Approximate Neumann Series or Exact Matrix Inversion for Massive MIMO?

Oscar Gustafsson, Erik Bertilsson, Johannes Klasson, and Carl Ingemarsson

LINKÖPING UNIVERSITY

Matrix Inversion in Massive MIMO

- N terminals, M antennas

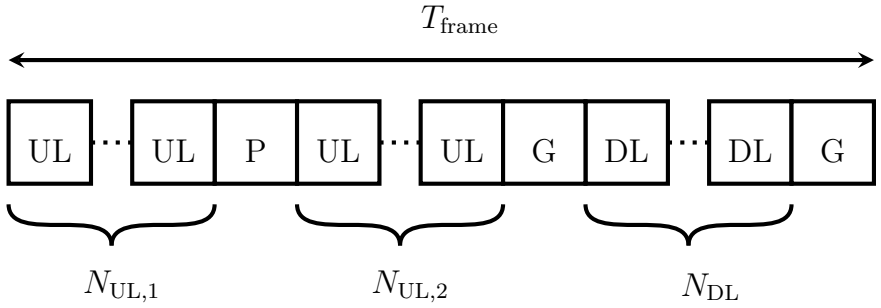
Matrix Inversion in Massive MIMO

- N terminals, M antennas
- Channel matrix, $\mathbf{H} \in \mathbb{C}^{M \times N}$
- Gram matrix, $\mathbf{X} = \mathbf{H}^H \mathbf{H} \in \mathbb{C}^{N \times N}$ to be inverted for zero forcing (or MMSE)

Matrix Inversion in Massive MIMO

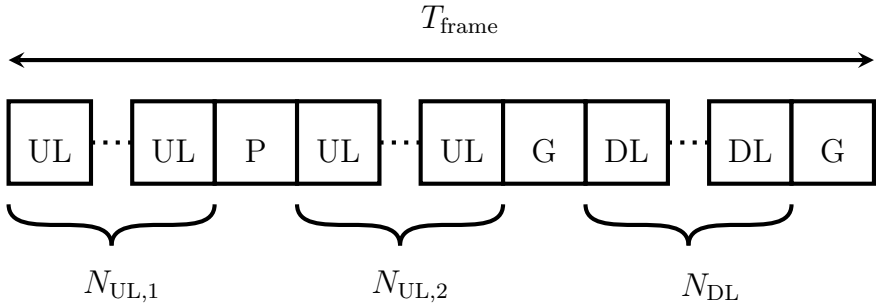
- N terminals, M antennas
- Channel matrix, $\mathbf{H} \in \mathbb{C}^{M \times N}$
- Gram matrix, $\mathbf{X} = \mathbf{H}^H \mathbf{H} \in \mathbb{C}^{N \times N}$ to be inverted for zero forcing (or MMSE)
- $\mathbf{X}$: conjugate symmetric (Hermitian) and semi-definite
- $\mathbf{X}$: with uncorrelated channels and $M \gg N$, diagonally dominant

Matrix Inversion in Massive MIMO



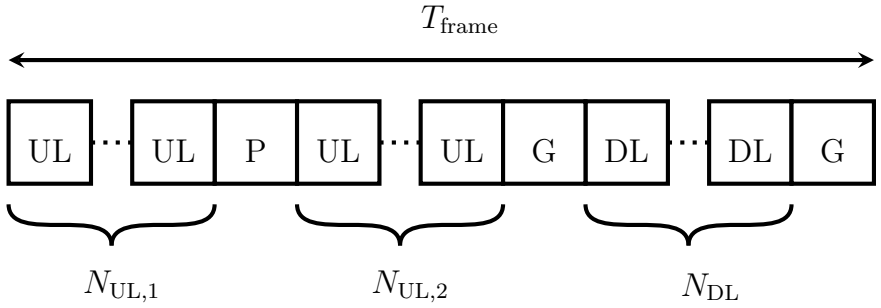
- One matrix inversion per frame

Matrix Inversion in Massive MIMO



- One matrix inversion per frame
- Computed between reception of pilot and transmission of first downlink data

Matrix Inversion in Massive MIMO



- One matrix inversion per frame
- Computed between reception of pilot and transmission of first downlink data
- Latency, not throughput

Algorithms for Matrix Inversion

- Exact algorithms
  - Numerical issues, especially in fixed-point, for close to singular (sub-)matrices
  - Division and/or square-roots
  - Cubic complexity

Algorithms for Matrix Inversion

- Exact algorithms
  - Numerical issues, especially in fixed-point, for close to singular (sub-)matrices
  - Division and/or square-roots
  - Cubic complexity
- LDL^T-decomposition
  - Lowest operation count
  - Reasonable fixed-point properties
  - No square-roots

Algorithms for Matrix Inversion

- Neumann series expansion
- Precondition matrix $\mathbf{A} \approx \mathbf{X}^{-1}$

$$\hat{\mathbf{X}}_K^{-1} = \left(\sum_{n=1}^{K} (\mathbf{I} - \mathbf{A}\mathbf{X})^{n-1} \right) \mathbf{A}, \qquad (1)$$

Algorithms for Matrix Inversion

- Neumann series expansion
- Precondition matrix $\mathbf{A} \approx \mathbf{X}^{-1}$

$$\hat{\mathbf{X}}_K^{-1} = \left(\sum_{n=1}^{K} (\mathbf{I} - \mathbf{A}\mathbf{X})^{n-1} \right) \mathbf{A}, \qquad (1)$$

- “High parallelism”
- “Low complexity”
- “No division”
- “Numerically stable”

Algorithms for Matrix Inversion

Diagonal precondition matrix

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{N,N} \end{bmatrix}$$

Algorithms for Matrix Inversion

Diagonal precondition matrix

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{N,N} \end{bmatrix}$$

$$a_{i,i} = 1/x_{i,i}$$

$$\mathbf{I} - \mathbf{AX} = \begin{bmatrix} 0 & y_{1,2} & \cdots & y_{1,N} \\ y_{2,1} & 0 & \cdots & y_{2,N} \\ \vdots & \ddots & \vdots & \vdots \\ y_{N,1} & y_{N,2} & \cdots & 0 \end{bmatrix}$$

Algorithms for Matrix Inversion

Tri-diagonal precondition matrix

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & 0 \\ 0 & a_{3,2} & a_{3,3} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & \cdots & a_{N,N} \end{bmatrix}$$

Algorithms for Matrix Inversion

Tri-diagonal precondition matrix

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & 0 \\ 0 & a_{3,2} & a_{3,3} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & \cdots & a_{N,N} \end{bmatrix}$$

Sequential computation of $\mathbf{A}$
Generic $\mathbf{I} - \mathbf{AX}$

Algorithms for Matrix Inversion

Diagonal + column precondition matrix

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ a_{N,1} & 0 & \cdots & a_{N,N} \end{bmatrix}$$

Algorithms for Matrix Inversion

Diagonal + column precondition matrix

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ a_{N,1} & 0 & \cdots & a_{N,N} \end{bmatrix}$$

$$\mathbf{I} - \mathbf{AX} = \begin{bmatrix} 0 & y_{1,2} & \cdots & y_{1,N} \\ 0 & y_{2,2bb} & \cdots & y_{2,N} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & y_{N,2} & \cdots & y_{N,N} \end{bmatrix}$$

Computational Complexity

- The latency (time to obtain the result) of an algorithm depends on two aspects:

Computational Complexity

- The latency (time to obtain the result) of an algorithm depends on two aspects:
  - Total number of operations $\rightarrow$ latency scales with number of processing elements (PEs)

Computational Complexity

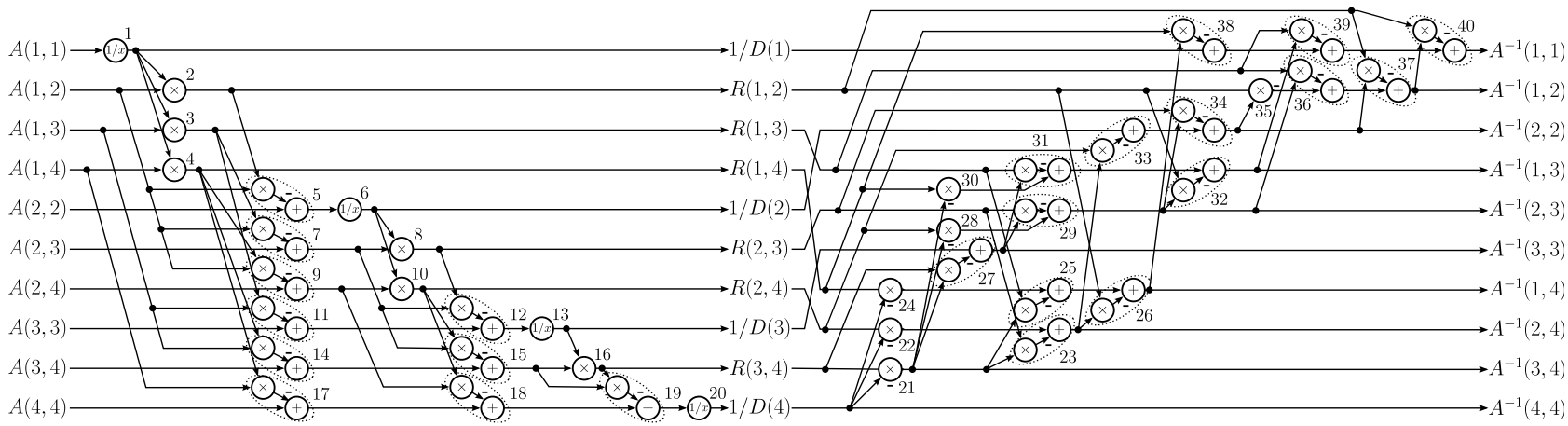
- The latency (time to obtain the result) of an algorithm depends on two aspects:
  - Total number of operations $\rightarrow$ latency scales with number of processing elements (PEs)
  - Number of sequential operations $\rightarrow$ latency does not scale with number of PEs

Computational Complexity

- The latency (time to obtain the result) of an algorithm depends on two aspects:
  - Total number of operations $\rightarrow$ latency scales with number of processing elements (PEs)
  - Number of sequential operations $\rightarrow$ latency does not scale with number of PEs
- Pipelining of the PEs
  - Increases clock frequency
  - Increases latency

Computational Complexity Example

4×4 exact matrix inversion based on LDL$^\mathsf{T}$



How Many Cycles?

- Assume multiply-and-add (MAD) operations
- Reciprocals performed using Newton-Raphson $\rightarrow$ a number of sequential MAD operations

How Many Cycles?

- Assume multiply-and-add (MAD) operations
- Reciprocals performed using Newton-Raphson $\rightarrow$ a number of sequential MAD operations
- Sum-of-products computed using sequential MADs

How Many Cycles?

- Assume multiply-and-add (MAD) operations
- Reciprocals performed using Newton-Raphson $\rightarrow$ a number of sequential MAD operations
- Sum-of-products computed using sequential MADs
- O operations, each with P pipeline stages implemented on Q processing elements (PEs) require

$$C_{\text{alg}} \geq \max\left\{\left\lceil\frac{O}{Q}\right\rceil + P - 1, PC_{\text{latency}}\right\} \text{ cycles.} \quad (2)$$

Algorithm Comparison – Complexity

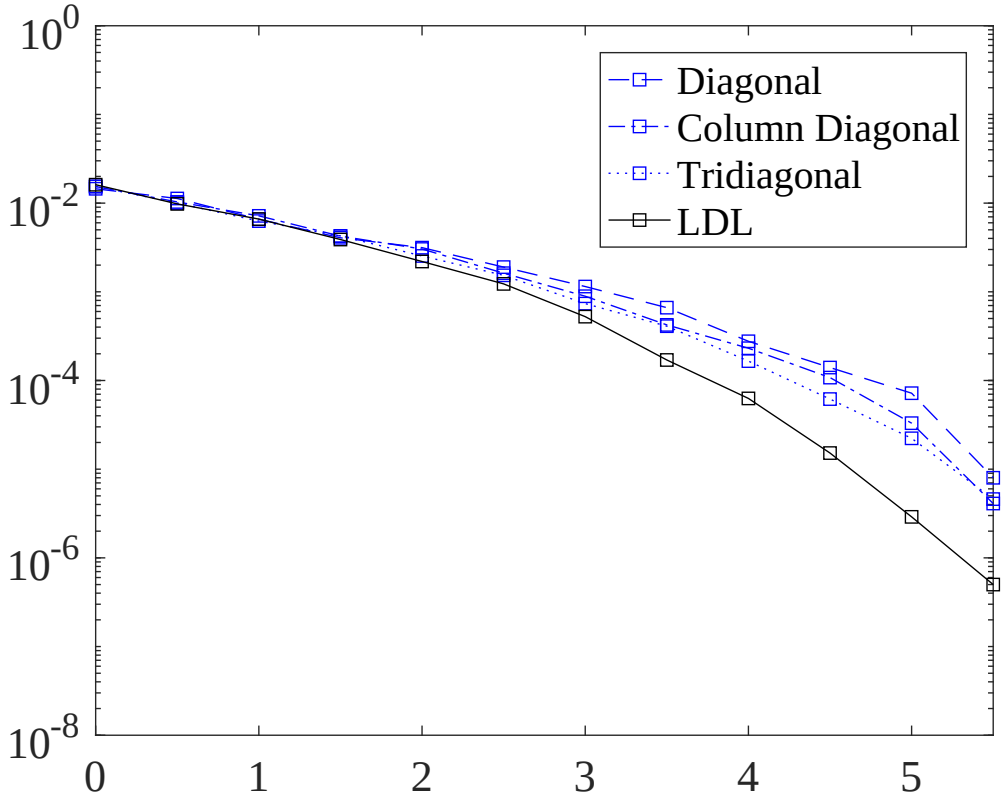
Method	MADs	Reciprocals
Exact method		
LDL$^\intercal$+EQU	$\frac{1}{2}N^3+\frac{1}{2}N^2-N$	N
Neumann series		
Diagonal, $K=2$	N^2-N	N
$K=3$	$\frac{1}{2}N^3+N^2-\frac{1}{2}N$	N
Tri-diagonals, $K=2$	$3N^2+7N-10$	$2N-1$
$K=3$	$\frac{1}{2}N^3+6N^2+\frac{1}{2}N-2$	$2N-1$
Diag. + column, $K=2$	$\frac{3}{2}N^2+\frac{5}{2}N-4$	N
$K=3$	$\frac{1}{2}N^3+\frac{5}{2}N^2-2N-1$	N

Algorithm Comparison – Latency

Method	MADs	Reciprocals
Exact method		
LDL$^\intercal$+EQU	$4N-4$	N
Neumann series		
Diagonal, $K=2$	2	1
$K=3$	$N+1$	1
Tri-diagonals, $K=2$	$2N+5$	N
$K=3$	$3N+5$	N
Diag. + column, $K=2$	$N+2$	1
$K=3$	$2N+1$	1

Results

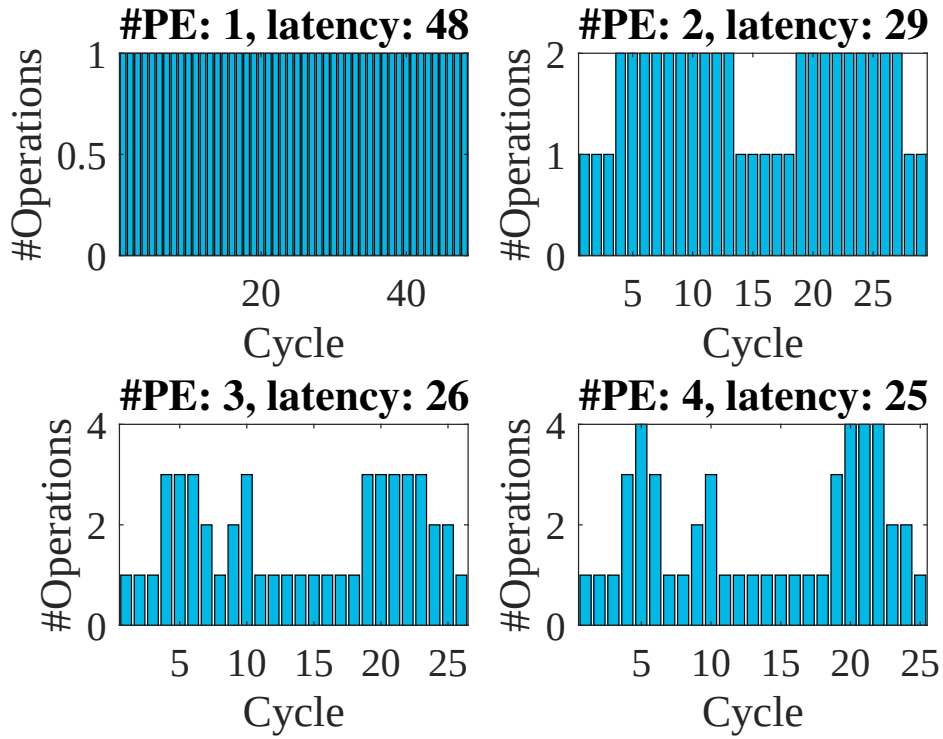
Bit-error rate for the four approaches, $N = 20$, $M = 120$



Results

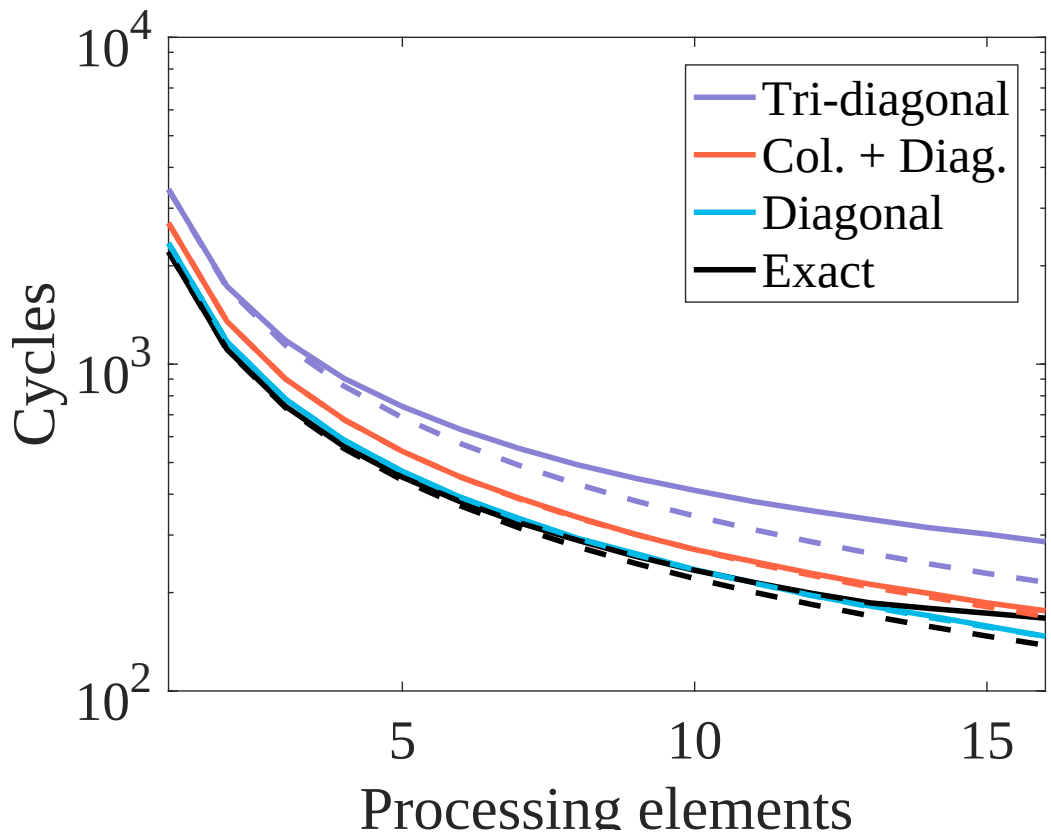
Reciprocal $\Rightarrow$ Three sequential MAD operations

4×4-matrix



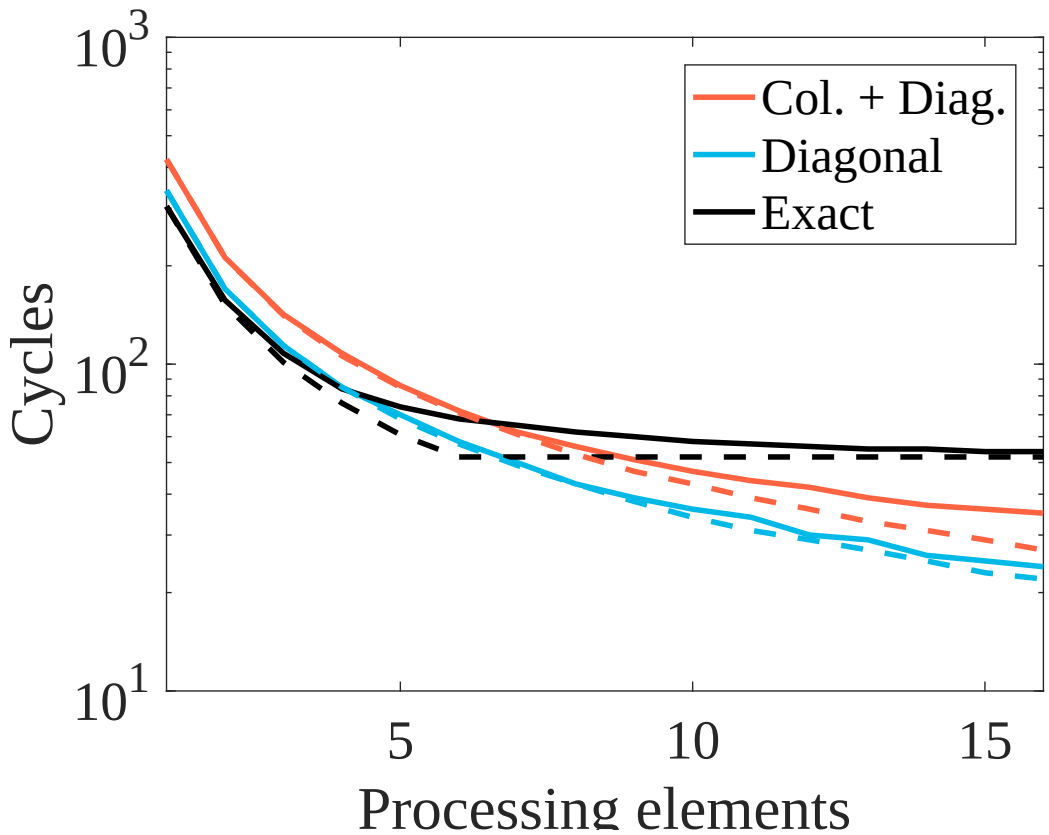
Results - 16×16

Solid: actual result, dashed: from equation



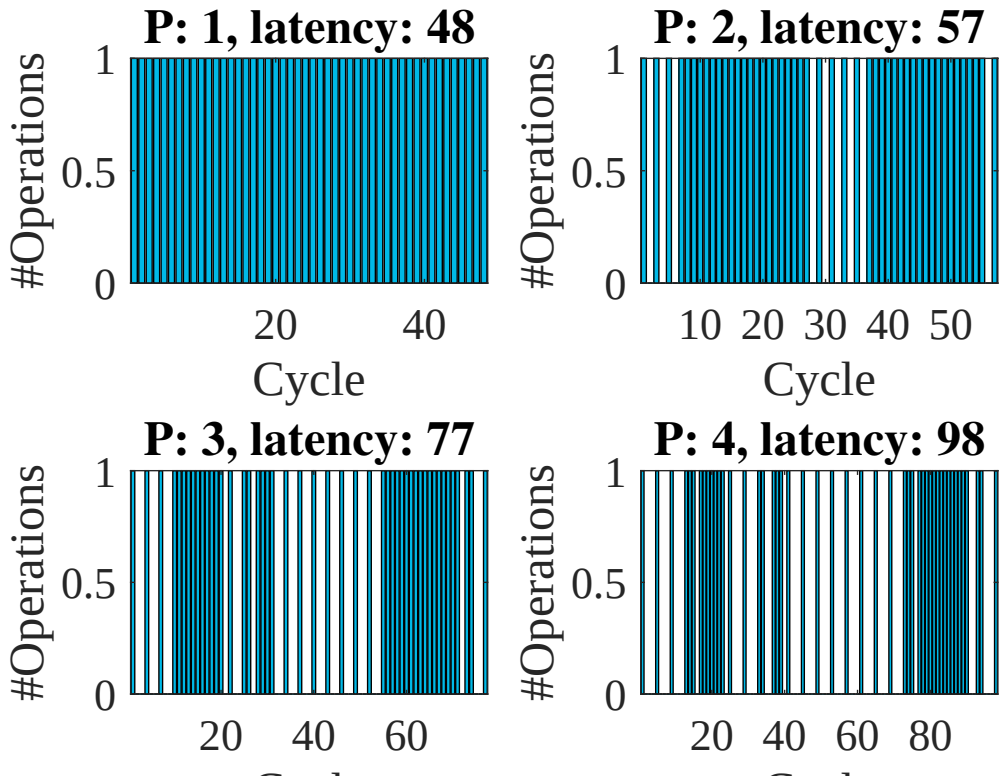
Results - 8×8

Solid: actual result, dashed: from equation



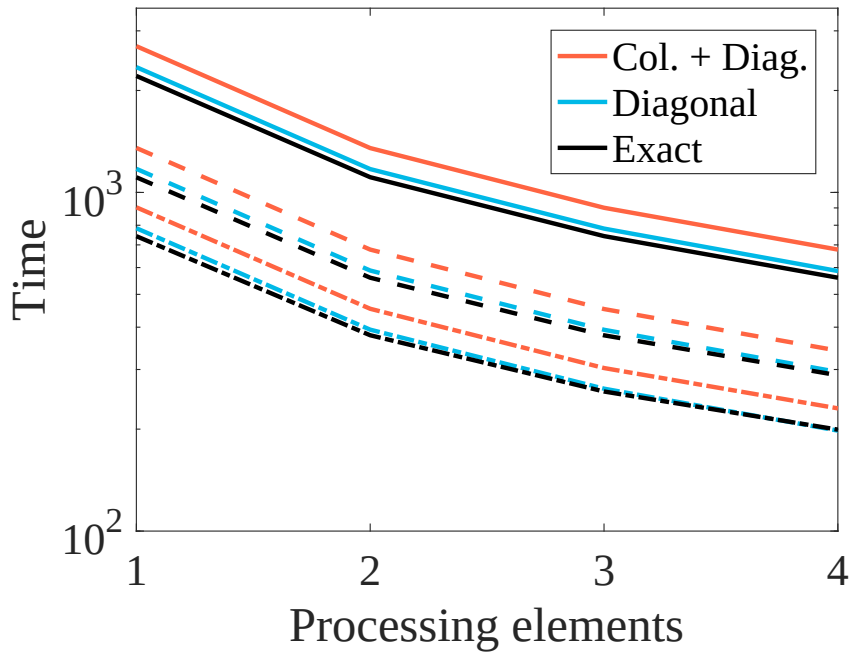
Results

With $P = 1, 2, 3, 4$ levels of pipelining
4×4-matrix



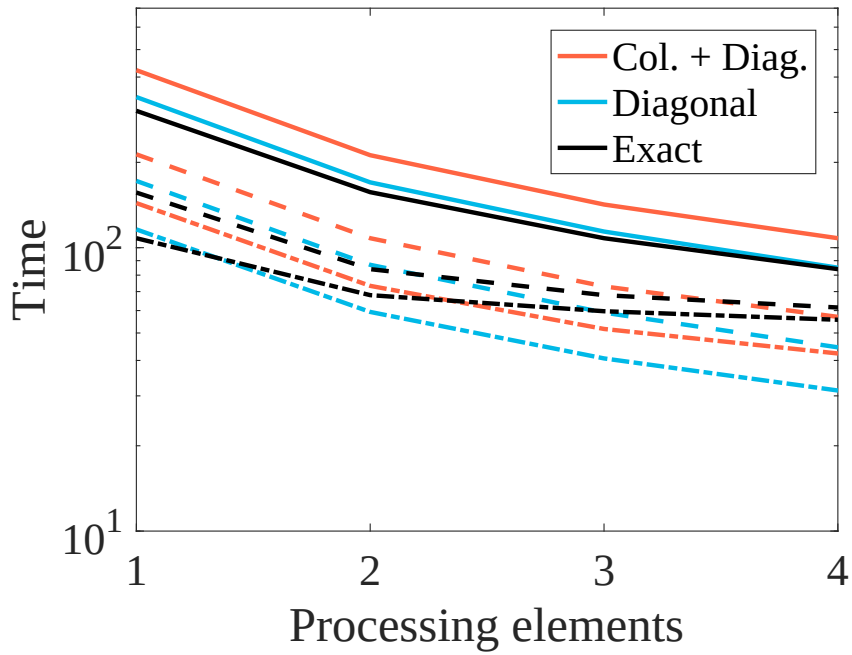
Results - 16×16

Time in single cycle latency operations, assuming pipelining increases speed linearly
Solid: $P = 1$, dashed: $P = 2$, dash-dotted: $P = 3$



Results - 8×8

Time in single cycle latency operations, assuming pipelining increases speed linearly
Solid: $P = 1$, dashed: $P = 2$, dash-dotted: $P = 3$



Design Example

- Assume a latency requirement of 0.05 ms (10% of an LTE-like frame with 2 UL and 2 DL symbols)

Design Example

- Assume a latency requirement of 0.05 ms (10% of an LTE-like frame with 2 UL and 2 DL symbols)
- For $N = 8$ and one PE, 304 cycles are required for the exact algorithm

Design Example

- Assume a latency requirement of 0.05 ms (10% of an LTE-like frame with 2 UL and 2 DL symbols)
- For $N = 8$ and one PE, 304 cycles are required for the exact algorithm
- One PE operating at $f_{\text{clk}} = 6.08$ MHz

Design Example

- Assume a latency requirement of 0.05 ms (10% of an LTE-like frame with 2 UL and 2 DL symbols)
- For $N = 8$ and one PE, 304 cycles are required for the exact algorithm
- One PE operating at $f_{\text{clk}} = 6.08$ MHz
- $N = 30 \Rightarrow f_{\text{clk}} \approx 280$ MHz

Design Example

- Assume a latency requirement of 0.05 ms (10% of an LTE-like frame with 2 UL and 2 DL symbols)
- For $N = 8$ and one PE, 304 cycles are required for the exact algorithm
- One PE operating at $f_{\text{clk}} = 6.08$ MHz
- $N = 30 \Rightarrow f_{\text{clk}} \approx 280$ MHz
- 2 kInv/s, idle 90% of the time

Is Neumann useful at all?

- If less than three terms are used, the complexity may be lower

Is Neumann useful at all?

- If less than three terms are used, the complexity may be lower
- Only compute parts of the third iteration

Is Neumann useful at all?

- If less than three terms are used, the complexity may be lower
- Only compute parts of the third iteration
- Allow increasing the number of terminals further
- But numerically most efficient when the ratio between number of antennas and terminals is high

Is Neumann useful at all?

- If less than three terms are used, the complexity may be lower
- Only compute parts of the third iteration
- Allow increasing the number of terminals further
- But numerically most efficient when the ratio between number of antennas and terminals is high
- May give a better result with singular or close to singular matrices (not correct result maybe not as bad as an exact algorithm)
- (Really) large matrices

Conclusions

- Latency, not throughput

Conclusions

- Latency, not throughput
- Complexity for Neumann series with $K = 3$ higher than best exact algorithm

Conclusions

- Latency, not throughput
- Complexity for Neumann series with $K = 3$ higher than best exact algorithm
- Few terms for Neumann when diagonally dominant

Conclusions

- Latency, not throughput
- Complexity for Neumann series with $K = 3$ higher than best exact algorithm
- Few terms for Neumann when diagonally dominant
  - Diagonally dominant $\Rightarrow$ well conditioned

Conclusions

- Latency, not throughput
- Complexity for Neumann series with $K = 3$ higher than best exact algorithm
- Few terms for Neumann when diagonally dominant
  - Diagonally dominant $\Rightarrow$ well conditioned $\Rightarrow$ exact algorithm behaves well
  - Few terminals $\Rightarrow$ more diagonally dominant

Conclusions

- Latency, not throughput
- Complexity for Neumann series with $K = 3$ higher than best exact algorithm
- Few terms for Neumann when diagonally dominant
  - Diagonally dominant $\Rightarrow$ well conditioned $\Rightarrow$ exact algorithm behaves well
  - Few terminals $\Rightarrow$ more diagonally dominant $\Rightarrow$ fewer Neumann terms (but also less complexity for exact algorithm)

Conclusions

- Latency, not throughput
- Complexity for Neumann series with $K = 3$ higher than best exact algorithm
- Few terms for Neumann when diagonally dominant
  - Diagonally dominant $\Rightarrow$ well conditioned $\Rightarrow$ exact algorithm behaves well
  - Few terminals $\Rightarrow$ more diagonally dominant $\Rightarrow$ fewer Neumann terms (but also less complexity for exact algorithm)
- With few PEs compared to matrix size, the limited parallelism of the exact algorithm is no problem

Conclusions

- Latency, not throughput
- Complexity for Neumann series with $K = 3$ higher than best exact algorithm
- Few terms for Neumann when diagonally dominant
  - Diagonally dominant $\Rightarrow$ well conditioned $\Rightarrow$ exact algorithm behaves well
  - Few terminals $\Rightarrow$ more diagonally dominant $\Rightarrow$ fewer Neumann terms (but also less complexity for exact algorithm)
- With few PEs compared to matrix size, the limited parallelism of the exact algorithm is no problem
- Required latency/parallelism determined by frame

Thank you!
Questions?

www.liu.se

LINKÖPING UNIVERSITY