Performance Modeling Tools for Parallel Sparse Linear Algebra Computations

X. Sherry Li
xsli@lbl.gov
Lawrence Berkeley National Laboratory

Pietro Cicotti, Scott Baden
University of California at San Diego

ParCo 2009, 1-4 September, 2009

Motivation

- **Parallel sparse matrix algorithms are needed in many large scale computational codes**
- **Why need accurate performance modeling / prediction tools?**
  - **Large design space of parallelization strategies**
  - **Can't afford to implement all algorithm choices**
  - **Performance depends on machines and input (sparsity pattern)**
- **Goals**
  1. **Predict performance of existing implementations on emerging architectures**
  2. **Design and prototype new algorithms, eliminate bad algorithm choices**

Sparse matrix algorithms

- **Characterization of the workload**
  - **CPU-bound, memory-bound, or mixture**

- **Matrix-vector multiplication, triangular solve**
  - **Purely memory-bound : matrix is read once, flops-to-memory ratio is O(1)**
  - **Can develop analytical cost models based solely on cache miss count s** [Vuduc et al.]
- **Factorizations (LU, Cholesky, QR, . . .)**
  - **Dense is CPU-bound: flops-to-memory ratio is O(N)**
  - **Sparse is a mixture of CPU-bound and memory-bound: Factors are sparser in the beginning, and denser later**
  - **Analytical cost models are inaccurate** [Ashcraft, Grigori et al.]

Performance modeling methods

- **Synthetic benchmarks : measure raw performance of the machine's individual components**
  - **PMTools , STREAM, MultiMAPS, IMB (Intel MPI Benchmarks)**
- **Analytical cost models**
- **Trace-based analysis : collect application attributes / addresses during execution, input to cache simulator**
  - **Can only analyze codes that are memory-bound**
- **Simulation-framework is more flexible and suitable for predicting performance**
  - **Write simulation code to mimic application's algorithm, advance simulation time by costs of memory access, operations, and communication**
  - **Not to worry implementation details (e.g., sparse data structures)**

Outline

- **Performance Modeling Tools (PMTools)**
  - **Calibrate speed of machine's individual components**

- **Application-specific simulations**
  - **Validate parallel sparse LU in SuperLU_DIST**
  - **Prototype parallel sparse Cholesky**

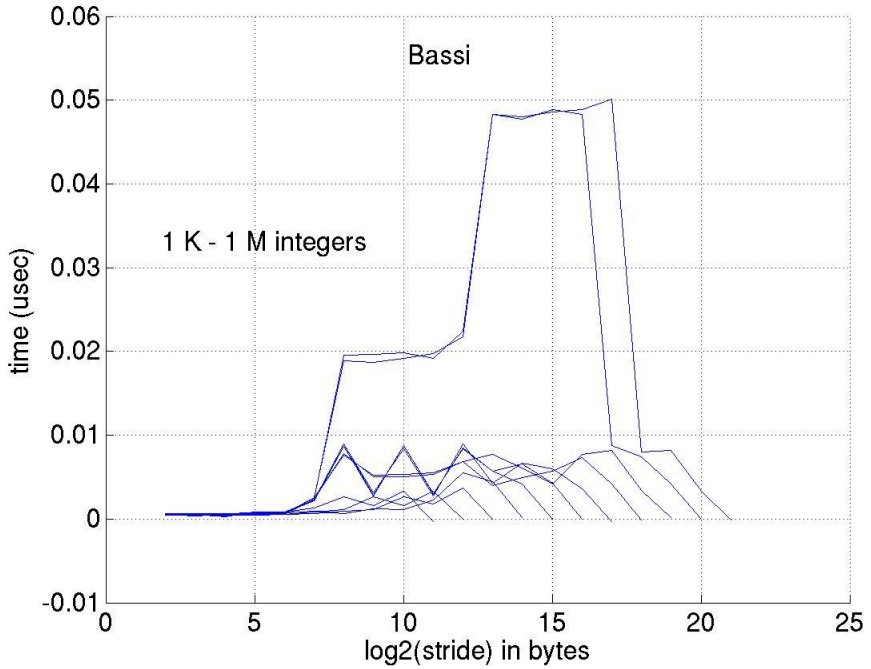
PMTools

- **Micro-benchmarks run off-line, times of basic operations are taken over a parameter space of interest, stored in tables. The omitted values are estimated using interpolation, extrapolation, or curve-fitting**
  - **Memory: timing memory updates with varying strides**
  - **BLAS: timing BLAS calls with typical dimensions**
  - **interconnect: timing point-to-point latency & bandwidth**

- **Cache simulator, with multiple levels**
  - **Maintains the state of the memory hierarchy at any time for each processor**
  - **Estimates the cost of each memory access**

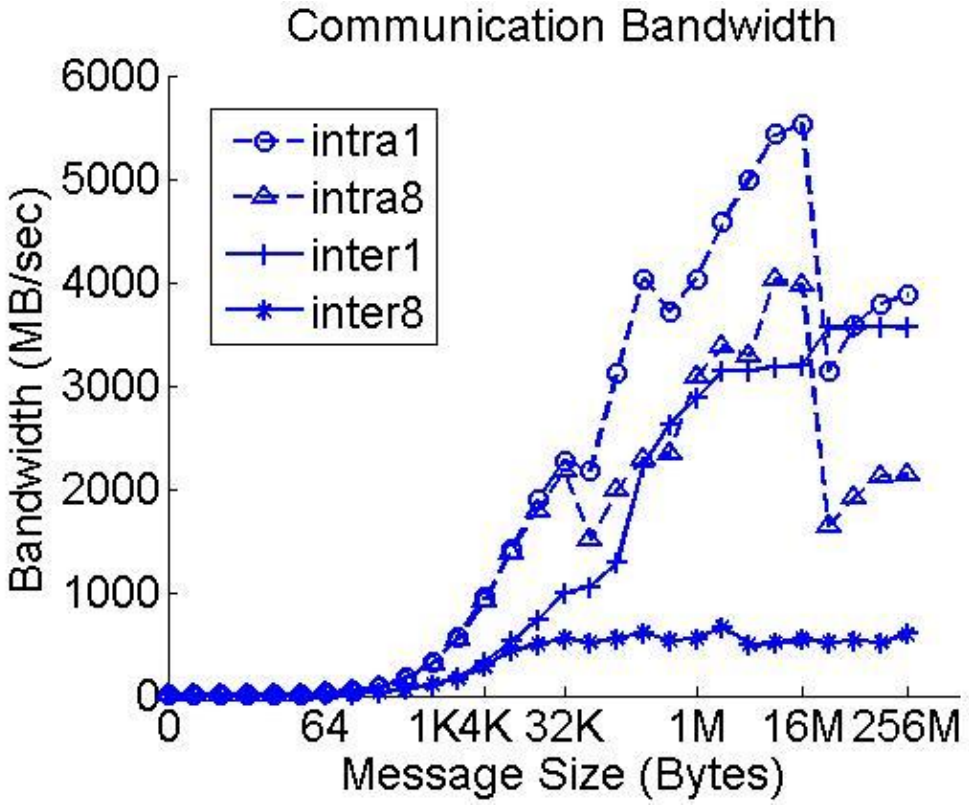
Memory micro-benchmark

- **Measure latency & bandwidth of memory hierarchy**
  - **Bandwidth: timing sequential updates**
  - **Latency: timing strided updates** [Saavedra]
    - **Repeated read/write N elements of a 1D array with stride s, plot the running time as a function of (N, s).**
- **IBM Power5: L1 (32 KB), L2 (1.92 MB)**

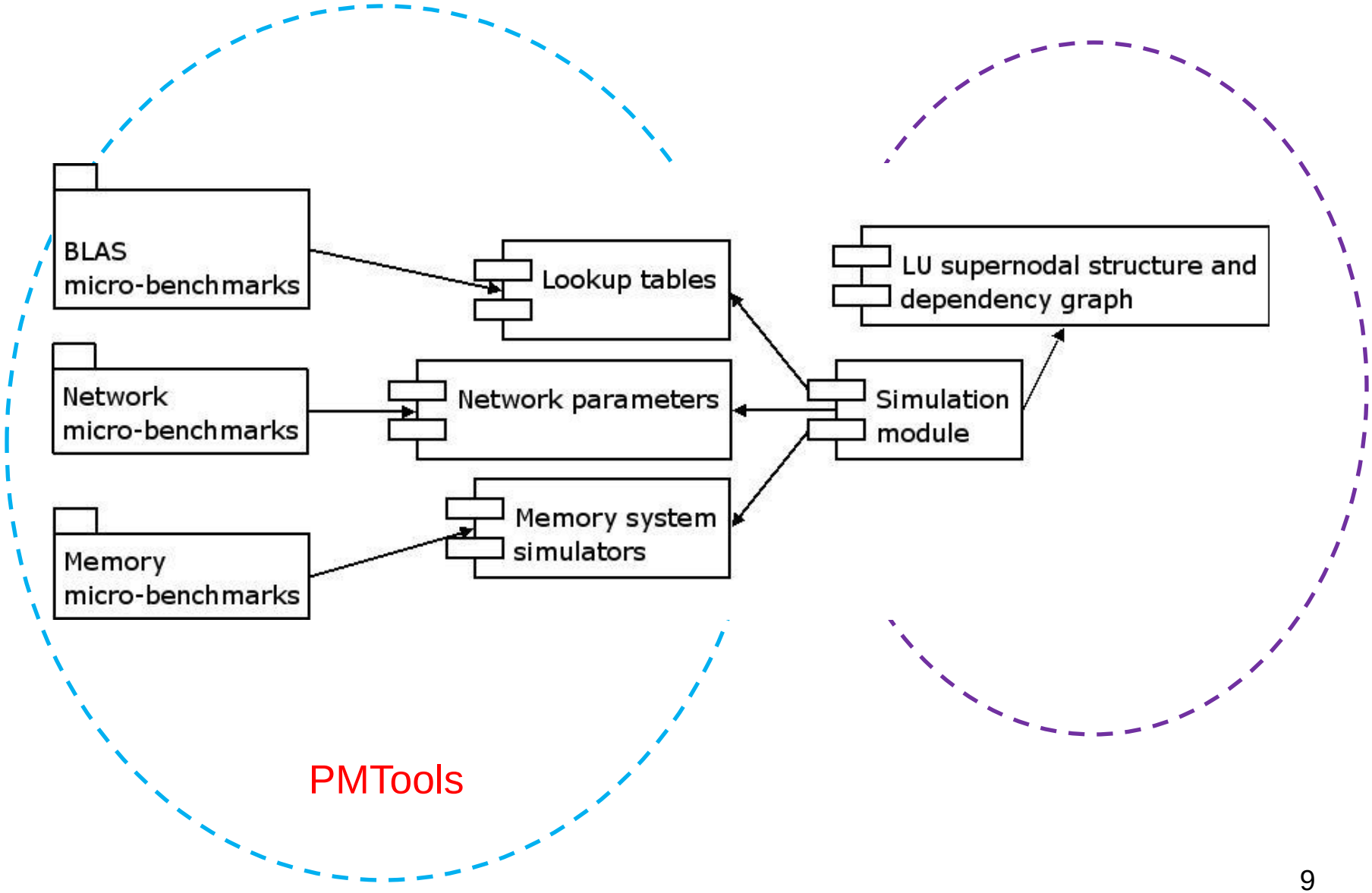


MPI micro-benchmark

- **Ping-pongs between a pair of processors, all pairs, intra-node, inter-node**
- **Bandwidth on IBM Power5: SMP node, 8 CPUs / node**

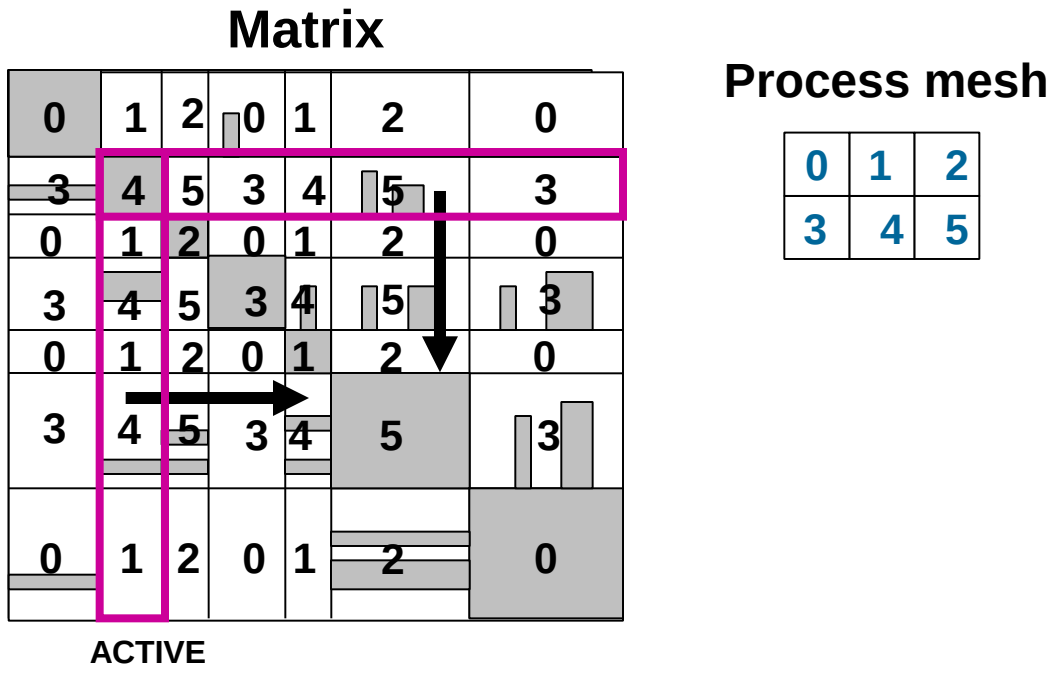


Putting together



Application simulation: sparse LU

- **SuperLU_DIST** [Li/Demmel/Grigori]
- **Right-looking – relatively more WRITEs than READs**
- **2D block cyclic layout**
- **One step look-ahead to overlap comm. & comp.**



Simulating update of the k-th block row

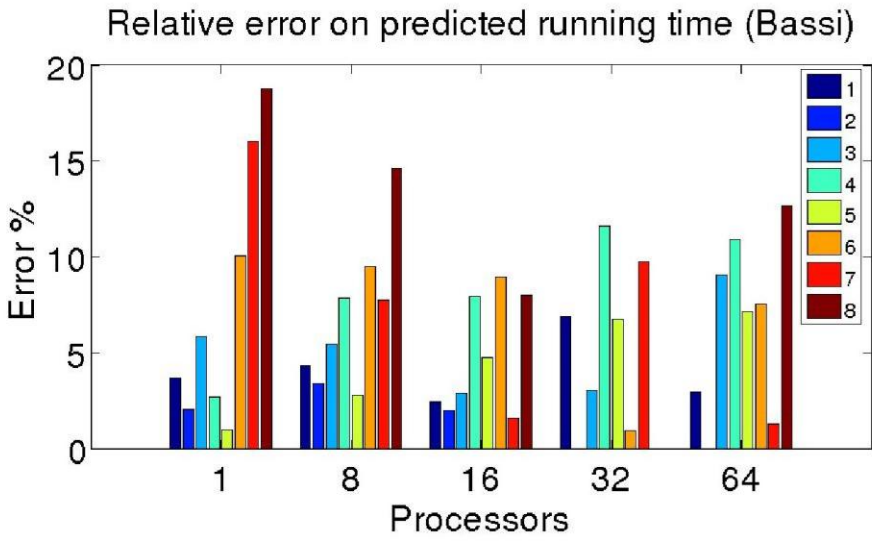
```
For  all block b in UBLOCKS(k) do
    p ← OWNER(b)
    time[p] += memory_read  (p, b)
    for  all colomn j in b do
        if  col  j not empty  then
            time[p] += lookup (dtrsv, sizeof(j))
        endif
    endfor
Endfor

For  all processor  p  that owns a block in row k  do
    time[p] += memory_update (p, stack)
endfo
```

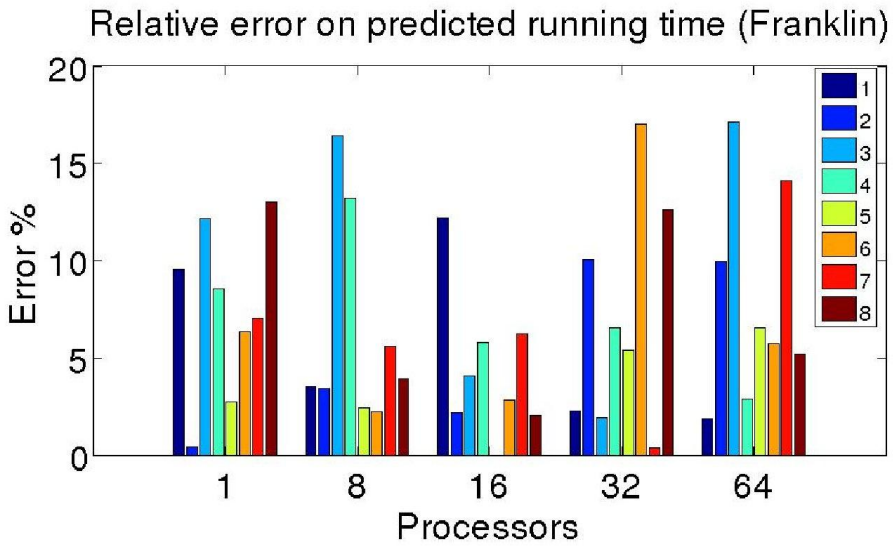
Validation

- **8 medium sized matrices from Univ. of Florida collection**
- **IBM Power5: 1.9 GHz, 8 CPUs/node**
  - **Average absolute prediction error 6.1%**
- **Cray XT4: 2.3 GHz, 4 cores/node**
  - **Average absolute prediction error 6.6%**

IBM Power5



Cray XT4



Prototyping new algorithms

- **Latency-reducing panel factorization**
- **Sparse Cholesky**

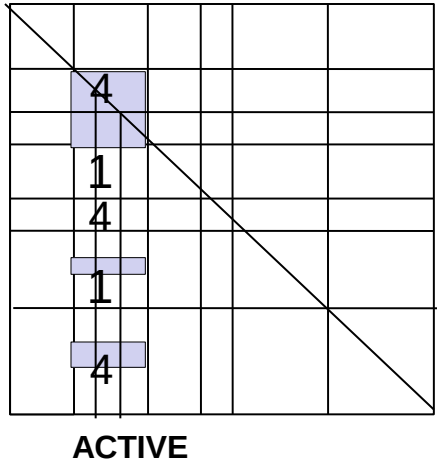
Latency-reducing panel factorization

- **Panel factorization often on the critical path**

  For each column within panel:
  - **Scale column**
  - **Rank-1 update for rest of columns**
- **Two approaches**

  1) Broadcast row for each rank-1 update

  2) Asynchronous Isend/Irecv

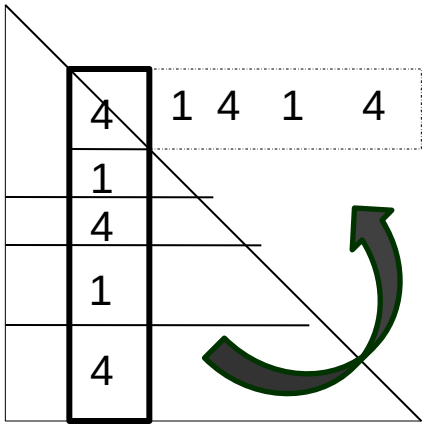


- **Simulation shows 2) has more messages but fewer synchronizations, more parallelism**
- **Implementation of 2) led to 25% speedup @ 64 procs**

New design: sparse Cholesky

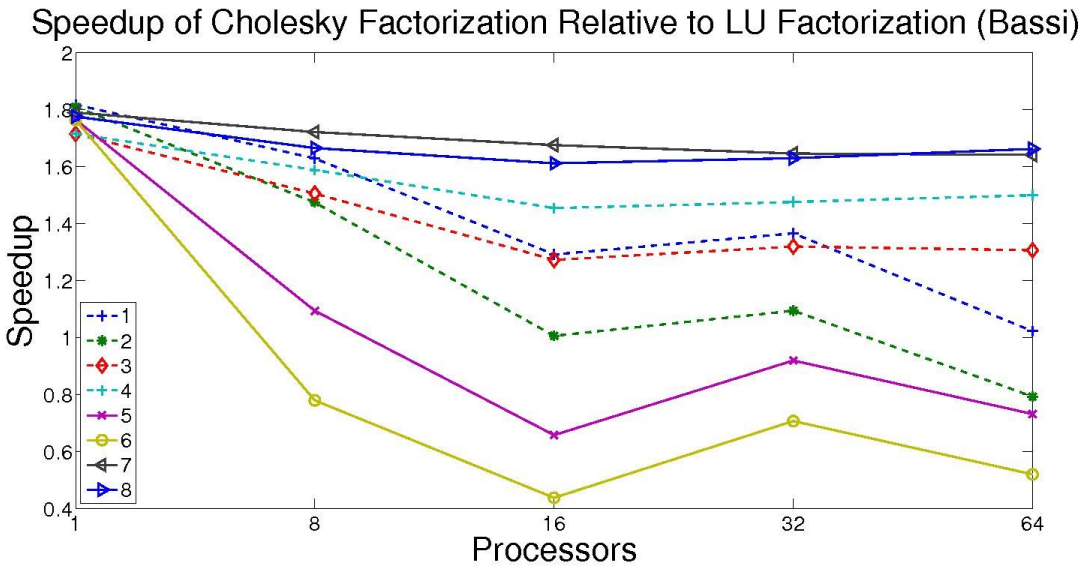
- Use the same data layout/mapping as in SuperLU_DIST
  - Block column of L, block row of U on each processor
- Store only half of the matrix; half operations
  - No need for block row, nor upper triangular update
- Simple strategy
  - Block column of L is aggregated and replicated on the column proccessors
- Advantage: no need to change sparse data structures
- Disadvantages
  - Larger communication volume
  - Synch. of procs along each block column

0	1	2
3	4	5



Simulated results of sparse Cholesky

- **Predicted that collective communication becomes major scalability bottleneck**
- **IBM Power5: Cholesky vs. LU, 8 matrices**



- **Future design: use truly block-oriented data format**
  - **Minimum comm. volume; no synch.**
  - **Comm. pattern not restricted within processor row**

Conclusions

- **Simulation-based performance modeling is very useful**
  - **Model complex applications that are input-dependent**
    - **Sparse LU: prediction errors mostly within 15%, average under 7%**
  - **Enable rapid prototyping of new algorithms**
  - **Evaluate new HPC system design**

- **Future work**
  - **Model resource contentions: memory, network**