Accelerating the Iterative Linear Solver for Reservoir Simulation

Wei Wu[1], Xiang Li[2], Lei He[1], Dongxiao Zhang[2]

1 Electrical Engineering Department, UCLA

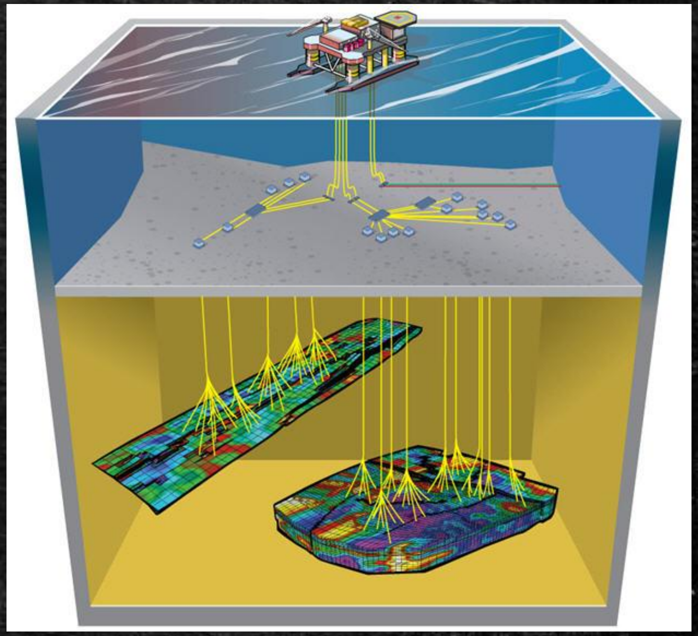
2 Department of Energy and Resources Engineering, College of Engineering, PKU

Outline

- Background
  - Reservoir simulation problems and its mathematical formulation
  - Similarities and differences with circuit simulation
- Method
- Experiment results

Background: Reservoir simulation

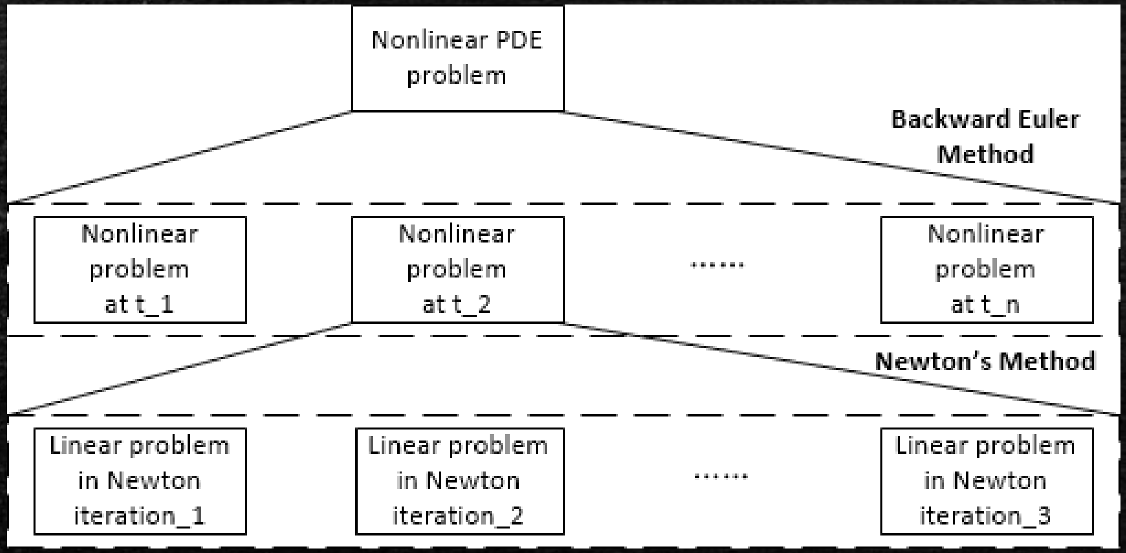
- Problem in petroleum engineering:
  - Petroleum reservoir is still the major energy supply in the modern society.
  - Modern petroleum reservoir simulation serves as a primary tool for quantitatively managing reservoir production and designing development plans for new fields.



- Petroleum reservoir simulation:
  - Reservoir simulators: A set of nonlinear partial differential equations (**PDE**) about mass and energy conduction. (Which is similar to circuit simulators)

Background: Reservoir simulation

- Petroleum reservoir simulation:
  - A nonlinear partial differential equations (**PDE**) problem.
  - The partial differential problem is solved using backward Euler method in several time steps.
  - In each time step, the nonlinear problem is solved by Newton's method, where each Newton step is solving a linear equation: Ax=b



$$\begin{cases} x' = F(x,t) \\ x(t_0) = x_0 \end{cases}$$

$$x(t_{k+1}) = x(t_k) + hF(t_{k+1}, x(t_{k+1}))$$

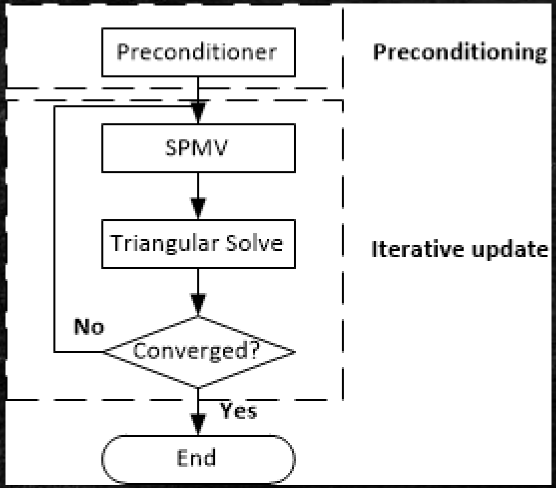
$$J_F(x_n)(x_{n+1} - x_n) = -F(x_n)$$

$J_F(x_n)$ is the Jacobian of $F(x_n)$

Solving a linear system Ax=b

- Iterative method
  - i.e.: Gassian Siedel, GMRES
  - 3 major components:
    - Preconditioner, triangular solve, SPMV

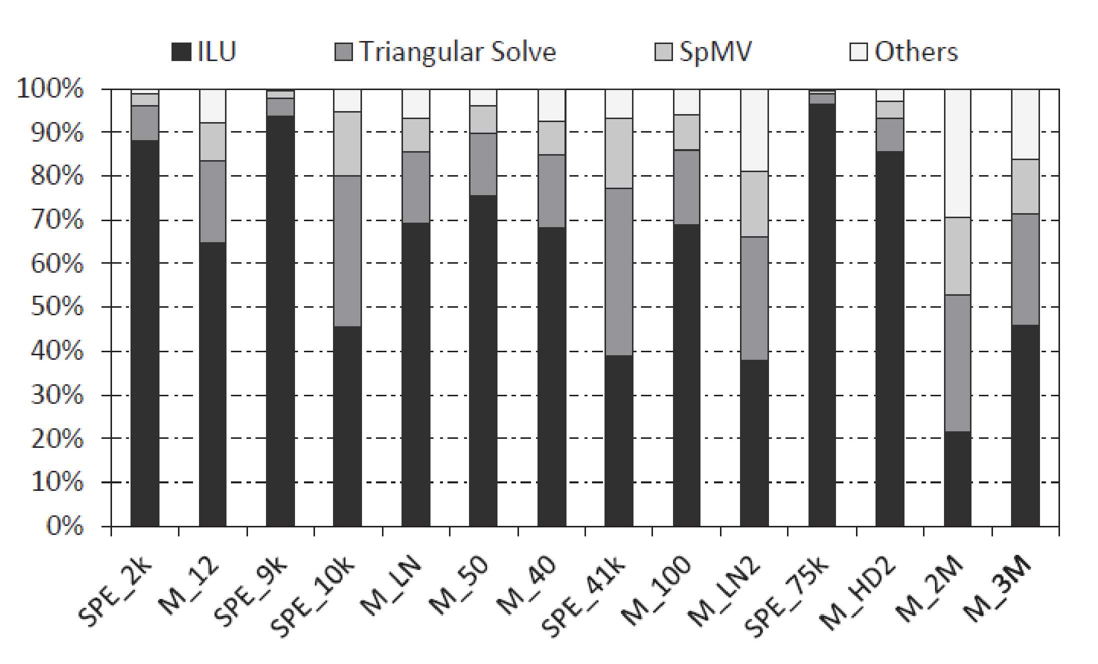
$$x_{i+1} = x_i - \omega \mathbf{P}^{-1}(\mathbf{A}x_i - b)$$



- Direct method
  - i.e. LU factorization
  - Calculate A=LU
  - Then solve Ax=b as Ly=b and Ux=y

"Hotspot" in Reservoir solver

- A profiling of Reservoir simulation
  - (**ILU preconditioner**, **triangular solve**, SPMV) consists of a large portion of the runtime in reservoir simulation
  - ILU and triangular solve are difficult to parallelize. (no existing parallel implementation released)



Outline

- Background
- **Method**
  - **Block-wise data operation**
  - **Parallel ILU(P) and triangular solve for reservoir simulation**
  - **Task dependency tree and hybrid parallel mode**
- Experiment results

Sparse LU Factorization

- Optimized sparse storage format?
  - Store in compressed sparse column format
    - Only store the nonzero entries
- Blockwise compressed sparse column is used for reservoir matrices
- Algorithms needs to consider the sparse formatting
  - **Fillins** are introduced when updating column “i” using a different column “j”
  - A sparse matrix might turn out to be dense after LU factorization

$$\begin{pmatrix} 1 & & & a & & & \\ & 2 & & & b & & \\ c & d & 3 & & & & \\ & e & & 4 & f & & \\ & & & & 5 & & g \\ & & & h & i & 6 & j \\ & & k & & l & & 7 \end{pmatrix}$$

nzval	1 c	2 d e	3 k	a 4 h	b f 5 i l	6	g j 7
rowind	1 3	2 3 4	3 7	1 4 6	2 4 5 6 7	6	5 6 7
colptr	1 3 6 8 11 16 17 20						

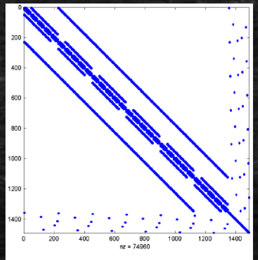
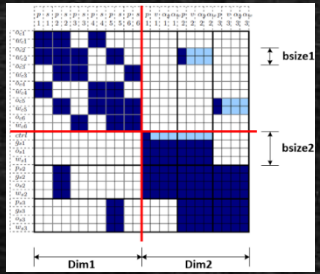
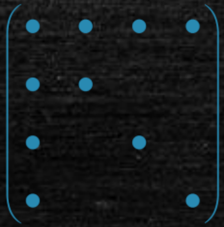
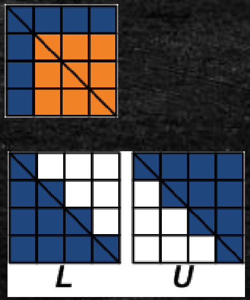


Figure 3.2: Typical matrix in reservoir simulation

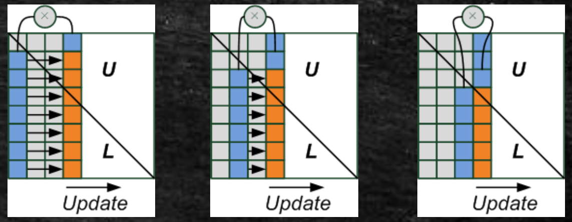


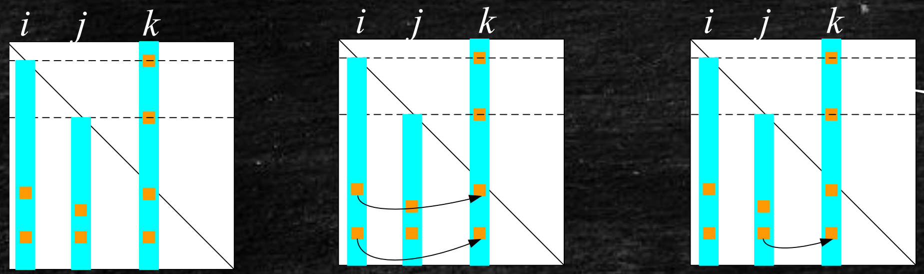




Sparse Incomplete LU factorization

- ILU algorithm?
  - Incomplete LU factorization
  - It will not introduce all the fillin during the factorization
    - As a special case, the ILU0 will not consider any fillin during the factorization.
    - If a column is going to be updated by another column in L, only the existing nonzeros will be considered.



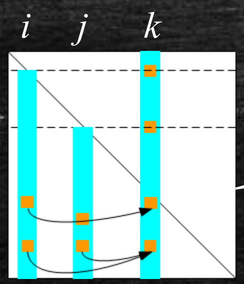


The NNZs in the upper part of a column determine which column is required for the updating of this column

- Why is it difficult to parallelize ILU?
  - Computations in ILU are sequential and data dependency exists. (i.e.)

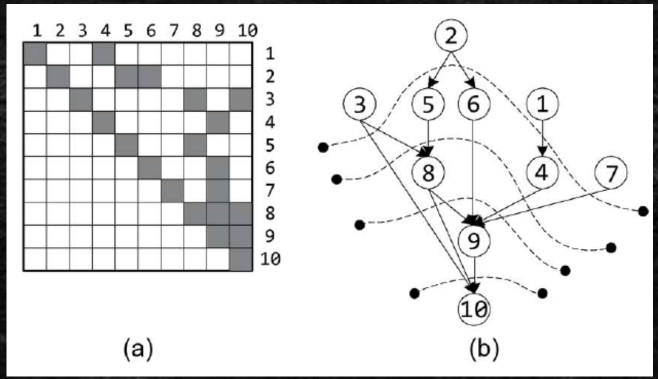
Parallel sparse ILU algorithm

- Analyze the data dependency:



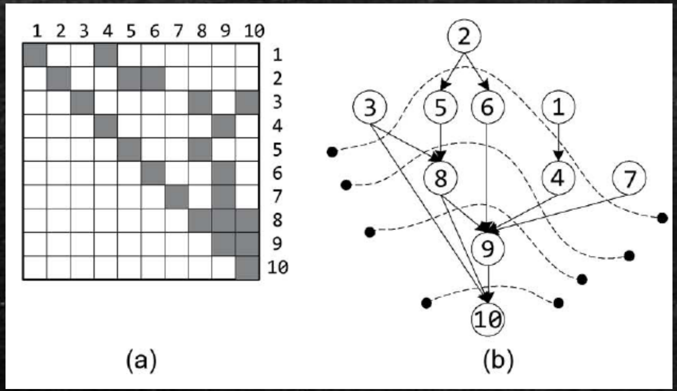
1 Considering process each column as a tasks
2 Task k depends on task i and j only when nonzeros exist in (i, k) and (j, k)

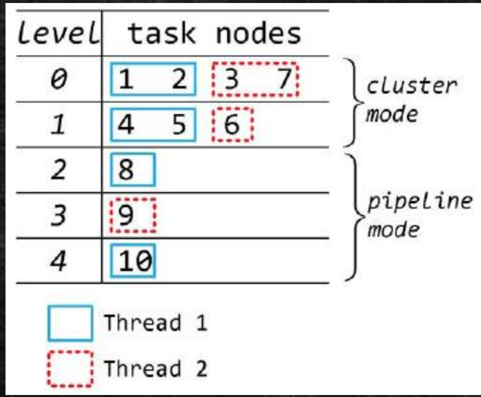
- Represent the data dependency in a data flow graph:
  - Exact describes the data dependency
  - Difficult to implement in practice



Parallel sparse ILU algorithm

- A better representation of data dependency for implementation



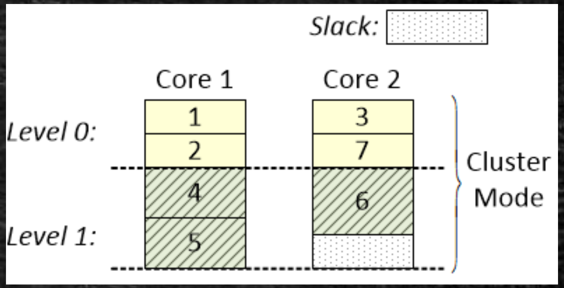


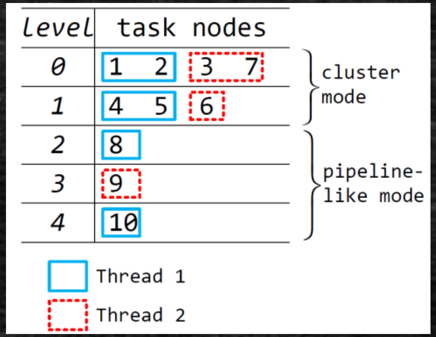
- Tasks in Elimination Graph are partitioned into different levels:
  - ***Level*** is actually the "earliest starting time" of each task
  - No data dependency exists between tasks in the same ***level***
  - The ***level i*** need the data from ***level 0~i-1***

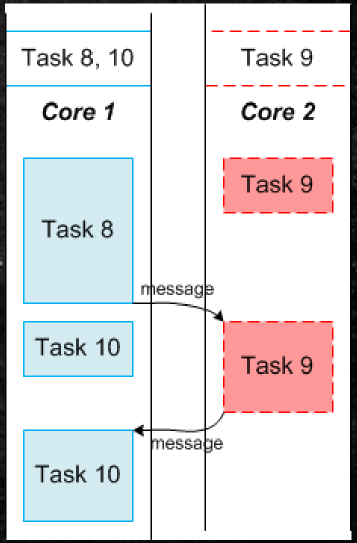
Xiaoming, C., Wei W., et al. (2011). "An EScheduler-Based Data Dependence Analysis and Task Scheduling for Parallel Circuit Simulation." Circuits and Systems II: Express Briefs, IEEE Transactions on 58(10): 702-706.

Two parallel modes

- Cluster mode:
  - Distribute independent tasks to multiple cores level by level
  - Low overhead
- Pipeline mode
  - Schedule the tasks with data dependency as a pipeline by inter-thread message passing
  - Try to overlap these tasks
  - High overhead







Parallel Triangular solve

- Triangular solve calculates x from Ax=LUx=b in two steps:
  - Solve: L(Ux)=Ly=b
  - Solve: Ux=y

- Each task is much “lighter” compared with the task in ILU
  - Similar task scheduling algorithm can be applied.
  - Thread synchronization in pipeline mode will dominate the runtime.
  - Only cluster mode task scheduling is applied.

Outline

- Background
- Method
- **Experiment results**
  - Experiment setup
  - Comparison between sequential version of block ILU and existing work (ITSOL and PARDISO)
  - Scalability with multicore processors

N. Li, B. Suchomel, D. Osei-Kuffuor, and Y. Saad, "ITSOL: Iterative solvers package," University of Minnesota, 2008.

Experiment setup

- Hardware platform: Intel Core™ i7-3820
  - 8 concurrent threads at 3.6 GHz
- Software included for comparisons:
  - PARDISO (direct solver), ITSOL (iterative solver)
  - Block ILU (Proposed)
    - Including both sequential and different parallel implementations
- Algorithm configurations
  - ITSOL and Block ILU
    - level of Fill is configured as 1 in ILU(p)
    - Tolerance for stopping iteration is set to 1e-6
  - PARDISO is loaded from Intel MKL library
- Test Matrices
  - 14 reservoir matrices dumped from an industrial reservoir simulator
  - The dimension is upto 3 million by 3 million

Speedup contribute by **blockwise data structure (Sequential version)**

Table I

COMPARISON ON THE SINGLE-THREAD RUNTIME ON MATRICES GENERATED FROM INDUSTRIAL RESERVOIR SIMULATOR

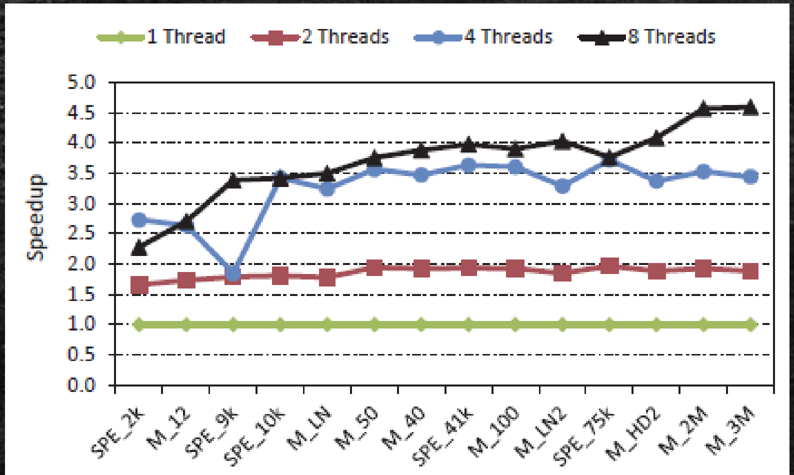
Test Cases	# of blcoks		Block size		# of rows	# of iterations	ILU(1) runtime (ms) and speedup			Total runtime (ms) and speedup			PARDISO
	RRP	WWP	RRP	WWP			ITSOL	block ILU[1]	Speedup	ITSOL	block ILU[1]	Speedup	
SPE_2k	2592	0	10	1	25920	12	440.0	87.5	5.0	660.0	158.8	4.2	1648.3
M_12	12344	59	3	4	37268	15	60.0	6.9	8.7	100.0	47.9	2.1	649.1
SPE_9k	9408	0	7	1	65856	4	610.0	128.0	4.8	1030.0	179.9	5.7	11472.0
SPE_10k	10368	0	10	1	103680	119	1820.0	327.9	5.5	2730.0	2945.3	0.9	24664.4
M_LN	43679	59	3	4	131273	12	250.0	27.5	9.1	380.0	146.0	2.6	5584.1
M_50	50000	20	5	3	250060	16	1250.0	327.0	3.8	2460.0	801.9	3.1	44496.8
M_40	100000	80	4	3	400240	12	680.0	416.6	1.6	1000.0	747.0	1.3	304841.8
SPE_41k	41472	0	10	1	414720	160	7780.0	1377.5	5.6	11580.0	15838.2	0.7	495332.9
M_100	100000	20	5	3	500060	22	2730.0	669.3	4.1	5230.0	1989.5	2.6	505445.2
M_LN2	260985	184	2	3	522522	31	N/A	97.0	N/A	N/A	991.8	N/A	35698.2
SPE_75k	75264	0	7	1	526848	2	5240.0	1031.7	5.1	7530.0	1285.4	5.9	1235692.9
M_HD2	368326	10	3	4	1105018	4	2170.0	246.1	8.8	2850.0	624.7	4.6	237800.7
M_2M	1094421	425	2	3	2190117	65	2560.0	493.8	5.2	5700.0	8141.0	0.7	N/A
M_3M	1094421	425	3	4	3284963	33	7790.0	1240.1	6.3	14780.0	9242.9	1.6	N/A

[1] **block ILU** refers to the single thread version of the proposed iterative solver with ILU

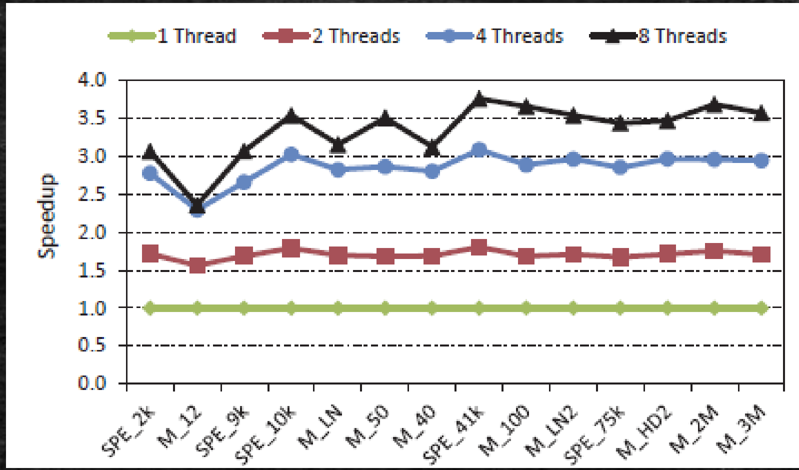
- Sequential version of block ILU(1) is 5.2x faster than ITSOL on geometric average.
  - Blockwise data structure can get better cache hit rate
- The iterative solver is much faster than direct solver (PARDISO) on these matrices.

Parallel Scalability

- Speedup of multi-thread program over single thread program



ILU1 preconditioner
2.3-4.6x speedup (3.6x on average)



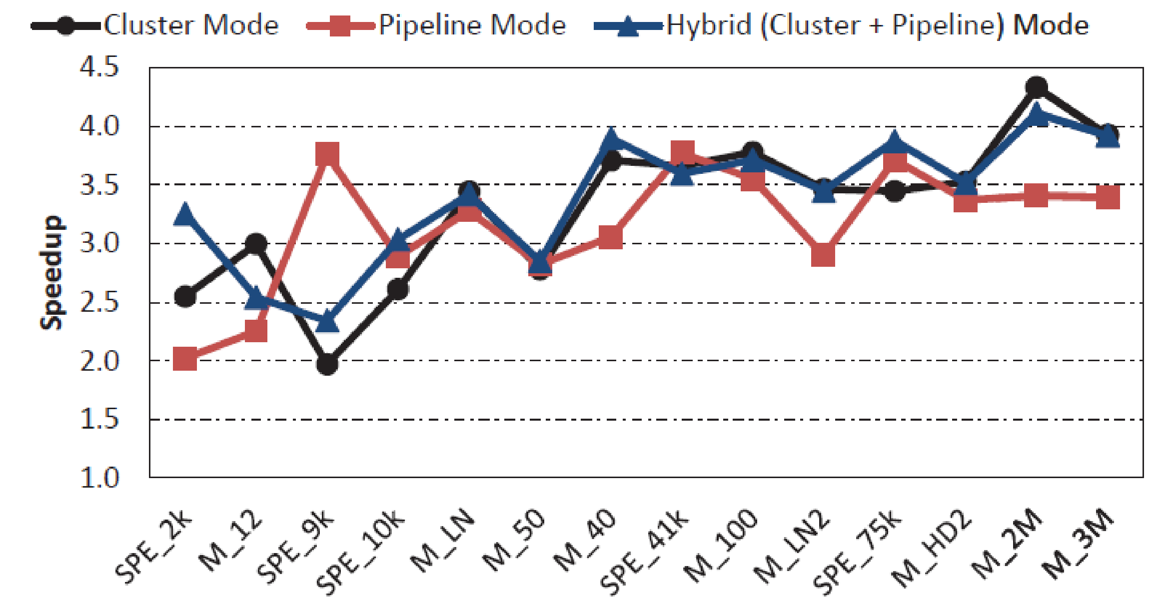
Triangular solve
2.4-3.8x speedup (3.3x on average)

- The total runtime of reservoir simulator is reduced to less than ½ on a 4-thread machine
- Better speedup for cases with
  - **Higher dimension** (comparatively less overhead)

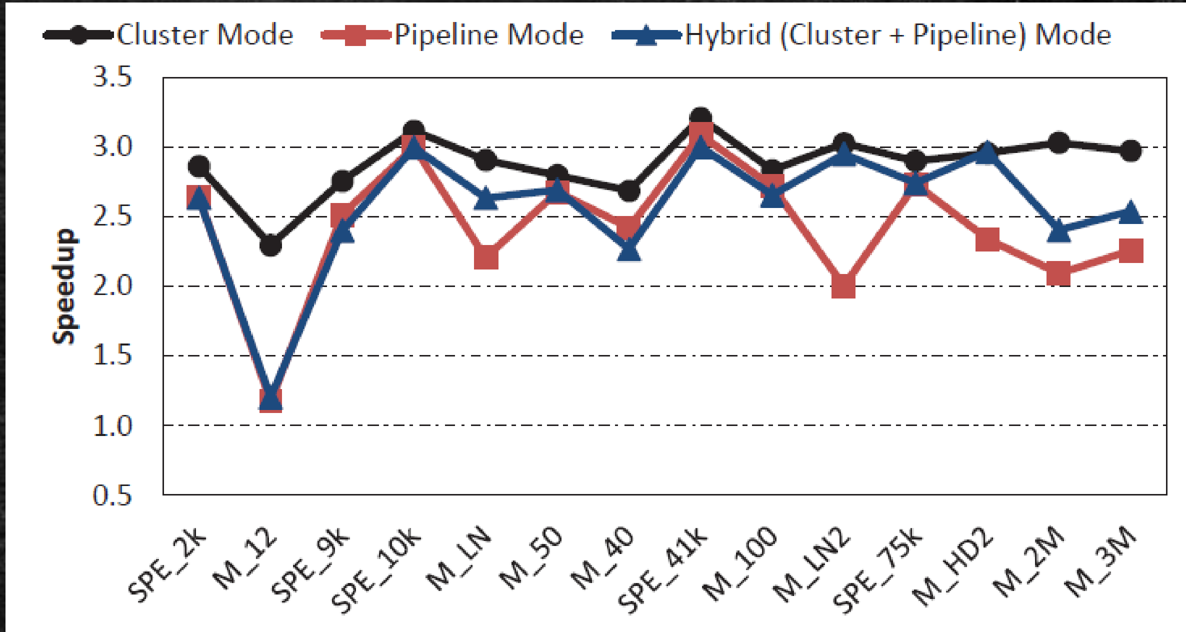
Speedup of ILU and triangular solve under different parallel mode

Hybrid mode is not always the winner!

Performance of **Pipeline, Cluster** and **Hybrid** mode



(a) ILU(1)

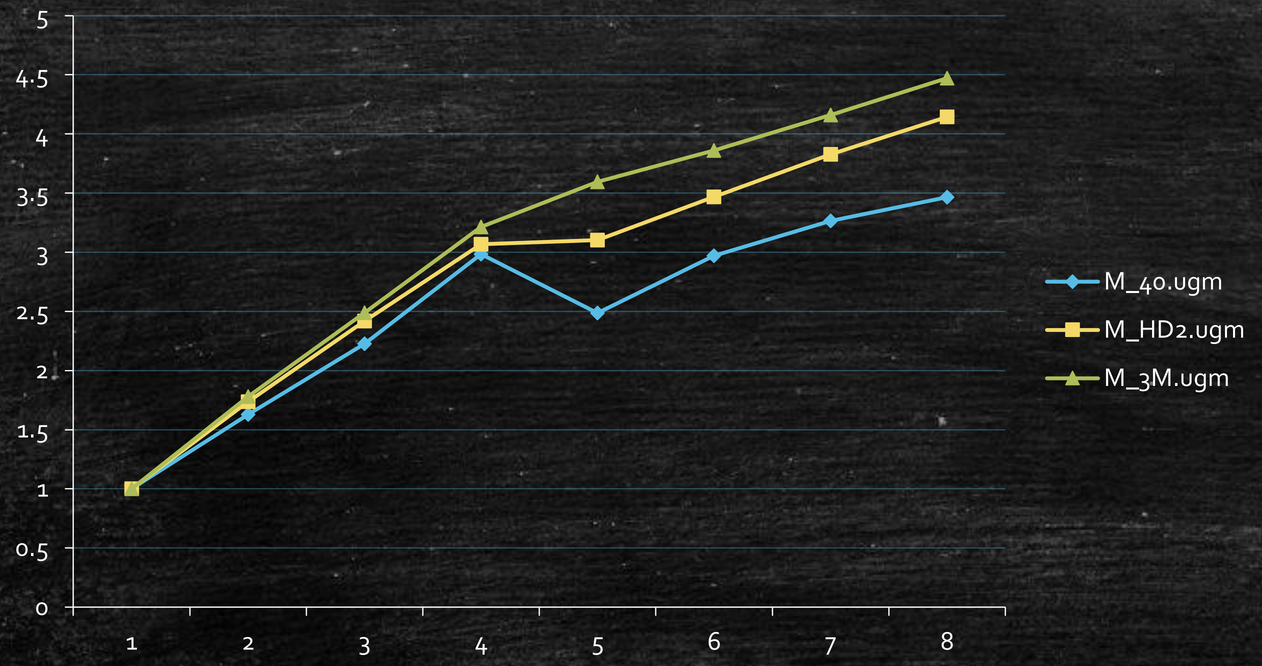


(b) Triangular solve

- Hybrid mode is the winner for ILU(1), but the cluster mode is the best choice for triangular solve.
  - ➢ **That is because the computation load of each task is much lighter, resulting to a high overhead of using pipeline.**

Experiment results – Scalability

- Speedup when # of threads increase



Speedup scales while the # of threads increase

We can still notice the trend of increase speedup at 8 threads.

Conclusion

- ILU0 can triangular solve are identified as the bottleneck of the reservoir simulators.

- Acceleration:
  - By taking advantage of the blockwise data structure, we can get 5.2x speedup on average.
  - More speedup can be achieved by scheduling ILU and solve based on the dependency tree:
    - 3.6x and 3.3x speedup are achieved on ILU and solve respectively
  - The total speedup of ILU(1) is 18.7x compared with the well-known ITSOL package

- Scalability:
  - Speedup scales with # of threads
  - Better speed up is achieved on larger problems and matrices with larger dimension.

Thanks for your attention!

Please address question to Wei Wu (weiw@seas.ucla.edu).