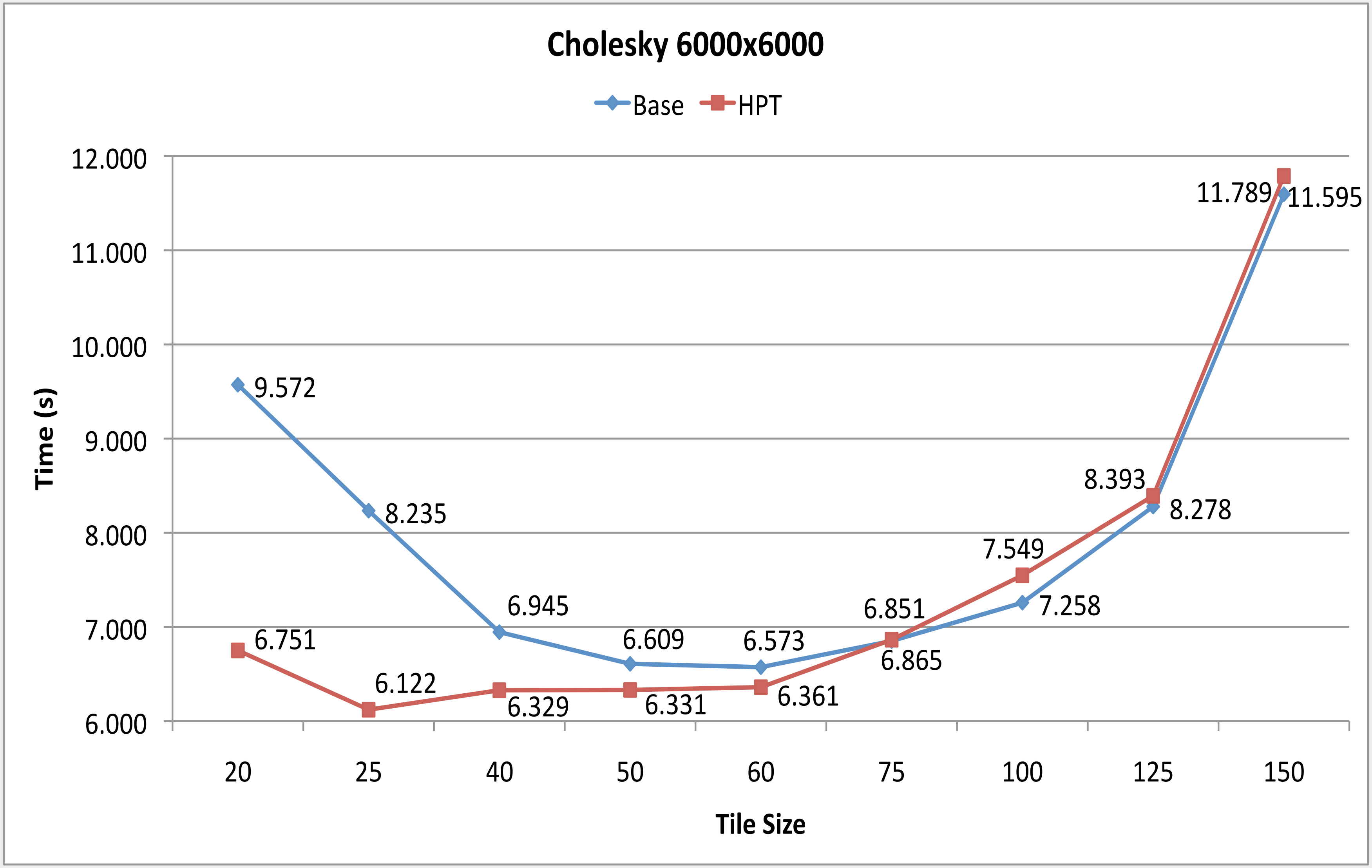
- Workers attached to leaf places
  - Bind to hardware core
- Each place has a queue
  - **async at**(<*pl*>) <*stmt*>: push task onto place *pl*'s queue




- A worker executes tasks from ancestor places from bottom-up
  - W0 executes tasks from PL3, PL1, PL0
- Tasks in a place queue can be executed by all workers in the place's subtree
  - Task in PL2 can be executed by workers W2 or W3

RICE

# Example: Cholesky Performance with HPT (12-core SMP)




*Reference: Runtime Systems for Extreme Scale Platforms. Sanjay Chatterjee. Ph.D Thesis, December 2013*

RICE

# LULESH with place annotation (can be selected by programmer, compiler, runtime)

```
finish {
Index_t i_len = numNode;
Index_t i_blk = HAB_C_BLK_SIZE;
int blk_per_child = (int)(i_len/num_children);
for (Index_t i_out = 0; i_out < i_len; i_out += i_blk) {
    Index_t i_end = ((i_out + i_blk) < i_len)?(i_out + i_blk) : i_len;
    place p = myAffinity(i_out, i_end);
    async at(p) {
        for(Index_t gnode = i_out ; gnode < i_end ; ++gnode ) {
         int xDir = 0;
         int yDir = 1;
         int zDir = 2;
        …
        }
    }
}
```

Reuse takes places across different loops in different functions

# LULESH Results w/ and w/o use of places in HPT

Timing in seconds on Intel Westmere (2x6cores) for 12 Threads with gcc –O3

| | w/o HPT | w/ HPT |
|---|---|---|
| LULESH (Problem Size=45) | 21.45 secs | 19.08 secs |

Hardware Performance Counter Ratio

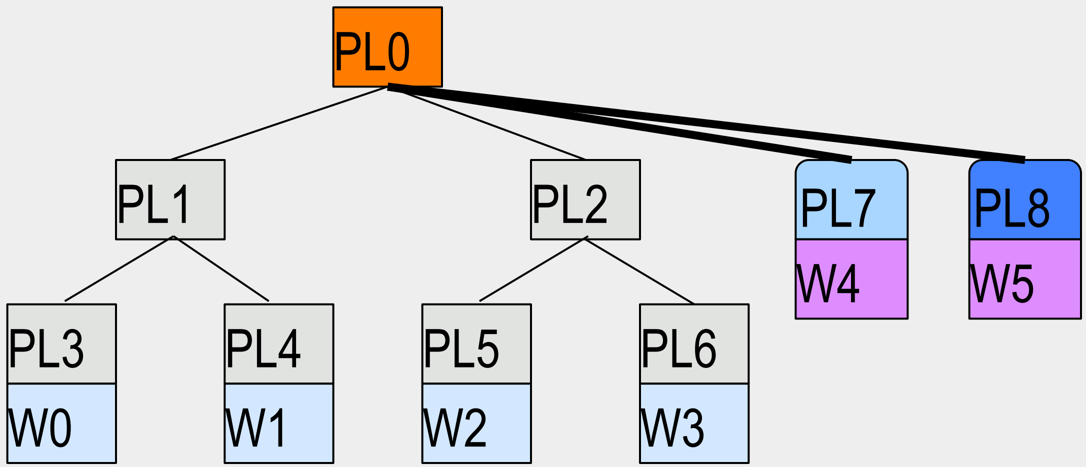
| Hardware Perf Counters | L1DCM | L2DCM | L3TCM | TLBDM |
|---|---|---|---|---|
| HC/HPT | 0.97 | 1.30 | 1.50 | 0.90 |

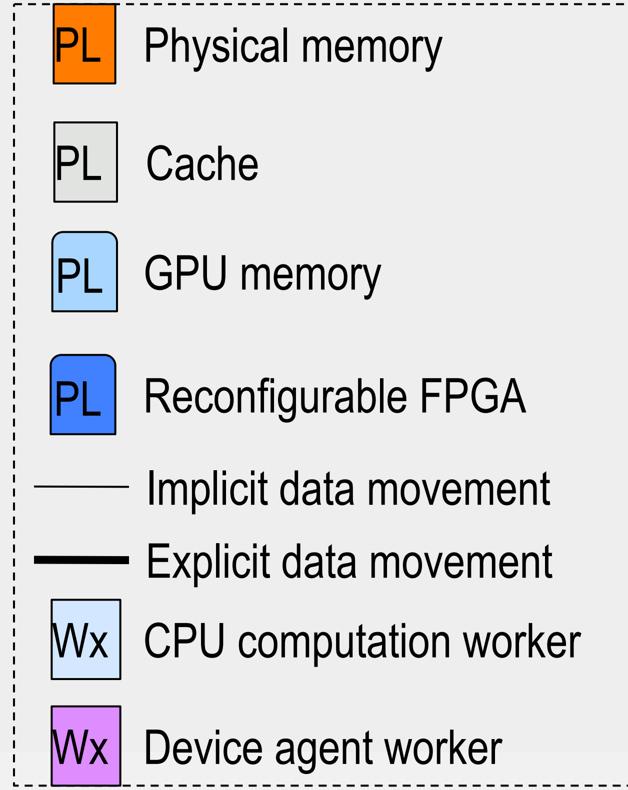
DCM: Data Cache Misses, DCA: Data Cache Accesses,

TCM: Total Cache Misses (Inst+Data),TLBDM: TLB Misses

In progress: figuring out why current HPT implementation decreases cache misses but increases TLB misses

# Habanero Hierarchical Place Trees for heterogeneous architectures and accelerators




- **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**
- **Explicit data transfer between main memory and device memory when needed**
  - **Use of copyin/copyout clauses to improve programmability of data transfers**
- **Device agent workers**
  - **Perform asynchronous data copy and task launching for device**

# Medical imaging application (Center for Domain-Specific Computing)

- New reconstruction methods
  - decrease radiation exposure (CT)
  - number of samples (MR)
- 3D/4D image analysis pipeline
  - Denoising
  - Registration
  - Segmentation
- Analysis
  - Real-time quantitative cancer assessment applications
- Potential:
  - order-of-magnitude performance improvement
  - power efficiency improvements
  - real-time clinical applications and simulations using patient imaging data

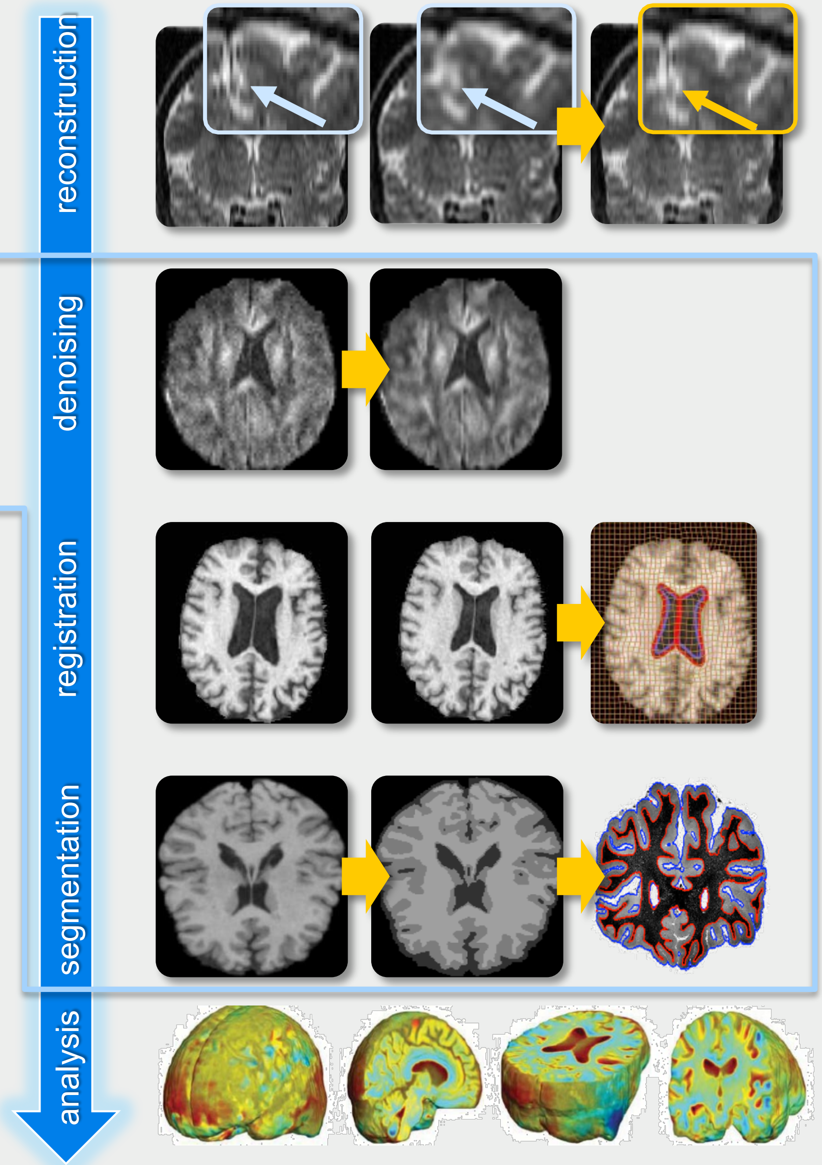
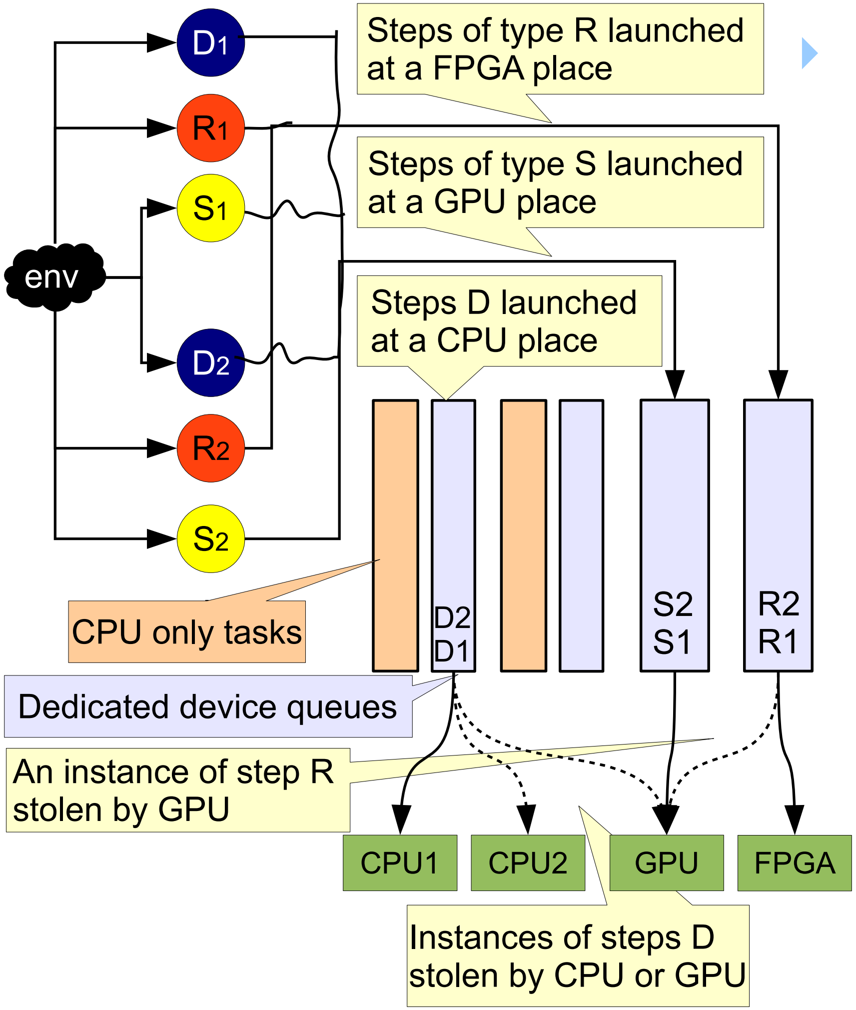



Figure credit: NSF Expeditions CDSC project

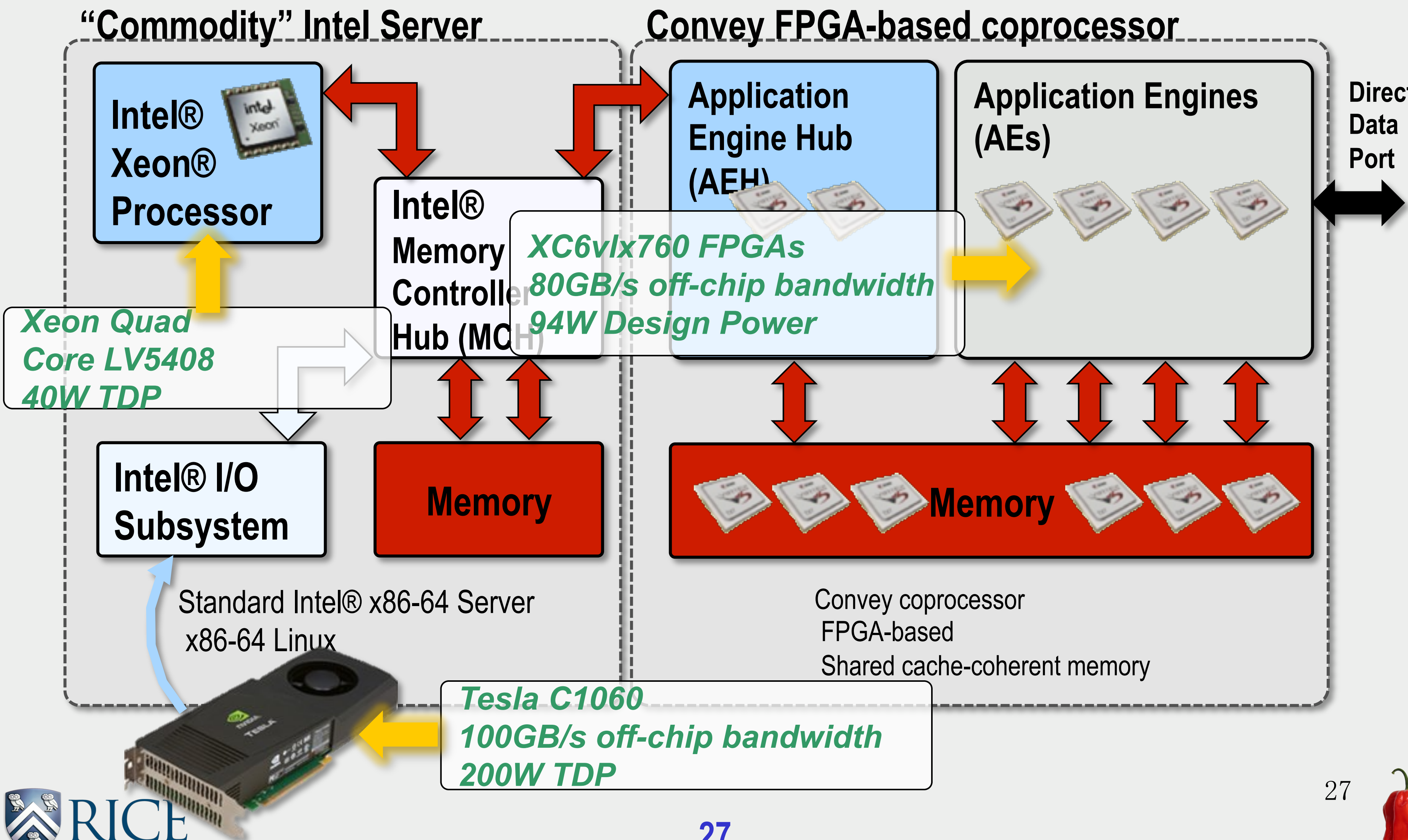
# Adding Affinity Annotations for Heterogeneous Computing




- CnC graph representation extended with tag functions and affinity annotations:
  - < C > :: ( D @CPU=20,GPU=10);
  - < C > :: ( R @GPU=5, FPGA=10);
  - < C > :: ( S @GPU=12);
  - [ IN : k-1 ] → ( D : k ) → [ IN2 : k+1 ];
  - [ IN2 : 2*k ] → ( R : k ) → [ IN3 : k/2 ];
  - [ IN3 : k ] → ( S : k ) → [ OUT : IN3[k] ];
  - env → [ IN : { 0 .. 9 } ], < C : { 0 .. 9 } >;
  - [ OUT : 1 ] → env;

"Mapping a Data-Flow Programming Model onto Heterogeneous Platforms." Alina Sbirlea, Yi Zou, Zoran Budimlic, Jason Cong, Vivek Sarkar. LCTES 2012

# Convey HC-1ex Testbed

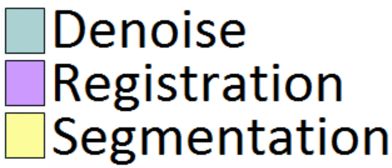


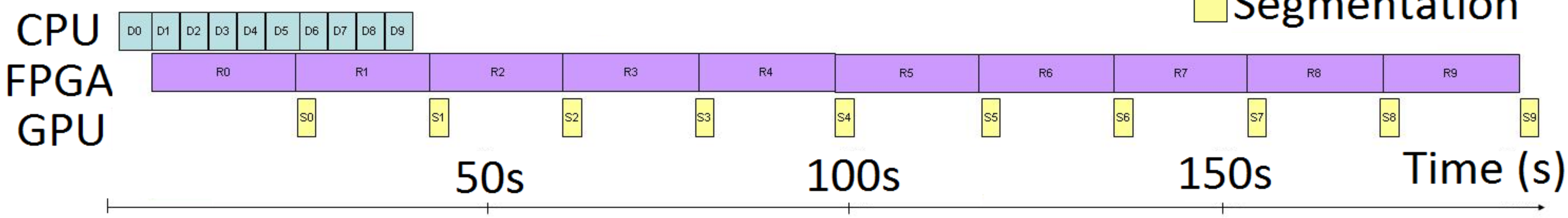

RICE

# Static vs Dynamic Scheduling

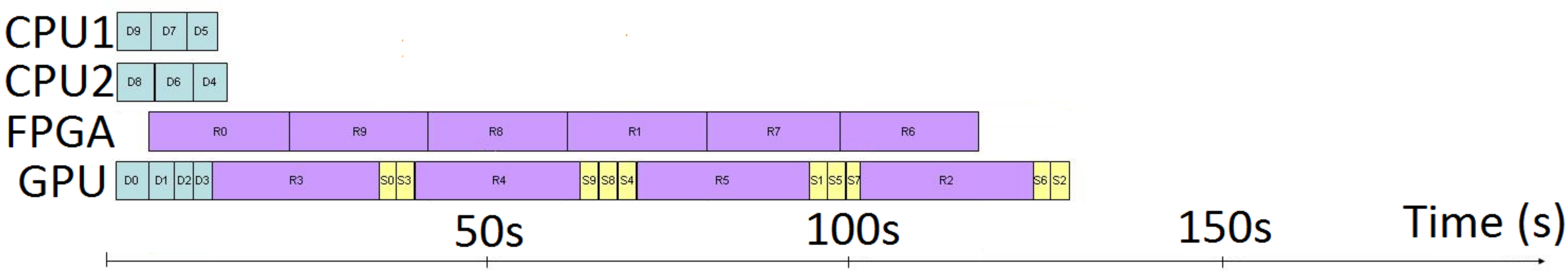
- < C > :: ( D @CPU=20,GPU=10);
- < C > :: ( R @GPU=5, FPGA=10);
- < C > :: ( S @GPU=12);

- Static Schedule



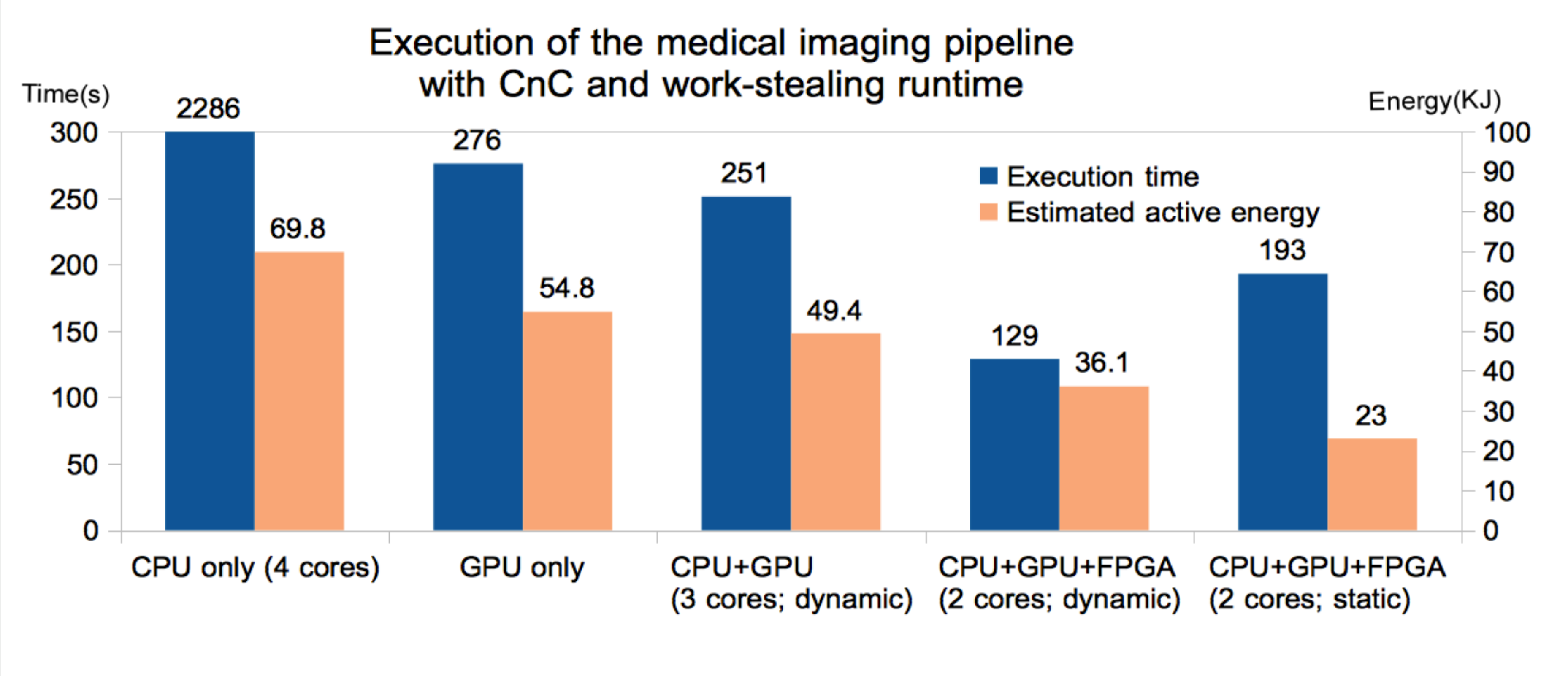




- Dynamic Schedule



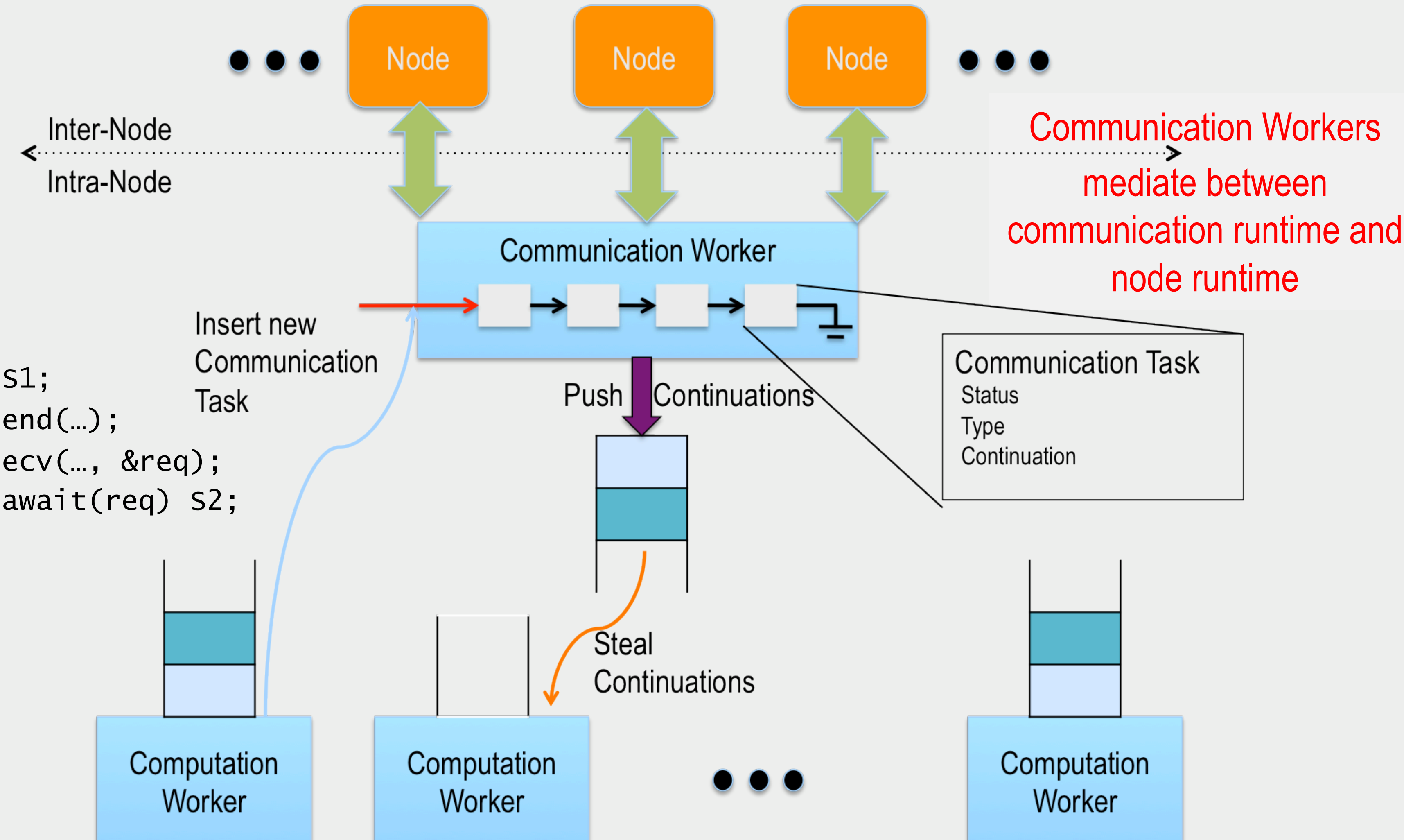
# Experimental results

- Execution times and active energy with dynamic work stealing




RICE

# Integrating Inter-node Communication with Intra-node Task Scheduling

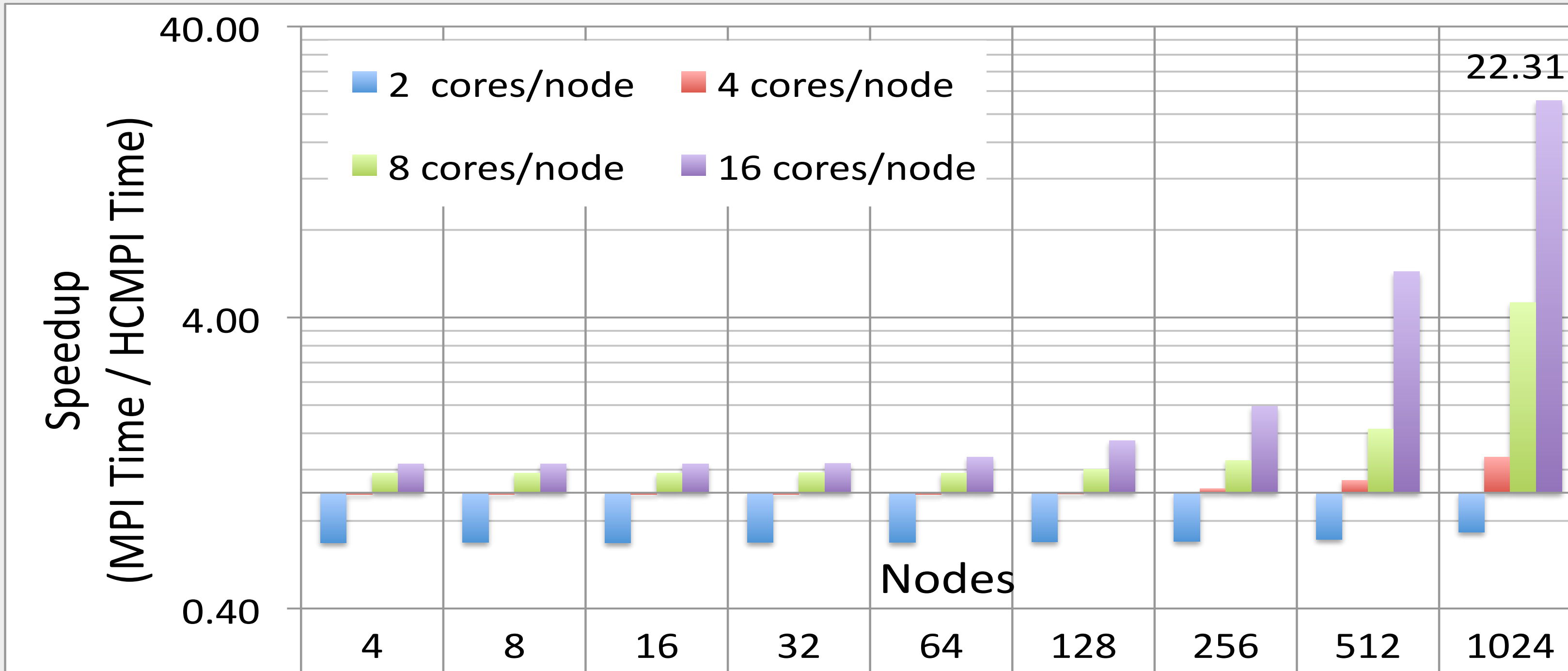



*Example:*

```
finish{
  async S1;
  MPI_Isend(…);
  MPI_Irecv(…, &req);
  async await(req) S2;
  S3;
}
...
```

“Integrating Asynchronous Task Parallelism with MPI.” Sanjay Chatterjee, Sagnak Tasirlar, Zoran Budimlic, Vincent Cave, Milind Chabbi, Max Grossman, Yonghong Yan, Vivek Sarkar. IPDPS 2013.

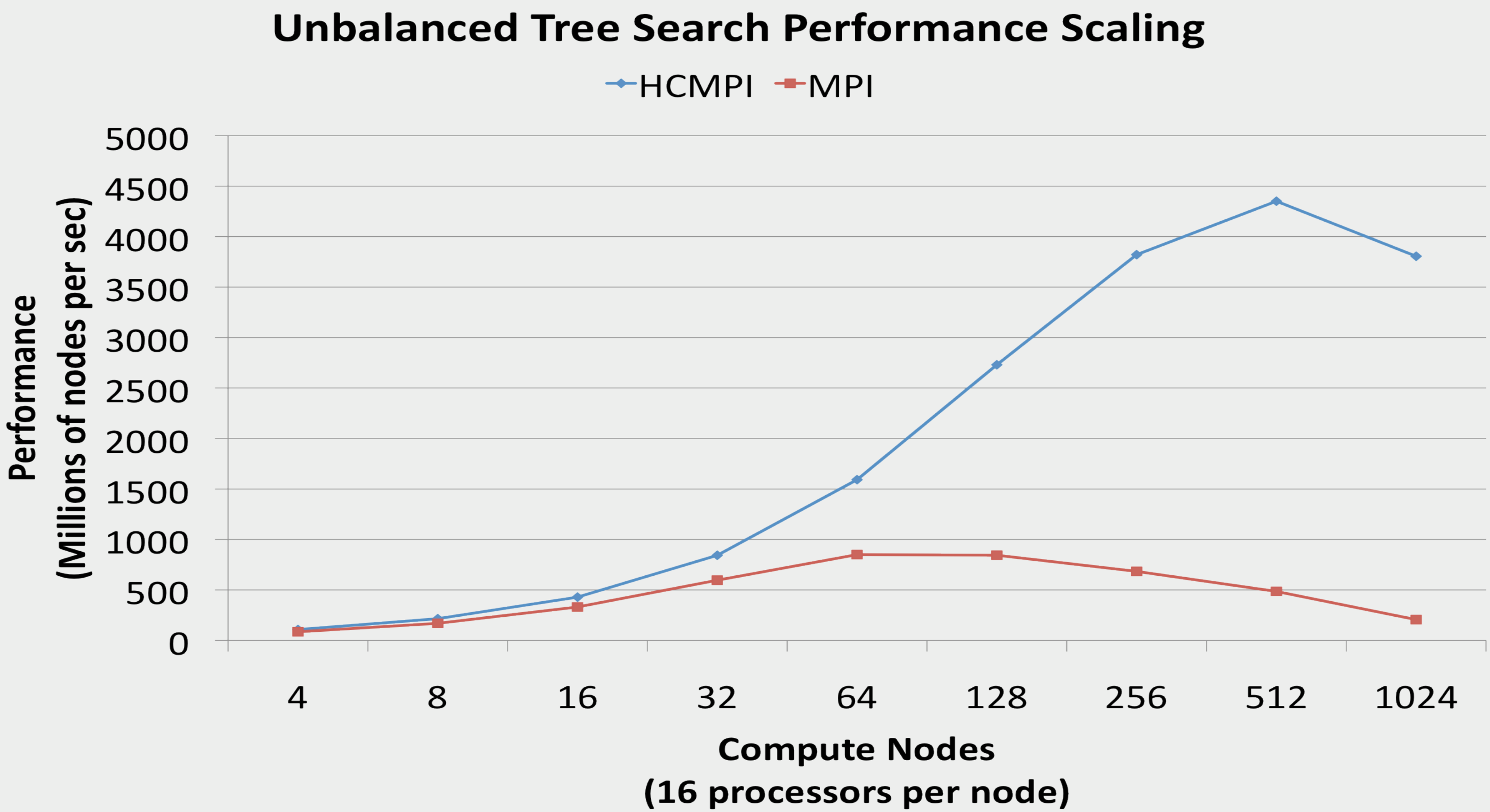
# UTS Performance on T1XXL




| | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|---|---|
| 2 cores/node | 0.67 | 0.67 | 0.67 | 0.67 | 0.67 | 0.68 | 0.68 | 0.69 | 0.73 |
| 4 cores/node | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.01 | 1.03 | 1.10 | 1.33 |
| 8 cores/node | 1.17 | 1.17 | 1.17 | 1.17 | 1.17 | 1.20 | 1.29 | 1.66 | 4.50 |
| 16 cores/node | 1.26 | 1.26 | 1.26 | 1.26 | 1.33 | 1.51 | 1.98 | 5.76 | 22.31 |

- Jaguar Supercomputer at ORNL
- 18688 nodes with Gemini Interconnect
- 16 core AMD Opteron nodes with 32 GB memory

# UTS Scaling on T1XXL



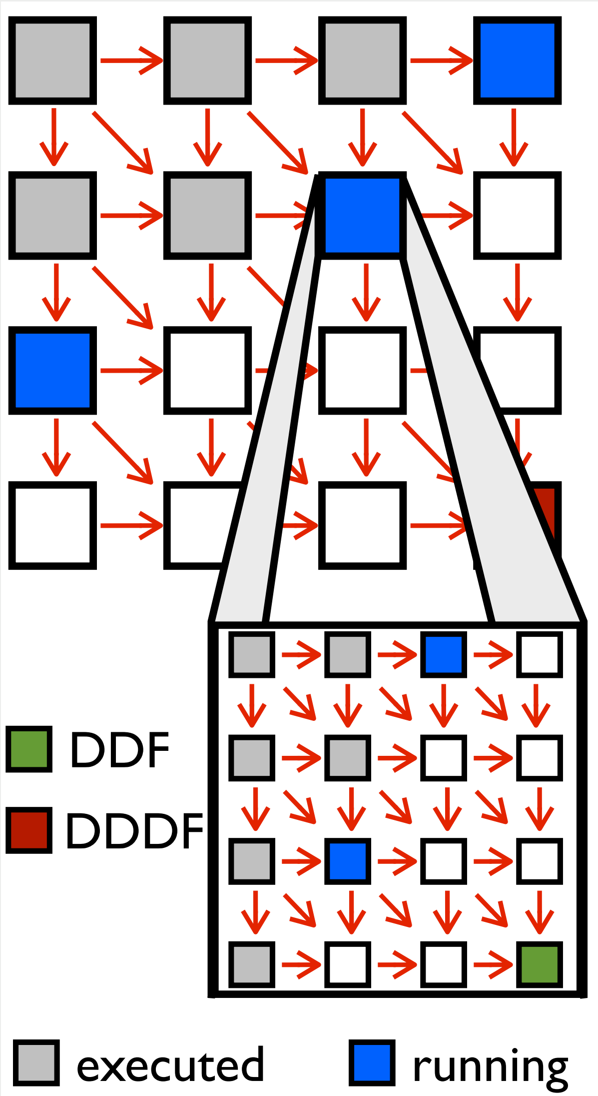

Failed steals lead to scalability bottleneck in MPI

- At 256 nodes: MPI suffers 2.35M failed steals while HCMPI suffers 0.82M
- At 1024 nodes: MPI suffers 94.75M failed steals while HCMPI suffers 8.83M

# APGNS Programming Model

- Philosophy :
  - In the Habanero Asynchronous Partitioned Global Name Space (APGNS) programming model, distributed tasks communicate via distributed data-driven futures, each of which has a globally unique id/name (guid).
  - Asynchronous one-sided communication model
  - APGNS can be implemented on a wide range of communication runtimes including MPI and GASNet, regardless of whether or not a global address space is supported.

# Multi-Node SmithWaterman

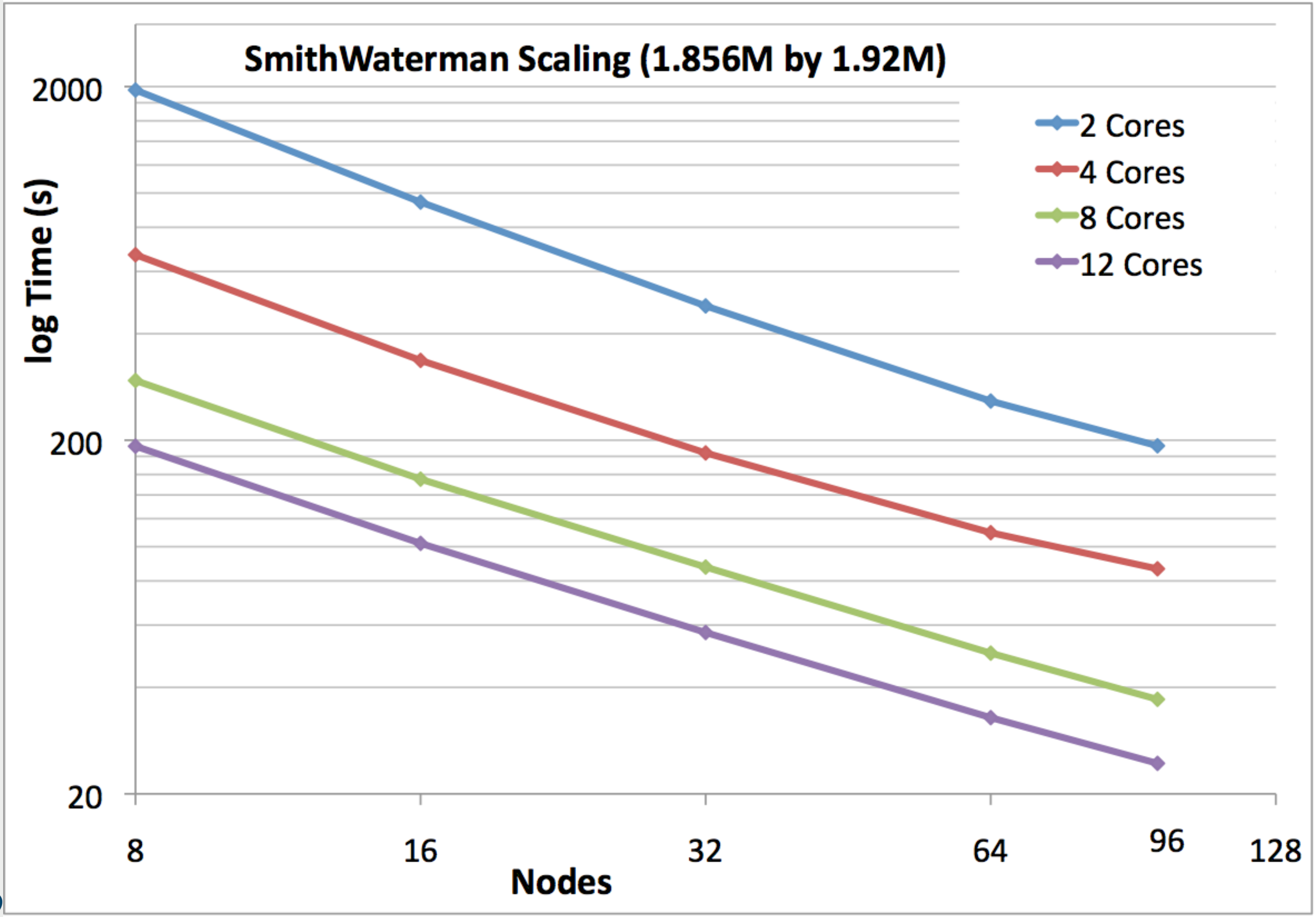



```
#define DDF_HOME(guid) . . .

for (i=0;i<H;++i)
  for (j=0;j<W;++j)
    matrix[i][j] = DDF_HANDLE(i*H+j);

doInitialPuts(matrix);
finish {
  for (i=0,i<H;++i) {
    for (j=0,j<W;++j) {
      DDF_t* curr = matrix[i][j];
      DDF_t* above = matrix[i-1][j];
      DDF_t* left = matrix[i][j-1];
      DDF_t* uLeft = matrix[i-1][j-1];
      if ( isHome(i,j) ) {
        async AWAIT (above, left, uLeft){
          Elem* currElem =
            init(DDF_GET(above),
                 DDF_GET(left),
                 DDF_GET(uLeft));
          compute(currElem);
          DDF_PUT(curr, currElem);
        }/*async*/
      }/*if*/
    }/*for*/
  }/*for*/
}/*finish*/
```

# Results for APGNS version of SmithWaterman (communication runtime uses MPI under the covers)




RICE

# 3) Mutual exclusion --- isolated statement

isolated <body>

- Like a critical section --- two tasks executing isolated statements must perform the isolated statements in mutual exclusion
  - ➔Weak atomicity guarantee: mutual exclusion only applies to (isolated, isolated) pairs of statement instances, not to (isolated,non-isolated) pairs
- Isolated statements may be nested, and may contain async and finish statements
  - See "Isolation for Nested Task Parallelism" [OOPSLA 2013] for details
- In case of an exception, all updates performed by <body> before throwing the exception will be observable after exiting <body>
- NOTE: mutual exclusion is intended for nondeterministic parallel programs

# Object-based isolation in HJ

```
isolated(<object-list>) <body>
```

- In this case, programmer specifies list of objects for which isolation is required
- Mutual exclusion is only guaranteed for instances of isolated statements that have a non-empty intersection in their object lists
  - Standard isolated is equivalent to isolated(*) by default i.e., isolation across all objects
- Implementation can choose to distinguish between read/write accesses for further parallelism
  - Current Habanero implementation supports object-based isolation, but does not exploit read/write distinction

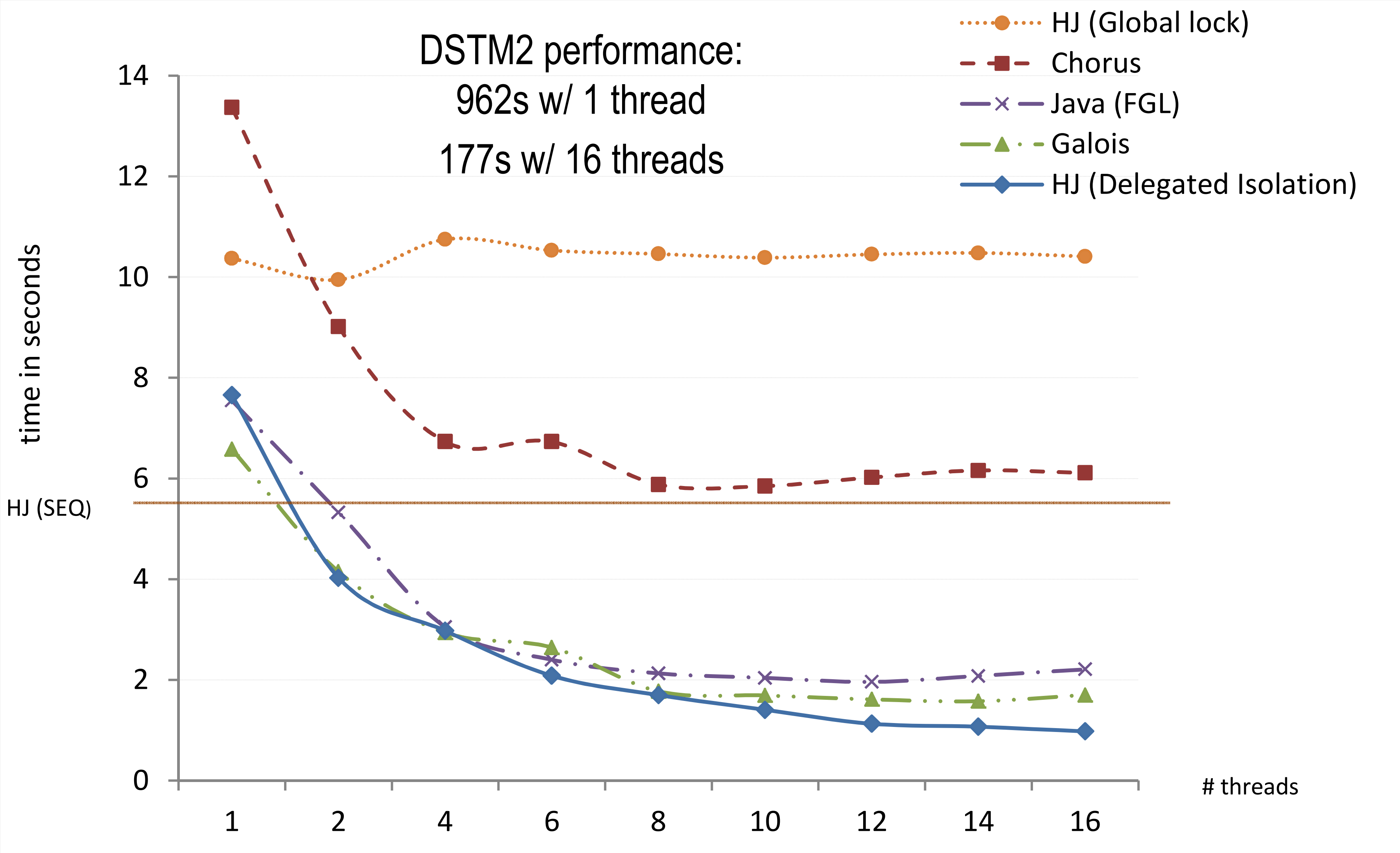
# Isolation by default

- Challenge: what if every async task could be isolated by default?
- Transactional memory approaches still incur too much overhead, and lack support for nested transactions
- Delegated Isolation approach:
  - 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
    - More complex rules for nested transactions (see OOPSLA '13 paper for details)
  - Deadlock-freedom and livelock-freedom guarantees
  - Open question: use of recent hardware TM capabilities


- "Delegated Isolation", R. Lublinerman, J. Zhao, Z. Budimlic, S. Chaudhuri, V. Sarkar, OOPSLA 2011
- "Isolation for Nested Task Parallelism", J. Zhao, R. Lublinerman, Z. Budimlic, S. Chaudhuri, V. Sarkar, OOPSLA 2013.

# Performance: DMR benchmark on 16-core Xeon SMP

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

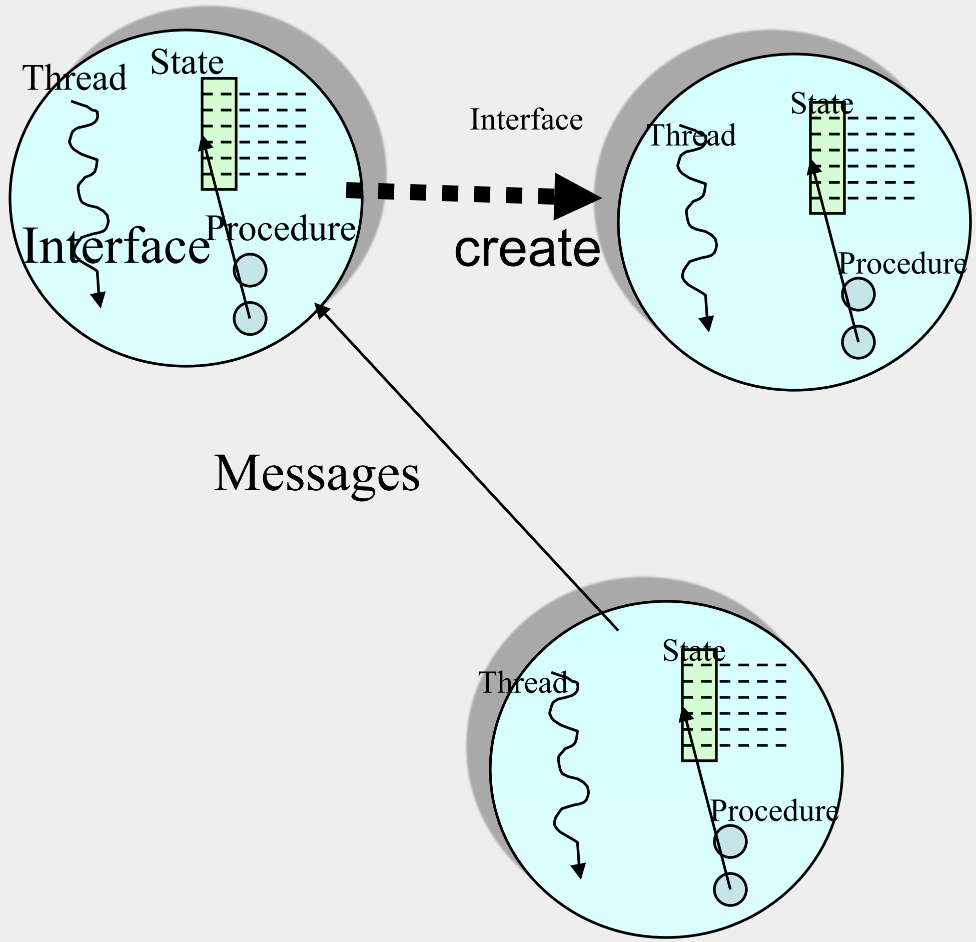



"Delegated Isolation", R. Lublinerman, J. Zhao, Z. Budimlic, S. Chaudhuri, V. Sarkar, OOPSLA 2011

# 3) Actors: an alternative approach to mutual exclusion by default

An actor may:

- process messages
- send messages
- change local state
- create new actors
- terminate (and release enclosing finish)




"Integrating Task Parallelism with Actors". Shams Imam, Vivek Sarkar, OOPSLA 2012.

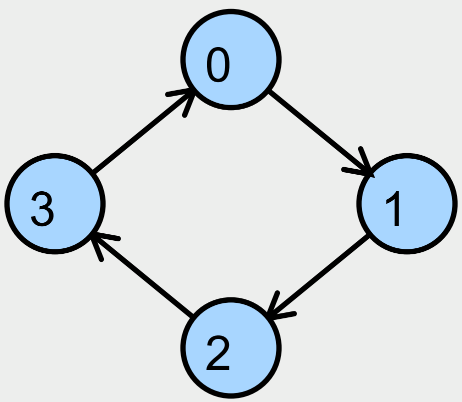
# Hello World Example

```
public class HelloWorld {
  public static void main(final String[] args) {
    finish(()-> {
      EchoActor actor = new EchoActor();
    actor.start(); // don't forget to start the actor
    actor.send("Hello"); // asynchronous send (returns immediately)
    actor.send("World");
    actor.send(EchoActor.STOP_MSG);
  });
}
private static class EchoActor extends Actor<Object> {
  static final Object STOP_MSG = new Object();
  private int messageCount = 0;
  protected void process(final Object msg) {
    if (STOP_MSG.equals(msg)) {
       println("Message-" + messageCount + ": terminating.");
       exit(); // never forget to terminate an actor
     } else {
       messageCount += 1;
       println("Message-" + messageCount + ": " + msg);
} } } }
```

Habanero actor model preserves order of messages between same sender and receiver

# ThreadRing Example

```
finish(() -> {
   int numThreads = 4;
   int numberOfHops = 10;
   ThreadRingActor[] ring =
     new ThreadRingActor[numThreads];
   for(int i=numThreads-1;i>=0; i--) {
     ring[i] = new ThreadRingActor(i);
     ring[i].start();
     if (i < numThreads - 1) {
       ring[i].nextActor(ring[i + 1]);
   } }
   ring[numThreads-1].nextActor(ring[0]);
   ring[0].send(numberOfHops);
}); // finish
```




```
class ThreadRingActor
     extends Actor<Object> {
   private Actor<Object> nextActor;
   private final int id;
   ...
   public void nextActor(
     Actor<Object> nextActor) {...}
   void process(Object theMsg) {
     if (theMsg instanceof Integer) {
       Integer n = (Integer) theMsg;
       if (n > 0) {
         println("Thread-" + id +
           " active, remaining = " + n);
         nextActor.send(n - 1);
       } else {
         println("Exiting Thread-"+ id);
         nextActor.send(-1);
         exit();
       }
     } else {
     /* ERROR - handle appropriately */
} } }
```

# 3) Asynchronous Collectives with Finish Accumulators (can be combined with Actors)

```
final FinishAccumulator ac =
                    newFinishAccumulator(Operator.SUM, int.class);
finish(ac) nqueens_kernel(new int[0], 0);
System.out.println("No. of solutions = " + ac.get())
. . .
void nqueens_kernel(int [] a, int depth) {
  if (size == depth) ac.put(1);
  else
    /* try each possible position for queen at depth */
    for (int i =  0; i < size; i++) async {
      /* allocate a temporary array and copy array a into it */
      int [] b = new int [depth+1];
      System.arraycopy(a, 0, b, 0, depth);
      b[depth] = i;
      if (ok(depth+1,b)) nqueens_kernel(b, depth+1);
    } // for-async
} // nqueens_kernel()
```

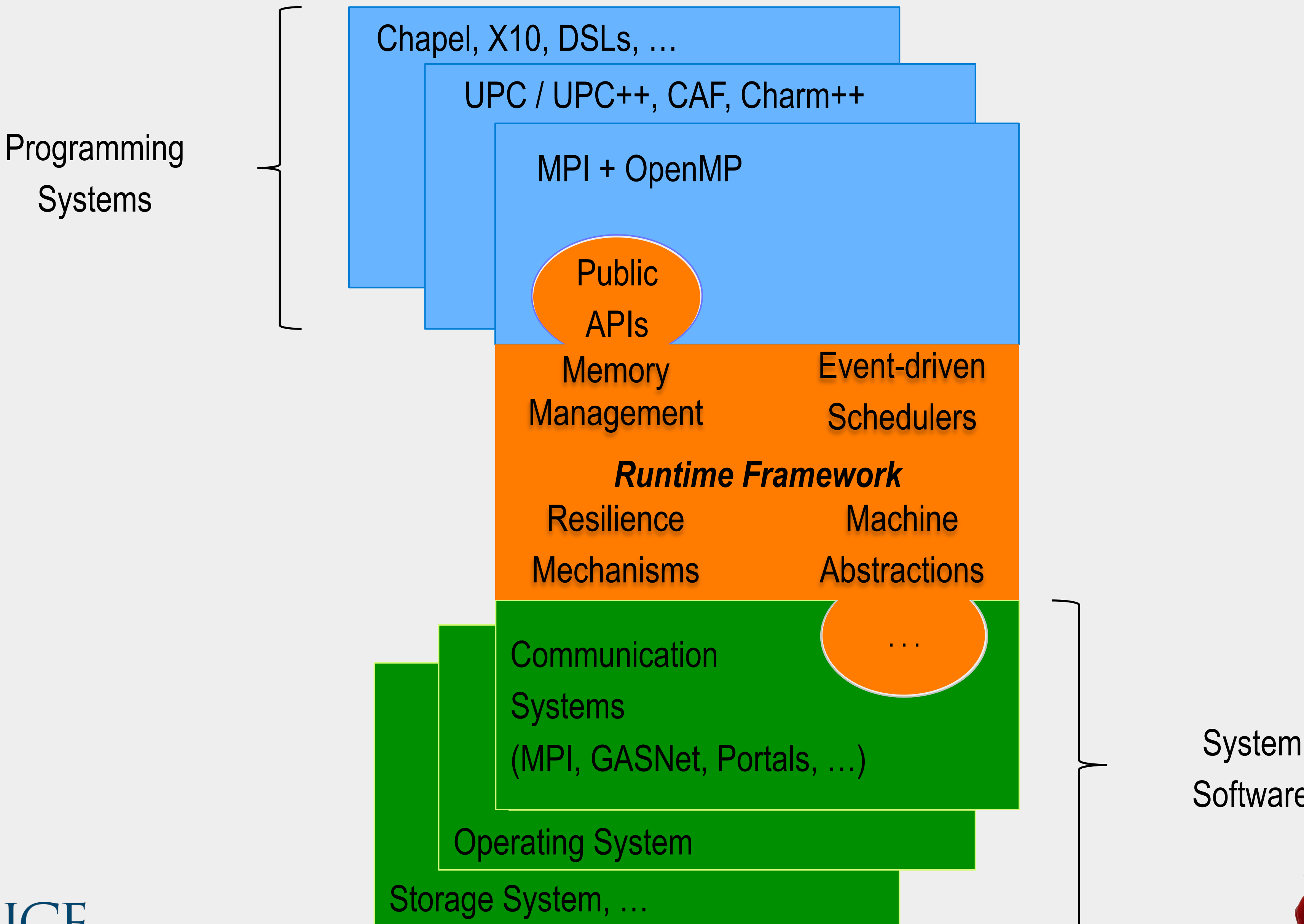
RICE

# Role of Runtime Systems

- Inherent variability and complexity of extreme scale platforms calls for a runtime system that is *abstract, asynchronous, user-controllable, adaptive, and portable*
- Bridging role between programming systems and system software brings multiple benefits
  - composability and hybridization by default,
  - improved performance for existing programming models,
  - enablement of new programming models,
  - simplified interfaces for system software,
  - improved use of system services (e.g., deadlock avoidance),
  - and cleaner code!

RICE

# Runtime Systems --- how to prime the pump?




RICE

# Motivation for an Open Community Runtime (OCR)

- Wide agreement that execution models for extreme scale systems will differ significantly from past execution models
- Shoehorning a new execution model into an old runtime system is counter-productive
- Instead, make a fresh start but carry forward reusable components from current runtime systems as appropriate

➔Motivation for Open Community Runtime framework that …

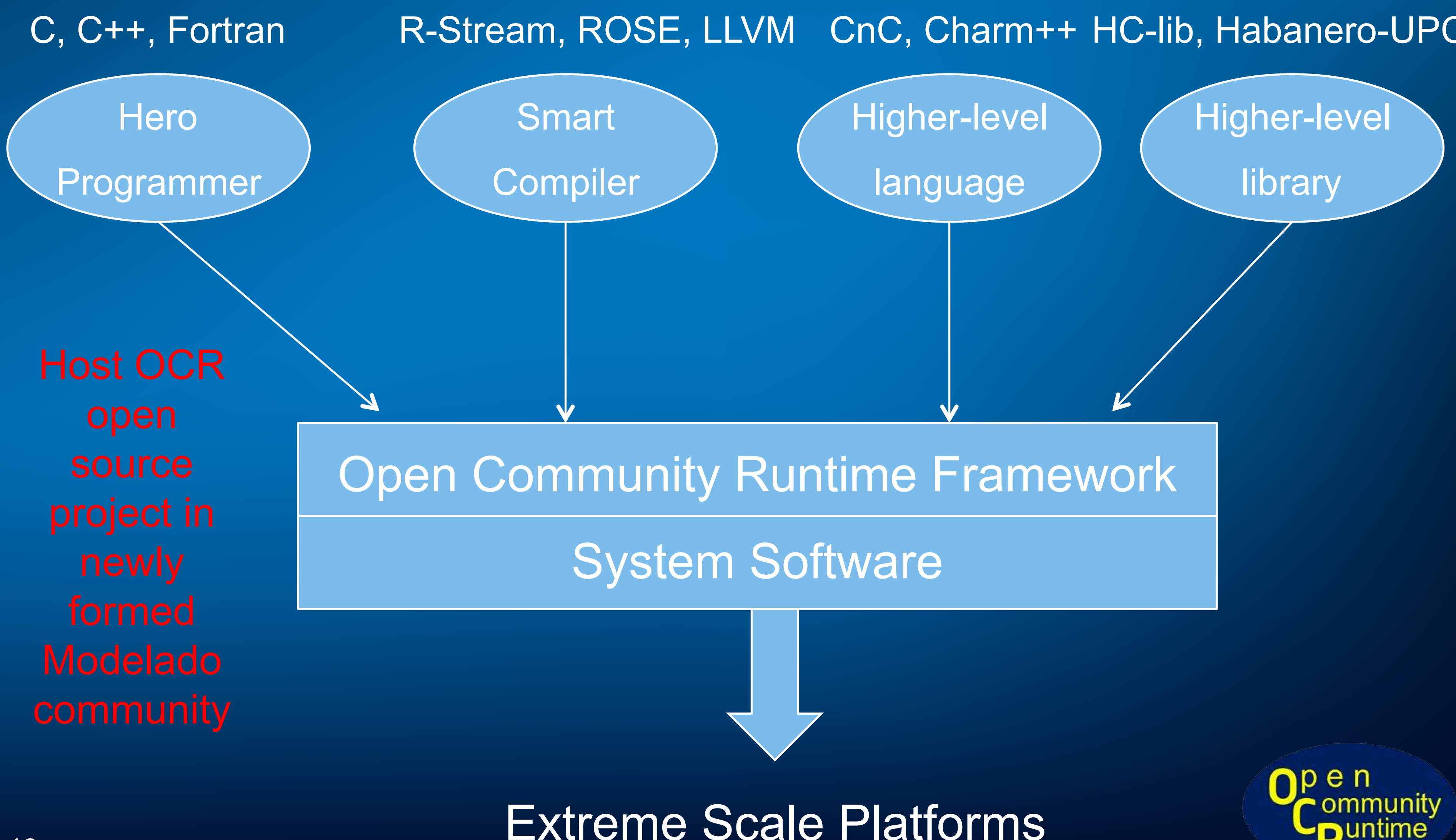
- is representative of future execution models
- can be targeted by multiple high-level programming systems
- can be mapped on to multiple extreme scale platforms
- is available as an open-source testbed
- reduces duplication of infrastructure efforts
- **enables us to address revolutionary challenges**

# Example API: Creating an Event-Driven Task (EDT)

- u8 **ocrEdtCreate**(ocrGuid_t * guid, ocrGuid_t templateGuid, u32 paramc, u64* paramv, u32 depc, ocrGuid_t *depv, u16 properties, ocrGuid_t affinity, ocrGuid_t *outputEvent);
  - guid [out]: the assigned guid
  - templateGuid: the template the EDT is an instance of
  - paramc: nb of u64 parameters
  - paramv: pointer to u64 parameters
  - depc: nb of guid parameters
  - depv: array of guid dependences (if known at creation or NULL)
  - properties: can specify if finish-edt here.
  - affinity: affinity guid
  - outputEvent [out]: edt completion notification

Open Community Runtime

# OCR Vision




Host OCR open source project in newly formed Modelado community

# Modelado Foundation

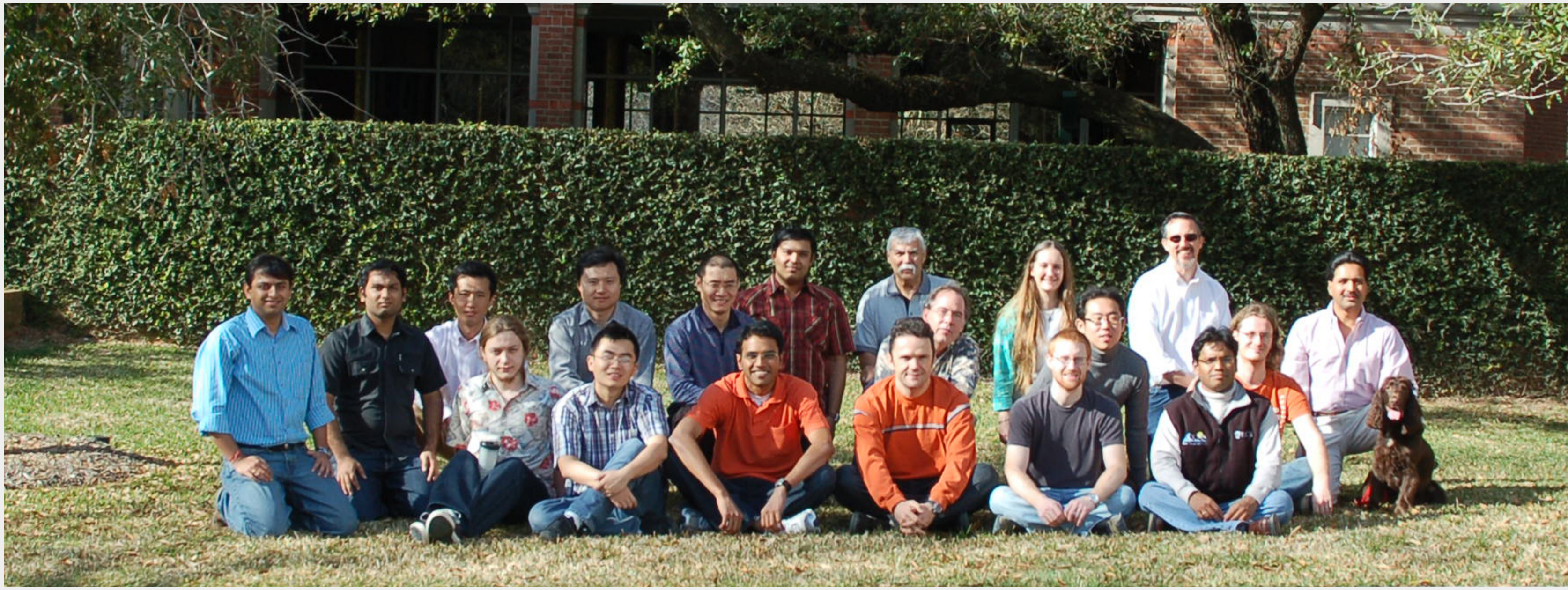
modelado

- A new Open Source Foundation for Parallel Computing
- Organization
  - Establish an open, transparent environment in which solutions are not pre-determined
  - Provide an organic process for community decision-making, ensuring the best solution wins (metrics)
  - Avoid a single player or clique dominating
  - Lower the barrier to participation by providing stable, reliable releases of candidate solutions to a broad audience
  - http://www.modelado.org
- Services
  - Project Team Infrastructure - e.g. source code control, tooling, debuggers, collaboration/communication
  - Release Engineering
  - Technical Support
  - IP management
  - Education, instruction and training
  - Community Development

# Conclusions

- Holistic redesign of software stack is needed to address concurrency, energy, and resiliency challenges of Extreme Scale systems
- Urgent need for execution models that integrate hybrid dimensions of parallelism and heterogeneity – multicore, accelerators, multi-node, HPC cluster, data center cluster
- Well-designed runtime primitives can provide foundation for new execution models, with synergistic innovation in languages, compilers, system software and system hardware
- OCR is a starting point for a strawman community effort --- let’s work as a community to extend/replace OCR components as needed!

# Habanero Team Pictures (http://habanero.rice.edu)



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