Bridging Parallel and Reconfigurable Computing with Multilevel PGAS and SHMEM+

CHREC
NSF Center for High-Performance Reconfigurable Computing

HPRCTA 2009

UF UNIVERSITY of FLORIDA
Virginia Tech VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
BYU BRIGHAM YOUNG UNIVERSITY
THE GEORGE WASHINGTON UNIVERSITY

Vikas Aggarwal
Alan D. George
Kishore Yalamanchalli
Changil Yoon
Herman Lam
Greg Stitt

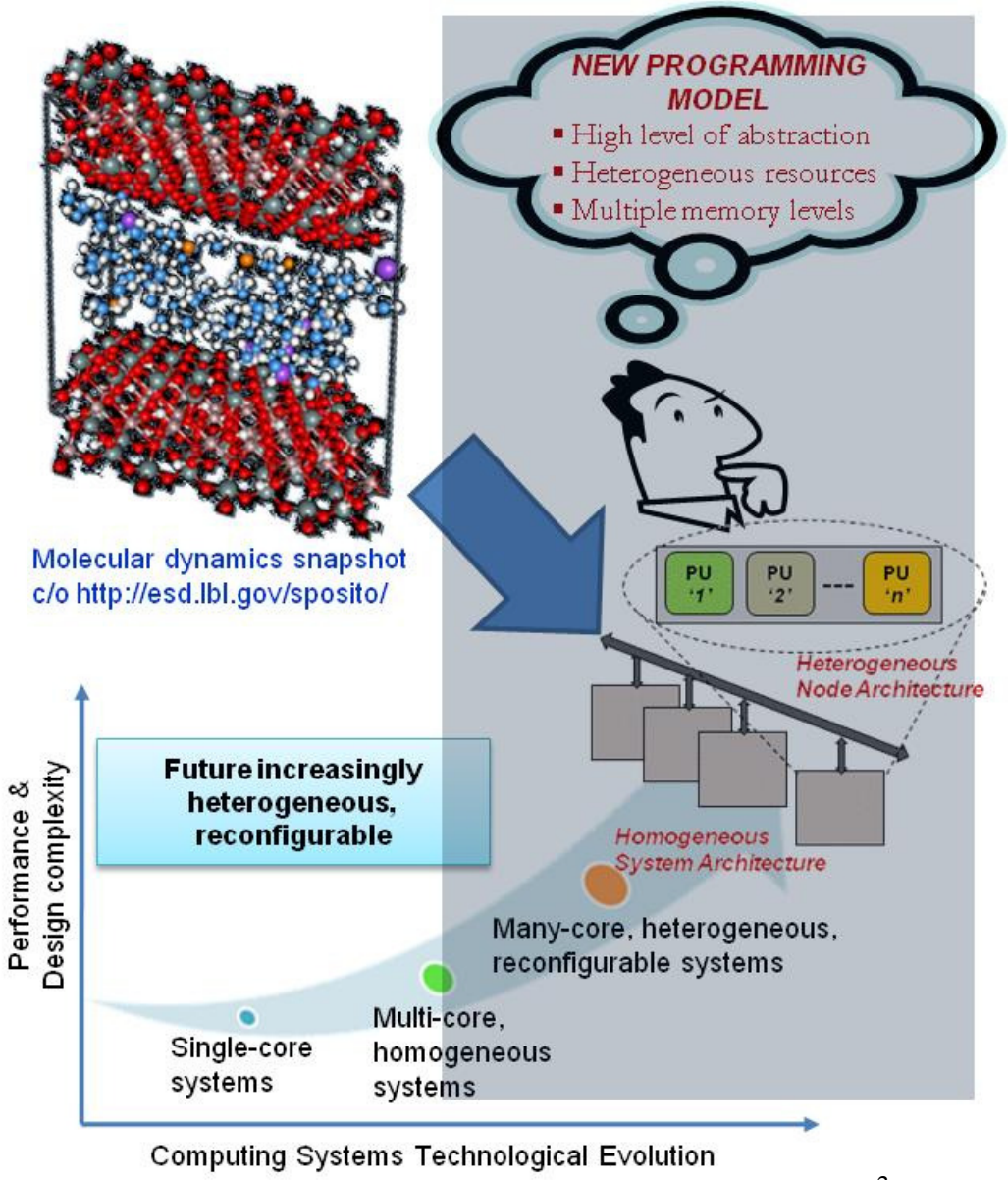
NSF CHREC Center
ECE Department, University of Florida

30 September 2009

Outline

- Introduction
  - Motivations
  - Background
- Approach
  - *Multilevel* PGAS model
  - SHMEM+
- Experimental Testbed and Results
  - Performance benchmarking
  - Case study: content-based image recognition (CBIR)
- Conclusions & Future Work

Motivations



RC systems offer much superior performance

- 10x to 1000x higher application speed, lower energy consumption

Characteristic differences from traditional HPC systems

- Multiple levels of memory hierarchy
- Heterogeneous execution contexts

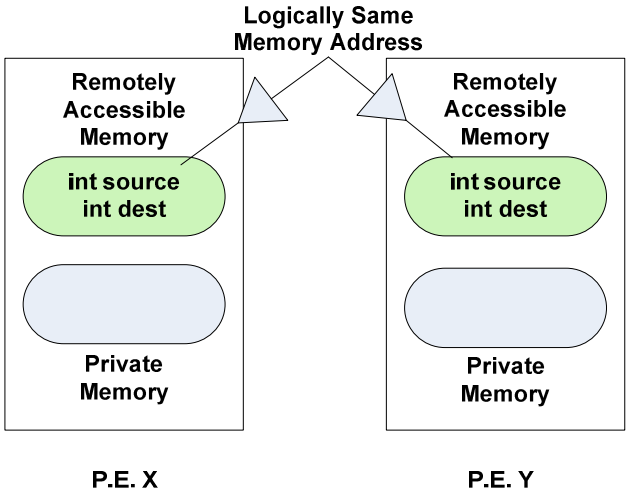
Lack of integrated, system-wide, parallel-programming models

- HLLs for RC do not address scalability/multi-node designs
- Existing parallel models insufficient; fail to address needs of RC systems
- Productivity for scalable, parallel, RC applications very low

Background

- Traditionally HPC applications developed using
  - Message-passing models, e.g. MPI, PVM, etc.
  - Shared-memory models, e.g. OpenMP, etc.
  - More recently, PGAS models, e.g. UPC, SHMEM, etc.
    - Extend memory hierarchy to include high-level global memory layer, partitioned between multiple nodes
- PGAS has common goals & concepts
  - Requisite syntax, and semantics to meet needs of coordination for reconfigurable HPC systems
  - SHMEM: Shared MEMory comm. library
- However, needs adaptation and extension for reconfigurable HPC systems
  - Introduce *multilevel* PGAS and SHMEM+

Background: SHMEM



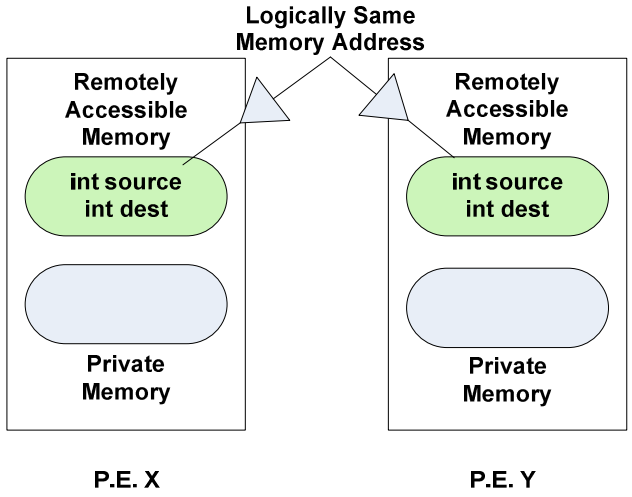
Shared and local variables in SHMEM

Why program using SHMEM

- Based on SPMD; easier to program in than MPI (or PVM)
- Low latency, high bandwidth one-sided **data transfers** (*put*s and *get*s)
- Provides **synchronization** mechanisms
  - Barrier
  - Fence, quiet
- Provides efficient **collective communication**
  - Broadcast
  - Collection
  - Reduction

Background: SHMEM

Array copy example



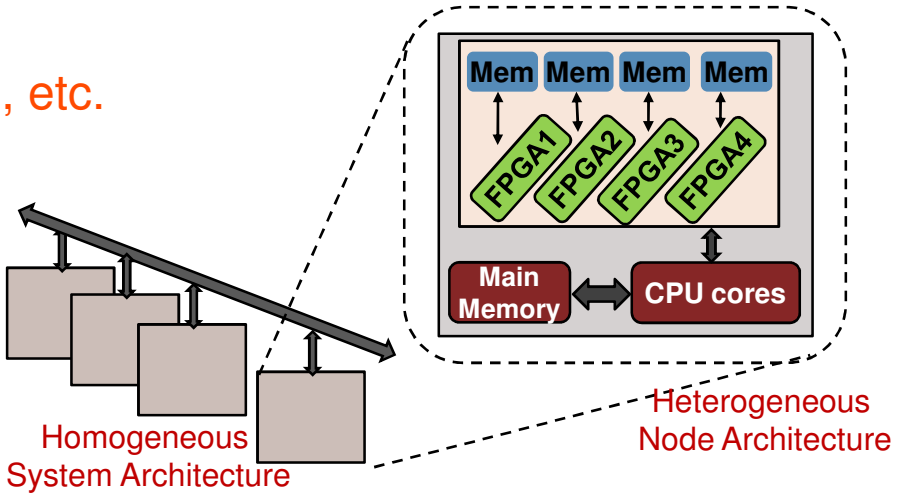

```
1. #include <stdio.h>
2. #include <shmem.h>
3. #include <intrinsics.h>
4.
6. int me, npes, i;
7. int *source, *dest;
8. main()
9. {
10. shmem_init();
11. /* Get PE information */
12. me = my_pe();
13. source = shmalloc(4*8);
14. dest = shmalloc(4*8);
```

```
15. /* Initialize and send on PE 1 */
16. if(me == 1) {
17. for(i=0; i<8; i++) source[i] = i+1;
18. /* put source data at PE1 to dest at PE0*/
19. shmem_putmem(dest, source, 8*sizeof(dest[0]), 0);
20. }

21. /* Make sure the transfer is complete */
22. shmem_barrier_all();
23.
24. /* Print from the receiving PE */
25. if(me == 0) {
26. printf(" DEST ON PE 0:");
27. for(i=0; i<8; i++)
28.       printf(" %d%c", dest[i], (i<7) ? ',' : '\n');
29. }}
```

Challenges/Issues

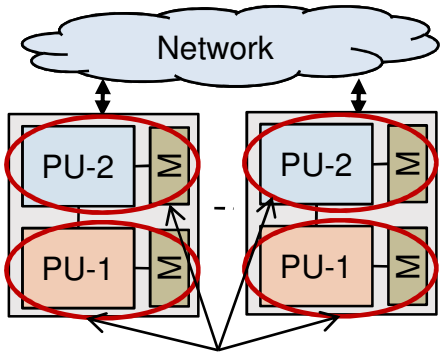
System architecture of a typical RC machine



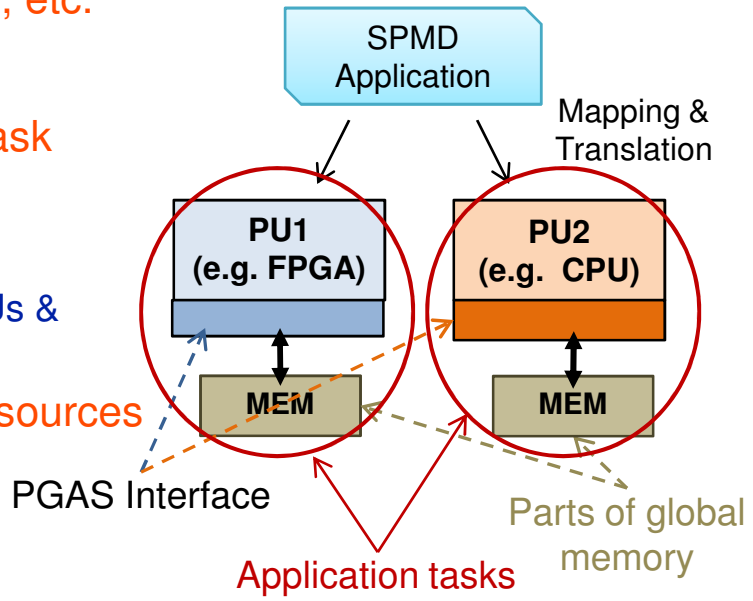
- Next-generation HPC systems
  - Multi-core CPUs, FPGA devices, Cell, etc.
  - Multiple levels becoming increasingly visible for comm., memory
- Challenges involved in providing PGAS abstraction
  - Semantics (PGAS, SPMD) in future computing systems
    - Traditional models/semantics fail
    - Redefine meaning & structure in multilevel heterogeneous system
    - What part of system arch. (node, device, core) is each SPMD task?
    - Where does PGAS exists in multitier memory arch.?
  - Interface abstraction
    - How do different computational resources provide abstraction for PGAS?

Approach – Model 1

- Each processing unit (PU) in a node executes a separate task of SPMD application
  - SPMD tasks translated to logically equivalent operations in different programming paradigms
- PGAS interface on each task presents a logically homogenous system view
  - PUs physically heterogeneous; e.g. CPUs, FPGAs, etc.
- Challenges:
  - PUs need to support complete functionality of its task
    - Provide PGAS interface
  - Account for different execution contexts
    - e.g. Notion & access patterns of data memory in CPUs & FPGAs are very different
  - Devices require independent access to network resources
  - Inefficient utilization of computing resources
    - FPGAs built for specific functionality

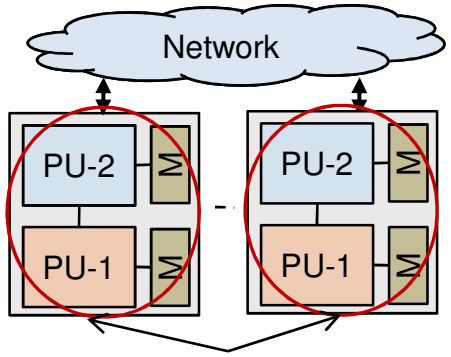


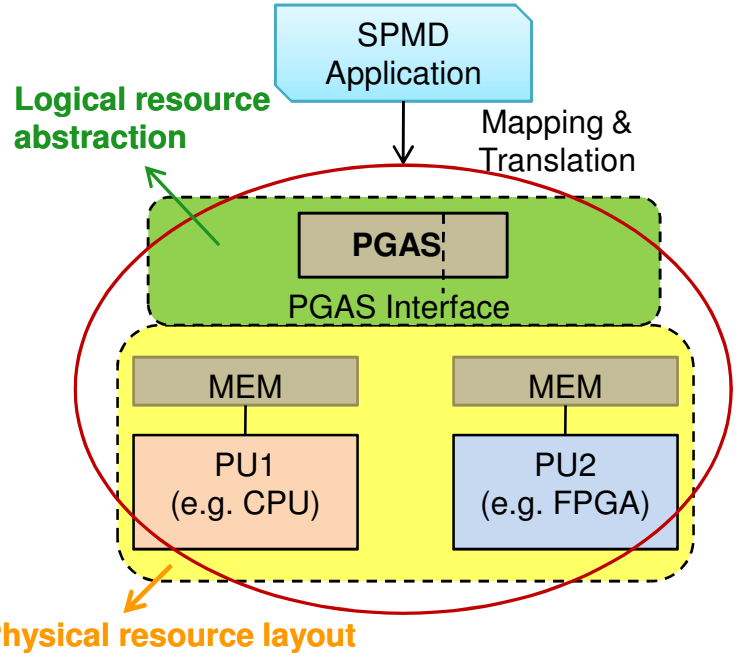
Independent tasks of parallel application



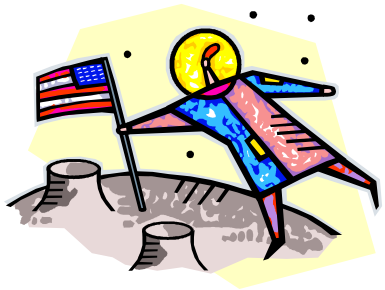
Approach – Model 2

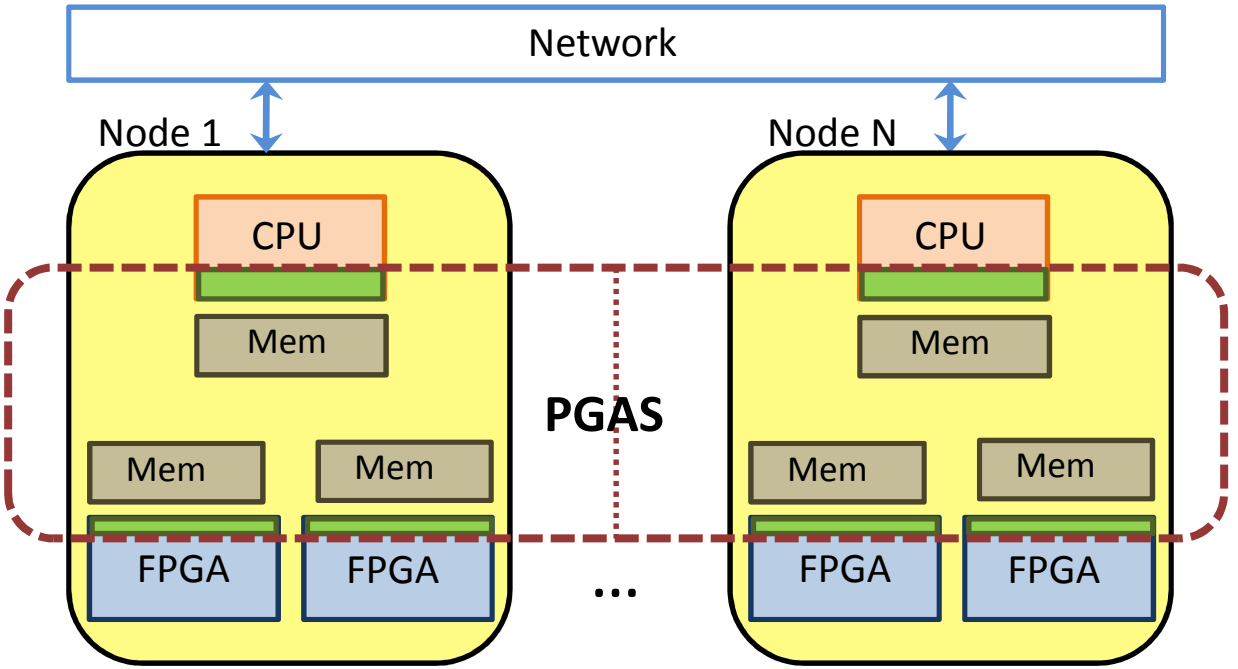
- Multiple PUs per node collectively execute a task
  - SPMD task HW/SW partitioned across computing resources within a node
- PGAS interface allows homogeneous system view
  - Goal is to presents logically unified memory abstraction
    - Shared memory physically split across multiple PUs in every node
- Challenges:
  - Providing virtual/logical unified abstraction of memory to users
    - Hide details of actual/physical distribution of resources
  - Distribute responsibilities of PGAS interface over multiple PUs in a node





Multilevel PGAS

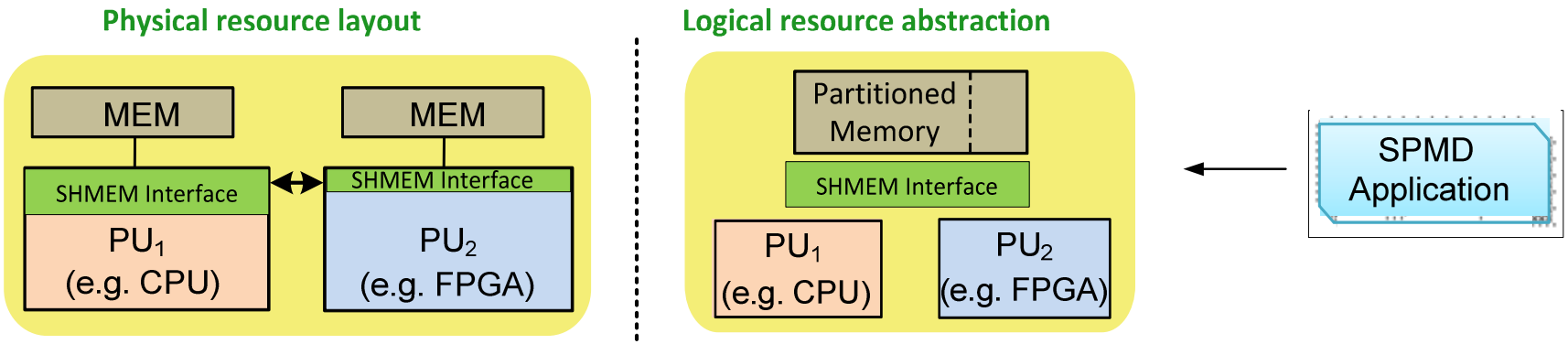




Distribution of resources in *multilevel* PGAS

- Model 2 seems to make most sense for RC systems
  - Integrates different levels of memory hierarchy into a single, partitioned, global address space
  - Each task of SPMD application executes over a mix of CPUs and FPGAs

Multilevel PGAS (contd.)

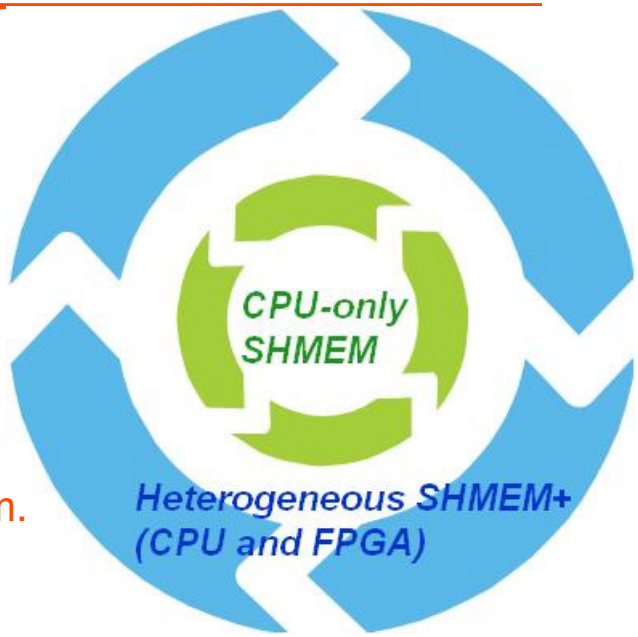


Intra-node view: Abstraction provided by *multilevel* PGAS

- Simplified logical abstraction to developer
  - PGAS physically distributed, but logically integrated
  - PUs collectively provide PGAS interface and its functionality
- Additional significant features:
  - Affinity of data to memory blocks
  - Explicit transfers between local memories

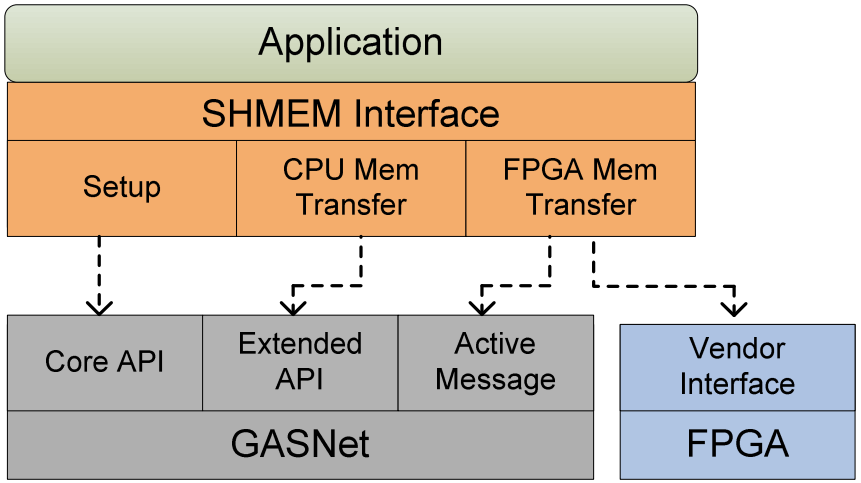
SHMEM+

Extended SHMEM for next-gen., heterogeneous, reconfigurable HPC systems



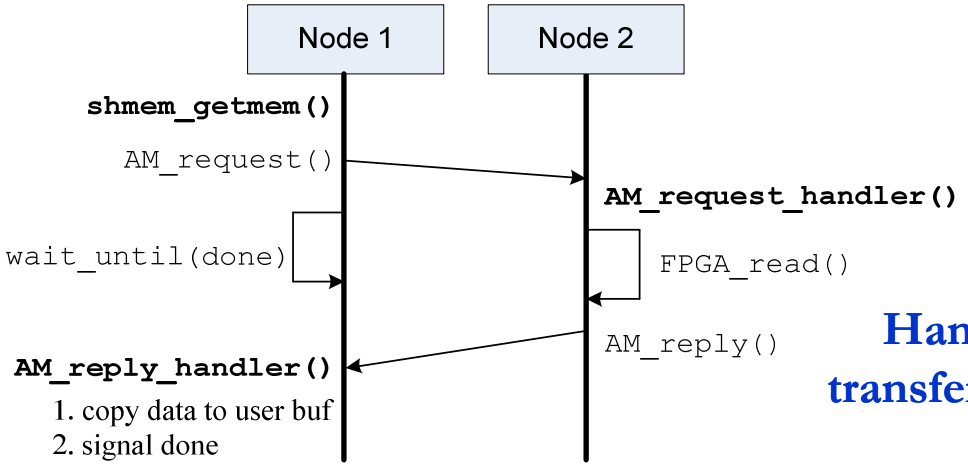
- SHMEM stands out as a promising candidate amongst PGAS-based languages/libraries
  - Simple, lightweight, function-based, fast one-sided comm.
  - Growth in interest in HPC community
- SHMEM+: First known SHMEM library that allows coordination between CPUs and FPGAs
  - Enables design of scalable RC applications without sacrificing productivity
- SHMEM+ is built over services provided by GASNet middleware from UC Berkeley and LBNL
  - GASNet is a lightweight, network-independent, language-independent, communication middleware
  - As a result, SHMEM+ offers high-degree of portability

SHMEM+ Software Architecture



- Various GASNet services are employed to provide underlying communication
  - Setup functions based on GASNet Core API
  - Data transfer to/from remote CPU's memory employs GASNet Extended API
    - Benefit from direct support for such high-level operations in GASNet
  - Data transfer to/from remote FPGA's memory employs Active Message (AM) services
    - Message handlers for AMs use FPGA vendor's APIs to communicate with FPGA board

SHMEM+: FPGA data-transfer



Handshake involved in transfers from remote FPGA

- All transfers to/from FPGA memory initiated by CPU (currently)
  - Based on medium AM's request-response transactions
  - Message handlers employ FPGA_read/write functions
    - Wrap vendor specific API to communicate with FPGA board into more generic read/write functions
  - Packetization required for large data transfers
    - FPGA transfers expected to have higher lat. & lower BW compared to CPU transfers
- FPGA-initiated transfers? (focus of future work)
  - Provide hardware engines to allow HDL applications to initiate transfers
  - Engines could interact with CPU to perform transfers over network

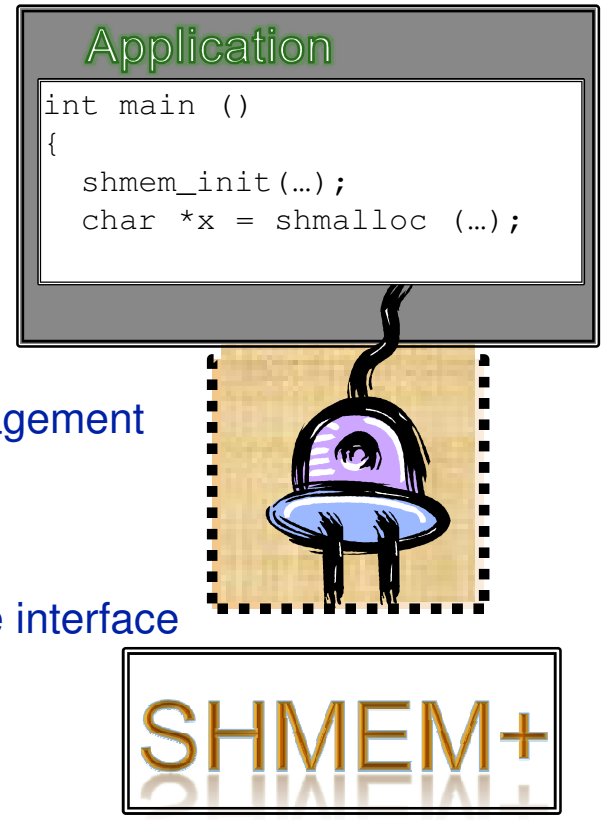
SHMEM+ Baseline Functions

Function	SHMEM Call	Type	Purpose
Initialization	*shmem_init*	Setup	Initializes a process to use SHMEM library
Communication rank	*my_pe*	Setup	Provides a unique node number for each process
Communication size	*num_pes*	Setup	Provides the number of PEs in the system
Finalize	*shmem_finalize*	Setup	De-allocates resources and gracefully terminates
Malloc	*shmalloc*	Setup	Allocates memory for shared variables
Get	*shmem_int_g*	P2P	Read single integer element from remote/local PE
Put	*shmem_int_p*	P2P	Write single integer element from remote /local PE
Get	*shmem_getmem*	P2P	Read any contiguous data type from a remote/local PE
Put	*shmem_putmem*	P2P	Write any contiguous data type to a remote/local PE
Barrier Synchronization	*shmem_barrier_all*	Collective	Synchronize all the nodes together.

- Focus first on basic functionality
  - 10 functions (5 setup, 4 point-to-point, 1 collective)
  - Focus on blocking communications (dominant mode in most applications)
  - Current emphasis on CPU-initiated data transfers

SHMEM+ Interface

- Objective: keep interface simple, familiar
- Extend functionality where needed
  - e.g. shmem_init(…)
    - Performs FPGA initialization and FPGA memory management
    - No change in user interface
  - Data transfer routines
    - Transfer to both local and remote FPGA through same interface



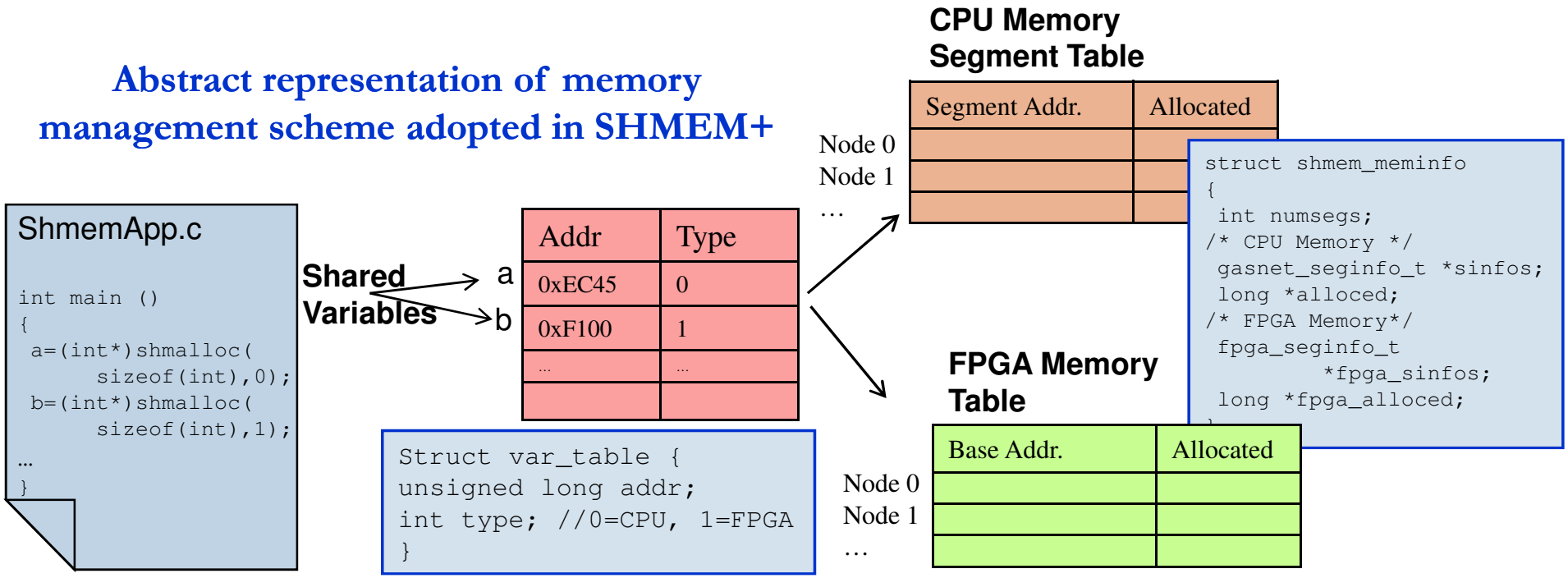
- Minor modifications
  - e.g. shmalloc(…)
    - Allows developers to specify affinity to memory blocks

```
void *shmalloc(size_t sz)
```

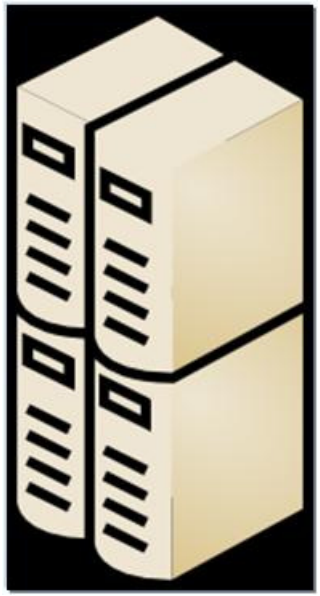
```
void *shmalloc(size_t sz, int type)
Type: 0=CPU/1=FPGA, etc.
```

Shared-Memory Management

- Developers oblivious to physical location of variables
  - Variable lookup table maintains location of each variable
- Memory management tables
  - Maintains information about remote node's CPU and FPGA memory

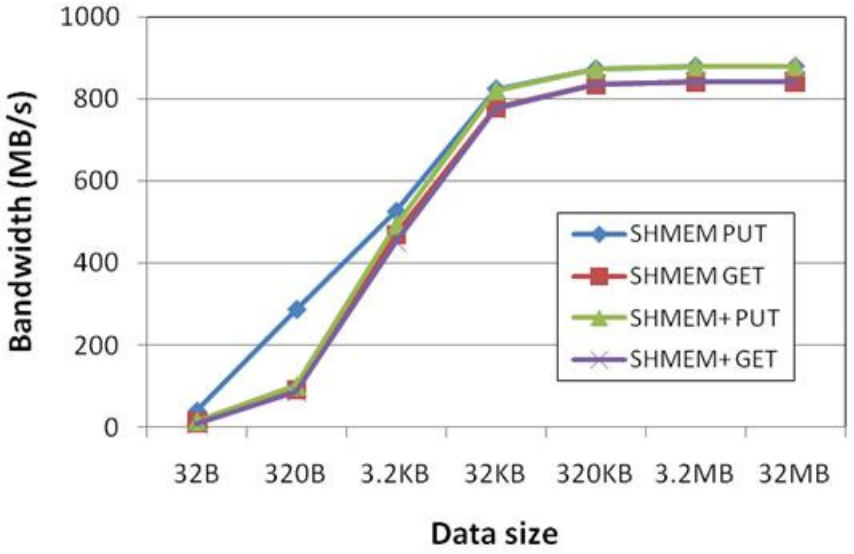


Experimental Results



- Experimental testbed
  - Four server nodes connected via QsNet^{II} interconnect technology from Quadrics
  - Each node: AMD 2GHz Opteron 246 processor and a GiDEL PROCStar-III FPGA board in a PCI-e ×8 slot
  - Each FPGA board: four Altera Stratix-III EP3SE260 FPGAs, each with two external DDR memory modules of 2GB and one on-board 256MB DDR2 SDRAM bank
  - We currently employ only one FPGA per board, though our design is extensible to more

SHMEM+: Performance



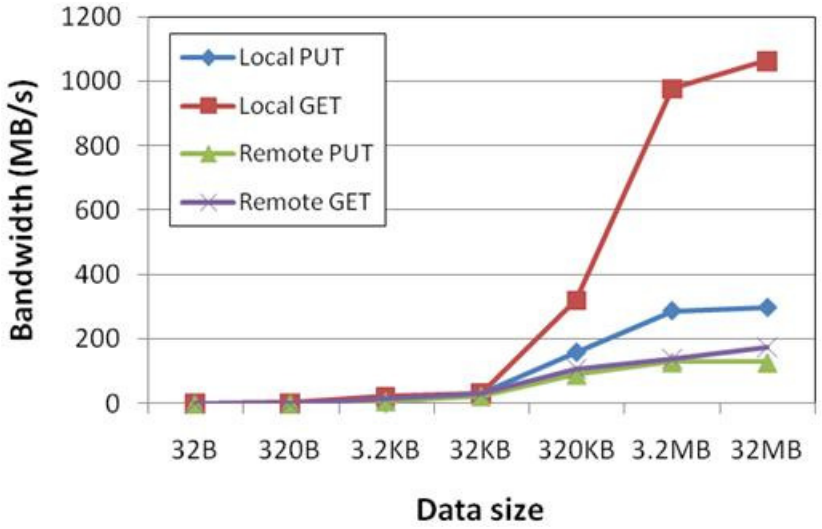
SHMEM+: Prototype of our extended SHMEM over GASNet

SHMEM : Proprietary version of SHMEM over Quadrics (CPU-only)

Transfers between two CPUs

- Performance on par with proprietary, vendor version of SHMEM
  - Benefit from direct support provided by GASNet
  - Minimal overheads
  - Throughput peaks at ~ 890 MB/s

SHMEM+: Performance (contd.)



Transfers between CPU and FPGA

Local :Transfers between a host CPU and its local FPGA on the same node

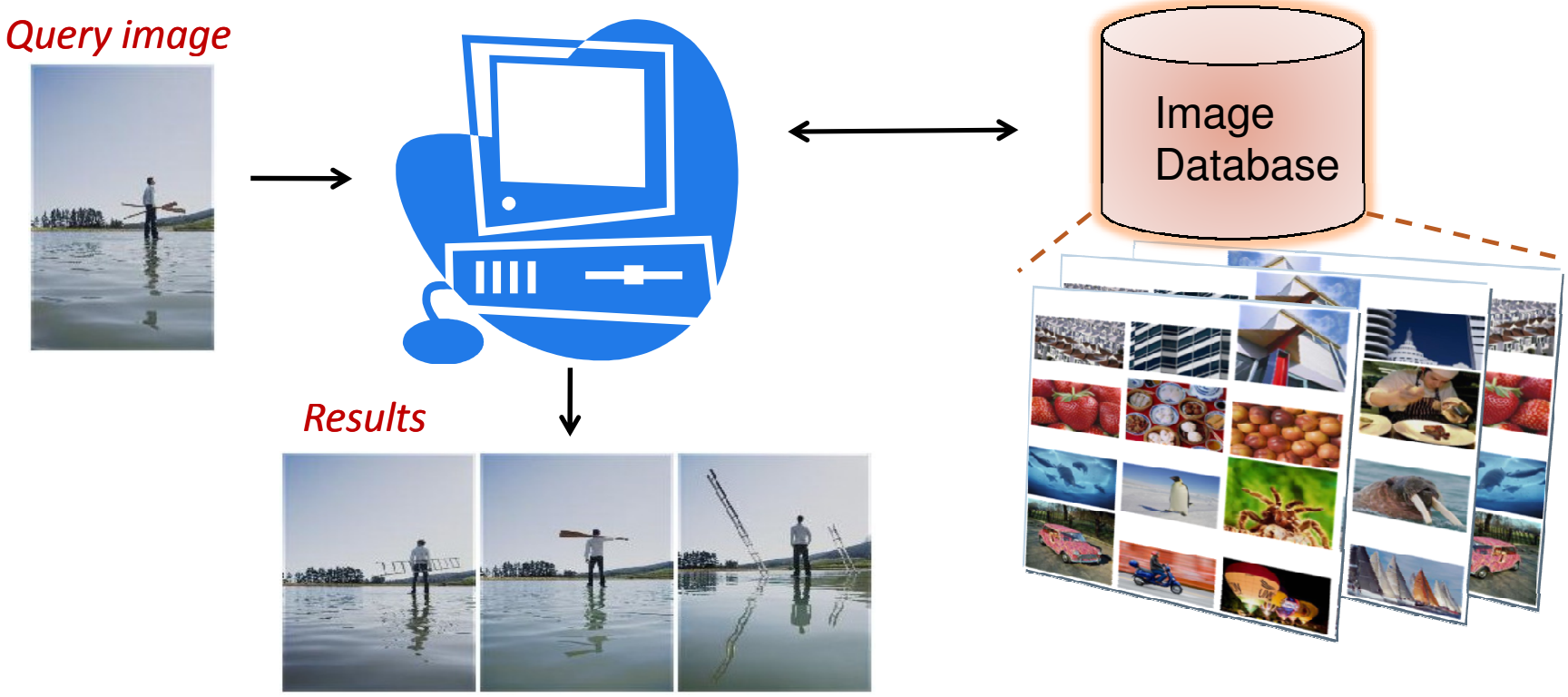
Remote :Transfers between a CPU and an FPGA on different node

All cases are CPU-initiated transfers

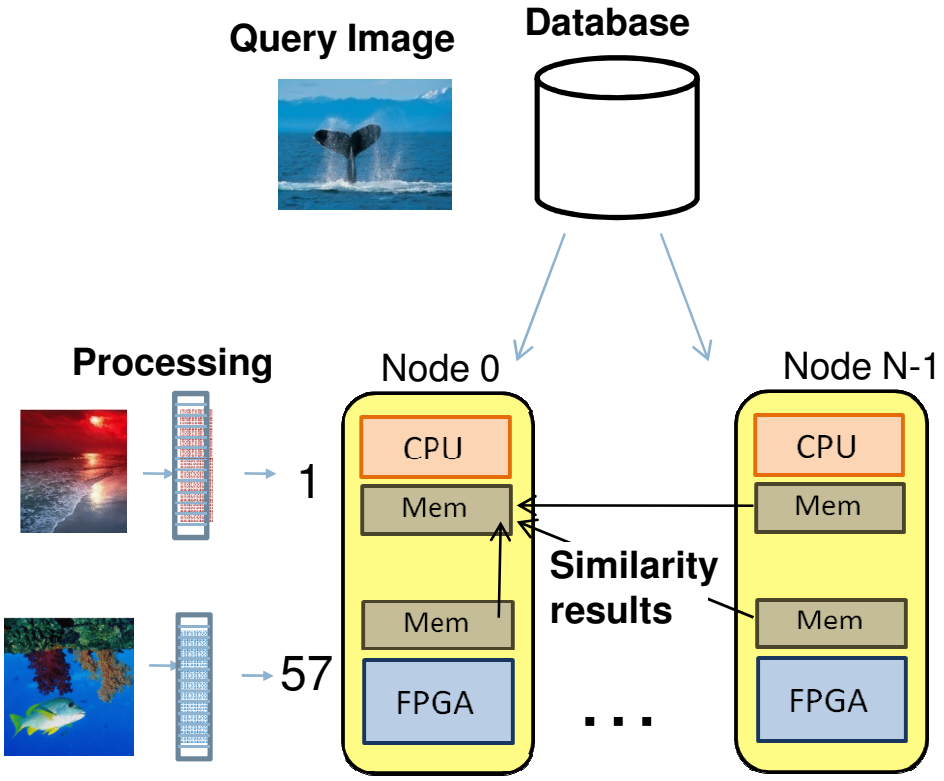
- Throughput peaks above 1GB/s for CPU read (get) from its local FPGA
- Lower performance for local writes and remote transfers
- Performance of transfers to/from FPGA dependent on FPGA board
  - Interconnect between CPU and FPGA (e.g. PCIe, PCI-X, PCI, HT, QPI, etc.)
  - Efficiency of communication controller on FPGA
  - Higher throughput for read ops; common characteristic for most RC systems

Case Study: CBIR Application

- Search a database of images for ones that are visually similar to input image
- Widely used in biomedicine, military, commerce, education, web image classification and many more.



Case Study: CBIR Algorithm



Read set of input images and feature vector of query image

↓

Transfer a subset of images to FPGA using *shmem_putmem,* for hardware acceleration

↓

All PUs compute feature vectors and similarity values

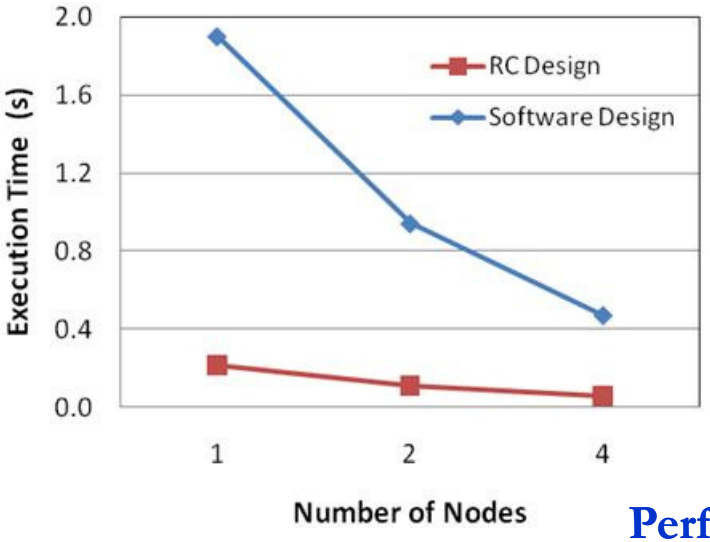
↓

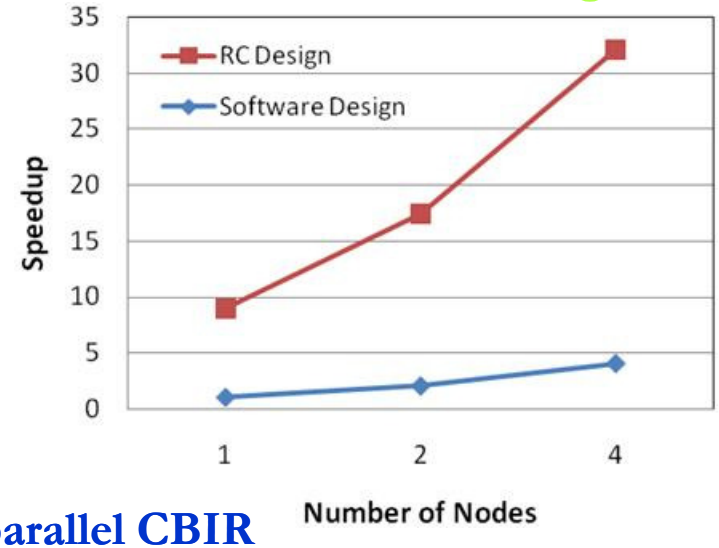
All nodes synchronize using *shmem_barrier_all*

Node 0, reads results from all the PUs Using *shmem_getmem,* and sorts them based on similarity values ↵

CBIR: Results

SW design: CPU devices only
RC design: Both CPU and FPGA devices

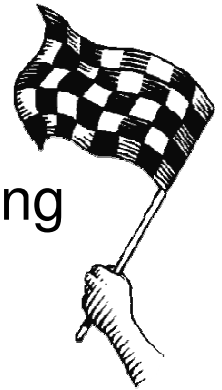




Performance of parallel CBIR application with SHMEM+

- Over 30× speedup using FPGA and CPU devices vs. serial S/W design
- More importantly,
  - SHMEM+ simplified design of scalable, parallel RC apps.
  - Integrated view of system, high level of abstraction led to increased productivity
  - Also enhances application portability: vendor-specific APIs are hidden

Conclusions & Future Work



- *Multilevel* PGAS programming model stands out as promising candidate for reconfigurable HPC
  - High-level abstraction of *multilevel* PGAS improves productivity
- SHMEM+: First version of SHMEM that allows coordination between CPUs and FPGAs
  - Enables development of scalable, parallel RC applications
  - Provides mechanisms akin to traditional methods for parallel app. development
  - Improves application portability
- Future Work
  - Scalability study on new Novo-G RC supercomputer
  - Investigate and explore FPGA-initiated transfers
  - Develop tools to support performance analysis for SHMEM+ apps

Acknowledgements

- NSF I/UCRC Program (Grant EEC-0642422), CHREC members
- Special thanks to ORNL for their ideas and support that made this work possible
- Rafael Garcia, M.S. Student, CHREC lab
- GiDEL for their tools and equipment

Questions