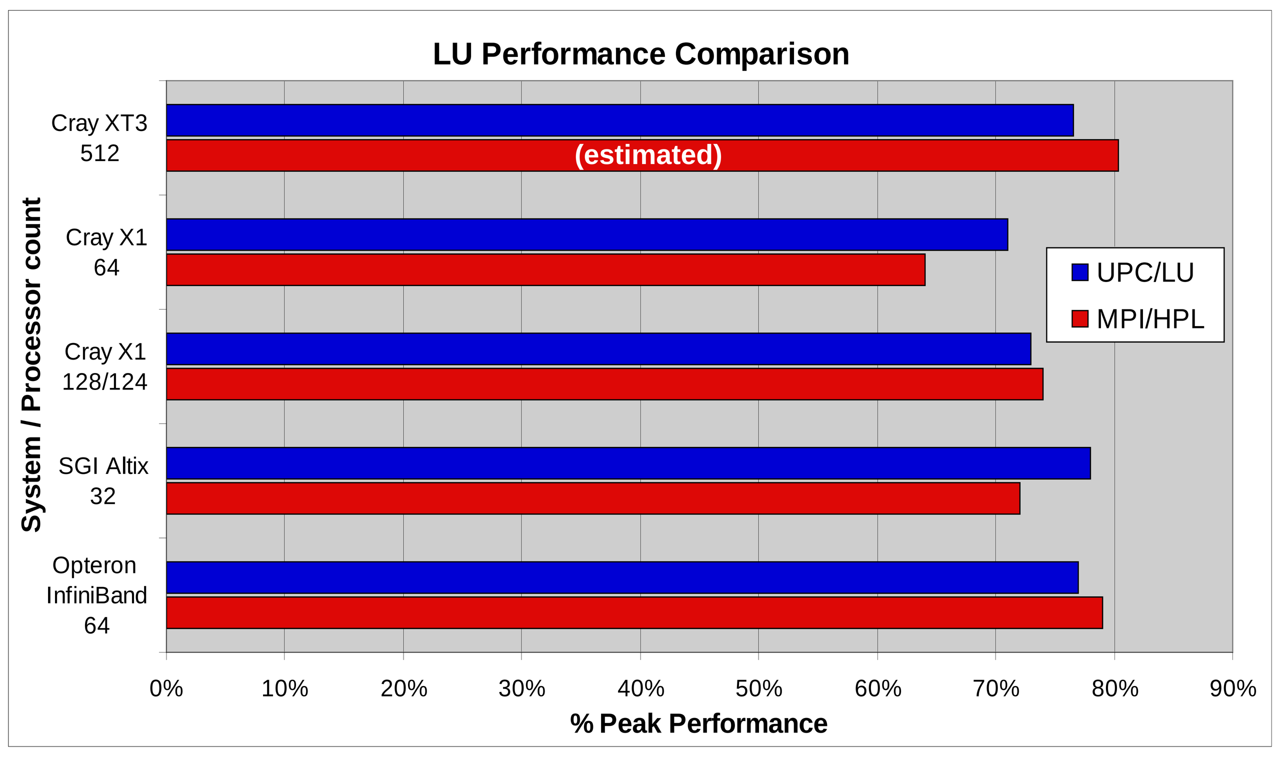
Berkeley UPC Applications

http://upc.lbl.gov

Goals of Application Projects

- Demonstrate that UPC can *outperform* other programming models
  - Take advantage of one-sided communication
  - Show performance advantage *on clusters* with RDMA hardware, as well as shared memory
- Demonstrate *scalability* of UPC
  - NAS FT: *.5 TFlops* on 512p Itanium/Elan4
  - Linpack: *4.4 Tflops* on 1024p Itanium/Elan4
    2 Tflops on 512p XT3/Portals
- Demonstrate *ease-of-use* on some challenging parallelization problems
  - Delaunay triangulation
  - Hyperbolic PDE Solver
  - Sparse Cholesky factorization (ongoing)

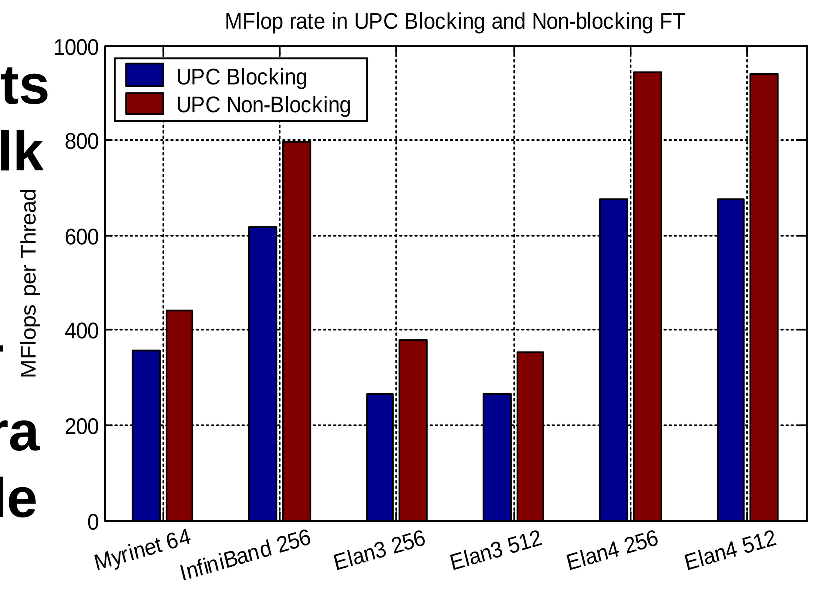
Linpack in UPC



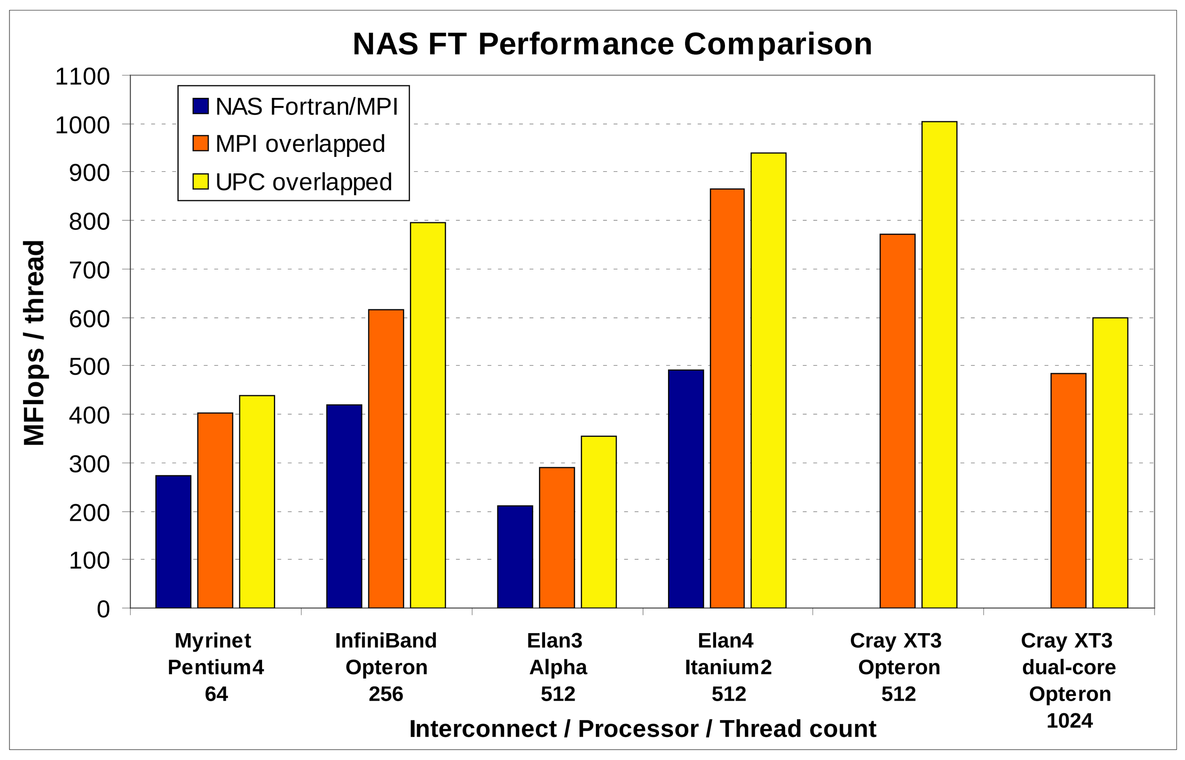
- *UPC Linpack* code is compliant with Top500 Benchmark (HPL)
- Dense case is warm-up for sparse factorizations
  - Dependencies, tuning of block sizes, overlap/lookahead are common challenges
- UPC Linpack is less than ½ the code size of MPI HPL
- Novel multi-threading on SPMD → latency tolerance
  - Portable co-operative thread package built using only function calls and returns
  - Each UPC process consists of multiple threads (one for each major operation) that yield on long latency communication operations
  - Threads also allow for *algorithmic* overlap
  - Dependencies tracked on each node using a scoreboard. Threads execute after all dependencies are satisfied
- Memory-constrained lookahead with deadlock avoidance allows for flexible execution schedule

3D FFTs in UPC

- FFT bottleneck is (all-to-all) communication
  - Limited by bisection bandwidth
  - Bisection bandwidth is increasingly expensive
    - ➔ Want to use "all the wires all the time"
    - ➔ Send early and often: same total data spread over longer period of time to avoid bottleneck
    - ➔ Fully leverage RDMA capabilities of modern networks
- Berkeley UPC compiler supports non-blocking bulk memory extensions
- Non-blocking FT version: ~30 extra lines of UPC code



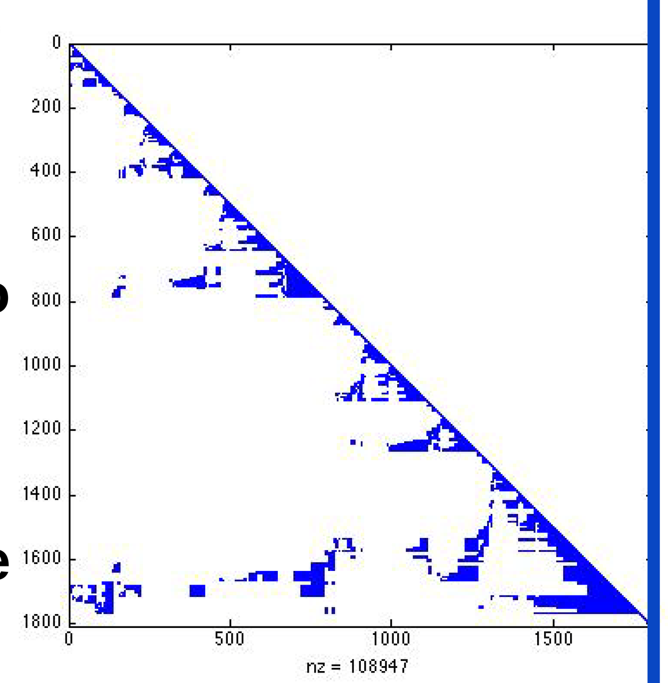
- Default NAS FT Fortran/MPI
  - communicates all at once in a big all-to-all
  - network is idle while processor computes
- UPC implementation *overlaps*
  - sends data as it becomes available
  - vary granularity of overlap: slabs or pencils



- Slabs win in MPI: overlap is good, but fine-grained overlap less effective due to high msg overheads
- Pencils win in UPC: low overhead + benefit of better local memory locality (smaller msgs)

Sparse Cholesky in UPC

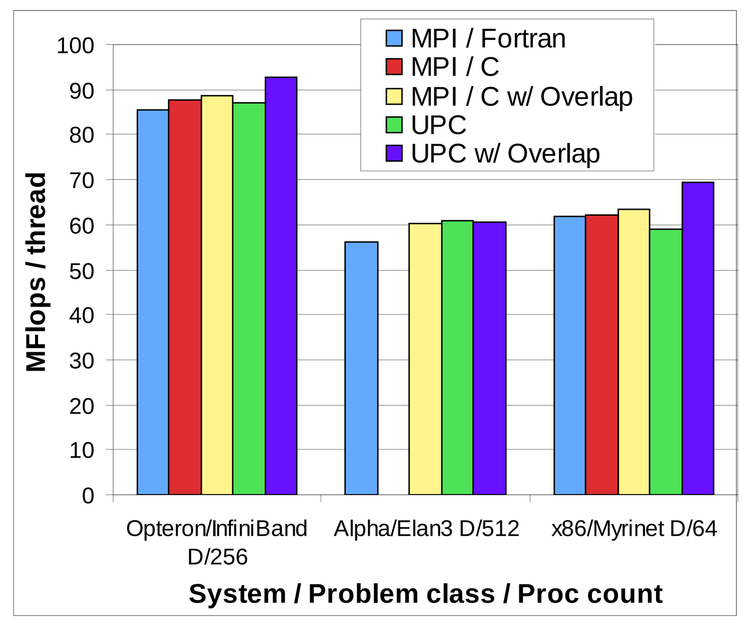
- Multithreaded UPC code
  - Based on left-looking, blocked serial code of Ng and Peyton
  - Choice of block size to enhance performance via level-3 BLAS operations
  - Block columns receive updates from earlier block columns



- After all updates are received a block column is factorized
- Code written and tuning underway

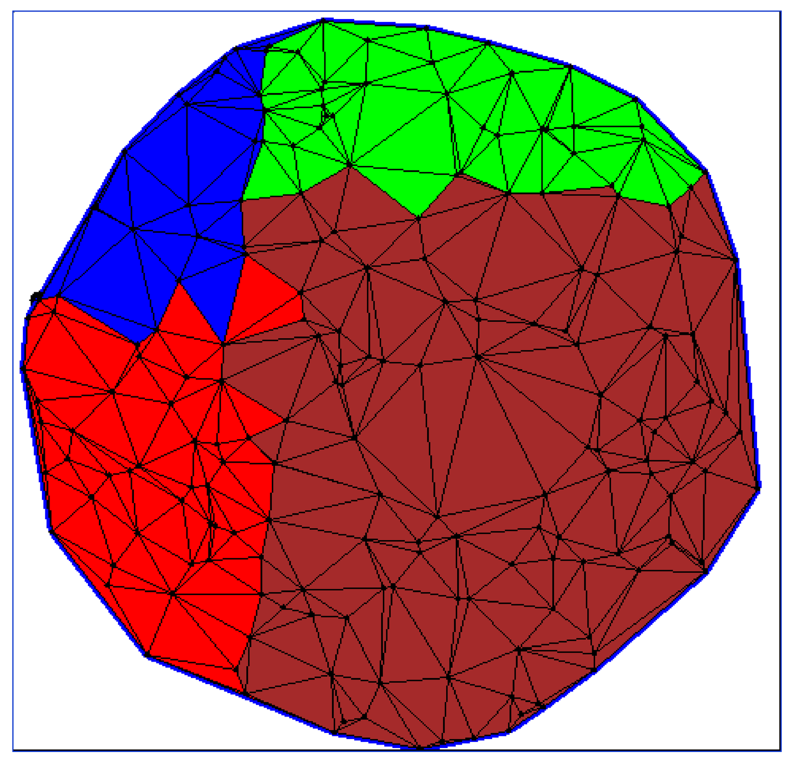
Conjugate Gradient in UPC

- CG: Iterative sparse solver w/ Sparse Matrix-Vector Multiply (SPMV)
- 2D (NAS-optimized) and 1D partitioned versions



- Bottleneck is reductions, which are *latency-limited*
- UPC version overlaps multi-word reductions with the local SPMV computations
- Outperforms MPI version by up to 10%

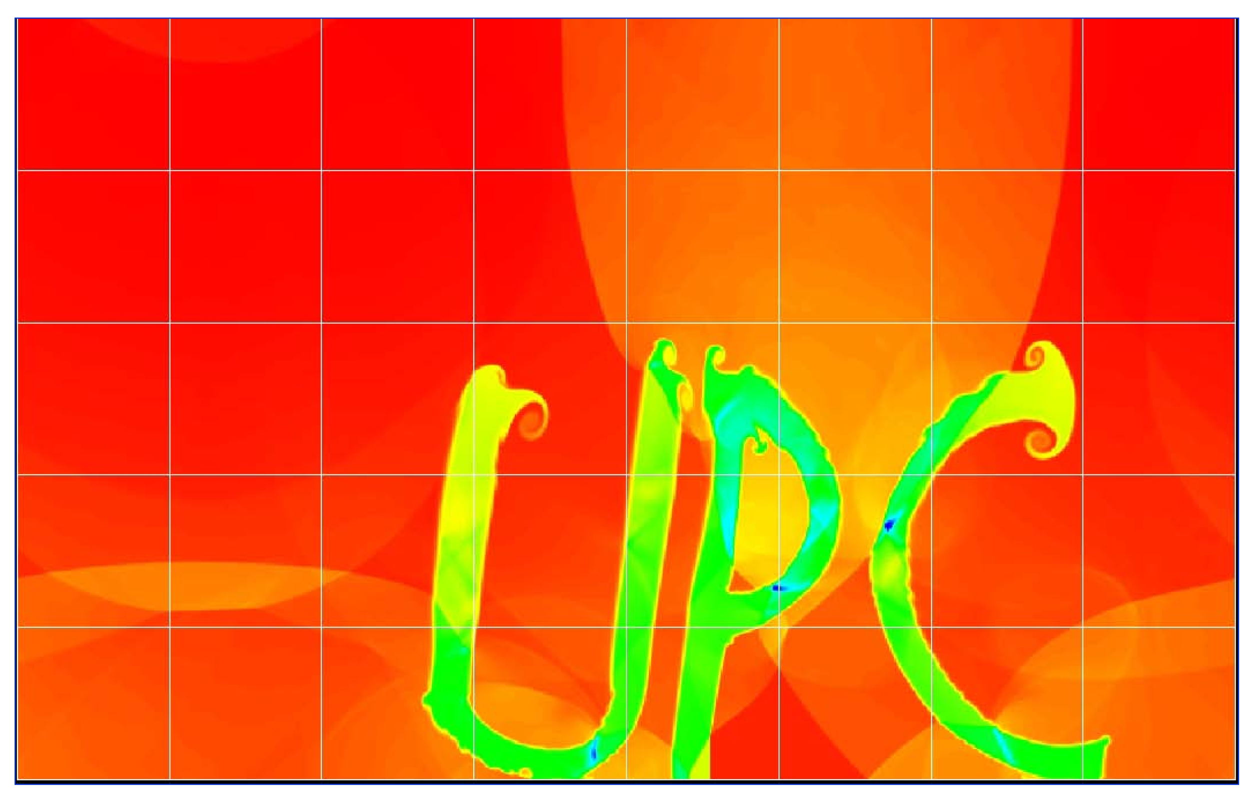
Triangulation in UPC



- 2D Delaunay triangulation
  - based on Triangle software
- Parallel version incorporates:
  - Dynamic load balancing
  - App-level software caching
  - Parallel sorting

Fluid Dynamics

- Finite difference hyperbolic solver in UPC
  - Numerics in FORTRAN*
  - Data/control structures in UPC



-Thanks to the ANAG group at LBL

- Warm-up for fully adaptive code
- Mach 2 wave in a 2-D periodic chamber with a dense fluid in the shape of the letters: U P C

BERKELEY LAB

BERKELEY UPC UNIFIED PARALLEL C

THE UNIVERSITY OF CALIFORNIA BERKELEY 1868