Compiler and Runtime Challenges for Memory-Centric Programming

Vivek Sarkar
Professor, School of Computer Science
Stephen Fleming Chair for Telecommunications, College of Computing
Georgia Institute of Technology
Habanero Extreme Scale Lab (as of Fall 2017)

- **Faculty**
  - Vivek Sarkar

- **Research Scientists**
  - Akihiro Hayashi, Jun Shirako, Jisheng Zhao

- **Postdoctoral Researchers**
  - Tiago Cogumbleiro, Max Grossman, Farzad Khorasani

- **Graduate Students**
  - Kumud Bhandari, Prasanth Chatarasi, Arghya Chatterjee, Sana Damani, Ankush Mandal, Sriraj Paul, Fangke Ye, Lechen Yu

- **Acknowledgments**
  - Research supported in part by DARPA, DoD, DOE, NSF
Habanero Extreme Scale Lab at Rice University

Habanero Team Photo, May 2016
GT PLSE group outing to Atlanta BeltLine, October 2017
Levels of Disruption in Moore’s Law End-Game and Post-Moore eras

At the far right (level 4) are non-von Neumann architectures, which completely disrupt all stack levels, from device to algorithm.

At the least disruptive end (level 1) are more “Moore” approaches, such as new transistor technology and 3D circuits, which affect only the device and logic levels.

Hidden changes are those of which the programmer is unaware.

Range of Approaches (and disruptions!) for Memory-Centric Processing
Memory-Centric Processing

Approach:
Memory-Centric Processing places computation closer to memory than conventional cores. These approaches are being explored at the in situ, sense amps, memory bank, on-memory, and near-memory levels.

Current & Future Promise:
Reduce memory bandwidth bottlenecks by performing lightweight specialized operations close to memory. Additional benefits include reduced latency, reduced energy of transport, faster atomic operations, and higher levels of concurrency.

Motivating applications:
Applications with memory–centric streaming operations, e.g., encryption/decryption, search, big data, big graphs, deep learning

Timeframe:
Above approaches demonstrated at the research level. Near-Memory Processing appears to be the most viable for the next level, due to its synergy with 3D stacking.

Key challenges:
How to maintain some level of coherence/consistency across data copies, how to support remote computations and a global address space, how to recognize completion of asynchronous operations, how to handle cases where data from separate memories need to be combined.
Focus of our research is on software enablement on a wide range of extreme scale hardware.

Structured-parallel 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

Languages: DFGL, OpenMP extensions
Libraries: Habanero-C/C++, Habanero-Java, Habanero-Scala

Compiler Extensions for explicit parallelism: Polyhedral+AST optimizers for C/C++ programs (src-to-src, LLVM), bytecode optimizers for Java

Runtime Extensions for asynchronous tasks, data transfers, heterogeneity: Habanero-C++, Open Community Runtime (OCR), HC-MPI, HJlib cooperative runtime

http://habanero.rice.edu
Habanero Execution Model underlies all our software research

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

Claim: these execution model primitives enable programmability, portability, performance, and debuggability for extreme scale software and hardware
Pedagogy using Habanero execution model

- Sophomore-level CS Course at Rice (http://comp322.rice.edu)
  - Approach: “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
    - Parallel algorithms & data structures including lists, trees, graphs, matrices
- 3-course introductory Coursera specialization on “Parallel, Concurrent and Distributed Programming in Java” launched on 7/31/2017 (http://bit.ly/pcdpjava)
  - Approach: use pedagogy from Rice COMP 322 course to target Java developers who have never before taken a course on parallel programming
- Future plan: create online specialization on parallel programming in C++ targeted to first-year graduate students (Masters & PhD), leveraging concepts from HClib and Kokkos
Example: Habanero abstraction of a CUDA kernel invocation

```plaintext
async at(GPU1)  forall(blockIdx)

async at(GPU2)  forallPhased(threadIdx)
```
Target Platforms

Habanero execution model is being mapped to a wide range of platforms

- Current/past systems
  - Multicore SMPs (IBM, Intel)
  - Discrete GPUs (AMD, NVIDIA), Integrated GPUs (AMD, Intel)
  - FPGA (Convey)
  - HPC Clusters, Hadoop/Spark Clusters
  - Embedded processors: TI Keystone

- Future systems
  - Members of “Rogues Gallery” in Georgia Tech’s Center for Research into Novel Computing Hierarchies (CRNCH): Emu Chick, 3D Stacked Memories w/ FPGAs, Neuromorphic Hardware, …
“Migratory Memory Side Processing” to exploit weak locality.

Data for graph edge attributes, documents, etc. reside nearby even if accessed irregularly.

Moving threads to data on reads means all accesses are local, common case needs to tolerate less latency.
3D Stacked Memories with FPGAs

- Rogues Gallery includes traditional FPGA platforms (Arria 10, Ultrascale +) and stacked memory variants.
- Enables “near-memory” and memory-centric processing.
- FPGA platforms enable prototypes of non-traditional accelerators like Automata, sparse data engines, etc.
- Current work is supported in part by Micron hardware donation.
Outline

- Type system & inference for place locality (Programming Model)
- Communication & Data Layout Optimizations (Compiler)
- Hierarchical Place Tree for Locality, Heterogeneity, and Distribution (Runtime)
Memory-Centric Programming with Places

Memory-Centric Programming with Places

HERE

T p

int x

T f

T g

T q

T r

at (r) r.x

THERE

objects don’t migrate, so migrate the computation over there to read a value
Memory-Centric Programming with Places

```
 at (r)xr.x
 at (q)xq.x
```

```
 int x
 T f
 T g
```

```
 T q
```

```
 T r
```

```
 HERE
```

```
 THERE
```

```
 Place Types
```
Example: Evenly Distributed Binary Tree
Example (contd.)

1. class DarkNode {
2.     DarkNode! left;
3.     DarkNode! right;
4. 
5.     int count () {
6.         return left.count() + 
7.             right.count() + 1;
8.     }
9. }

1. class LightNode {
2.     Node? left;
3.     Node! right;
4. 
5.     int count () {
6.         int l = at (left) left.count();
7.         int r = right.count();
8.         return l + r + 1;
9.     }
10. }
Summary of results

- Type system for places: in a distributed program with *at* statements, ensure that all dereferences are local
  - Based on an equality-based constraint system
  - Therefore admits standard type inference machinery
  - Includes support for distributed arrays

- Algorithm for automatic inference of place types
Outline for rest of talk

- Type system & inference for place locality (Programming Model)

- Communication & Data Layout Optimizations (Compiler)

- Hierarchical Place Tree for Locality, Heterogeneity, and Distribution (Runtime)
Distributed Object Model

- Serialization/deserialization of data for remote activities created using `at` and `async` constructs
  - Objects
    - Only global instance fields (immutable) of an object and the transitive closure of the object graph are serialized and deserialized
    - Serialized objects contain remote references (RR) to the original objects
  - Structs and functions (closures)
    - All the data members and their transitive closures are serialized (since they are implicitly immutable)
Key Contributions

- We introduce high-level compiler optimizations to
  - reduce communication overheads
    - Scalar replacement for global variables and arrays
    - Class splitting
    - Loop splitting to separate local and remote communications
  - reduce synchronization overheads
    - Strip-mining of distributed loops
      - Scalar expansion
      - Async coalescing
Communication Optimization: Scalar Replacement for Global Variables

// 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 ...;
}

// 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 ...;
}
Experimental Results for MolDyn: Execution time in seconds for different numbers of nodes

Summary of speedup due to communication optimization (MolDyn):
- BlueGene/P: 34.46X
- Nehalem: 2.99X
- Power7: 2.73X
Performance depends on data layouts, not just code transformations (CPU examples)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Layout 1</th>
<th>Layout 2</th>
<th>Layout 3</th>
<th>Layout 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM POWER7</td>
<td>27 × 1</td>
<td>9 × 3</td>
<td>3 × 9</td>
<td>1 × 27</td>
</tr>
<tr>
<td>AMD APU</td>
<td>1.00</td>
<td>4.66</td>
<td>4.66</td>
<td>4.71</td>
</tr>
<tr>
<td>Intel Sandybridge</td>
<td>1.00</td>
<td>1.26</td>
<td>1.38</td>
<td>1.40</td>
</tr>
<tr>
<td>IBM BG/Q</td>
<td>1.00</td>
<td>1.06</td>
<td>1.10</td>
<td>1.10</td>
</tr>
</tbody>
</table>

Example: performance of IRSmk kernel relative to default 27x1 layout for different CPUs – opposite trends are seen on GPUs!
Separation of Concerns:

- User specifies data layouts for different architectures in “meta files”
- Automatic Layout Generator can be used to recommend best layouts for different architectures
- Code is unchanged!
## Manual vs. Automated Layout Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Power7 8 Threads</th>
<th>AMD APU 4 Threads</th>
<th>Sandybridge 8 Threads</th>
<th>BG/Q 64 Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRSmk Best Manual Layout</td>
<td>4.70x</td>
<td>1.46x</td>
<td>1.11x</td>
<td>2.20x</td>
</tr>
<tr>
<td>IRSmk Automated Layout</td>
<td>4.67x</td>
<td>1.43x</td>
<td>1.10x</td>
<td>2.08x</td>
</tr>
<tr>
<td>LULESH Best Manual Layout</td>
<td>1.43x</td>
<td>1.50x</td>
<td>1.02x</td>
<td>1.10x</td>
</tr>
<tr>
<td>LULESH Automated Layout</td>
<td>1.58x</td>
<td>1.46x</td>
<td>0.96x</td>
<td>1.07x</td>
</tr>
<tr>
<td>SRAD Best Manual Layout</td>
<td>1.35x</td>
<td>3.13x</td>
<td>1.00x</td>
<td>1.08x</td>
</tr>
<tr>
<td>SRAD Automated Layout</td>
<td>1.20x</td>
<td>2.55x</td>
<td>0.46x</td>
<td>0.98x</td>
</tr>
</tbody>
</table>

*Speedup of Best Manual and Automated Layout relative to base layout*
Automatic Data Layout Problem for CPU+GPU architectures

- Meta-data framework limits a single data-layout for the entire program

- Large programs with multiple kernels might require different layouts for different parts of the program
  - The best data layout could be a single layout for the entire program or multiple layouts for each kernel with data-remapping in between

“Automatic Data Layout Generation and Kernel Mapping for CPU+GPU Architectures.”
Deepak Majeti, Kuldeep Meel, Raj Barik and Vivek Sarkar. CC 2016.
Illustrative Example

```c
struct ABCD{
float x; float y; float z; float w;
};
float *x, *y, *z, *w, *e;
init(x, y, z, w, e);

//Kernel-1 on GPU with AoS layout: 5.3 msec
//Kernel-1 on GPU with SoA layout: 11.5 msec
finish forasync point(j) range(0:M) at(dev){
  if(.....){
    e[j] = ((x[j] + y[j] + z[j]) / w[j])
      + ((x[j+1] + y[j+1] + z[j+1]) / w[j+1])
      + ((x[j+2] + y[j+2] + z[j+2]) / w[j+2])
      + ((x[j+3] + y[j+3] + z[j+3]) / w[j+3]);
  }
}

//Remap from AoS to SoA: 3.3 msec
remap(xyzw, x, y, z, w);

//Kernel-2 on GPU with AoS layout: 8.4 msec
//Kernel-2 on GPU with SoA layout: 3.0 msec
finish forasync point(j) range(0:M) at(dev){
  x[j] = (y[j] + e[j] * 1.432);
}
```

AoS: 5.3 + 8.4 = 13.7 msec
SoA: 11.5 + 3.0 = 14.5 msec
Best: AoS + Remap + SoA:
5.3 + 3.3 + 3.0 = 11.6 msec
**ADHA SDL Results**

Y Axis is speedup compared to default SoA layout

Kernels are statically mapped to CPU and GPU

**CPU: Intel_CPU_X5660**

**GPU: NVIDIA_GPU_M2050**
ADHA PDL Results

Kernels are statically mapped to CPU and GPU

Y Axis is overall speedup

Medical: 1.51 1.41
LBM: 1 1.2
K-Means: 2.7 2.69
NBody: 1.2 1.23
GEOMEAN: 1.49 1.54

CPU: Intel_CPU_X5660
GPU: NVIDIA_GPU_M2050
Outline for rest of talk

- Type system & inference for place locality (Programming Model)
- Communication & Data Layout Optimizations (Compiler)
- Hierarchical Place Tree for Locality, Heterogeneity, and Distribution (Runtime)
Load Balancing with Work-Stealing Schedulers

Urgent need for extensions that support locality with work-stealing

Place 0

Place 1

Place N-1
Distributed Work-Stealing

**Inter-node** steals are much costlier than **intra-node** steals

Intra-node steals

Inter-node steals

Optimized Distributed Work-Stealing. Vivek Kumar, Karthik Murthy, Vivek Sarkar and Yili Zheng. IA^3 2016
Our Approach

- Use HabaneroUPC++ PGAS library for multicore cluster [Kumar et. al., PGAS 2014]
  - Several asynchronous tasking APIs
- Provide a programming model to express irregular computation
- Implement a high performance distributed work-stealing runtime that **completely removes** all inter-node failed steal attempts
HabaneroUPC++ Programming Model

asyncAny ( [=] { 

irregular_computation();

}); //distributed work-stealing

- C++11 lambda-function based API
- Provides **serial elision** and improves productivity
Distributed Work-Stealing Runtime

- Two different implementations in HabaneroUPC++
- BaselineWS
  - Uses prior work + some optimizations
- SuccessOnlyWS
  - Extends BaselineWS by using a novel victim selection policy that completely removes all inter-node failed steals
- See paper for details
Locality-aware Scheduling for CPUs using the Hierarchical Place Tree (HPT) abstraction

- Model target system as a tree/hierarchy
- Workers attached to leaf places
- Each place has a queue
  - async at(<pl>) <stmt>: push task on place pl’s queue
  - Destination place can be determined by programmer, compiler or runtime
- Current policy (can also be applied to distributed work-stealing)
  - 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 its subtree
      - Task in PL2 can be executed by workers W2 or W3

Hierarchical Place Trees: A Portable Abstraction for Task Parallelism and Date Movement.
Yonghong Yan, Jisheng Zhao, Yi Guo, Vivek Sarkar. LCPC 2009.
Extending the HPT abstraction for heterogeneous architectures & accelerators

- Devices (GPU or FPGA) are represented as memory module places and agent workers
  - GPU memory configuration is fixed, while FPGA memory can be reconfigured at runtime
- async at(P) S
  - Creates new task to execute statement S at place P
Medical imaging applications (NSF Expeditions 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 and energy efficiency improvements
  - real-time clinical applications and simulations using patient imaging data

Figure credit: NSF Expeditions CDSC project
<table>
<thead>
<tr>
<th>Domain-specific modeling</th>
<th>UCLA</th>
<th>Rice</th>
<th>UCSB</th>
<th>Ohio State</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHP creation</td>
<td>Bui, Reinman, Potkonjak</td>
<td>Sarkar, Baraniuk</td>
<td>Cheng</td>
<td>Sadayappan</td>
</tr>
<tr>
<td>CHP mapping</td>
<td>Chang, Cong, Reinman</td>
<td>Sarkar</td>
<td>Cheng</td>
<td>Sadayappan</td>
</tr>
<tr>
<td>Application drivers</td>
<td>Cong, Palsberg, Potkonjak</td>
<td>Baraniuk</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Experimental systems</td>
<td>Aberle, Bui, Chien, Hsu, Vese</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>

All (led by Cong & Bui)
Adding Affinity Annotations for Heterogeneous Computing to CnC-inspired Dataflow Model

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

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

- Steps are compiled for execution on CPU, GPU or FPGA
  - Aim for single-source multi-target compilation!
- Designate a CPU core as a proxy worker for heterogeneous device
- Device inbox is now a concurrent queue and tasks can be stolen by CPU or other proxy workers

Device tasks created from CPU worker via

`async at(gpl) IN() OUT() { ... }`

Continuations stolen by CPU workers

Device tasks stolen by CPU and other device workers
Convey HC-1ex Testbed

“Commodity” Intel Server

Intel® Xeon® Processor

Intel® Memory Controller Hub (MCH)

Intel® I/O Subsystem

Standard Intel® x86-64 Server

x86-64 Linux

Convey FPGA-based coprocessor

Application Engine Hub (AEH)

Application Engines (AEs)

XC6vlx760 FPGAs
80GB/s off-chip bandwidth
94W Design Power

Memory

Convey coprocessor
FPGA-based
Shared cache-coherent memory

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

Xeon Quad Core LV5408
40W TDP

46
Static vs Dynamic Scheduling

- Static Schedule

- Dynamic Schedule
Experimental Results for Medical Imaging Workload

- Execution times and active energy for dynamic vs. static scheduling on heterogeneous processors

![Bar chart showing execution times and active energy for different configurations of processors.](chart.png)
Multi-Node SmithWaterman using Data-Driven Tasks

1. `#define DDF_HOME(guid) . . .`
2. . . .
3. `for (i=0;i<H;++i)`
4. `for (j=0;j<W;++j)`
5. `matrix[i][j] = DDF_HANDLE(i*H+j);`
6. . . .
7. `finish { // matrix is a 2-D array of DDFs`
8. `for (i=0,i<H;++i) {`
9. `for (j=0,j<W;++j) {`
10. `DDF_t* curr = matrix[i][j];`
11. `DDF_t* above = matrix[i-1][j];`
12. `DDF_t* left = matrix[i][j-1];`
13. `DDF_t* uLeft = matrix[i-1][j-1];`
14. `async AWAIT (above, left, uLeft){`
15. `Elem* currElem = . . .`
16. `DDF_PUT(curr, currElem);`
17. `} /*async*/ } /*for-j*/ } /*for-i*/`
18. `} /*finish*/`
Integrating Inter-node Communication with Intra-node Task Runtime Systems (HCMPI)

Example:

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

Results for APGNS version of SmithWaterman (communication runtime uses MPI under the covers)
UTS Performance on T1XXL

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

<table>
<thead>
<tr>
<th>Nodes</th>
<th>2 cores/node</th>
<th>4 cores/node</th>
<th>8 cores/node</th>
<th>16 cores/node</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.67</td>
<td>1.00</td>
<td>1.17</td>
<td>1.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.67</td>
<td>1.00</td>
<td>1.17</td>
<td>1.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.67</td>
<td>1.00</td>
<td>1.17</td>
<td>1.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0.67</td>
<td>1.00</td>
<td>1.17</td>
<td>1.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>0.68</td>
<td>1.01</td>
<td>1.20</td>
<td>1.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>0.68</td>
<td>1.03</td>
<td>1.29</td>
<td>1.51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>0.69</td>
<td>1.10</td>
<td>1.66</td>
<td>1.98</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>0.73</td>
<td>1.33</td>
<td>4.50</td>
<td>5.76</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>22.31</td>
<td>22.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HiPER Platform Model (generalization of HPT)

An undirected graph of places across which to distribute work.

Max Grossman, Vivek Kumar, Nick Vrvilo, Zoran Budimlic, Vivek Sarkar. AsHES’17.
HiPER Pluggable Modules

Platform model & generalized work-stealing handle work discovery and execution.

Modules add user APIs that spawn tasks on the platform model, (work creation).

Built-in system module adds basic task-parallel, future-based, etc. APIs.

Developers add custom modules for third-party software packages (CUDA, MPI, UPC++, and OpenSHMEM).
HiPER Generalized Work-Stealing

Generalized work-stealing controls work discovery on the platform model.

Consists of:
• Persistent thread pool
• User-defined pop, steal paths
Conclusions

- Habanero execution model enables multiple levels of parallelism and heterogeneity by bridging from programming models to extreme scale systems.
- Well-designed execution model primitives can enable synergistic innovation across all levels of software+hardware stack.
- Habanero group is gearing up to address programming model, compiler and runtime system challenges for memory-centric processors, as exemplified by the CRNCH Rogues Gallery.