We present a parallel distributed-memory sparse symmetric solver, symPACK, based on an asynchronous task paradigm using one-sided communication and dynamic scheduling for computing the Cholesky factorization. The performance and resource usage of sparse matrix factorizations are critical to time-to-solution and maximum problem size solvable on a given platform. Exploiting symmetry in sparse matrices reduces both the amount of work and storage cost required for factorization. On large-scale distributed memory platforms, communication cost is critical to performance.

symPACK relies on efficient and flexible communication primitives provided by the UPC++ library. Experiments show good scalability and that symPACK often outperforms state-of-the-art parallel distributed memory factorization packages.

Right-looking Sparse Cholesky factorization

- Fill-in reducing ordering
  - Multiple Minimum Degree
  - Approximate Minimum Degree
  - Nested-dissection (PARMETIS and PT-Scotch graph partitions)
- Symmetric permutation
- Supernodal elimination tree
  - A Supernode is a set of columns with a dense diagonal block and some off-diagonal non-zero row structure
  - The Elimination tree describes dependencies between supernodes.
- At every node in the elimination tree:
  - Supernode factorized locally
  - Updates some ancestors in the tree
  - Linear solve phase
  - Forward and backward solve: bottom-up and top-down tree traversals

Fan-in, Fan-out and Fan-both factorization algorithms

- Three families of parallel Cholesky algorithms [1]:
  - Fan-in: Updates from a column k to other columns j computed on the processor owning column k. Processor owning i will have to “fan-in” (or collect) updates from previous columns.
  - Fan-out: Updates from k computed on processors owning columns j. Processor owning k has to “fan-out” (or broadcast) column k of the factor.
  - Fan-both: Updates are to be performed on any processors. Relies on computation maps.

Asynchronous Task Execution

- Dynamic task scheduling within distributed memory node:
  - local task queue (LTQ), contains all local tasks awaiting execution,
  - ready task queue (RTQ), contains all local tasks ready for execution.
- One sided “pull” communication protocol using UPC++:
  - Sender notifies available data
  - Recipients periodically get incoming data
  - One-sided communications without interrupting sender/recipient
  - Remote temporary data deallocation

Numerical Results on NERSC Edison, 2x 12-core processors per node

- PASTIX 5.2.2 uses 24 threads per node
- Multi-threading helps PASTIX a lot
- symPACK has no hybrid implementation yet but coming soon

Fan-in and Fan-out Task dependencies for three columns j, i and h

- Three types of tasks:
  - Factorization: Compress column i of the Cholesky factor.
  - Update U(j,i): Compute the update from f(j,i) to column j, with i < j such that f(j,i) ≠ 0, and put it to an aggregate vector f.
  - Aggregation: A(j,i): apply all aggregate vectors f(i) from columns i < j, with f(j,i) ≠ 0, to column j.

Data layout and computation mapping

- ID cyclic Supernodal layout, sequential BLAS is called within each supernode
- Supernodes assigned to nodes based on estimated work (proportional mapping)
- Computation map determines node ranks where tasks are mapped/executed

Fan-in, Fan-out and Fan-both task dependencies for three columns j, i and h

- Three families of parallel Cholesky algorithms [1]:
  - Fan-in: Updates from a column k to other columns j computed on the processor owning column k. Processor owning i will have to “fan-in” (or collect) updates from previous columns.
  - Fan-out: Updates from k computed on processors owning columns j. Processor owning k has to “fan-out” (or broadcast) column k of the factor.
  - Fan-both: Updates are to be performed on any processors. Relies on computation maps.

Asynchronous Task Execution

- Dynamic task scheduling within distributed memory node:
  - Local task queue (LTQ), contains all local tasks awaiting execution.
  - Ready task queue (RTQ), contains all local tasks ready for execution.
- One sided “pull” communication protocol using UPC++:
  1. Sender notifies available data
  2. Recipients periodically get incoming data
  3. One-sided communications without interrupting sender/recipient
  4. Remote temporary data deallocation

Numerical Results on NERSC Edison, 2x 12-core processors per node

- PASTIX 5.2.2 uses 24 threads per node
- Multi-threading helps PASTIX a lot
- symPACK has no hybrid implementation yet but coming soon

Data exchange protocol in symPACK

- Dynamic task scheduling within distributed memory node:
  - Local task queue (LTQ), contains all local tasks awaiting execution,
  - Ready task queue (RTQ), contains all local tasks ready for execution.
- One sided “pull” communication protocol using UPC++:
  1. Sender notifies available data
  2. Recipients periodically get incoming data
  3. One-sided communications without interrupting sender/recipient
  4. Remote temporary data deallocation

Numerical Results on NERSC Edison, 2x 12-core processors per node

- PASTIX 5.2.2 uses 24 threads per node
- Multi-threading helps PASTIX a lot
- symPACK has no hybrid implementation yet but coming soon

References


http://www.sympack.org