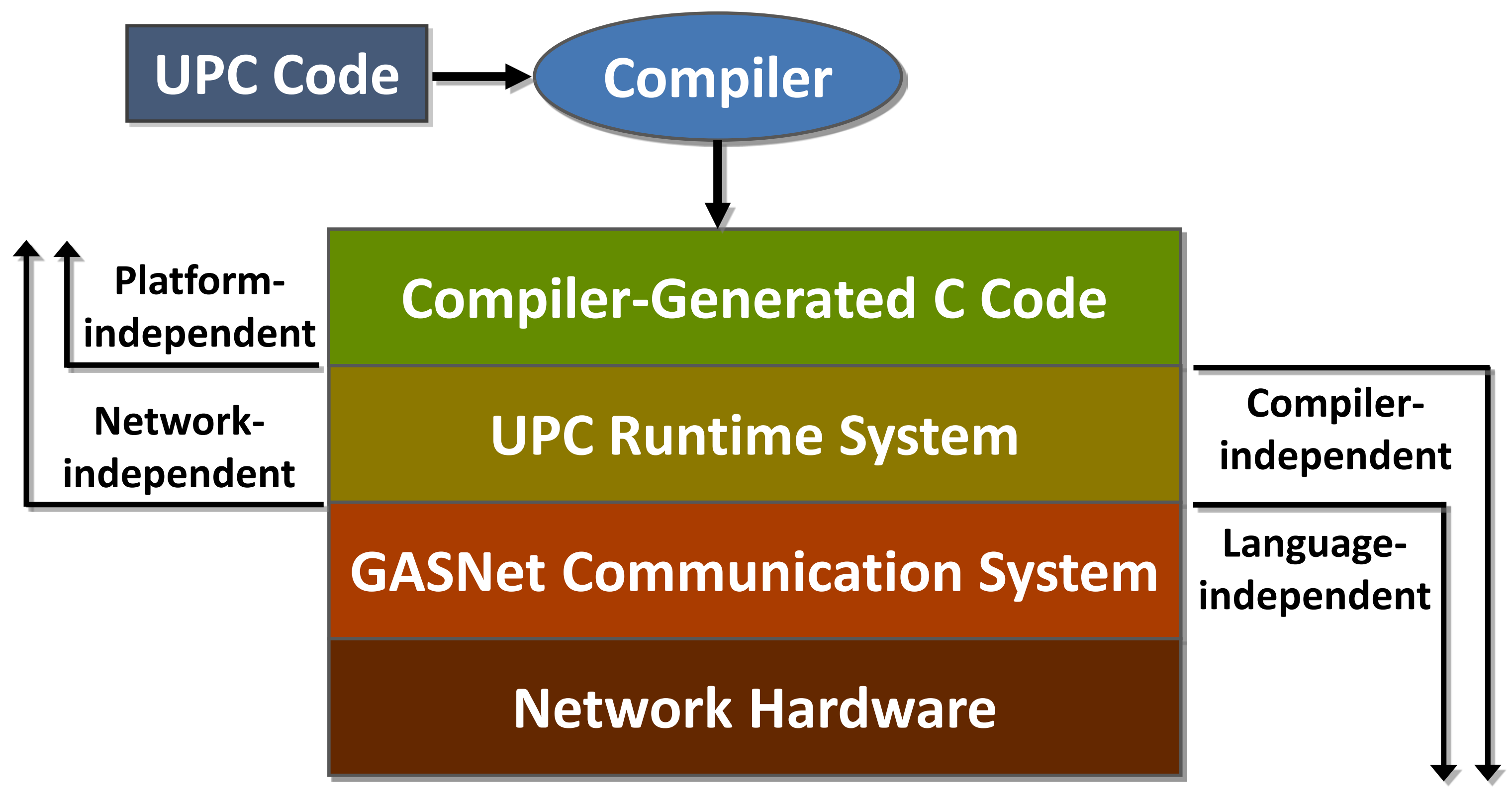
UPC at Berkeley

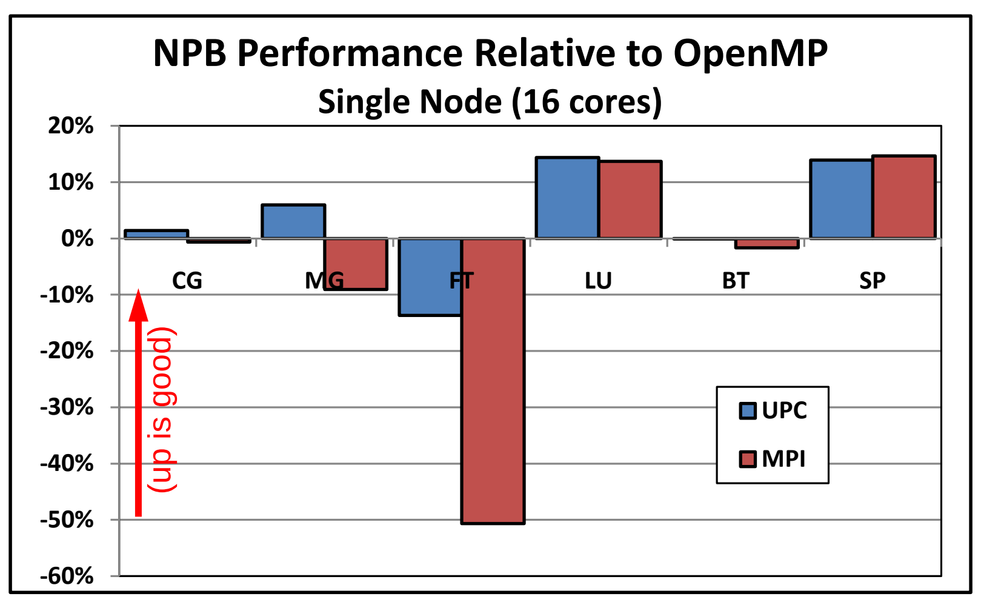
http://upc.lbl.gov

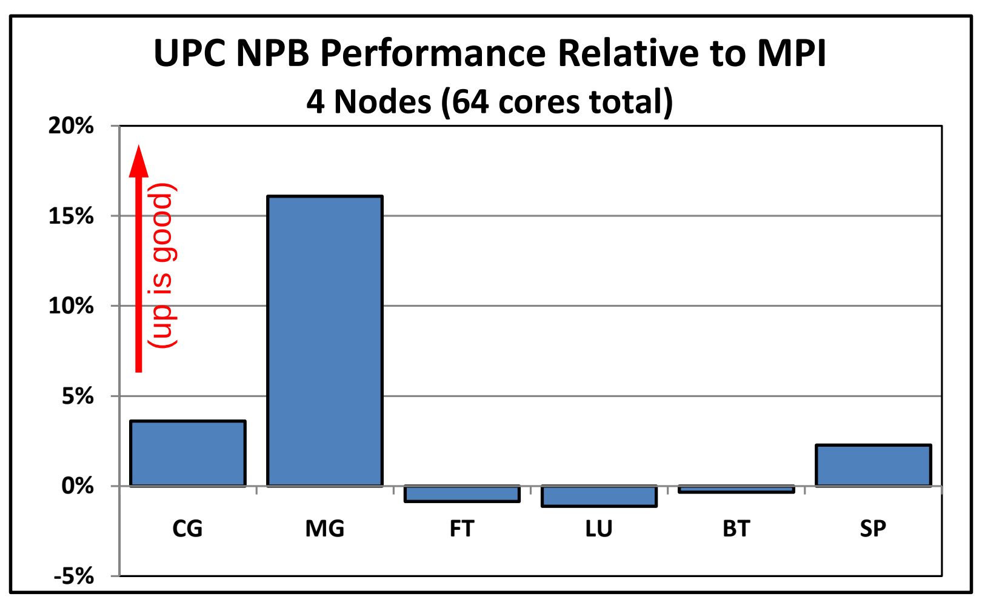
Berkeley UPC Compiler

- Open Source Software (Windows/Mac/UNIX), installation DVD available in this booth
- A portable and high-performance UPC implementation, compliant with UPC 1.2 spec



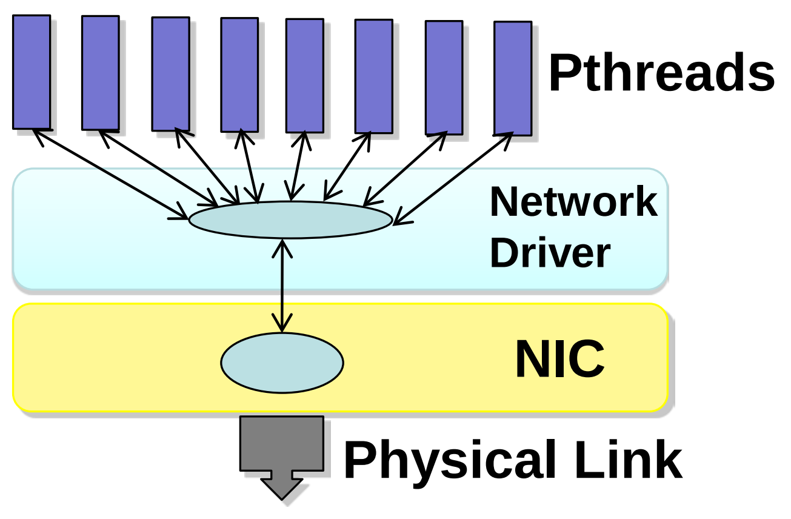
- Runtime libraries support a wide range of platforms
- Implementation Features:
  - UPC Collectives with auto-tuning
  - Extensions for performance and programmability
  - Compiler optimizations for application scalability
  - Debugging and tracing support (GASP, PPW, TotalView)
  - Interoperability with other programming environments
    - UPC calls to/from C, C++, FORTRAN, MPI
- Well-documented runtime interface, used by the Berkeley UPC and Intrepid GCC/UPC compilers
- Berkeley GASNet used for communication:
  - Performance from inline functions, macros, and network-specific implementations
  - Scalability demonstrated to 32K cores of BG/P
  - Also used by Cray: UPC, CAF & Chapel; and Rice: CAF2.0
- NPB performance comparisons for Class C on TACC Ranger:

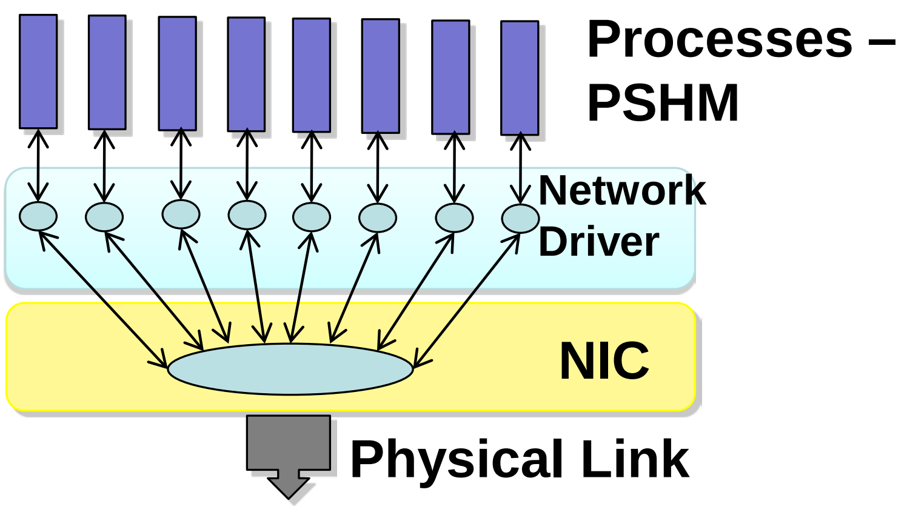


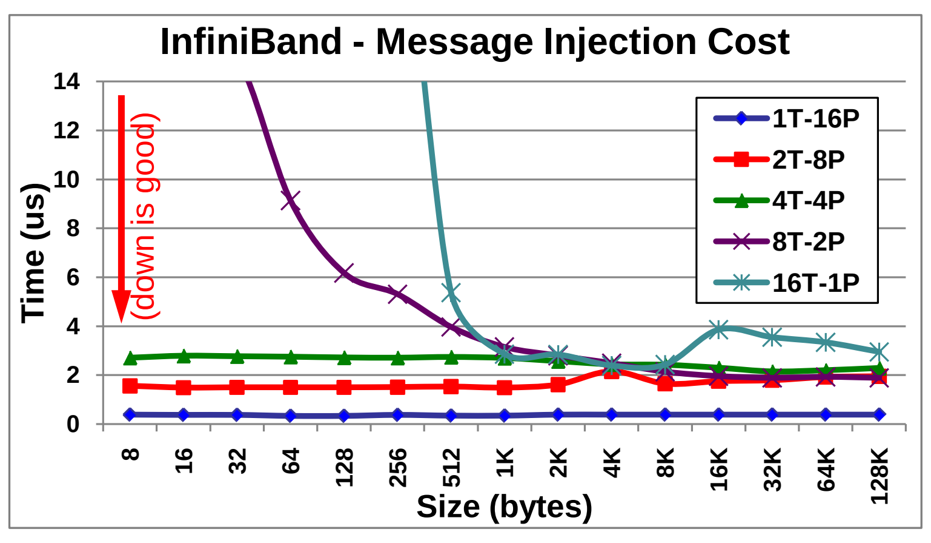


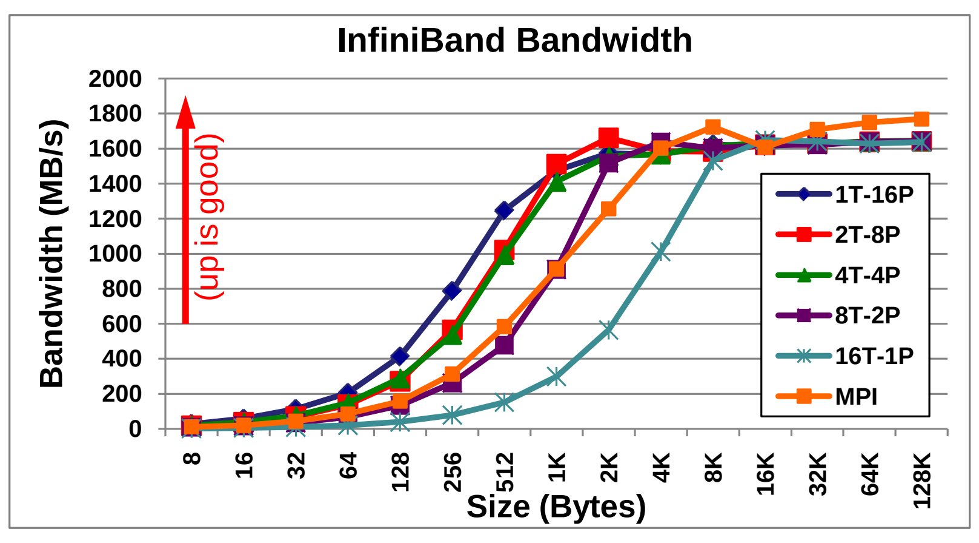
Hybrid Shared/Distributed Memory

- BUPC allows programs to use arbitrary combinations of Pthreads and Processes with shared memory
- Mixing Pthreads and Processes is required for:
  - Interoperability with MPI and non thread-safe libraries
  - Hybrid/hierarchical parallelism (for best performance)
- New PSHM layer – Process SHared Memory
  - Shared memory comms through POSIX, SYSV or mmap()
  - Shared memory "network" for Active Messages support
  - Hybrid processes/pthreads execution







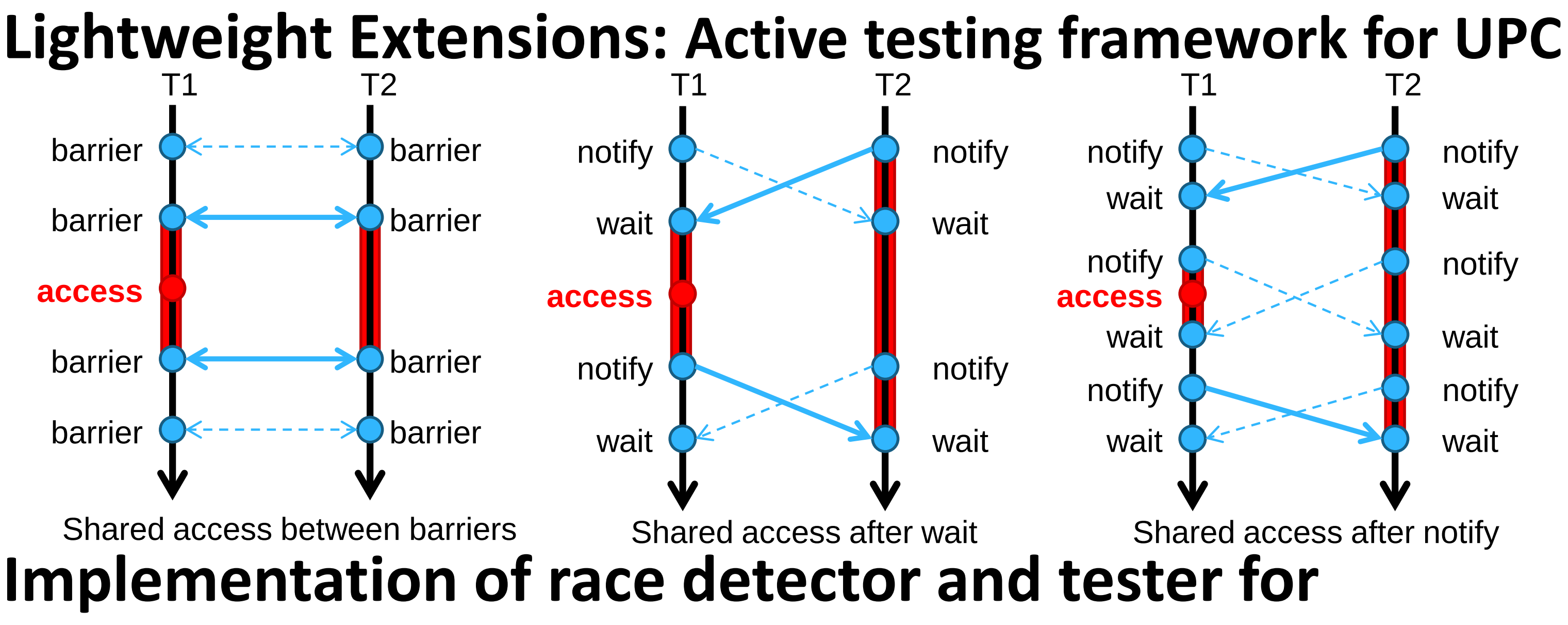


Both figures are for 2 nodes (16 cores each), a total of 32 UPC threads perform pair-wise inter-node communication. Various runtime configurations observed. Example: "4T-4P" is 4 processes of 4 Pthreads each, on each node.

For more details, please see: "Hybrid PGAS Runtime Support for Multicore Nodes", F. Blagojevic, P. Hargrove, C. Iancu and K. Yelick, 4th Conference on Partitioned Global Address Space Programming Model, New York NY, Oct 2010.

Active Testing in UPC

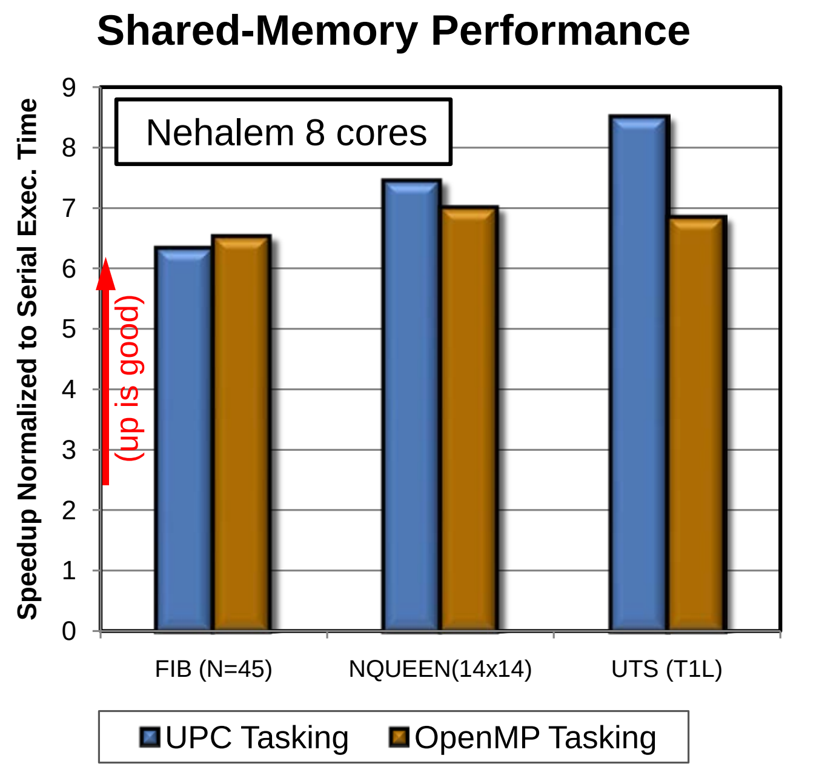
- Active Testing: Leverage program analysis to make testing quickly find real concurrency bugs
  - Phase 1: Use imprecise static or dynamic program analysis to find abstract states where a potential concurrency bug can happen (Race Detector)
  - Phase 2: Directed testing based on the abstract states obtained from phase 1 (Race Tester)
- THRILLE – THRead Interposition Library and Lightweight Extensions: Active testing framework for UPC

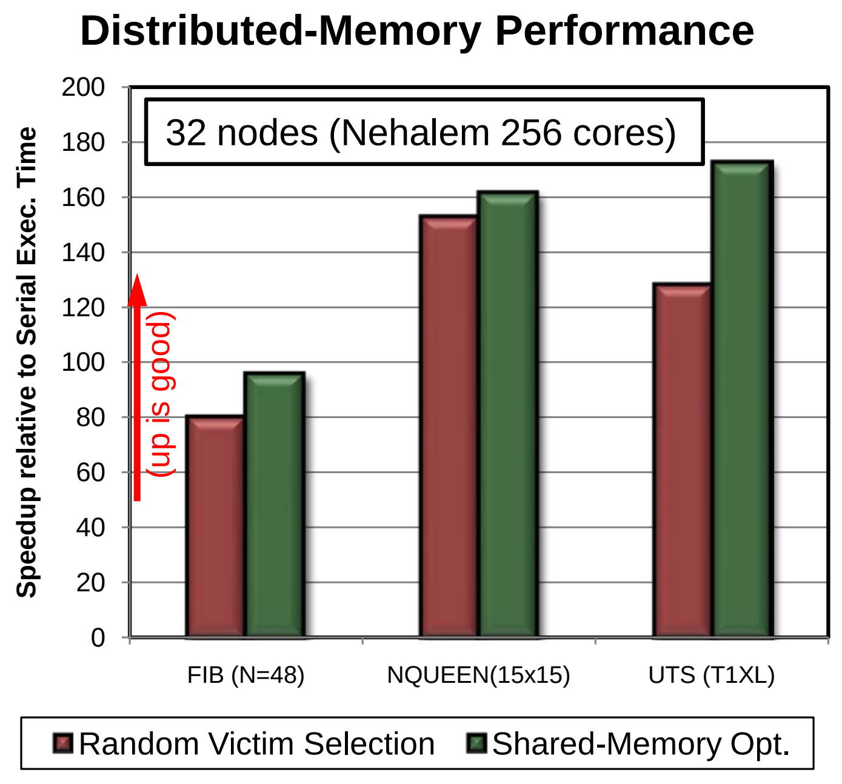


- Implementation of race detector and tester for programs written in shared memory style
  - Support for collectives and local aliases in progress
  - Additional analyses for deadlocks, memory consistency, and others in the future
- Download available at http://upc.lbl.gov/thrille.shtml

Dynamic Tasking in UPC

- UPC task library supports dynamic tasking in UPC
- Task is defined as a task function + pointers to in/out
  - void *task_func* (void *in, void *out) { /* *task body* */ }
  - Tasks are stored in the global task queue
- Task programming models
  - Parallel-for parallelism and Fork-Join parallelism
  - Dependent task graph with task synchronization
- Performance improvement by dynamic load balancing
  - Shared-memory optimized: steals within a node first

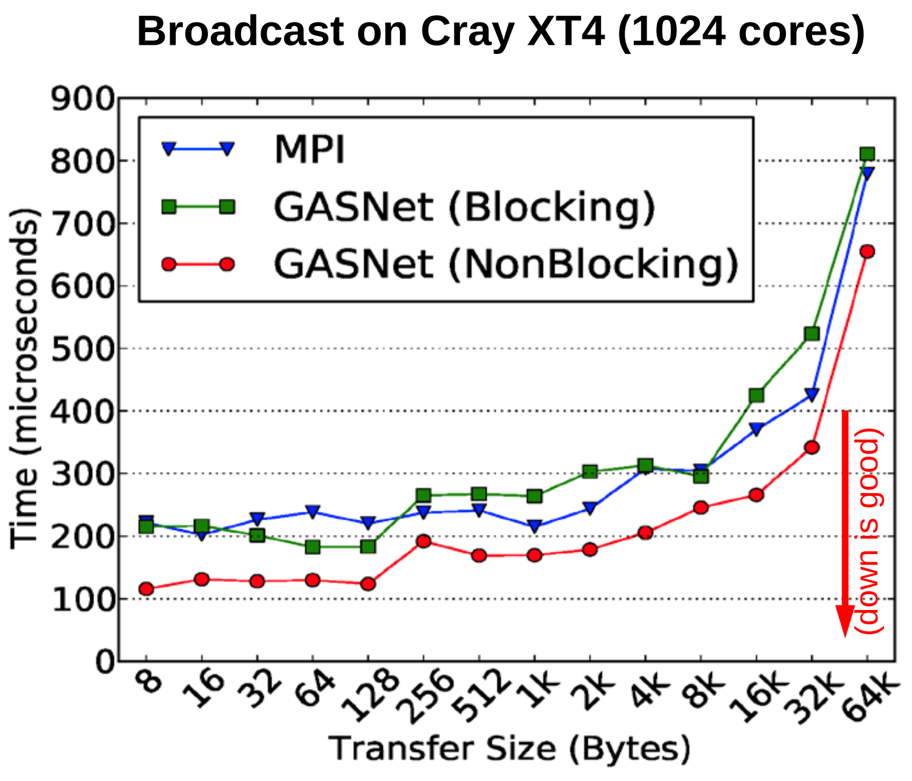


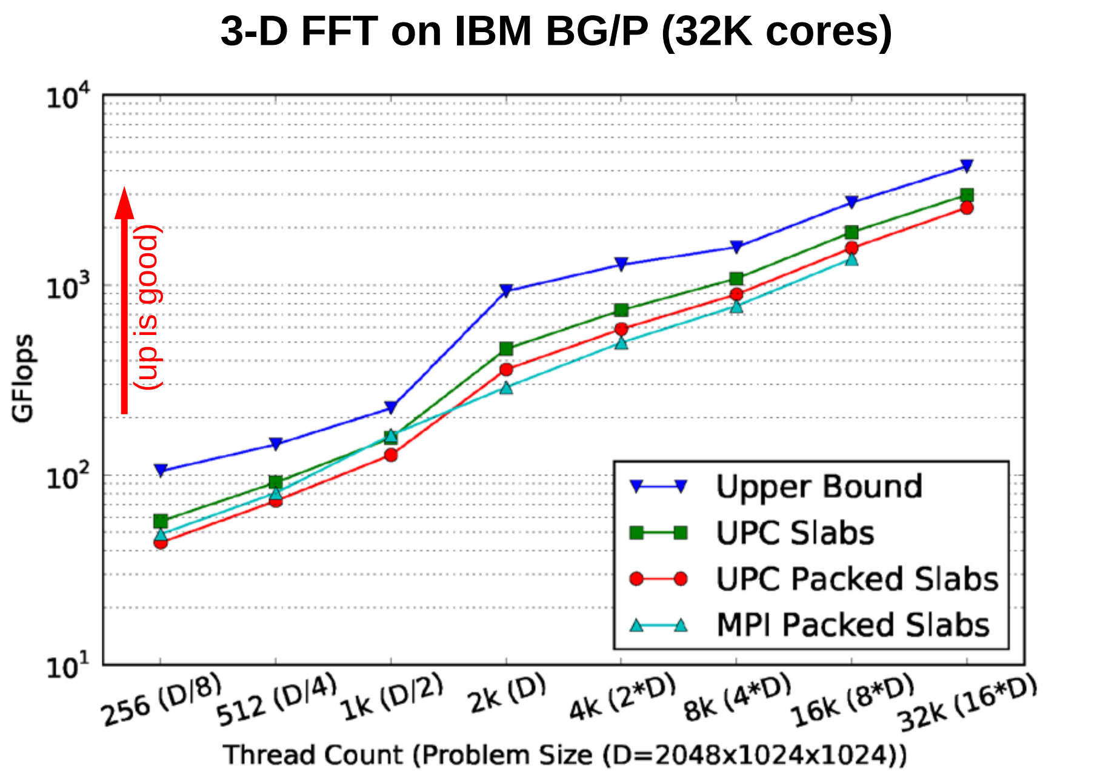


- For more information – http://upc.lbl.gov/task.shtml

Collective Communication

- UPC collectives implemented by GASNet collectives
- GASNet collectives features:
  - Auto-tuning
    - Support online and offline search
    - A repository of popular algorithms with tunable parameters
    - Portable performance without user intervention
  - Non-blocking multi-threaded collectives
    - Overlap communication with computation/communication
    - Improve load balance with threaded communication
  - Teams
    - Enable efficient sub-group collective communication
    - Multi-threaded team collectives (research prototype)





This research used resources of the National Energy Research Scientific Computing Center and the Argonne Leadership Computing Facility at Argonne National Laboratory supported by the Office of Science of the U.S. Department of Energy under Contracts DE-AC02-05CH11231 and DE-AC02-06CH11357.
The authors also acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing computational resources.