Structure-Aware Parallel Algorithm for Solution of Sparse Triangular Linear Systems

Ehsan Totoni

Advisors:

Michael T. Heath

Laxmikant V. Kale

{totoni2, heath, kale}@illinois.edu

Triangular Solution

$$x_i = (b_i - \sum_{j=1}^{i-1} l_{ij}\, x_j)/l_{ii}, \quad i = 1, \ldots, n$$

Used in solution of linear systems, least squares
- Many times iteratively
- Both direct and iterative methods

Example: Preconditioned Conjugate Gradient (PCG)
- With Incomplete-Cholesky as preconditioner
- Same number of non-zeros as coefficient matrix or more
- Usually, half of iteration's operations is triangular solve
-If triangular solve doesn't scale:
  Parallel PCG's speedup is at most **(2)**
  According to Amdahl's law

Very resistant to parallelism!
- Minimal concurrency
  Lots of structural dependencies
- Small work per data
  Just one multiply-add for most entries!

Standard linear algebra packages very slow
- E.g. HYPRE, SuperLU_DIST
- Slower than sequential many cases

New algorithm to extract parallelism
- By adapting to matrix's sparse structure

Reference:

Structure-Adaptive Parallel Solution of Sparse Triangular Linear Systems, Totoni et al., PPL Technical Report 12-42, 2012.

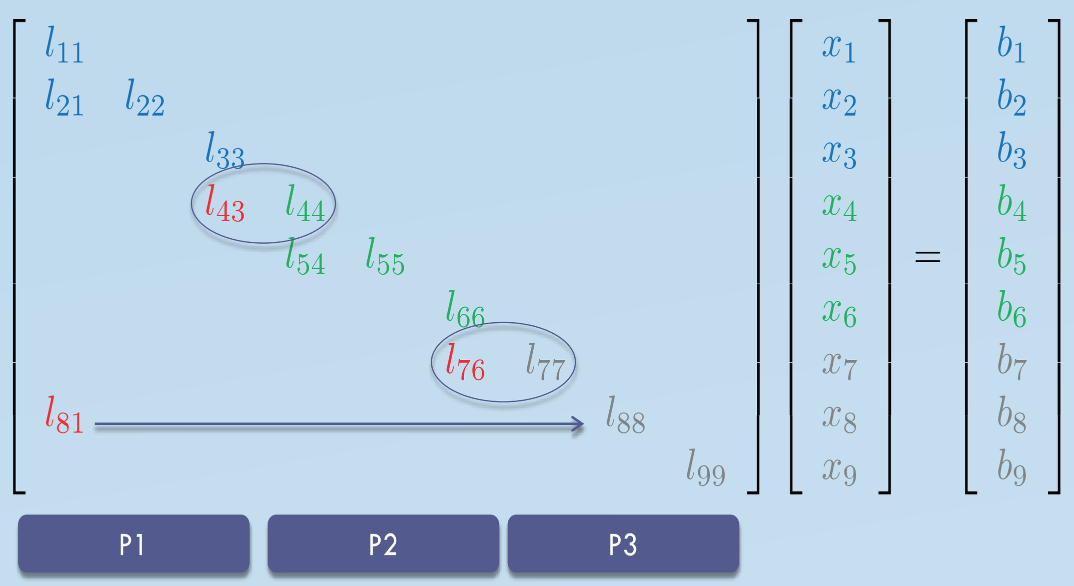
Code:

https://charm.cs.illinois.edu/benchmarks/triangularsolver.git

http://charm.cs.illinois.edu

Our Parallel Algorithm

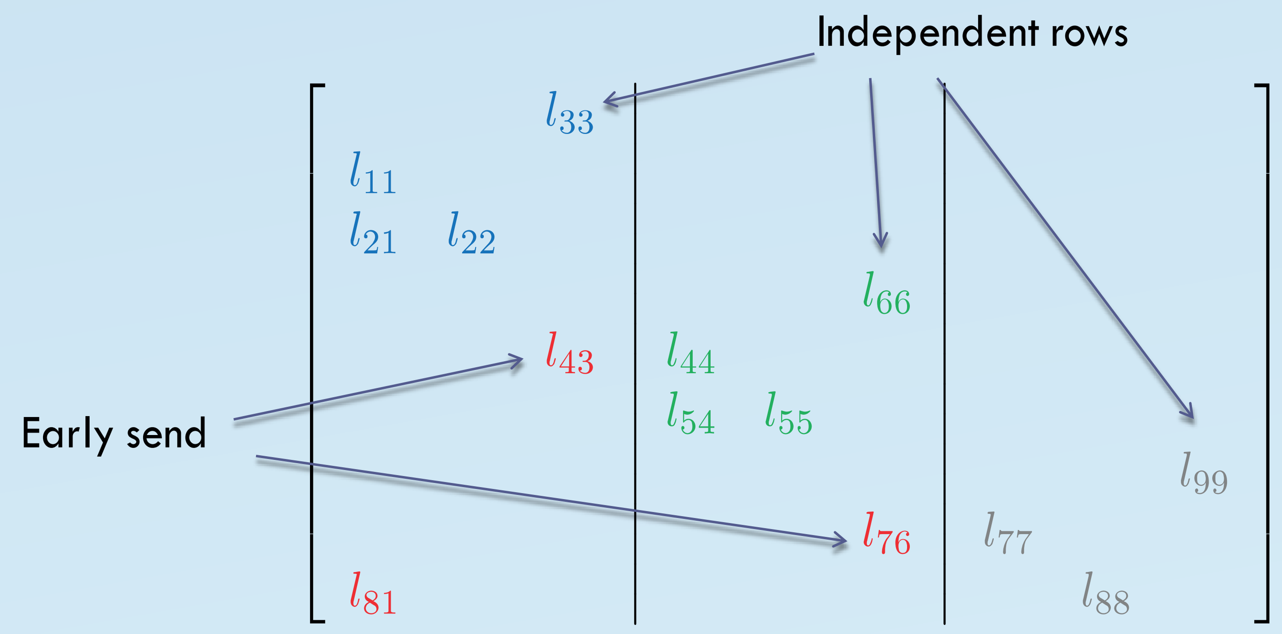
Data decomposition: blocks of columns
- Typical dependencies between processors shown



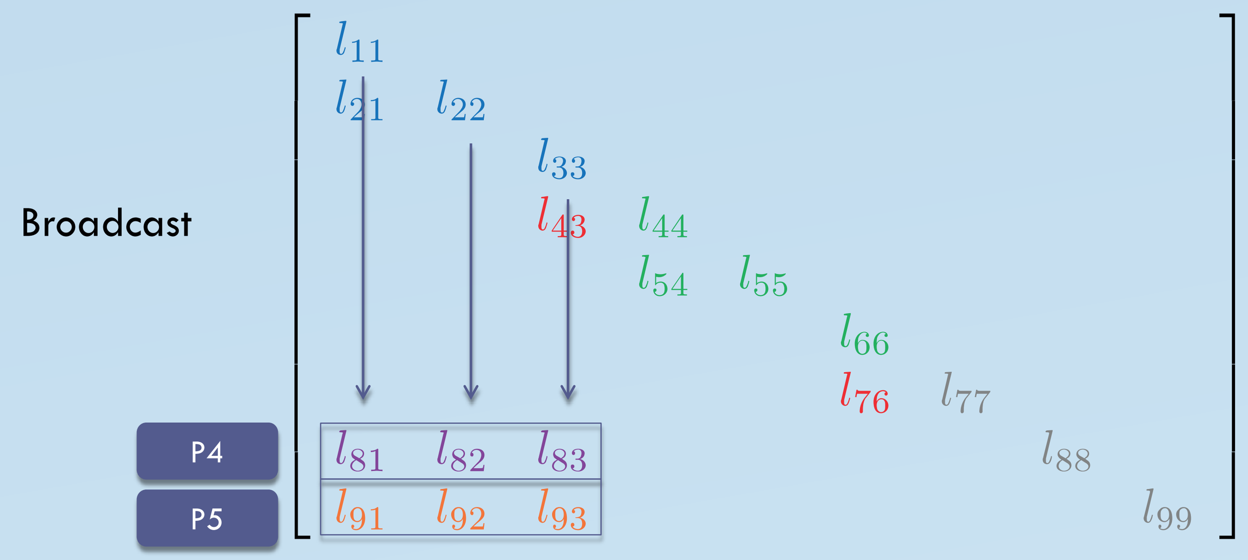
Three strategies for more parallelism:

1- Reordering rows to find independent rows
- Some rows of a processor don't depend on other processors
  (no non-zero on left)
- Also don't depend on other rows that are dependant
  (no non-zero on those columns)

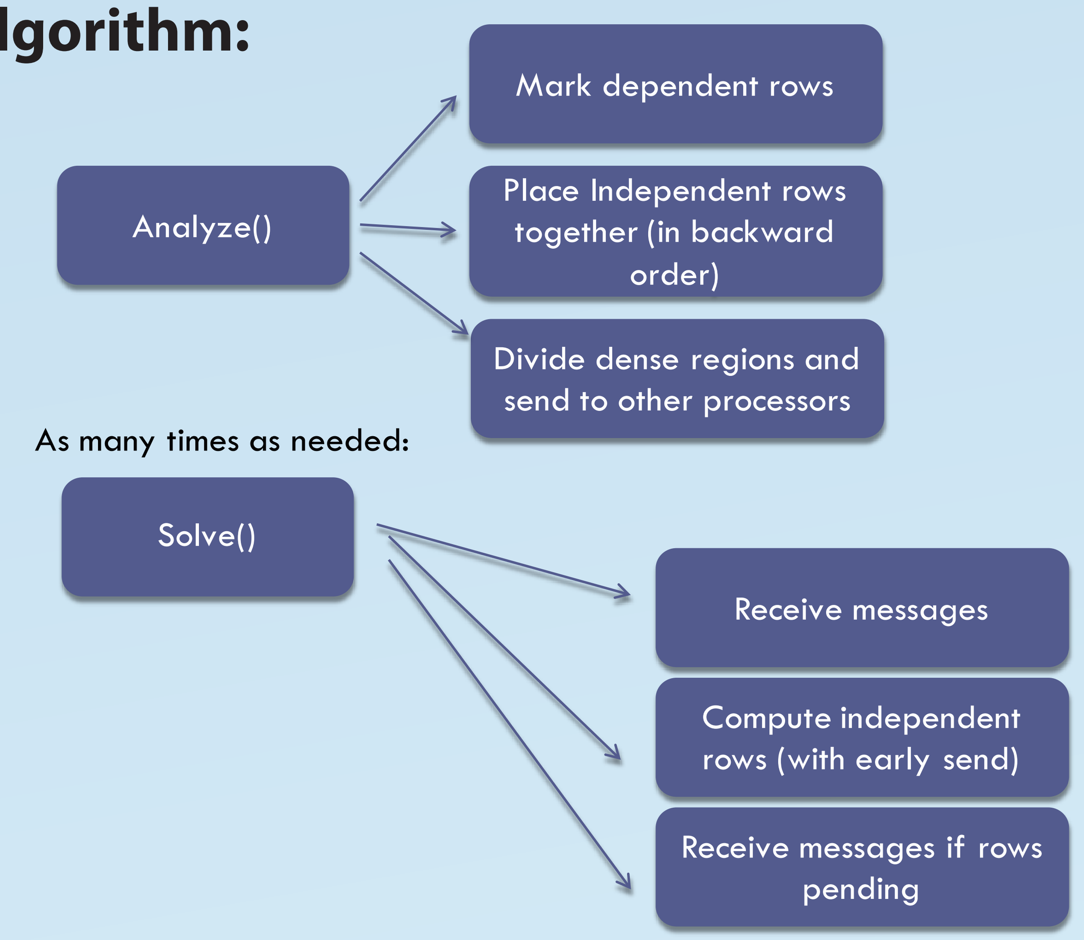
2- Early send of critical data
- Processors typically depend on only some rows of others
- So we process those rows first and send them earlier
- Progress along critical path is accelerated



3- Divide dense regions and send to other processors for more parallelism
- "Dense" only means enough non-zeros to amortize the cost
- Broadcast needed x values when computed



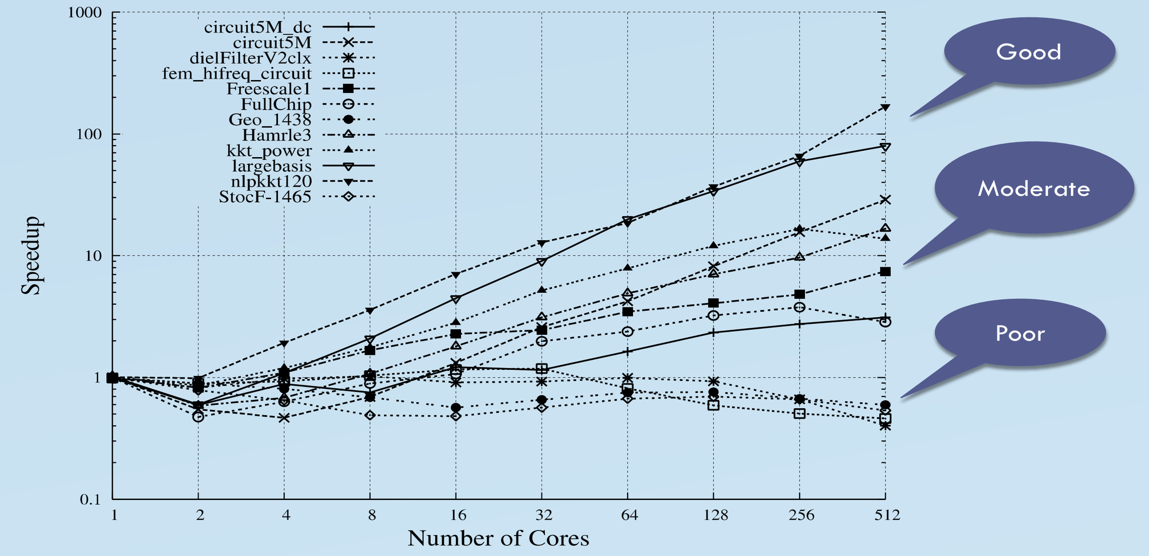
Algorithm:



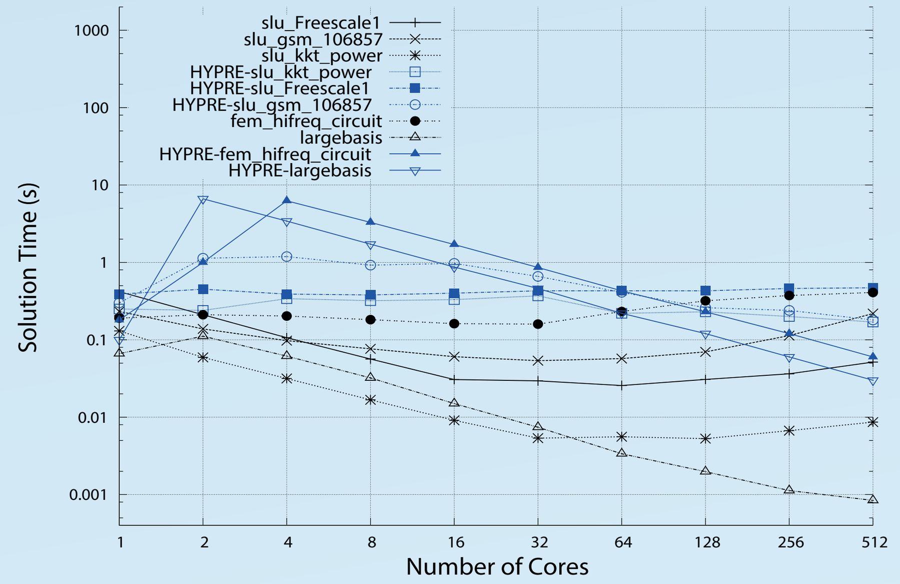
Implementation:
- In Charm++, only 692 Source Lines Of Code (SLOCs)
- Integration to an MPI package in progress
  Using Charm++ interoperability
- Some other optimizations:
  Over-decomposition for more overlap
  Message priorities for faster critical path progress
  Message aggregation

Evaluation

Strong scaling evaluation:
- Using real application matrices from Florida Collection
- On 512 nodes of BlueGene/P (1 core per node used)
- Speedups compared with best sequential code
- Performance highly depends on structure of matrix



Comparison to HYPRE's triangular solver:
- Our algorithm is 35 times faster than HYPRE
  (blue curves are HYPRE) for "largebasis" on
  512 cores
- SuperLU_DIST is even slower (not shown)



Comparison to Level-set algoritm
- Barriers of level-set are bottlenecks
- Longest chain of communication steps (critical path)

for a sample of matrices:

Matrix	Our algorithm	level-set algorithm
circuit5M	2	18
kkt_power	3	17
Freescale1	18	216
Hamrle3	25	31083
Geo_1438	87	5823