Cooperative Multithreading and Remote Function Invocation in UPC

Parry Husbands (LBNL)
Esmond Ng (LBNL)
Katherine Yelick (LBNL/UCB)

Motivation & Context

- Parallel programming challenges
  - Expressibility
    - Many algorithmic constructs tortuous to implement
  - Performance
    - Synchronous codes spend an excessive amount of time waiting
- Asynchronous memory operations boost performance
  - Modern out-of-order processors
  - MPI_Isend()/MPI_Irecv()
- How do we organize programs with many outstanding requests?
  - Threads have a natural latency tolerance

Write distributed memory code in a multithreaded style!

The System

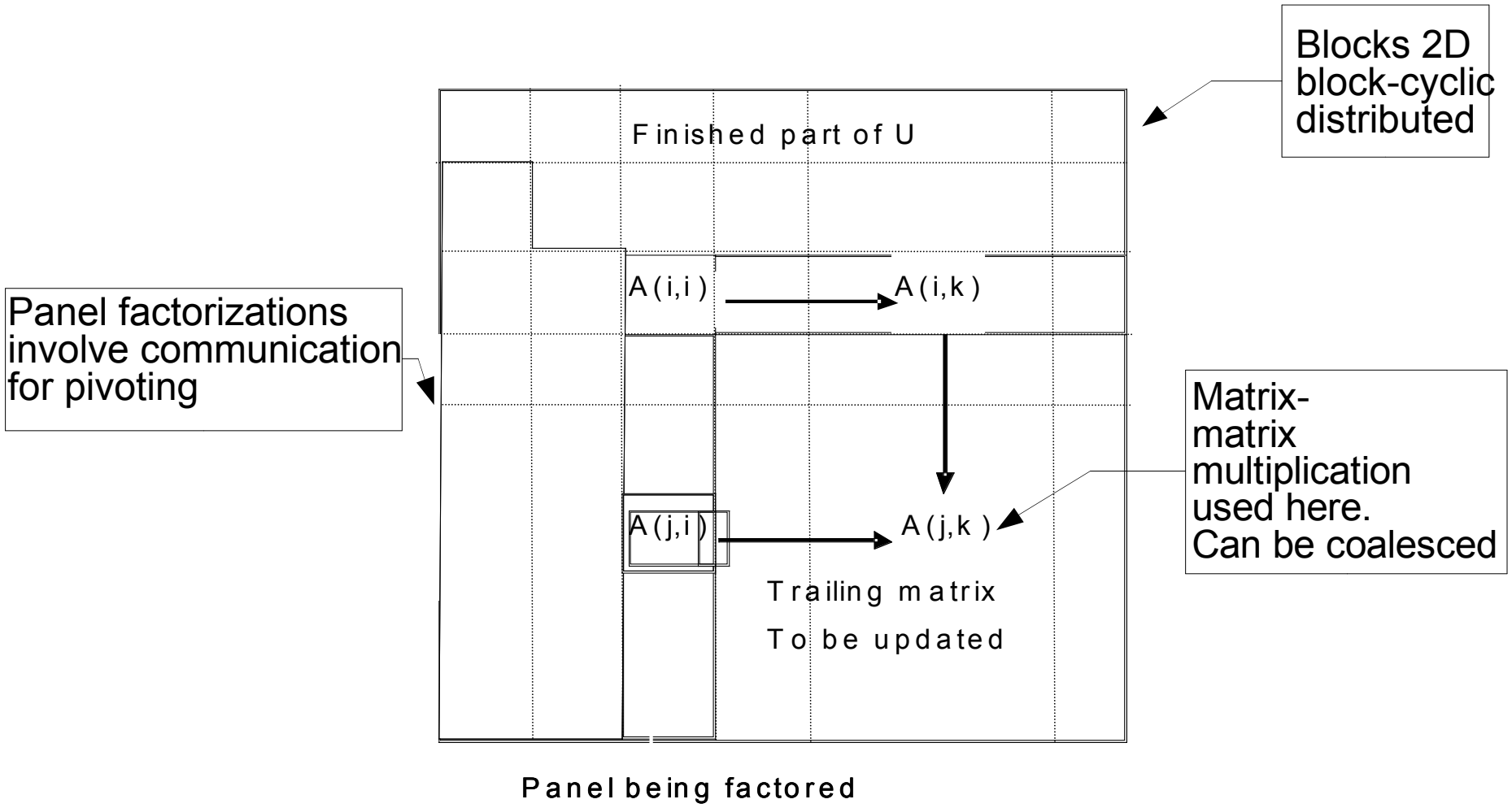
- Co-operative threads
  - Remove need to maintain integrity of data structures throughout program
  - Experimented with GNU Pth, POSIX Threads,
    Hand rolled user-level threads for portability
    - Uses only function calls and returns
    - "Interesting" use of Duff's Device
    - Macros: PTP_SPAWN, PTP_FUNCALL, PTP_YIELD,
      PTP_START, PTP_END
    - Suspend, resume, priorities
    - Custom script expands, computes jumps, rewrites local (stack) accesses, creates functions for arguments, etc.
- Remote Queues + A poll & dispatch thread
  - Lots of implementation freedom
    - Locks, Single Reader Single Writer, Active Messages, Floating Functions, Enqueue on NIC, ...
- AND non-blocking remote operations

Our test applications

- Parallel Dense LU Factorization and Sparse Cholesky?
- Why? Challenging to implement concisely yet efficiently

- Our Strategy
  - Don't overconstrain the execution
  - Express algorithm in terms of local computations and dependencies
  - Asynchronous style

LU Factorization



LU Operations and Constraints

- Panel factorization
  - Including pivots!
- Update to a block of U
  - Need to apply pivots to entire block column
- Trailing submatrix updates

- Panel Fact. needs all trailing updates
- Update of U needs Panel Fact.
- Pivots need previous pivots and updates
- Trailing updates need U and pivots complete

- Use a "scoreboard" to keep track of local dependencies
  - Keep track of number of remaining updates to each block. When count reaches zero, can begin a panel factorization
- Need to receive remote event notifications

Like Herding Cats

- Some order needs to be imposed on the execution schedule
- Critical operation: Panel Factorization
  - need to satisfy its dependencies as soon as possible
  - perform trailing matrix updates with low block numbers first
    - Use a Priority Queue to schedule these

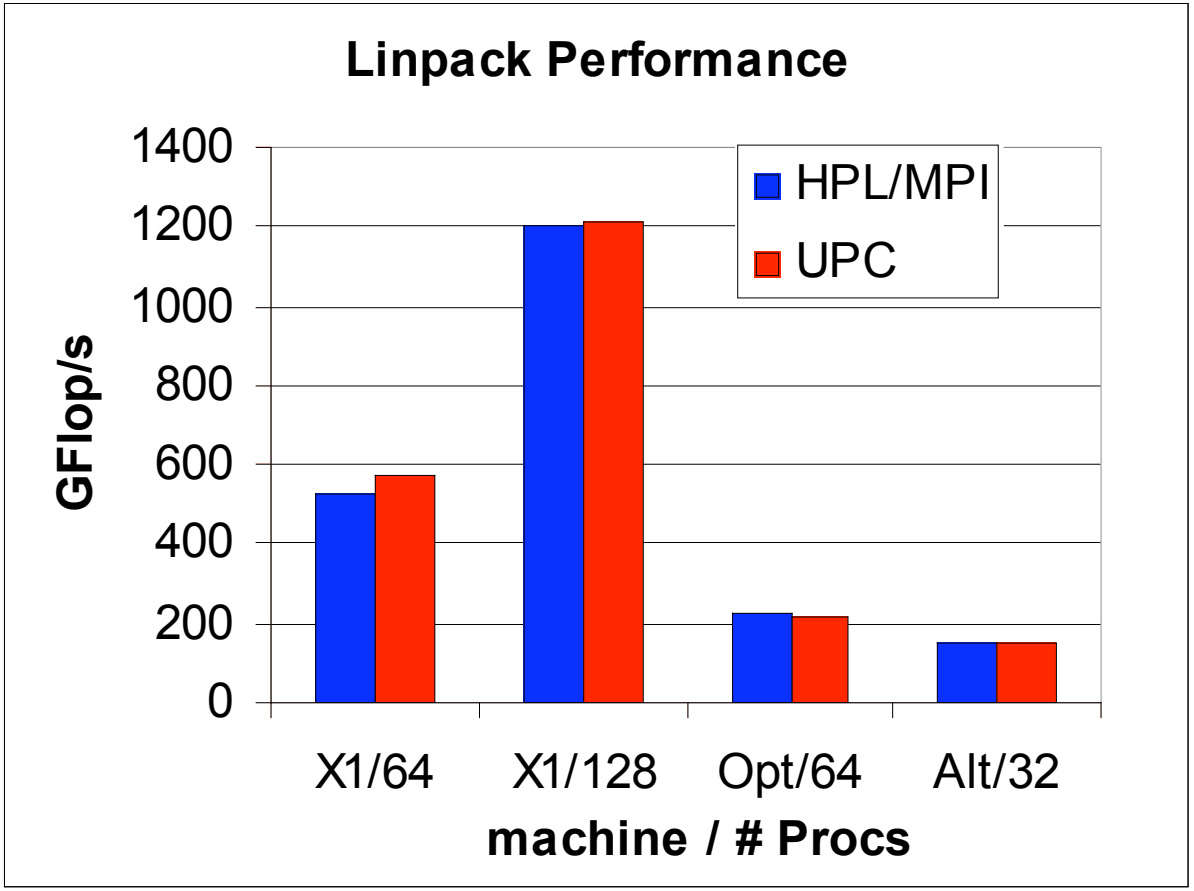
BUT

- How do we buffer these updates? Do we have enough memory? Deadlock in allocator?

Solution:

- Keep pieces of L and U around for the trailing updates as long as needed
- Allocate memory in increasing order of factorization and don't skip any!
- Thread blocks until enough memory available

Parallel Performance



- Itanium 2/Elan 4.1 – 2.25 TFlop/s, 78.5% of peak on 512p
- 1p Itanium 2 1.5 GHz – 91.8% of peak
- 1p Opteron 2.2GHz – 81.9% of peak

Sparse Cholesky Decomposition

- Based on left-looking, blocked serial code of Ng and Peyton
  - Block columns receive updates from earlier block columns
  - After all updates are received, a block column is factorized
- Complications due to data dependent dataflow graph
  - Scoreboard no longer simple
  - What's the critical path?
- Our Strategy
  - Use analysis to figure out dataflow graph and importance of each update
  - Threads for block column-block column updates
  - Set thread priorities based on importance
  - Similar strategy to limit memory usage
- Status: Code written and tuning underway

Conclusion and Open Questions

- Portable addition of cooperative threads and remote function invocation to UPC
- High performance version of Linpack Benchmark in ~5K LOC
- Sparse Cholesky work ongoing

Future Investigations:

- How do things change with pre-emptive threads?
- Can we get support for remote enqueue and spawning?
- How to exert control over the local schedule in a principled way?
- Deadlock avoidance in resource allocation?

Extras