George R. Brown School of Engineering
Computer Science

Overview of DEGAS Programming Models area

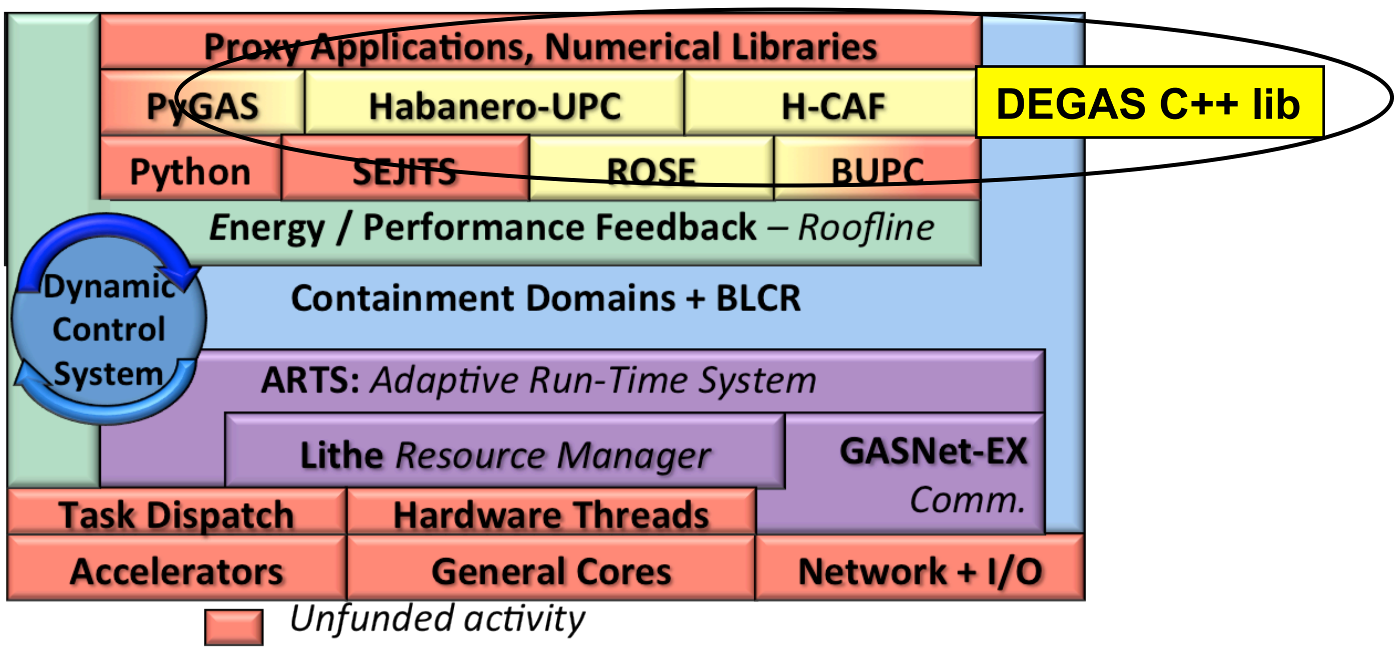
Vivek Sarkar

Rice University

June 3, 2013

Context

- Exascale systems will impose a fresh set of requirements on programming models including
  - targeting nodes with hundreds of homogeneous and heterogeneous cores with limited memory per core
  - severe bandwidth, energy, locality and resiliency constraints within and across nodes.
- DEGAS = Dynamic Exascale Global Address Space



Programming Model Goals

- Programmability: ease of use by application partners
- Performance: effective exploitation of unique aspects of DEGAS stack
  - Dynamic + Hierarchical + One-sided
- Portability:
  - Unified primitives for synchronization, communication and parallelism
    - Homogeneous/heterogeneous, intra-node/inter-node, SIMD/SIMT, SPMD/dynamic, synchronous/ asynchronous, …
  - Tight integration with leading-edge processors and interconnects from multiple vendors
- Success: DEGAS programming systems used in production context on leading-edge hardware by application partners

Pushing the boundaries

- Asynchrony
  - One-sided communications, function shipping
  - Data-driven tasks
  - PGAS, APGNS (Async Partitioned Global Name Space)
- Hierarchy and Locality
  - Hierarchical Teams
  - Hierarchical CAF
  - Hierarchical Place Tree
  - Containment Domains
- Heterogeneity
  - Automatic generation of CUDA & OpenCL
  - Dynamic scheduling for homogeneous and heterogeneous processors

DEGAS Programming System Gaps being addressed by other X-Stack projects

- Domain Specific Languages
- Debugging tools
- Performance tools
- Auto-tuning
- . . .

- … but we’re interested in these topics too!

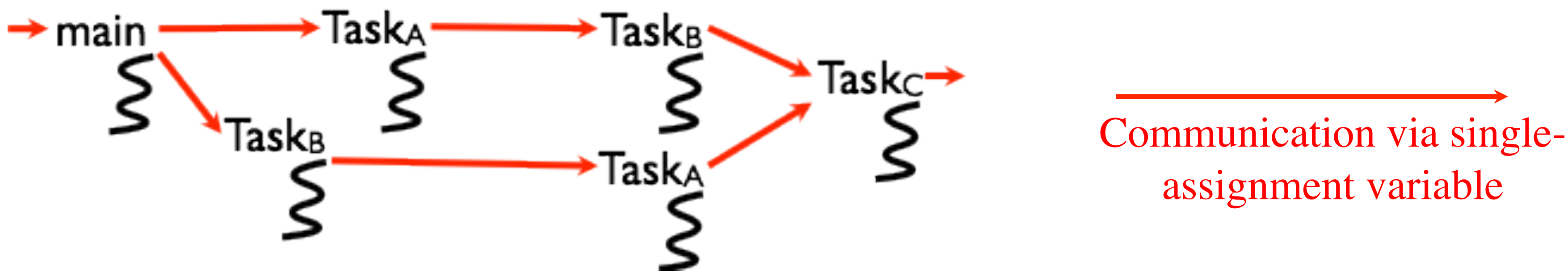
Programming Model Talks

- MG Code for language and system design (Sam, Nick)
- Hierarchical teams (Amir)
- CAF Overview (John)
- DEGAS programming system via C++ library extension (Yili)
- Future of Scientific Python (Fernando)

Background: Summary of Habanero-C (HC)

- HC is a parallel programming system (language + compiler + runtime) developed in the Rice Habanero Multicore Software research project
- Five classes of parallel programming primitives in HC:
  1. Dynamic task creation & termination
     - async, finish, forasync
  2. Data-Driven Tasks (DDTs) and Data-Driven Futures (DDFs)
     - await, put(), get()
  3. Support for affinity control and heterogeneous processors
     - hierarchical places
  4. Collective and point-to-point synchronization for SPMD parallelism
     - phasers
  5. Distribution
     - Partitioned Global Name Space (PGNS) model with Distributed Data-Driven Futures (DDDFs)
     - Integration of task parallelism with communication (HCMPI)

Productivity Benefits of Dataflow Programming



- "Macro-dataflow" = extension of dataflow model from instruction-level to task-level operations
- General idea: build an arbitrary task graph, but restrict all inter-task communications to single-assignment variables
- Static dataflow ==> graph fixed when program execution starts
- Dynamic dataflow ==> graph can grow dynamically
- Semantic guarantees: race-freedom, determinism
- Deadlocks are possible due to unavailable inputs (but they are deterministic)

RICE

Data-Driven Futures (DDFs) and Data-Driven Tasks (DDTs)

```
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)
```

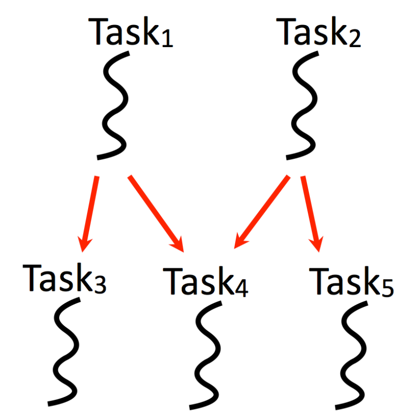
- Return value stored in ddfA
- No blocking needed --- should only be performed by async's that contain ddfA in their AWAIT clause, or when some other synchronization (e.g., finish) guarantees that DDF_PUT must have been performed.

DDFs can be implemented more efficiently than classical futures

RICE

Example Habanero-C code fragment with Data-Driven Futures (Dag Parallelism)

```
DDF_t* left = DDF_CREATE();
DDF_t* right = DDF_CREATE();
finish {
  async AWAIT(left) leftReader(DDF_GET(left)); // Task3
  async AWAIT(right) rightReader(DDF_GET(right)); // Task5
  async AWAIT(left,right) // Task4
        bothReader(DDF_GET(left), DDF_GET(right));
  async DDF_PUT(left,leftWriter()); //Task1
  async DDF_PUT(right,rightWriter());//Task2
}
```



- AWAIT clauses capture data flow relationships

Smith Waterman example (Single Node)

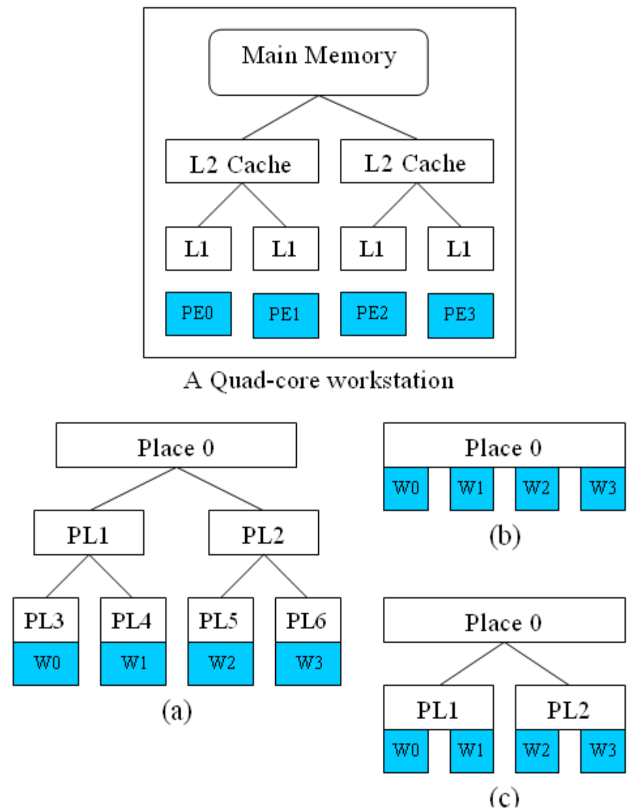
```
finish { // matrix is a 2-D array of DDFs
  for (i=0,i<H;++i) {
    for (j=0,j<W;++j) {
      DDF_t* curr = matrix[i][j];
      DDF_t* above = matrix[i-1][j];
      DDF_t* left = matrix[i][j-1];
      DDF_t* uLeft = matrix[i-1][j-1];
      async AWAIT (above, left, uLeft){
          Elem* currElem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

Background: Summary of Habanero-C (HC)

- HC is a parallel programming system (language + compiler + runtime) developed in the Rice Habanero Multicore Software research project
- Five classes of parallel programming primitives in HC:
  1. Dynamic task creation & termination
     - async, finish, forasync
  2. Data-Driven Tasks (DDTs) and Data-Driven Futures (DDFs)
     - await, put(), get()
  3. Support for affinity control and heterogeneous processors
     - hierarchical places
  4. Collective and point-to-point synchronization
     - phasers
  5. Extensions for distributed-memory parallelism
     - Partitioned Global Name Space (PGNS) model with Distributed Data-Driven Futures (DDDFs)
     - Integration of task parallelism with communication (HCMPI)

Hierarchical Place Trees (HPT)

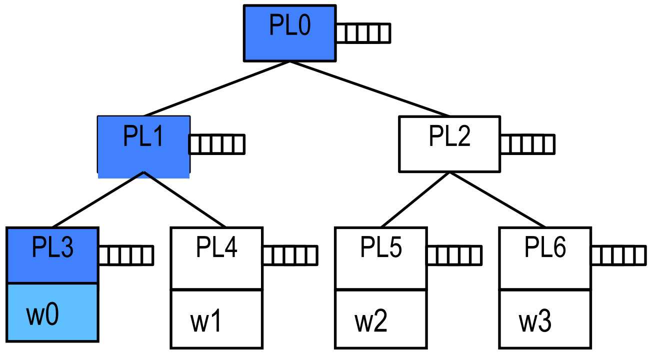
- HPT approach
  - Hierarchical memory + Dynamic parallelism
- Place denotes affinity group at memory hierarchy level
  - L1 cache, L2 cache, CPU memory, GPU memory, …
- Leaf places include worker threads
  - e.g., W0, W1, W2, W3
- Explore multiple HPT configurations
  - For same hardware and application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

Locality-aware Scheduling using the HPT (Logical View)

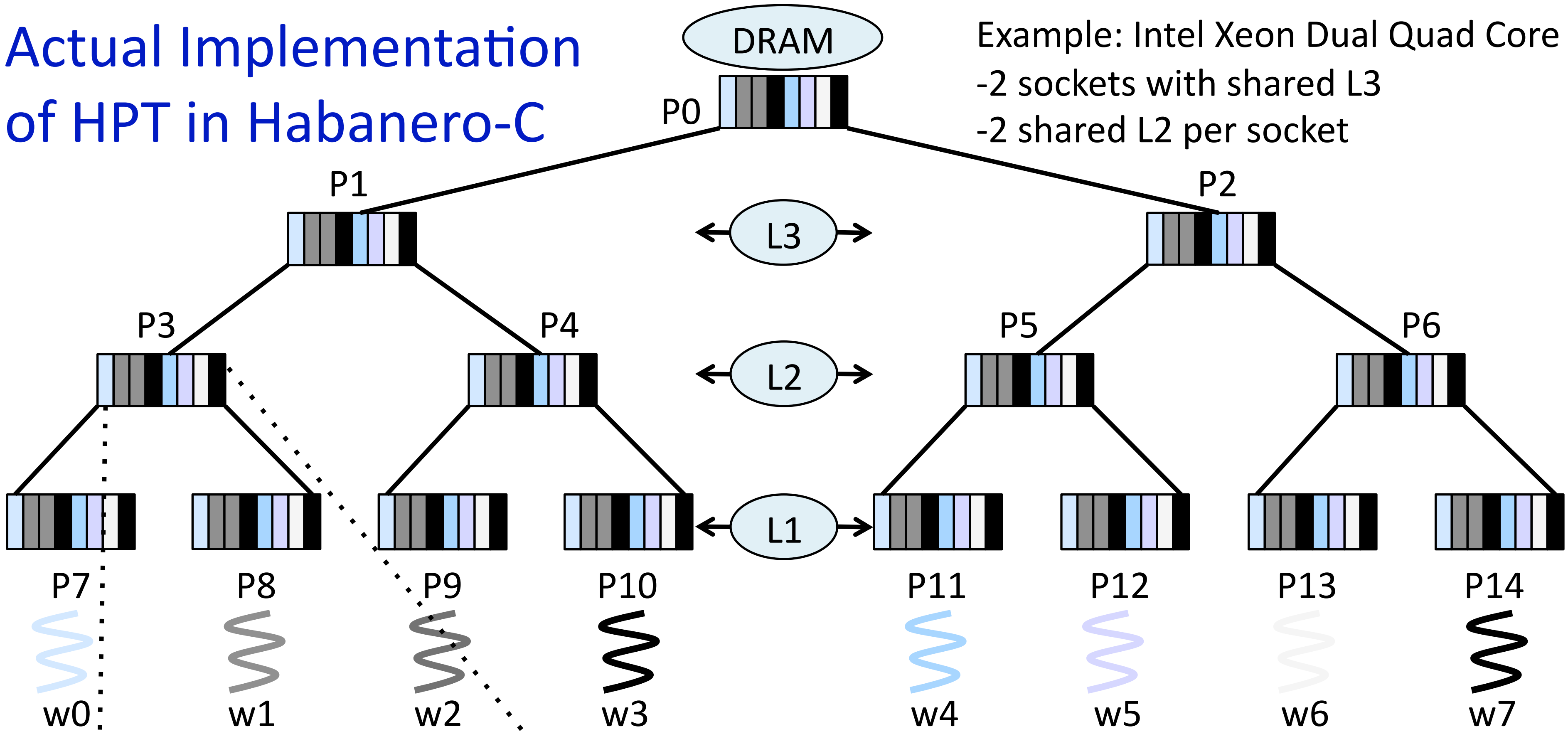
- Workers attached to leaf places
  - Bind to hardware core
- Each place has a queue
  - **async at**(<*pl*>) <*stmt*>: push task onto place *pl*'s queue

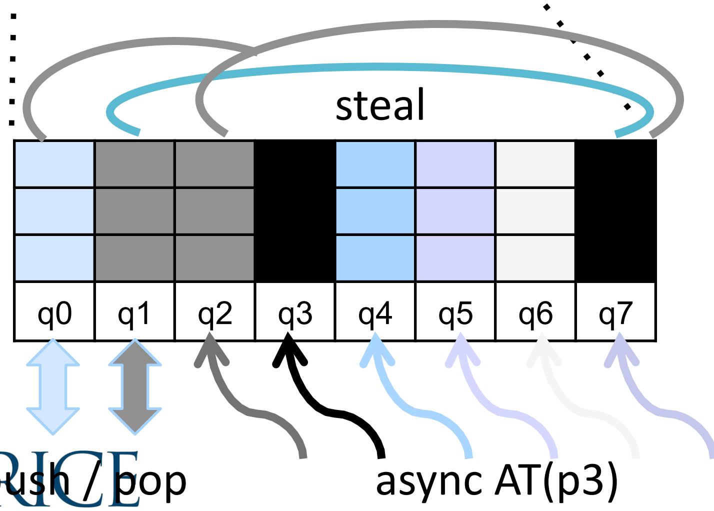


- A worker executes tasks from ancestor places from bottom-up
  - W0 executes tasks from PL3, PL1, PL0
- Tasks in a place queue can be executed by all workers in the place's subtree
  - Task in PL2 can be executed by workers W2 or W3

Actual Implementation of HPT in Habanero-C

Example: Intel Xeon Dual Quad Core
-2 sockets with shared L3
-2 shared L2 per socket

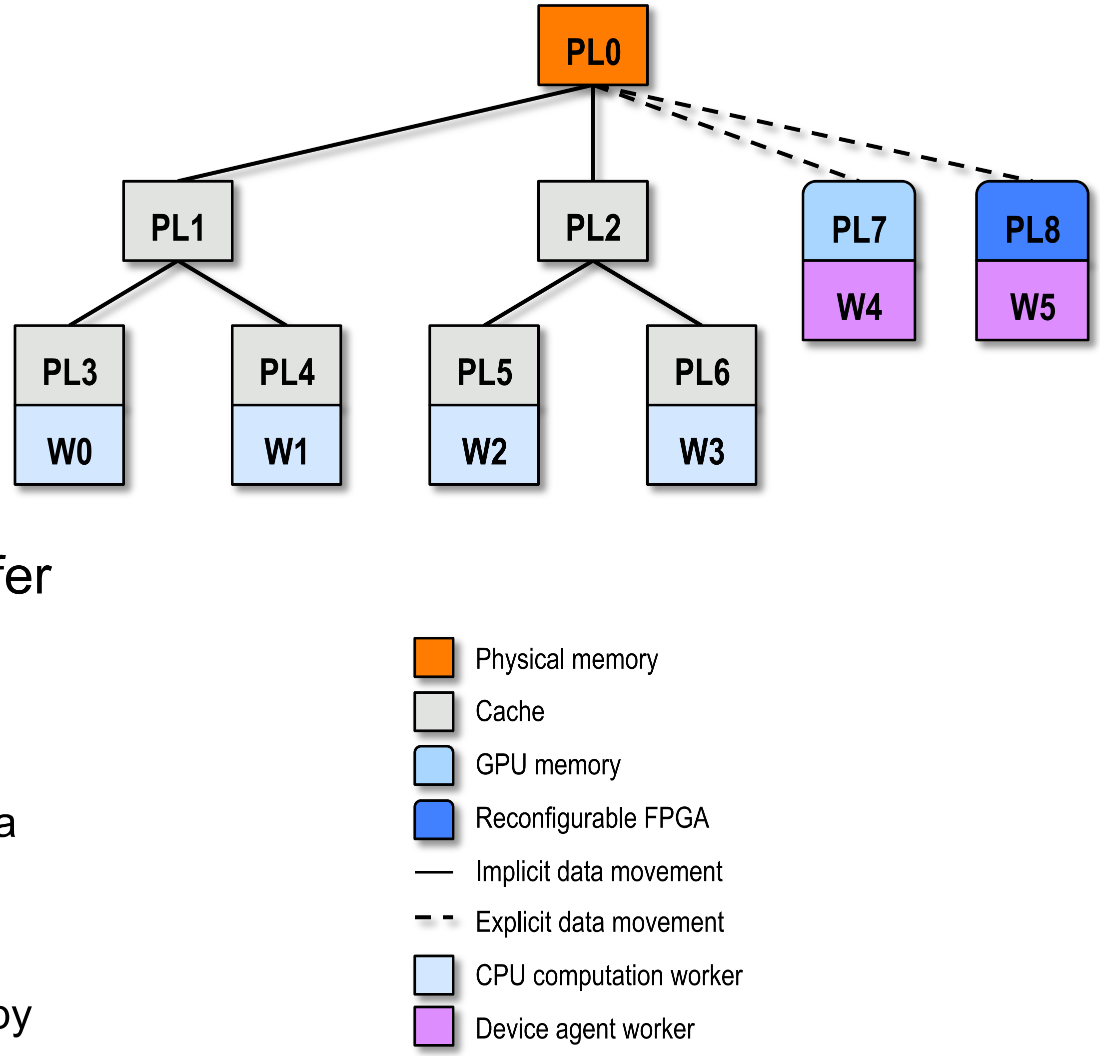




- Each place has one queue per worker
    - Ensures non-synchronized push and pop
- Workers bound to cores (leaf places)
- Any worker can push a task at any place
- Pop / steal access permitted to subtree workers
- Workers traverse path from leaf to root
- Tries to pop, then steal, at every place
- After successful pop / steal worker returns to leaf

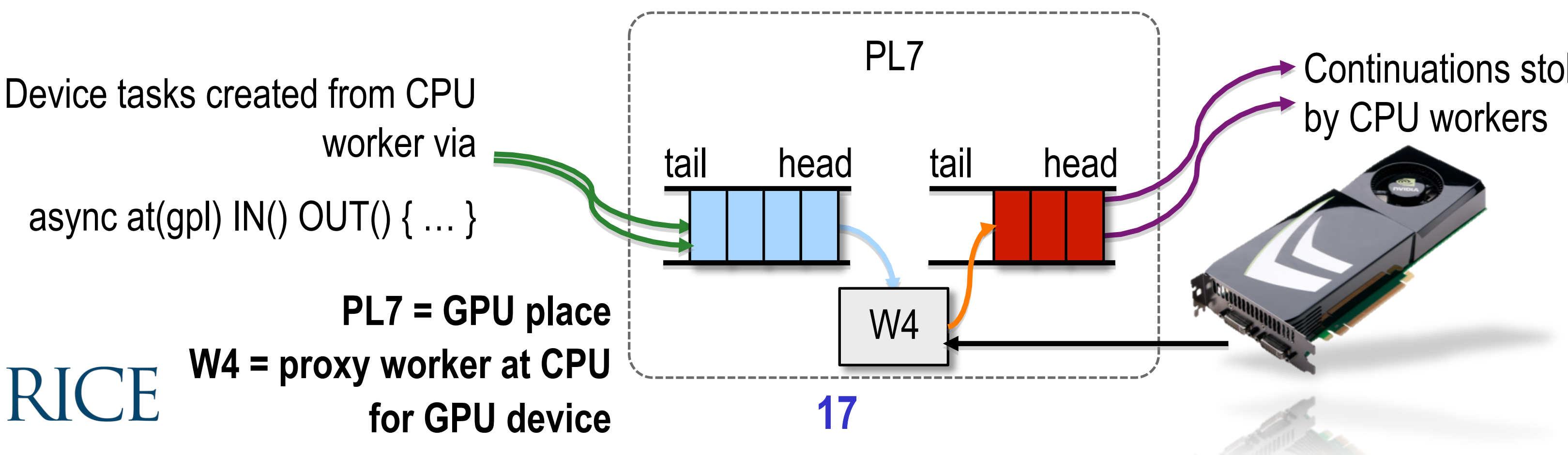
HC Hierarchical Place Trees for Heterogeneous Architectures

- 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
- 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



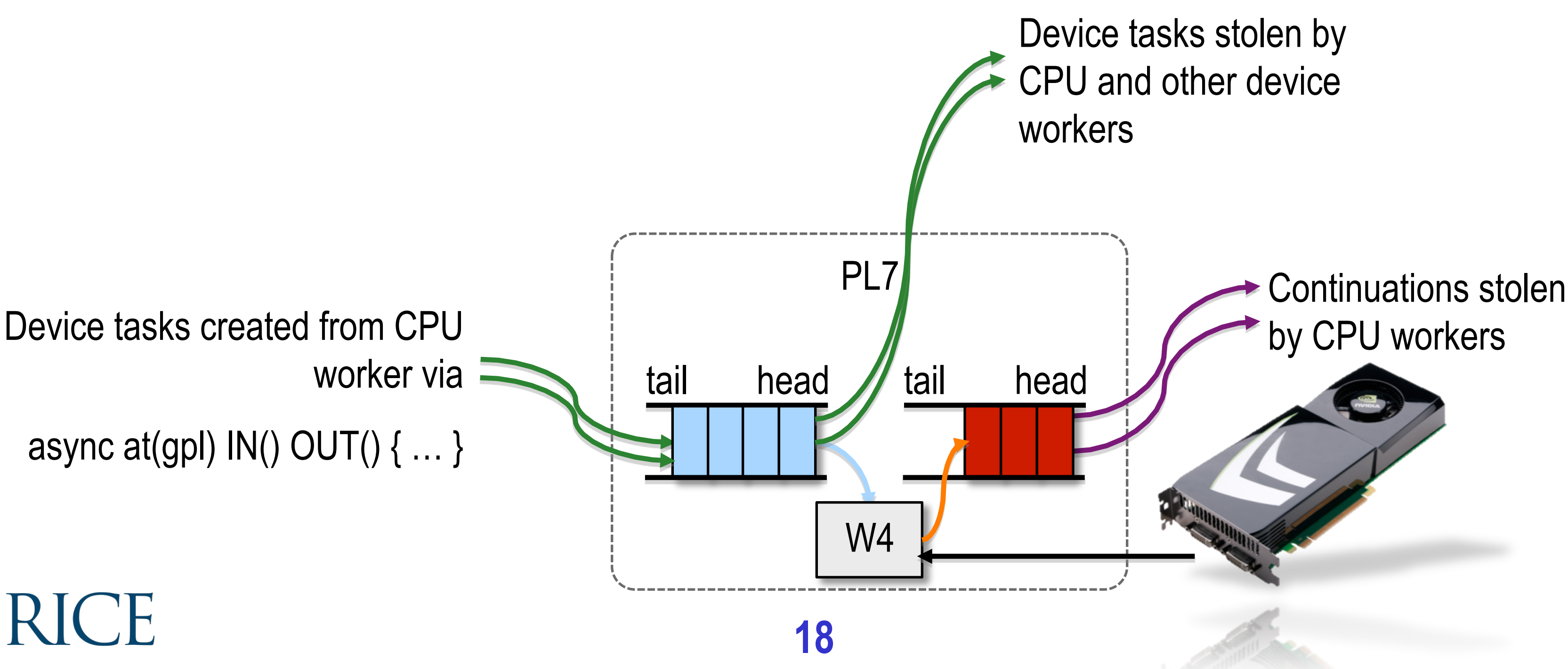
Hybrid Scheduling for Heterogeneous Nodes

- Device place has two HC (half-concurrent) mailboxes: inbox (green) and outbox (red)
  - No locks – highly efficient
- Inbox maintains asynchronous device tasks (with IN/OUT)
  - Concurrent enqueuing device tasks by CPU workers from tail
  - Sequential dequeuing tasks by device “proxy” worker
- Outbox maintains continuation of the finish scope of tasks
  - Sequential enqueuing continuation by “proxy” worker
  - Concurrent dequeuing (steal) by CPU workers

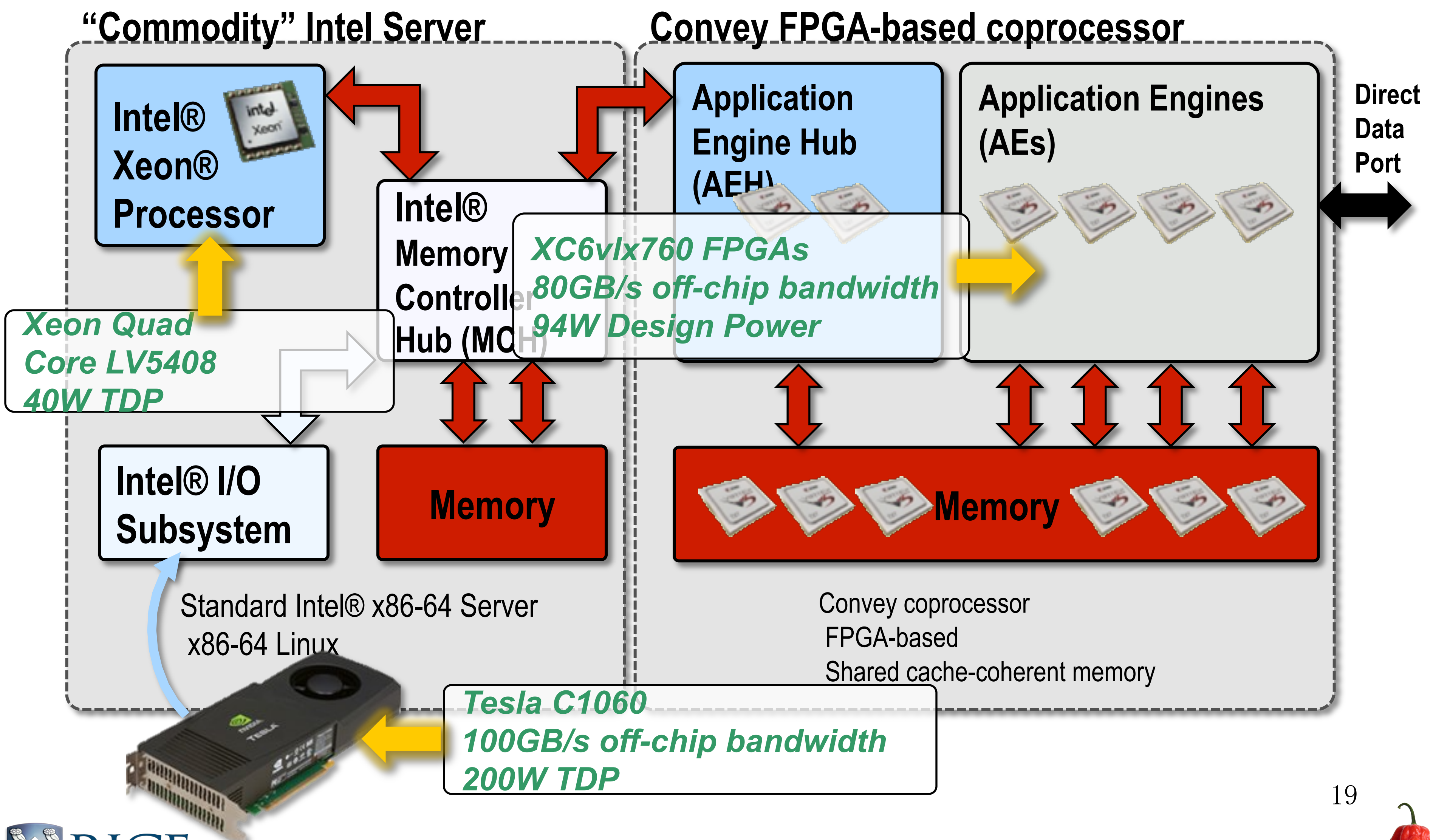


Hybrid Scheduling with Cross-Platform Work Stealing

- Steps are compiled for execution on CPU, GPU or FPGA
  - Same-source multiple-target compilation in future
- Device inbox is now a concurrent queue and tasks can be stolen by CPU or other device workers
  - Multitasks, range stealing and range merging in future

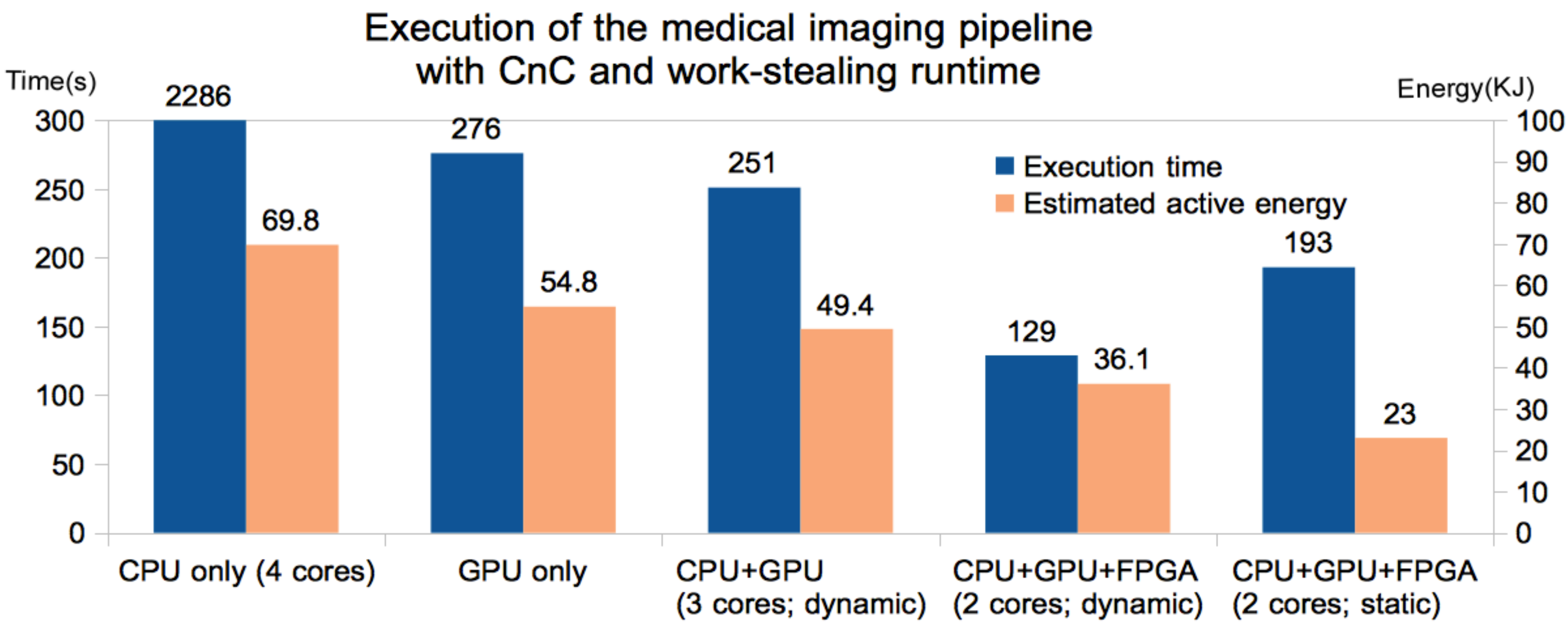


Convey HC-1ex Testbed



Experimental results

- Execution times and active energy with dynamic work stealing

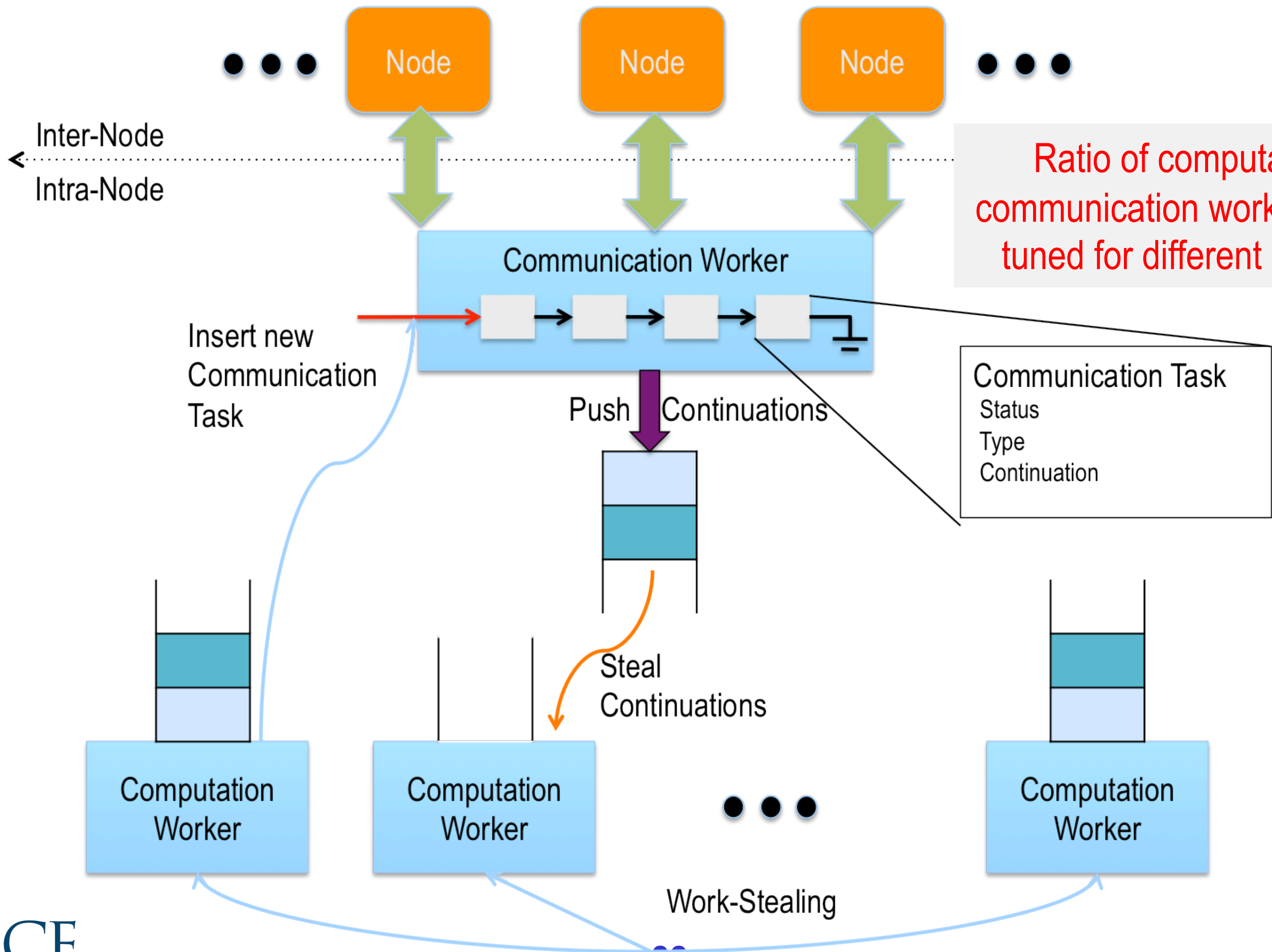


Background: Summary of Habanero-C (HC)

- HC is a parallel programming system (language + compiler + runtime) developed in the Rice Habanero Multicore Software research project
- Five classes of parallel programming primitives in HC:
  1. Dynamic task creation & termination
     - async, finish, forasync
  2. Data-Driven Tasks (DDTs) and Data-Driven Futures (DDFs)
     - await, put(), get()
  3. Support for affinity control and heterogeneous processors
     - hierarchical places
  4. Collective and point-to-point synchronization
     - phasers
  5. <u>Extensions for distributed-memory parallelism</u>
     - Asynchornos Partitioned Global Name Space (APGNS) model with Distributed Data-Driven Futures (DDDFs)
     - Integration of task parallelism with communication (HCMPI)

RICE

From Locality to Communication --- Integrating Inter-node Communication with Intra-node Task Scheduling



Ratio of computation to communication workers can be tuned for different platforms

APGNS Programming Model

- Philosophy :
  - In the 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).
  - APGNS can be implemented on a wide range of communication runtimes including GASNet and MPI, regardless of whether or not a global address space is supported.

RICE

Distributed Data-Driven Futures (DDDFs)

```
int DDF_HOME (int guid) {…};
```

- a globally unique DDDF id ➔ *home* rank

```
int DDF_SIZE (int guid) {…};
```

- a globally unique DDDF id ➔ size of DDDF in bytes

```
DDF_t* ddfA = DDF_HANDLE(guid); (contrast with DDF_CREATE of shared memory)
```

- Allocate an instance of a <u>distributed data-driven-future</u> object (container)
- Every rank has a handle, *home* rank can `put,` every rank can `get`

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

- Create a new <u>data-driven-task</u> to start executing Stmt after all of ddfA, ddfB, … become available (i.e., after task becomes "enabled")
- Seamless usage of distributed and shared memory DDFs
- Await registration handles the communication implicitly

Distributed Data-Driven Futures (DDDFs, contd)

`DDF_PUT(ddfA, V);`

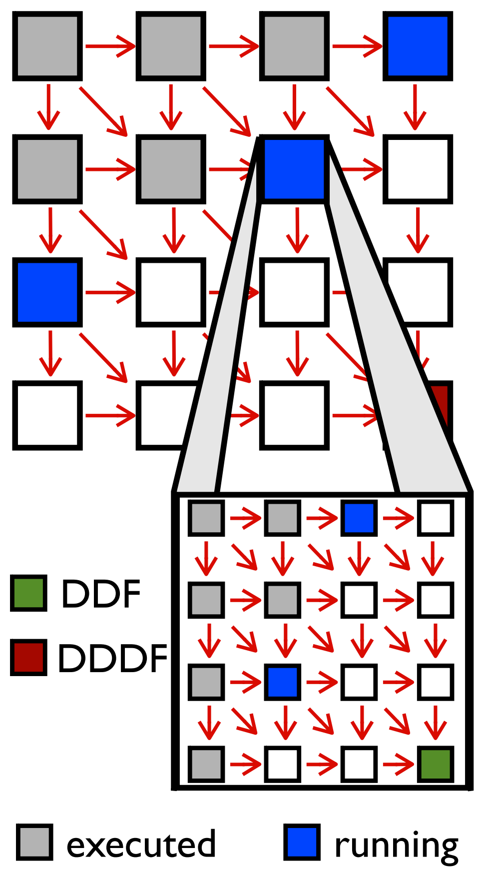
- Store object V in ddfA, thereby making ddfA available
- Single-assignment rule: at most one put is permitted on a given DDF
- Restricted only to *home* rank
- Handles communication to registrants implicitly

`DDF_GET (ddfA)`

- Return value stored in ddfA
- Ensured to be safely performed by async's that contain ddfA in their await clause
- needs to be preceded by await clause on ddfA if the producer is remote
  - await can be in a different task provided local synchronization ensures the await precedes get

RICE

Multi-Node SmithWaterman

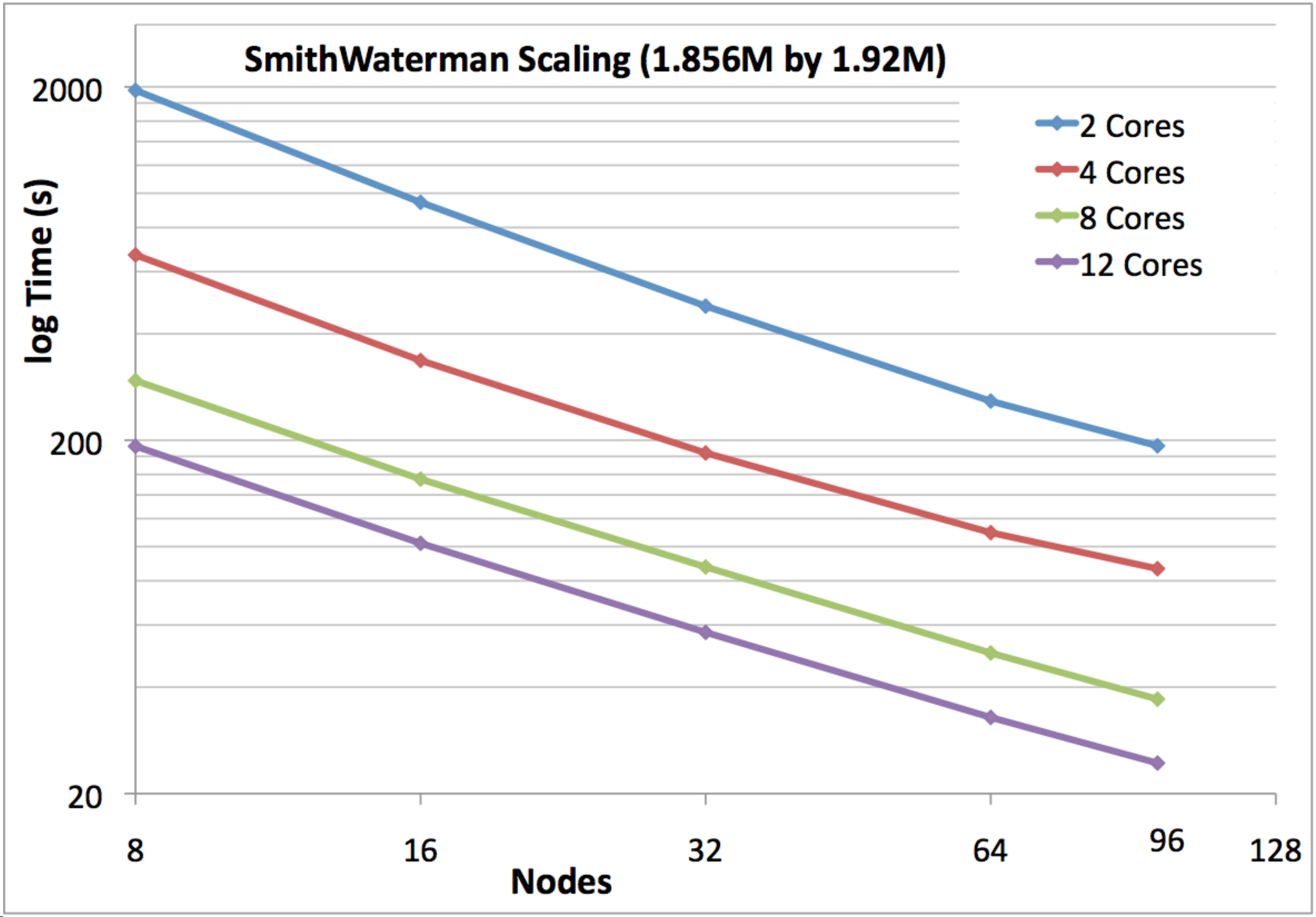



```
#define DDF_HOME(guid) (guid%NPROC)
#define DDF_SIZE(guid) (sizeof(Elem))

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



Habanero Posters

- Sanjay Chatterjee
  - The Habanero Asynchronous Partitioned Global Name Space (APGNS) Programming Model
- Deepak Majeti
  - Programming Heterogeneous Platforms with Habanero-C
- Nick Vrvilo
  - Comparison of MPI and UPC overheads for MG benchmarks

Programming Model Discussion Topics

- How can Lithe be used to enable Habanero-UPC and H-CAF to interoperate with MPI + OpenMP?
- How should Containment Domains be integrated with DEGAS programming models?
- Programming model and compiler support for CA?
- Next steps
  - Demonstrations of DEGAS programming models on MG code
    - DEGAS C++ lib version
    - Habanero-UPC version
    - Hierarchical-CAF version
  - Integration of HClib with DEGAS C++ library?
- . . .