清華大學
Tsinghua University

Nested Dissection Based Parallel Transient Power Grid Analysis on Public Cloud Virtual Machines

Jiawen Cheng, Zhiqiang Liu, Lingjie Li, Wenjian Yu

Department of Computer Science and Technology,
Tsinghua University, Beijing 100084, China

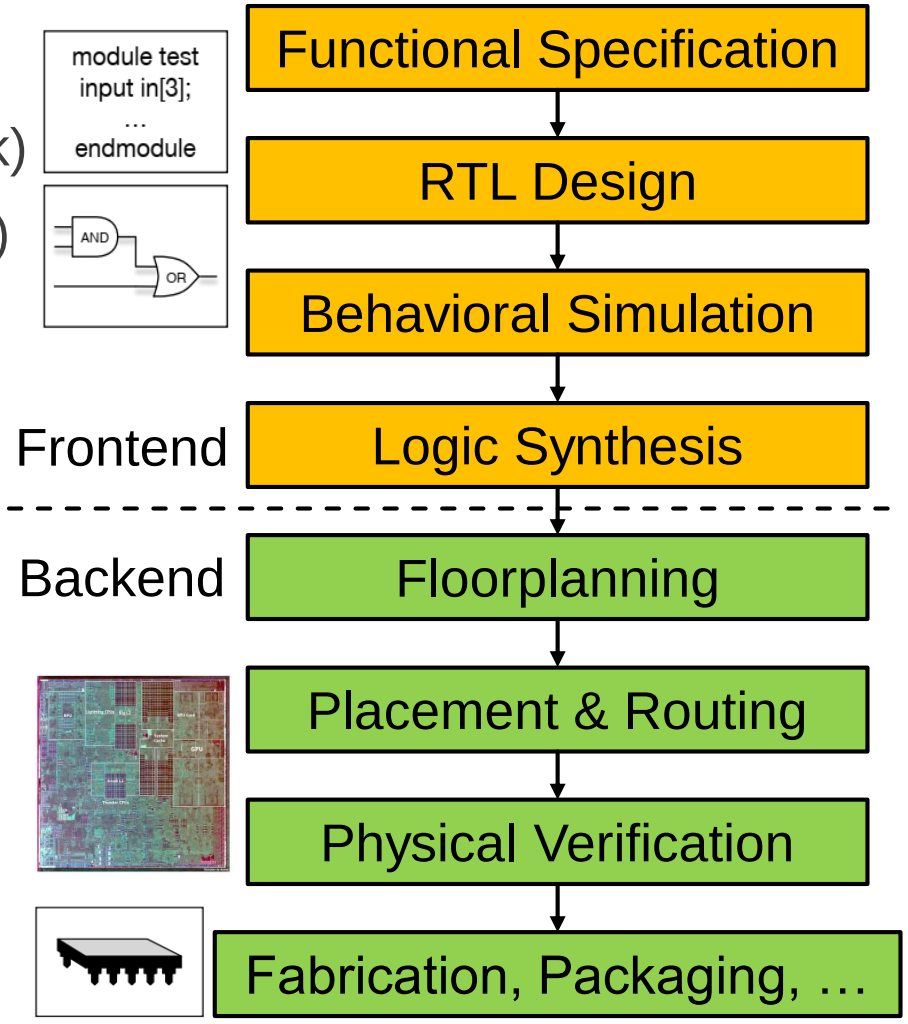
2024-01-24

Outline

- Background
- Nested Dissection Based Parallel Transient Power Grid Analysis with Cloud Computing
- Experimental Results
- Conclusion

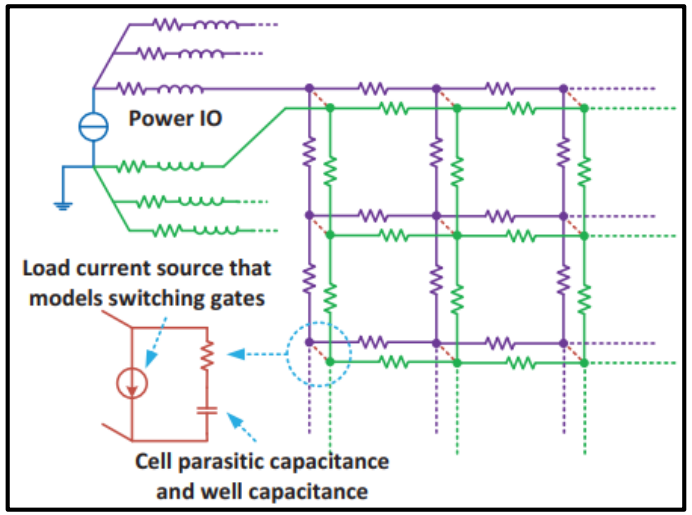
Integrated Circuit (IC) Design Flow

- Backend related EDA technology
  - Physical design optimization (P&R, power delivery network)
  - Physical modelling analysis (RC extraction, delay analysis)
  - Signoff verification (signal integrity, **circuit simulation**)
  - Layout verification
  - Mask optimization
- Challenge of accurate circuit simulation
  - **Parasitic effect** from interconnects with widths < 0.25μm
  - Extremely **time consuming** due to large scale



Power Grid Analysis

- Compute the IR drop under various current load
- Critical for analyzing the robustness of the circuit
- Challenge
  - circuit scale↗, IR drop margin↘, demand for accuracy↗
- **Efficient**, **accurate**, **memory-saving** software
- Classification
  - Direct current: only consider resistors
  - Transient: consider resistors, capacitors and inductors (RLC) **accurate** √



Transient Power Grid Analysis

- Modeled as an RLC network, a first-order ODE system is derived

$$C\dot{x}(t) + Gx(t) = Bu(t) \tag{1}$$

C: capacitance and inductance matrix

G: conductance matrix $u(t)$: input sources

B: input selector matrix $x(t)$: node voltages branch currents

- Apply trapezoidal rule with time step h, (1) becomes $Ax = b$ problem!

$$\left(\frac{C}{2} + \frac{G}{2}\right)x(t+h) = \left(\frac{C}{2} - \frac{G}{2}\right)x(t) + B\frac{u(t+h) + u(t)}{2} \tag{2}$$

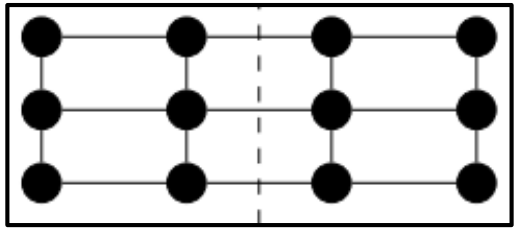
Solving Linear Equation Systems

	Direct: LU/Cholesky + Fwd/Bwd Substitution	Iterative: Preconditioning + CG/GMRES
Pros	Fast substitution, accurate	Variable time step, memory saving
Cons	Fixed time step, fill-ins	Accuracy loss, convergence issue

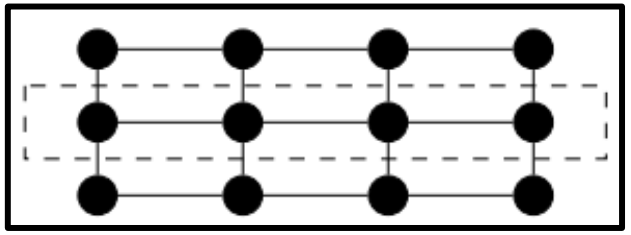
- Motivation
  - Design an efficient, accurate and robust solver $\Longrightarrow$ **Direct method**
  - Accelerate solving and reduce memory cost $\Longrightarrow$ **Distributed computing**
  - Public cloud computing platforms make distributed computing feasible
  - Internal structure of the circuit won't be exposed $\Longrightarrow$ **Public cloud computing**

Domain Decomposition Method (DDM)

- Symmetric matrix⟺Undirected graph
- Use the edge/vertex separator to partition the graph
- Arranging subdomains and separator together



Edge separator



Vertex separator

$$\begin{pmatrix} D_1 & & & F_1 \\ & D_2 & & F_2 \\ & & \ddots & \vdots \\ & & D_m & F_m \\ E_1 & E_2 & \cdots & E_m & S \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \\ g \end{pmatrix} \Longrightarrow$$

Schur complement

$$(S - ED^{-1}F)y = g - ED^{-1}f$$

$$D_i x_i = f - F_i y$$

- Solve x_i in parallel😊 Solving y is bottleneck when m is large😢
- Learn from nested dissection to reduce the size of Schur complement

Our contribution

- **Efficient, accurate and robust transient PG analysis**
  - Multi-level distributed parallel LU factorization and forward/backward substitution approach based on nested dissection
  - Scheme for efficient RHS updating to further exploit parallelism
- **Leverage cloud computing for distributed parallelism**
  - Use public cloud computing while ensuring security
  - Ideal for deployment on cloud computing platforms without InfiniBand

Outline

- Background
- Nested Dissection Based Parallel Transient Power Grid Analysis with Cloud Computing
- Experimental Results
- Conclusion

Overall Idea – LU Factorization

- Top level of partition:

$$\begin{pmatrix} \boldsymbol{D}_1 & & \boldsymbol{F}_1 \\ & \boldsymbol{D}_2 & \boldsymbol{F}_2 \\ \boldsymbol{E}_1 & \boldsymbol{E}_2 & \boldsymbol{S} \end{pmatrix} = \begin{pmatrix} \boldsymbol{L}_{11} & & \\ & \boldsymbol{L}_{22} & \\ \boldsymbol{L}_{31} & \boldsymbol{L}_{32} & \boldsymbol{L}_{33} \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_{11} & & \boldsymbol{U}_{13} \\ & \boldsymbol{U}_{22} & \boldsymbol{U}_{23} \\ & & \boldsymbol{U}_{33} \end{pmatrix}$$

- Solve the submatrices in L, U

$$\boldsymbol{L}_{31} = \boldsymbol{E}_1 \boldsymbol{U}_{11}^{-1}, \boldsymbol{L}_{32} = \boldsymbol{E}_2 \boldsymbol{U}_{22}^{-1} \quad (3)$$

$$\boldsymbol{U}_{13} = \boldsymbol{L}_{11}^{-1} \boldsymbol{F}_1, \boldsymbol{U}_{23} = \boldsymbol{L}_{22}^{-1} \boldsymbol{F}_2 \quad (4)$$

$$\boldsymbol{L}_{33} \boldsymbol{U}_{33} = \boldsymbol{S} - \boldsymbol{L}_{31} \boldsymbol{U}_{13} - \boldsymbol{L}_{32} \boldsymbol{U}_{23} \quad (5)$$

- Factorize D_1, D_2 recursively in the same manner $\Longrightarrow$ nested dissection
- Opportunities for parallelization: (3), (4) and factorization of D_1, D_2
- Acceptable cost of solving (5) on a single process

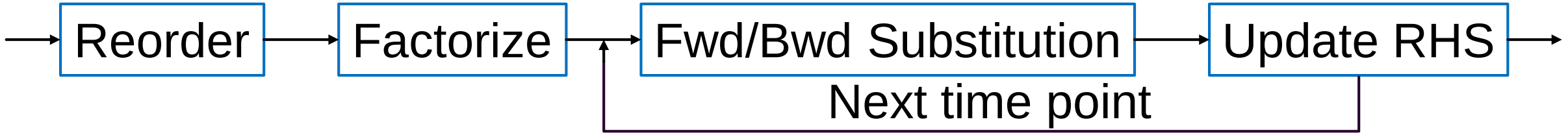
Overall Idea – Forward/Backward Substitution

- Forward substitution

$$\begin{pmatrix} L_{11} & & \\ & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ g \end{pmatrix} \Longrightarrow \boxed{L_{11}x_1 = f_1}, \boxed{L_{22}x_2 = f_2}, \boxed{L_{33}y = g - L_{31}x_1 - L_{32}x_2}$$

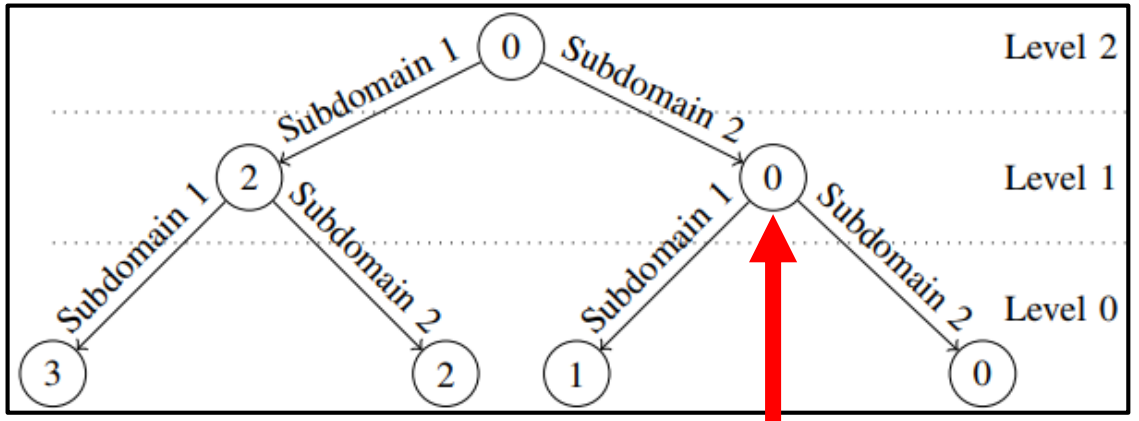
Parallel

  - Parallel solving x_1, x_2 in the same manner recursively
  - Acceptable cost of solving y on a single process
- Backward substitution is similarly done from the top down

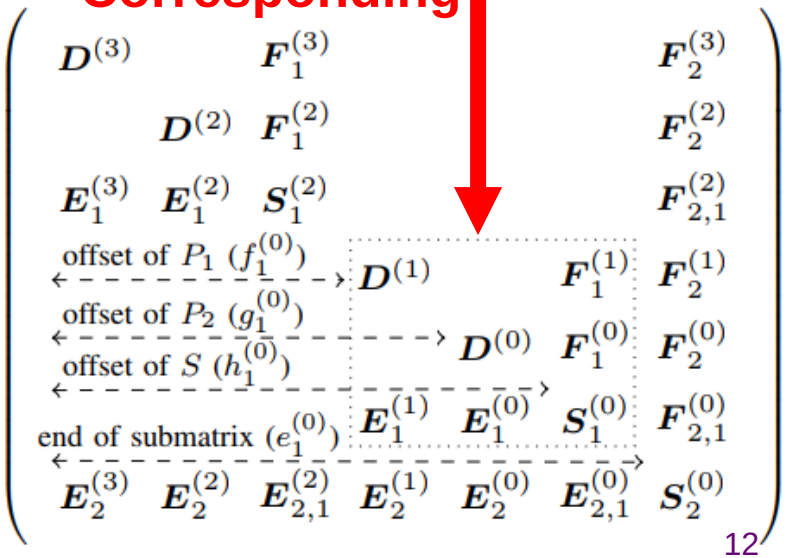


Parallel Reordering into Nested Dissection Form

- Notations
  - l_m: max nesting level 2^{l_m}: number of processes
  - Superscript (r): data resides in process r
  - Subscript l: data describes information of level l
- Task assignment for process r in level l
  - Level l_m: partition the full matrix if $r = 0$
  - Level 0: compute fill-in reducing reordering for $D^{(r)}$
  - Others: partition the subdomain from level $l - 1$ if r divides 2^l
- Record additional information for subsequent solving
  - f, g, h: offset of subdomain 1, subdomain 2, separator in A
  - e: end of submatrix



Corresponding



Parallel LU Factorization

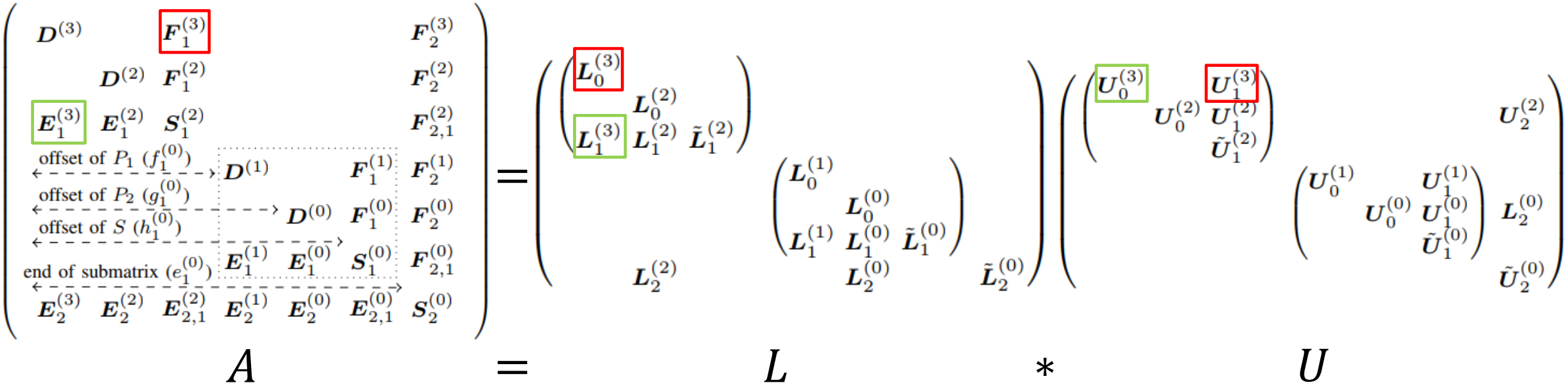
- Level 0
  - $D^{(r)} = L_0^{(r)} U_0^{(r)} \Rightarrow$ Each process r factorizes $D^{(r)}$ to get $L_0^{(r)}, U_0^{(r)}$

$$
\begin{pmatrix}
\boldsymbol{D}^{(3)} & & \boldsymbol{F}_1^{(3)} & & & \boldsymbol{F}_2^{(3)} \\
 & \boldsymbol{D}^{(2)} & \boldsymbol{F}_1^{(2)} & & & \boldsymbol{F}_2^{(2)} \\
\boldsymbol{E}_1^{(3)} & \boldsymbol{E}_1^{(2)} & \boldsymbol{S}_1^{(2)} & & & \boldsymbol{F}_{2,1}^{(2)} \\
\text{offset of } P_1\ (f_1^{(0)}) & & & \boldsymbol{D}^{(1)} & & \boldsymbol{F}_1^{(1)} & \boldsymbol{F}_2^{(1)} \\
\text{offset of } P_2\ (g_1^{(0)}) & & & & \boldsymbol{D}^{(0)} & \boldsymbol{F}_1^{(0)} & \boldsymbol{F}_2^{(0)} \\
\text{offset of } S\ (h_1^{(0)}) & & & \boldsymbol{E}_1^{(1)} & \boldsymbol{E}_1^{(0)} & \boldsymbol{S}_1^{(0)} & \boldsymbol{F}_{2,1}^{(0)} \\
\text{end of submatrix } (e_1^{(0)}) & & & & & & \\
\boldsymbol{E}_2^{(3)} & \boldsymbol{E}_2^{(2)} & \boldsymbol{E}_{2,1}^{(2)} & \boldsymbol{E}_2^{(1)} & \boldsymbol{E}_2^{(0)} & \boldsymbol{E}_{2,1}^{(0)} & \boldsymbol{S}_2^{(0)}
\end{pmatrix}
=
\begin{pmatrix}
\begin{pmatrix} \boldsymbol{L}_0^{(3)} & & \\ & \boldsymbol{L}_0^{(2)} & \\ \boldsymbol{L}_1^{(3)} & \boldsymbol{L}_1^{(2)} & \tilde{\boldsymbol{L}}_1^{(2)} \end{pmatrix} & & \\
 & \begin{pmatrix} \boldsymbol{L}_0^{(1)} & & \\ & \boldsymbol{L}_0^{(0)} & \\ \boldsymbol{L}_1^{(1)} & \boldsymbol{L}_1^{(0)} & \tilde{\boldsymbol{L}}_1^{(0)} \end{pmatrix} & \\
 & \boldsymbol{L}_2^{(2)} \quad \boldsymbol{L}_2^{(0)} & \tilde{\boldsymbol{L}}_2^{(0)}
\end{pmatrix}
\begin{pmatrix}
\begin{pmatrix} \boldsymbol{U}_0^{(3)} & & \boldsymbol{U}_1^{(3)} \\ & \boldsymbol{U}_0^{(2)} & \boldsymbol{U}_1^{(2)} \\ & & \tilde{\boldsymbol{U}}_1^{(2)} \end{pmatrix} & & \boldsymbol{U}_2^{(2)} \\
 & \begin{pmatrix} \boldsymbol{U}_0^{(1)} & & \boldsymbol{U}_1^{(1)} \\ & \boldsymbol{U}_0^{(0)} & \boldsymbol{U}_1^{(0)} \\ & & \tilde{\boldsymbol{U}}_1^{(0)} \end{pmatrix} & \boldsymbol{L}_2^{(0)} \\
 & & \tilde{\boldsymbol{U}}_2^{(0)}
\end{pmatrix}
$$

$$A = L * U$$

Parallel LU Factorization

- Level 1
  - $E_1^{(r)} = L_1^{(r)} U_0^{(r)}, F_1^{(r)} = L_0^{(r)} U_1^{(r)} \Rightarrow$ Each process r solves triangular equations to get $L_1^{(r)}, U_1^{(r)}$



Parallel LU Factorization

- Level 1
  - $E_1^{(r)} = L_1^{(r)} U_0^{(r)}, F_1^{(r)} = L_0^{(r)} U_1^{(r)} \Rightarrow$ Each process r solves triangular equations to get $L_1^{(r)}, U_1^{(r)}$
  - $S_1^{(r)} = L_1^{(r+1)} U_1^{(r+1)} + L_1^{(r)} U_1^{(r)} + \tilde{L}_1^{(r)} \tilde{U}_1^{(r)} \Rightarrow$ Even-numbered process r factorizes the Schur complement to get $\tilde{L}_1^{(r)}, \tilde{U}_1^{(r)}$

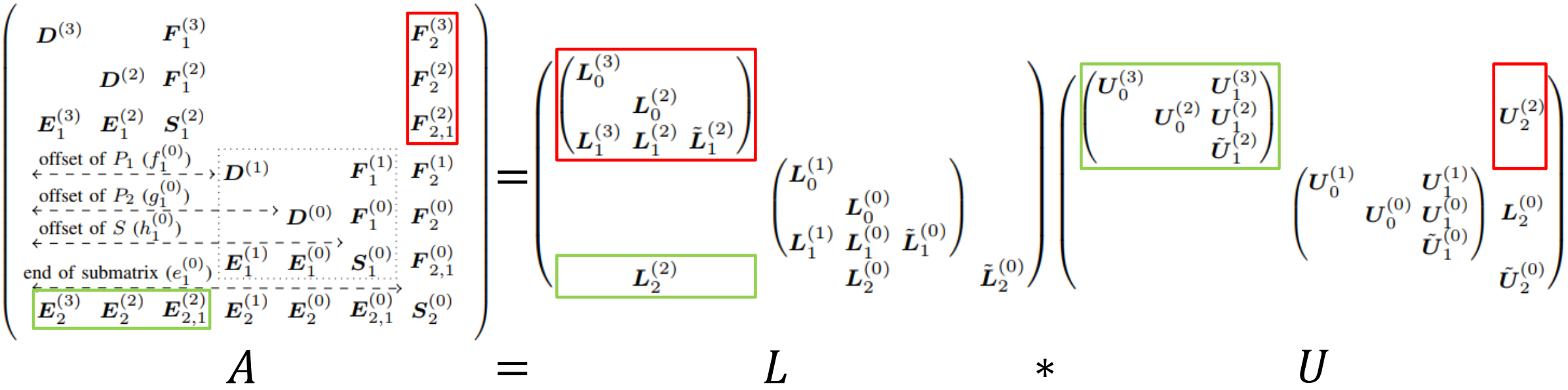
Matrices sent from $r+1$ to r

$$
\begin{pmatrix}
D^{(3)} & & F_1^{(3)} & & & & F_2^{(3)} \\
 & D^{(2)} & F_1^{(2)} & & & & F_2^{(2)} \\
E_1^{(3)} & E_1^{(2)} & S_1^{(2)} & & & & F_{2,1}^{(2)} \\
\text{offset of } P_1\ (f_1^{(0)}) & & & D^{(1)} & & F_1^{(1)} & F_2^{(1)} \\
\text{offset of } P_2\ (g_1^{(0)}) & & & & D^{(0)} & F_1^{(0)} & F_2^{(0)} \\
\text{offset of } S\ (h_1^{(0)}) & & & E_1^{(1)} & E_1^{(0)} & S_1^{(0)} & F_{2,1}^{(0)} \\
\text{end of submatrix } (e_1^{(0)}) & & & & & & \\
E_2^{(3)} & E_2^{(2)} & E_{2,1}^{(2)} & E_2^{(1)} & E_2^{(0)} & E_{2,1}^{(0)} & S_2^{(0)}
\end{pmatrix}
=
\begin{pmatrix}
\begin{pmatrix} L_0^{(3)} & & \\ & L_0^{(2)} & \\ L_1^{(3)} & L_1^{(2)} & \tilde{L}_1^{(2)} \end{pmatrix} & & \\
 & \begin{pmatrix} L_0^{(1)} & & \\ & L_0^{(0)} & \\ L_1^{(1)} & L_1^{(0)} & \tilde{L}_1^{(0)} \end{pmatrix} & \\
 & L_2^{(2)} \quad \tilde{L}_2^{(0)} & \\
\end{pmatrix}
\begin{pmatrix}
\begin{pmatrix} U_0^{(3)} & & U_1^{(3)} \\ & U_0^{(2)} & U_1^{(2)} \\ & & \tilde{U}_1^{(2)} \end{pmatrix} & & U_2^{(2)} \\
 & \begin{pmatrix} U_0^{(1)} & & U_1^{(1)} \\ & U_0^{(0)} & U_1^{(0)} \\ & & \tilde{U}_1^{(0)} \end{pmatrix} & L_2^{(0)} \\
 & & \tilde{U}_2^{(0)}
\end{pmatrix}
$$

$$A = L * U$$

Parallel LU Factorization

- Level $l > 1$
  - $E = L_l^{(r)} U, F = L U_l^{(r)} \Rightarrow$ Each process r dividing 2^{l-1} solves triangular equations to get $L_l^{(r)}, U_l^{(r)}$



Parallel LU Factorization

- Level $l > 1$
  - $E = L_l^{(r)} U, F = L U_l^{(r)} \Rightarrow$ Each process r dividing 2^{l-1} solves triangular equations to get $L_l^{(r)}, U_l^{(r)}$
  - $S_l^{(r)} = L_l^{(r+2^{l-1})} U_l^{(r+2^{l-1})} + L_l^{(r)} U_l^{(r)} + \tilde{L}_l^{(r)} \tilde{U}_l^{(r)} \Rightarrow$ Each process r dividing 2^l factorizes the Schur complement to get $\tilde{L}_l^{(r)}, \tilde{U}_l^{(r)}$

Matrices sent from $r + 2^{l-1}$ to r

$$
\begin{pmatrix}
\boldsymbol{D}^{(3)} & & \boldsymbol{F}_1^{(3)} & & & & \boldsymbol{F}_2^{(3)} \\
& \boldsymbol{D}^{(2)} & \boldsymbol{F}_1^{(2)} & & & & \boldsymbol{F}_2^{(2)} \\
\boldsymbol{E}_1^{(3)} & \boldsymbol{E}_1^{(2)} & \boldsymbol{S}_1^{(2)} & & & & \boldsymbol{F}_{2,1}^{(2)} \\
\text{offset of } P_1\ (f_1^{(0)}) & & & \boldsymbol{D}^{(1)} & & \boldsymbol{F}_1^{(1)} & \boldsymbol{F}_2^{(1)} \\
\text{offset of } P_2\ (g_1^{(0)}) & & & & \boldsymbol{D}^{(0)} & \boldsymbol{F}_1^{(0)} & \boldsymbol{F}_2^{(0)} \\
\text{offset of } S\ (h_1^{(0)}) & & & \boldsymbol{E}_1^{(1)} & \boldsymbol{E}_1^{(0)} & \boldsymbol{S}_1^{(0)} & \boldsymbol{F}_{2,1}^{(0)} \\
\text{end of submatrix } (e_1^{(0)}) & & & & & & \\
\boldsymbol{E}_2^{(3)} & \boldsymbol{E}_2^{(2)} & \boldsymbol{E}_{2,1}^{(2)} & \boldsymbol{E}_2^{(1)} & \boldsymbol{E}_2^{(0)} & \boldsymbol{E}_{2,1}^{(0)} & \boldsymbol{S}_2^{(0)}
\end{pmatrix}
=
\begin{pmatrix}
\begin{pmatrix} \boldsymbol{L}_0^{(3)} & & \\ & \boldsymbol{L}_0^{(2)} & \\ \boldsymbol{L}_1^{(3)} & \boldsymbol{L}_1^{(2)} & \tilde{\boldsymbol{L}}_1^{(2)} \end{pmatrix} & & \\
& \begin{pmatrix} \boldsymbol{L}_0^{(1)} & & \\ & \boldsymbol{L}_0^{(0)} & \\ \boldsymbol{L}_1^{(1)} & \boldsymbol{L}_1^{(0)} & \tilde{\boldsymbol{L}}_1^{(0)} \end{pmatrix} & \\
\boldsymbol{L}_2^{(2)} & \boldsymbol{L}_2^{(0)} & \tilde{\boldsymbol{L}}_2^{(0)}
\end{pmatrix}
\begin{pmatrix}
\begin{pmatrix} \boldsymbol{U}_0^{(3)} & & \boldsymbol{U}_1^{(3)} \\ & \boldsymbol{U}_0^{(2)} & \boldsymbol{U}_1^{(2)} \\ & & \tilde{\boldsymbol{U}}_1^{(2)} \end{pmatrix} & & \boldsymbol{U}_2^{(2)} \\
& \begin{pmatrix} \boldsymbol{U}_0^{(1)} & & \boldsymbol{U}_1^{(1)} \\ & \boldsymbol{U}_0^{(0)} & \boldsymbol{U}_1^{(0)} \\ & & \tilde{\boldsymbol{U}}_1^{(0)} \end{pmatrix} & \boldsymbol{L}_2^{(0)} \\
& & \tilde{\boldsymbol{U}}_2^{(0)}
\end{pmatrix}
$$

$$A = L * U$$

Parallel Forward/Backward Substitution

- Level 0
  - $L_0^{(r)} y_0^{(r)} = b_0^{(r)} \Rightarrow$ Each process r solves triangular equations to get $y_0^{(r)}$

$$\begin{pmatrix} \begin{pmatrix} \boldsymbol{L_0^{(3)}} & & \\ & \boldsymbol{L_0^{(2)}} & \\ \boldsymbol{L_1^{(3)}} & \boldsymbol{L_1^{(2)}} & \boldsymbol{\tilde{L}_1^{(2)}} \end{pmatrix} & & \\ & \begin{pmatrix} \boldsymbol{L_0^{(1)}} & & \\ & \boldsymbol{L_0^{(0)}} & \\ \boldsymbol{L_1^{(1)}} & \boldsymbol{L_1^{(0)}} & \boldsymbol{\tilde{L}_1^{(0)}} \end{pmatrix} & \\ & & \boldsymbol{\tilde{L}_2^{(0)}} \end{pmatrix} \begin{pmatrix} \boldsymbol{y_0^{(3)}} \\ \boldsymbol{y_0^{(2)}} \\ \boldsymbol{y_1^{(2)}} \\ \boldsymbol{y_0^{(1)}} \\ \boldsymbol{y_0^{(0)}} \\ \boldsymbol{y_1^{(0)}} \\ \boldsymbol{y_2^{(0)}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b_0^{(3)}} \\ \boldsymbol{b_0^{(2)}} \\ \boldsymbol{b_1^{(2)}} \\ \boldsymbol{b_0^{(1)}} \\ \boldsymbol{b_0^{(0)}} \\ \boldsymbol{b_1^{(0)}} \\ \boldsymbol{b_2^{(0)}} \end{pmatrix}$$

$$L \quad * \quad y \quad = \quad b$$

Parallel Forward/Backward Substitution

- Level $l > 0$
  - $L_l^{(r+2^{l-1})} y_l^{(r+2^{l-1})} + L_l^{(r)} y_l^{(r)} + \tilde{L}_l^{(r)} y_l^{(r)} = b_l^{(r)} \Rightarrow$ Each process r dividing 2^l solves triangular equations to get $y_l^{(r)}$

Vector sent from $r + 2^{l-1}$ to r

$$\begin{pmatrix} \begin{pmatrix} L_0^{(3)} & & \\ & L_0^{(2)} & \\ L_1^{(3)} & L_1^{(2)} & \tilde{L}_1^{(2)} \end{pmatrix} & & \\ & \begin{pmatrix} L_0^{(1)} & & \\ & L_0^{(0)} & \\ L_1^{(1)} & L_1^{(0)} & \tilde{L}_1^{(0)} \end{pmatrix} & \\ & & \begin{pmatrix} L_2^{(0)} & & \\ & L_2^{(2)} & \\ & & \tilde{L}_2^{(0)} \end{pmatrix} \end{pmatrix} \begin{pmatrix} y_0^{(3)} \\ y_0^{(2)} \\ y_1^{(2)} \\ y_0^{(1)} \\ y_0^{(0)} \\ y_1^{(0)} \\ y_2^{(0)} \end{pmatrix} = \begin{pmatrix} b_0^{(3)} \\ b_0^{(2)} \\ b_1^{(2)} \\ b_0^{(1)} \\ b_0^{(0)} \\ b_1^{(0)} \\ b_2^{(0)} \end{pmatrix}$$

$$L \quad * \quad y \quad = \quad b$$

Parallel Forward/Backward Substitution

- Backward substitution is similarly done from the top down
- Properties of the processes
  - Own a **continuous** segment of solution vector ⇒ **Communication friendly**
  - Read and write the **same** segment of solution vector ⇒ **RHS updating friendly**

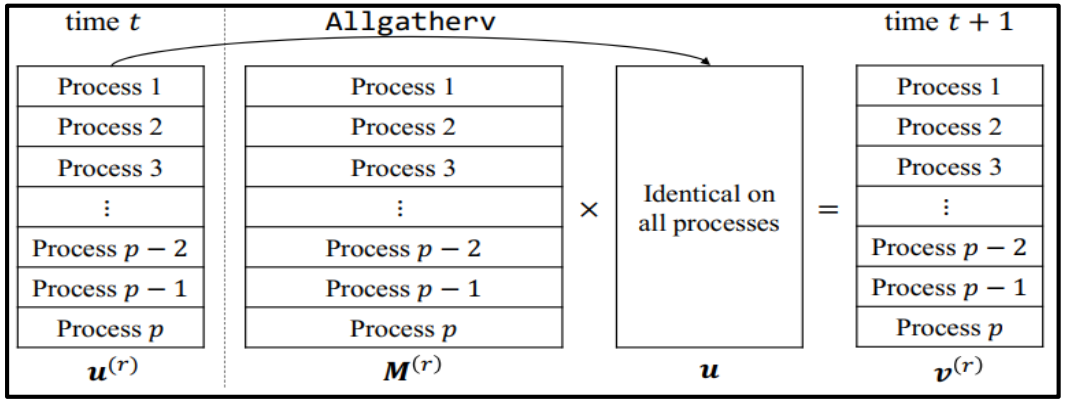
$$\begin{pmatrix} \begin{pmatrix} L_0^{(3)} & & \\ & L_0^{(2)} & \\ L_1^{(3)} & L_1^{(2)} & \tilde{L}_1^{(2)} \end{pmatrix} & & \\ & \begin{pmatrix} L_0^{(1)} & & \\ & L_0^{(0)} & \\ L_1^{(1)} & L_1^{(0)} & \tilde{L}_1^{(0)} \end{pmatrix} & \\ & L_2^{(2)} & \tilde{L}_2^{(0)} \end{pmatrix} * \begin{pmatrix} y_0^{(3)} \\ y_0^{(2)} \\ y_1^{(2)} \\ y_0^{(1)} \\ y_0^{(0)} \\ y_1^{(0)} \\ y_2^{(0)} \end{pmatrix} = \begin{pmatrix} b_0^{(3)} \\ b_0^{(2)} \\ b_1^{(2)} \\ b_0^{(1)} \\ b_0^{(0)} \\ b_1^{(0)} \\ b_2^{(0)} \end{pmatrix}$$

$$L * y = b$$

Scheme for Efficient Right-Hand Side Updating

- $b = \left(\frac{C}{2} - \frac{G}{2}\right) x(t) + B\frac{u(t+h)+u(t)}{2} \triangleq Mx + Bu$
- Fwd/bwd substitution **read and write the same segment** of solution vector
- Optimization strategy
  - No need to store the full M, B $\Longrightarrow$ **Reduce memory overhead**
  - Smaller SpGEMV and only 1 `Allgatherv` $\Longrightarrow$ **Increase parallelism**
  - Allocate shared memory for solution vectors on the same computing node

$\Longrightarrow$ **Reduce communication overhead**



Outline

- Background
- LU Factorization & Forward/Backward Substitution Based on Nested Dissection
- Experimental Results
- Conclusion

Experimental Configurations

- Environment
  - 4-node Huawei Cloud cluster with Ethernet connectivity
  - Each node with a 32-core Intel Xeon Platinum 8378A CPU and 256GB RAM
- Test data: ibmpg3-6t, thupg1,3,5,8,10t (500 time points of 10ps each)
- Implementation: OpenMPI + Eigen + MKL + METIS
- Methods compared to: NICSLU, edge separator based DDM
- Allocate 2 threads to each process for DDM and our method

Results on a Single Computing Node

- Notations
  - N_{th}: # of threads T_s: total time of serial solving
  - T_f, T: time of factorization, total time
  - N_s : size of Schur complement
  - $\overline{N}_s$: average size of Schur complement in level 0
  - Sp1,2: speedup compared to NICSLU, DDM
- Analysis
  - All fail to complete the simulation of thupg8,10t
  - **2.06X** speedup over NICSLU
  - **1.43X** speedup over DDM

Case	N_{th}	NICSLU	DDM			Proposed				
		T_s	T	T_f	T	N_S	T_f	T	$\overline{N}_S$	Sp1	Sp2
ibmpg3t $N=1.0E6$ $nnz=4.5E6$	4	74.4	47.8	9.07	24.8	0	8.88	25.9	0	1.84	0.96
	8		40.7	5.60	18.5	2325	6.08	16.3	561	2.49	1.13
	16		32.2	5.43	23.1	6288	4.25	13.1	377	2.45	1.76
	32		34.8	5.47	27.3	10860	3.83	10.6	261	3.30	2.58
ibmpg4t $N=1.2E6$ $nnz=5.8E6$	4	94.5	56.2	12.6	34.4	0	12.3	34.5	0	1.63	1.00
	8		43.1	7.82	24.9	3334	9.13	25.0	723	1.72	0.99
	16		37.3	6.51	27.2	6410	5.55	16.4	335	2.28	1.66
	32		41.9	8.43	39.1	13112	4.79	14.6	353	2.86	2.67
ibmpg5t $N=1.6E6$ $nnz=6.4E6$	4	75.5	50.1	11.1	32.5	540	11.43	39.2	0	1.28	0.83
	8		40.1	6.87	23.4	2469	6.87	23.6	330	1.70	0.99
	16		36.0	5.78	23.7	4670	5.78	16.7	204	2.15	1.41
	32		43.8	5.25	26.9	9388	5.25	13.9	294	3.15	1.93
ibmpg6t $N=2.4E6$ $nnz=9.7E6$	4	103	68.9	12.8	40.0	0	12.8	39.2	0	1.76	1.02
	8		53.2	7.64	27.8	1864	7.64	30.4	266	1.75	0.91
	16		48.3	5.23	23.7	4178	5.23	22.9	296	2.11	1.04
	32		61.6	4.51	22.6	7309	4.51	20.6	167	2.99	1.10
thupg1t $N=5.3E6$ $nnz=2.3E7$	4	458	276	66	176	2953	92.5	193	1401	1.43	0.92
	8		221	87.1	185	8792	66.0	142	1404	1.55	1.30
	16		187	69.4	166	15064	44.6	95.4	661	1.96	1.74
	32		200	51.6	170	25981	36.5	80.5	697	2.49	2.12
thupg3t $N=1.2E7$ $nnz=5.4E7$	4	1243	761	213	487	4646	323	610	2270	1.25	0.80
	8		587	374	611	14110	316	493	2268	1.19	1.24
	16		539	218	453	23615	168	300	994	1.80	1.51
	32		599	188	496	41340	138	252	1034	2.38	1.97
thupg5t $N=2.0E7$ $nnz=8.8E7$	4	2193	1932	533	1013	6777	626	1061	3234	1.82	0.95
	8		1203	511	915	17513	425	695	2445	1.73	1.32
	16		1058	482	891	30558	327	539	1576	1.96	1.65
	32		1211	522	1068	51244	263	445	1210	2.72	2.40
Average	-	-	-	-	-	-	-	-	-	2.06	1.43

Results on Multiple Computing Nodes

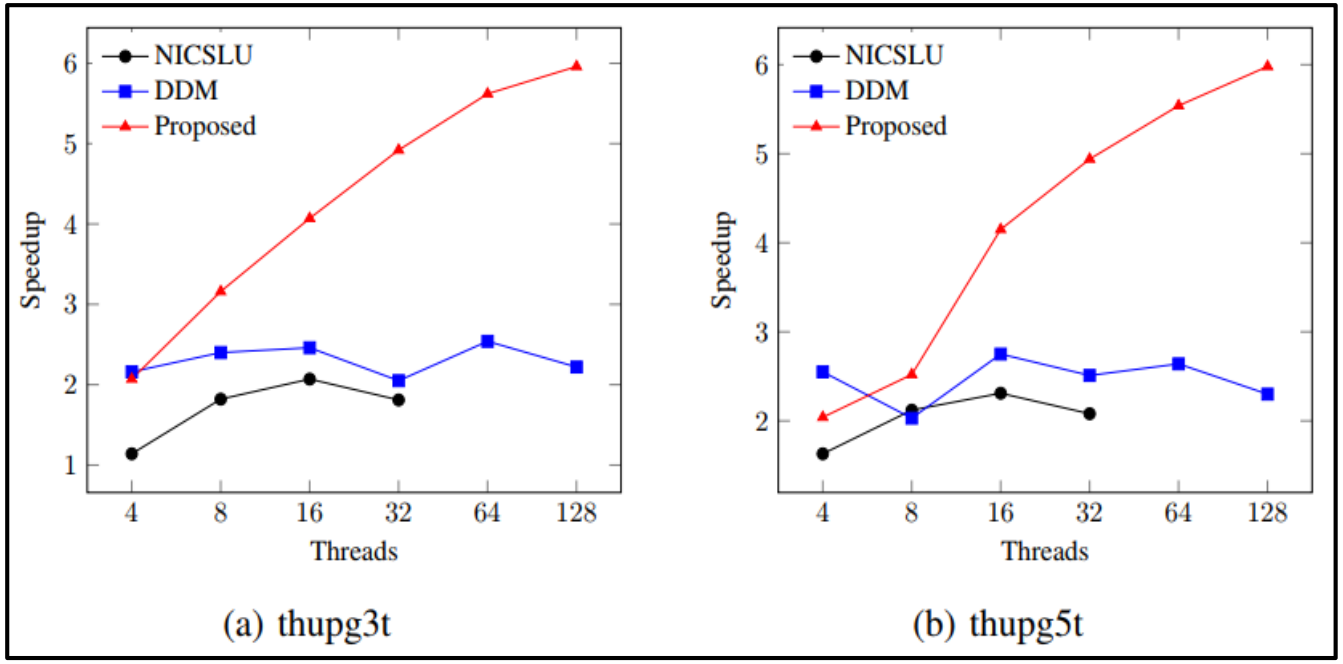
- 64, 128 processes assigned to 2, 4 computing nodes respectively
- V_f, V_{tr}: communication volume of factorization, fwd&bwd substitution
- Analysis
  - Size of Schur complement effectively controlled
  - Efficient
    - **2.84X** speedup over DDM > 1.43X on a single node
    - High parallel scalability
  - Accurate and robust
    - Relative error of solution compared to NICSLU < 1e-7
  - Low communication volume

Case	N_{th}	DDM			Proposed					
		T_f	T	N_S	T_f	T	$\overline{N}_S$	V_f	V_{tr}	Sp
ibmpg3t	64	6.67	36.0	17508	3.96	10.9	162	0.50	0.06	3.29
	128	10.2	56.1	26646	4.16	11.5	126	0.93	0.07	4.88
ibmpg4t	64	8.15	45.5	19966	4.47	13.6	196	0.65	0.07	3.35
	128	13.1	64.8	32215	5.01	14.0	172	1.19	0.09	4.63
ibmpg5t	64	5.57	30.8	14865	5.57	13.9	159	0.76	0.09	2.22
	128	10.5	51.3	24295	5.84	14.4	138	1.11	0.12	3.57
ibmpg6t	64	5.73	30.4	14441	5.73	18.2	163	0.77	0.13	1.67
	128	9.98	48.4	22794	6.15	18.9	111	1.50	0.18	2.56
thupg1t	64	40.2	175	38348	31.8	75.0	324	3.14	0.31	2.33
	128	47.8	212	58838	33.2	78.3	315	5.55	0.38	2.71
thupg3t	64	142	472	60959	116	224	516	7.78	0.68	2.10
	128	197	541	91613	108	208	489	13.5	0.88	2.60
thupg5t	64	291	863	77062	224	391	763	12.9	1.13	2.21
	128	371	987	117742	209	368	594	22.3	1.47	2.68
thupg8t $N=4.1E7$ $nnz=1.8E8$	64	859	2714	106472	806	1175	1049	27.4	2.33	2.31
	128	801	2875	142819	750	1089	874	47.1	2.97	2.64
thupg10t $N=6.3E7$ $nnz=2.7E8$	128	1239	4375	181934	1058	1695	1034	73.0	4.57	2.58
Average	-	-	-	-	-	-	-	-	-	2.84

The notations of $N_{th}, T_f, T, N_S, \overline{N}_S$ and Sp are the same as Table I. V_f denotes the communication volume of matrix reordering and factorization and V_{tr} denotes the average communication volume of forward/backward substitution and RHS updating per time point.

Parallel scalability

- X-axis: # of threads Y-axis: speedup compared to serial NICSLU
- No improvement of speedup of NICSLU and DDM
- **Increasing speedup** of our method, achieving up to **6.0X** using 4 computing nodes
- Cloud computing allows for faster simulation
- **Cost of adding computing nodes may be less than the benefit of saving time**



Conclusion

- Leverage the public cloud computing for transient PG analysis while ensuring security
- Multi-level distributed parallel LU factorization and forward/backward substitution approach based on nested dissection
- Scheme for efficient RHS updating to further exploit parallelism
- Ideal for deployment on cloud computing platforms without InfiniBand
- **2.06X** speedup compared to NICSLU on a single computing node
- **2.84X** speedup compared to DDM on multiple computing nodes
- Good parallel scalability up to **6.0X** speedup over serial simulation

清華大學
Tsinghua University

Thank you! Q & A

Jiawen Cheng 2024-01-24