Communication Avoiding and Overlapping for Numerical Linear Algebra

Evangelos Georganas[1], Jorge González-Domínguez[2], Edgar Solomonik[1], Yili Zheng[3], Juan Touriño[2], Katherine Yelick[1,3]

[1]Department of Electrical Engineering and Computer Sciences, UC Berkeley
[2]Department of Electronics and Systems, University of A Coruña
[3]Lawrence Berkeley National Laboratory

June 4, 2013

Table of contents

1. Introduction
    - Communication is Expensive
    - Communication avoidance vs Overlapping
    - CA and CO in Linear Algebra
2. Linear algebra algorithms
    - Matrix Multiplication
    - Cholesky factorization
    - Triangular Solve (TRSM)
3. Performance modeling
    - Motivation
    - Methodology
4. Conclusions
    - Conclusions

Communication is Expensive

Communication has two components:

- **Bandwidth cost:** # of words moved / bandwidth
- **Latency cost:** # messages $\times$ latency

Communication is Expensive

Communication has two components:

- **Bandwidth cost:** # of words moved / bandwidth
- **Latency cost:** # messages $\times$ latency

Communication exists in **memory hierarchy** and **network**
Things are bad and **getting worse**:

$$\mathit{flop\ time} \ll 1/\mathit{bandwidth} \ll \mathit{latency}$$

Annual improvements [FOSC]			
Flop time		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

Communication is Expensive

Communication has two components:

- **Bandwidth cost:** # of words moved / bandwidth
- **Latency cost:** # messages $\times$ latency

Communication exists in **memory hierarchy** and **network**
Things are bad and **getting worse**:

$$\textit{flop time} \ll 1/\textit{bandwidth} \ll \textit{latency}$$

Annual improvements [FOSC]			
Flop time		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

Communication is also expensive in **energy**

Communication Avoidance vs Overlapping

Two techniques to minimize the impact of communication:

Communication Avoidance vs Overlapping

Two techniques to minimize the impact of communication:

1. **Communication Avoidance (CA):**

Communication Avoidance vs Overlapping

Two techniques to minimize the impact of communication:

1. **Communication Avoidance (CA):**
    - Leads to provably optimal algorithms

Communication Avoidance vs Overlapping

Two techniques to minimize the impact of communication:

1. **Communication Avoidance (CA):**
   - Leads to provably optimal algorithms
   - There might be a trade-off in bandwidth and latency

Communication Avoidance vs Overlapping

Two techniques to minimize the impact of communication:

1. **Communication Avoidance (CA):**
   - Leads to provably optimal algorithms
   - There might be a trade-off in bandwidth and latency
2. **Communication Overlapping (CO):**

Communication Avoidance vs Overlapping

Two techniques to minimize the impact of communication:

1. **Communication Avoidance (CA):**
   - Leads to provably optimal algorithms
   - There might be a trade-off in bandwidth and latency
2. **Communication Overlapping (CO):**
   - Reduces the impact of each communication event by overlapping it with computation

Communication Avoidance vs Overlapping

Two techniques to minimize the impact of communication:

1. **Communication Avoidance (CA):**
   - Leads to provably optimal algorithms
   - There might be a trade-off in bandwidth and latency
2. **Communication Overlapping (CO):**
   - Reduces the impact of each communication event by overlapping it with computation
   - Hides bandwidth and/or latency cost

Communication Avoidance vs Overlapping

Two techniques to minimize the impact of communication:

1. **Communication Avoidance (CA):**
   - Leads to provably optimal algorithms
   - There might be a trade-off in bandwidth and latency
2. **Communication Overlapping (CO):**
   - Reduces the impact of each communication event by overlapping it with computation
   - Hides bandwidth and/or latency cost
   - Most effective if communication & computation are balanced

CA and CO in Linear Algebra

- We studied these techniques in three Linear Algebra routines:
  - Matrix Multiplication (SUMMA and Cannon's algorithms)
  - Cholesky factorization
  - Triangular Solve

CA and CO in Linear Algebra

- We studied these techniques in three Linear Algebra routines:
  - Matrix Multiplication (SUMMA and Cannon's algorithms)
  - Cholesky factorization
  - Triangular Solve
- Prior algorithms but novel implementations

Optimizations / Algorithm	Overlapping	Avoidance	Overlapping & Avoidance
SUMMA	PRIOR	PRIOR	
Cannon's	PRIOR	PRIOR	
Cholesky	PRIOR		
TRSM	PRIOR		

CA and CO in Linear Algebra

- We studied these techniques in three Linear Algebra routines:
  - Matrix Multiplication (SUMMA and Cannon's algorithms)
  - Cholesky factorization
  - Triangular Solve
- Prior algorithms but novel implementations

Algorithm \ Optimizations	Overlapping	Avoidance	Overlapping & Avoidance
SUMMA	PRIOR	PRIOR	NEW
Cannon's	PRIOR NEW: One sided communication	PRIOR NEW: One sided communication	NEW
Cholesky	PRIOR	NEW	NEW
TRSM	PRIOR	NEW*	NEW*

*Uses replication but not communication optimal

CA and CO in Linear Algebra

Three major questions arise:

CA and CO in Linear Algebra

Three major questions arise:

1. Which is more important? CA or CO?

CA and CO in Linear Algebra

Three major questions arise:

1. Which is more important? CA or CO? It depends

CA and CO in Linear Algebra

Three major questions arise:

1. Which is more important? CA or CO? It depends
2. Is there a benefit from combining both techniques?

CA and CO in Linear Algebra

Three major questions arise:

1. Which is more important? CA or CO? It depends
2. Is there a benefit from combining both techniques? Yes

CA and CO in Linear Algebra

Three major questions arise:

1. Which is more important? CA or CO? It depends
2. Is there a benefit from combining both techniques? Yes
3. Can we explain the behavior of CA and CO under different situations?

CA and CO in Linear Algebra

Three major questions arise:

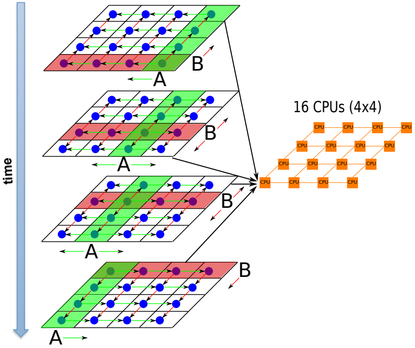
1. Which is more important? CA or CO? It depends
2. Is there a benefit from combining both techniques? Yes
3. Can we explain the behavior of CA and CO under different situations? Yes - Performance models

Table of contents

1. Introduction
   - Communication is Expensive
   - Communication avoidance vs Overlapping
   - CA and CO in Linear Algebra
2. Linear algebra algorithms
   - Matrix Multiplication
   - Cholesky factorization
   - Triangular Solve (TRSM)
3. Performance modeling
   - Motivation
   - Methodology
4. Conclusions
   - Conclusions

2D Matrix Multiplication (SUMMA)

[Van De Geijn and Watts 97]



- Outer product form of Mat Mul
- Partitions A, B and C in 2 dimensions
- Row and column broadcast on 2D grid
- Costs:
  - $O(n^3/p)$ flops
  - $O(n^2/\sqrt{p})$ words moved
  - $O(\sqrt{p})$ messages

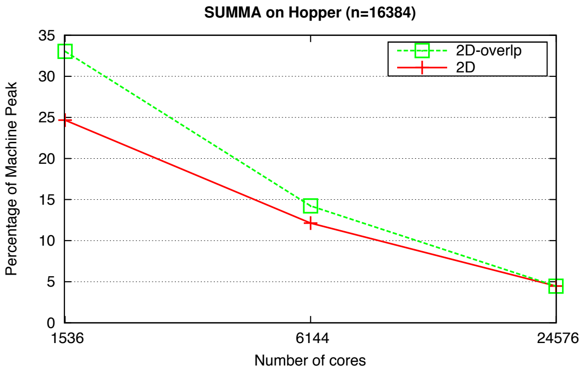
2D Matrix Multiplication with CO (SUMMA)

- We overlap the broadcasts of next iteration with the local Mat.Mul computation of current iteration
- Theoretically the execution time becomes $t_{exec} = O(\max(t_{computation}, t_{communication}))$
- If communication and computation are balanced we can achieve up to $2\times$ speedup
- **Additional communication buffers are needed**

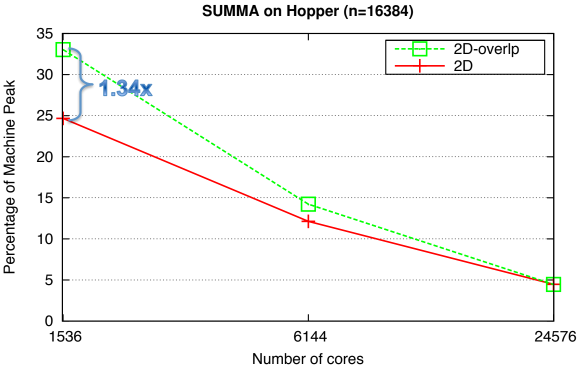
Experimental setup

- Experiments on Hopper, a Cray XE6 system (153,216 cores)
- We will focus on communication-limited problems (i.e. small problems on large machine configurations)
- Strong scaling experiments

Overlapping yields more benefits at medium scale



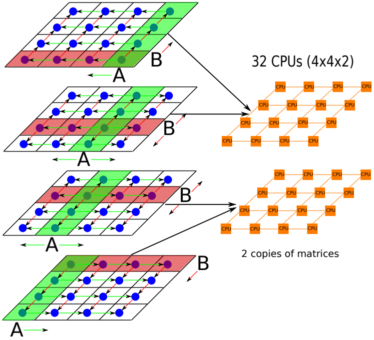
Overlapping yields more benefits at medium scale



- At medium scale where communication and computation are balanced we observe larger benefits (1.34× speedup)
- At large scale overlapping does not help

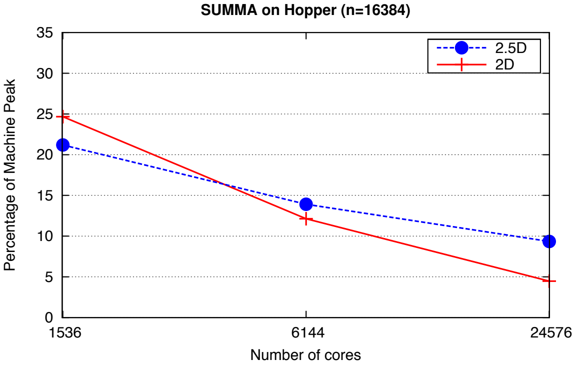
2.5D Matrix Multiplication (SUMMA)

[McColl and Tiskin 99], [Solomonik and Demmel 11]

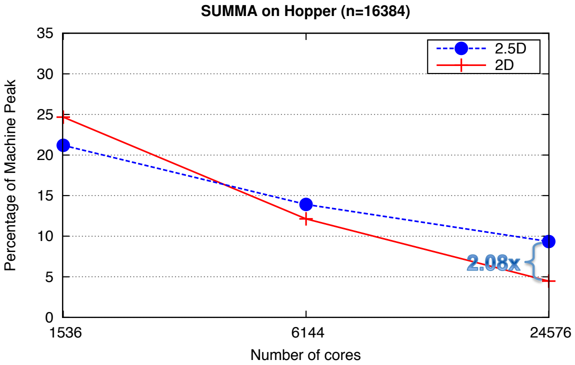


- The 2.5D algorithm uses extra memory to reduce communication
- Each one of the c layers of processors computes a different contribution to the matrix C
- Works for c copies, $c \in [1, p^{1/3}]$
- Costs:
  - $O(n^3/p)$ flops
  - $O(n^2/\sqrt{c \cdot p})$ words moved
  - $O(\sqrt{p/c^3})$ messages

Communication avoidance helps a lot at large scale



Communication avoidance helps a lot at large scale

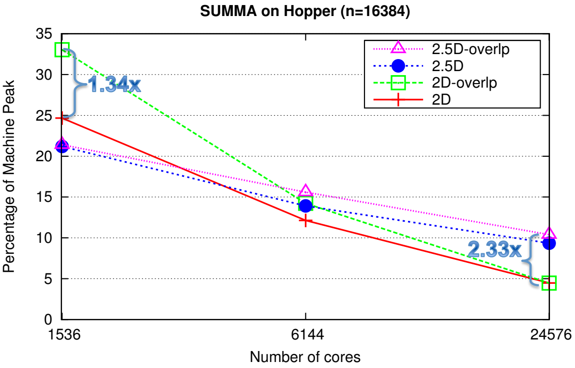


- At large scale avoidance helps a lot (2.08× speedup) (there is a lot of communication to avoid!)
- At medium scale communication avoidance may yield slowdown

2.5D Matrix Multiplication with CO (SUMMA)

- We overlap the broadcasts of the next iteration with the local Mat.Mul computation of current iteration **on each** of the c processor layers
- **Additional communication buffers are needed on top of the extra memory needed for replication**

Putting everything together



- **At medium scale overlapping itself yields best performance**
- **At large scale combining both techniques pays off**

Cannon's algorithm

- In SUMMA we use collective communication operations

Cannon's algorithm

- In SUMMA we use collective communication operations
- In Cannon's algorithm the communication needed is shifting

Cannon's algorithm

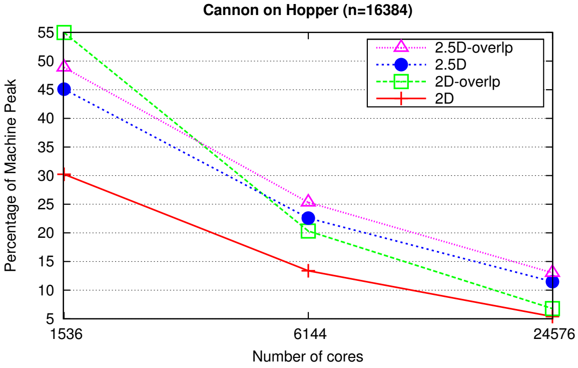
- In SUMMA we use collective communication operations
- In Cannon's algorithm the communication needed is shifting
- The same techniques can be applied for Cannon's algorithm

Cannon's algorithm

- In SUMMA we use collective communication operations
- In Cannon's algorithm the communication needed is shifting
- The same techniques can be applied for Cannon's algorithm
- Use fast **one-sided communication** provided by UPC

Cannon's algorithm

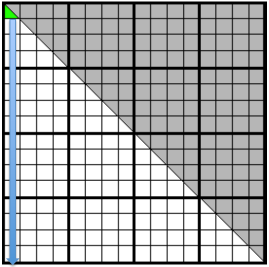
- In SUMMA we use collective communication operations
- In Cannon's algorithm the communication needed is shifting
- The same techniques can be applied for Cannon's algorithm
- Use fast **one-sided communication** provided by UPC



2D Cholesky factorization

- Factorize a symmetric positive definite matrix A into $A = L \cdot L^T$, where L is lower triangular
- Take advantage of symmetry and store only half of matrix A
- Employ block-cyclic layout for load-balance

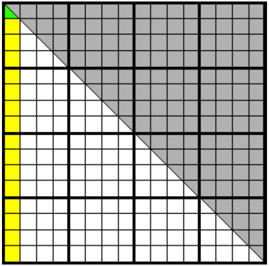
2D Cholesky factorization



1. Factorize the upper-left (green) block & broadcast it to the column
2. Update via TRSM the (yellow) block column
3. Broadcast the factorized column in two phases & update the trailing matrix (white blocks)
4. Continue with the factorization of the second block column and repeat previous steps until all matrix is factorized

- Communication involved is row and column broadcasts
- There are dependencies among rows and column during factorization

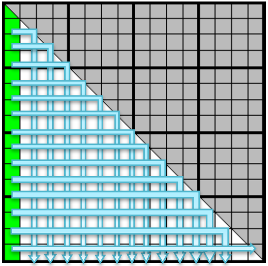
2D Cholesky factorization



1. Factorize the upper-left (green) block & broadcast it to the column
2. Update via TRSM the (yellow) block column
3. Broadcast the factorized column in two phases & update the trailing matrix (white blocks)
4. Continue with the factorization of the second block column and repeat previous steps until all matrix is factorized

- Communication involved is row and column broadcasts
- There are dependencies among rows and column during factorization

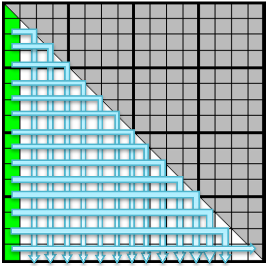
2D Cholesky factorization



1. Factorize the upper-left (green) block & broadcast it to the column
2. Update via TRSM the (yellow) block column
3. Broadcast the factorized column in two phases & update the trailing matrix (white blocks)
4. Continue with the factorization of the second block column and repeat previous steps until all matrix is factorized

- Communication involved is row and column broadcasts
- There are dependencies among rows and column during factorization

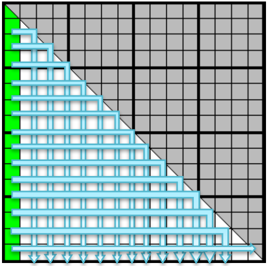
2D Cholesky factorization



1. Factorize the upper-left (green) block & broadcast it to the column
2. Update via TRSM the (yellow) block column
3. Broadcast the factorized column in two phases & update the trailing matrix (white blocks)
4. Continue with the factorization of the second block column and repeat previous steps until all matrix is factorized

- Communication involved is row and column broadcasts
- There are dependencies among rows and column during factorization

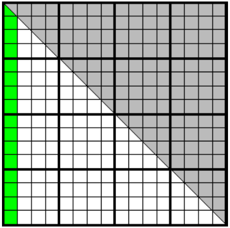
2D Cholesky factorization



1. Factorize the upper-left (green) block & broadcast it to the column
2. Update via TRSM the (yellow) block column
3. Broadcast the factorized column in two phases & update the trailing matrix (white blocks)
4. Continue with the factorization of the second block column and repeat previous steps until all matrix is factorized

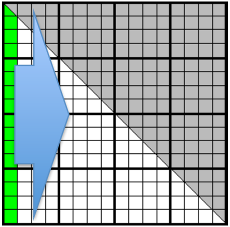
- Communication involved is row and column broadcasts
- There are dependencies among rows and column during factorization

2D Cholesky factorization with overlapping



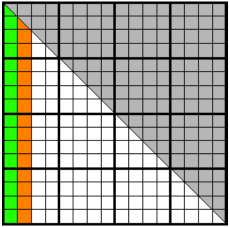
1. Factorize the 1^{st} block-column
2. Broadcast factorized column
3. Update & factorize **only** the 2^{nd} block-column
4. **Overlap**
   - broadcast of the 2^{nd} block-column
   - update of the rest trailing matrix (using 1^{st} block-column)

2D Cholesky factorization with overlapping



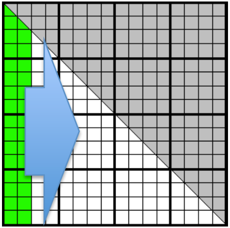
1. Factorize the 1^{st} block-column
2. Broadcast factorized column
3. Update & factorize **only** the 2^{nd} block-column
4. **Overlap**
   - broadcast of the 2^{nd} block-column
   - update of the rest trailing matrix (using 1^{st} block-column)

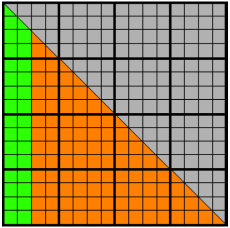
2D Cholesky factorization with overlapping



1. Factorize the 1^{st} block-column
2. Broadcast factorized column
3. Update & factorize **only** the 2^{nd} block-column
4. **Overlap**
   - broadcast of the 2^{nd} block-column
   - update of the rest trailing matrix (using 1^{st} block-column)

2D Cholesky factorization with overlapping



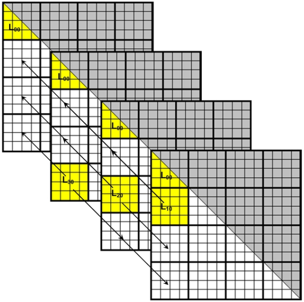


1. Factorize the 1^{st} block-column
2. Broadcast factorized column
3. Update & factorize **only** the 2^{nd} block-column
4. **Overlap**
   - broadcast of the 2^{nd} block-column
   - update of the rest trailing matrix (using 1^{st} block-column)

2.5D Cholesky factorization

[McColl and Tiskin 99], [Solomonik and Demmel 11]

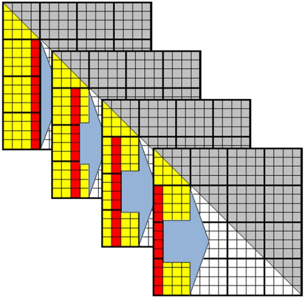
- Employ two levels of blocking: “Fat panels” and “blocks”



1. All layers jointly factorize a “fat” panel
2. Broadcast different subpanels within each layer & update trailing matrices
3. All-reduce the next "fat" panel to accumulate the updates

2.5D Cholesky factorization

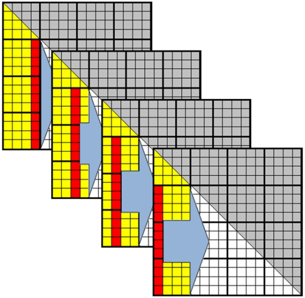
[McColl and Tiskin 99], [Solomonik and Demmel 11]



1. All layers jointly factorize a “fat” panel
2. Broadcast different subpanels within each layer & update trailing matrices
3. All-reduce the next ”fat” panel to accumulate the updates

2.5D Cholesky factorization

[McColl and Tiskin 99], [Solomonik and Demmel 11]

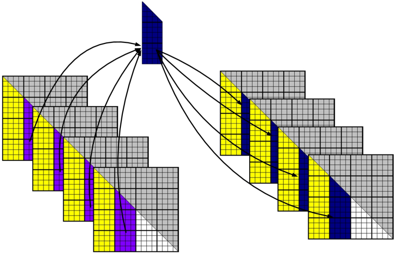


1. All layers jointly factorize a “fat” panel
2. Broadcast different subpanels within each layer & update trailing matrices

3. All-reduce the next ”fat” panel to accumulate the updates

2.5D Cholesky factorization

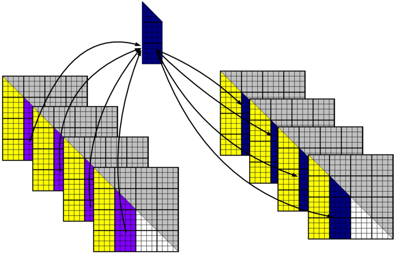
[McColl and Tiskin 99], [Solomonik and Demmel 11]



1. All layers jointly factorize a “fat” panel
2. Broadcast different subpanels within each layer & update trailing matrices
3. All-reduce the next "fat" panel to accumulate the updates

2.5D Cholesky factorization

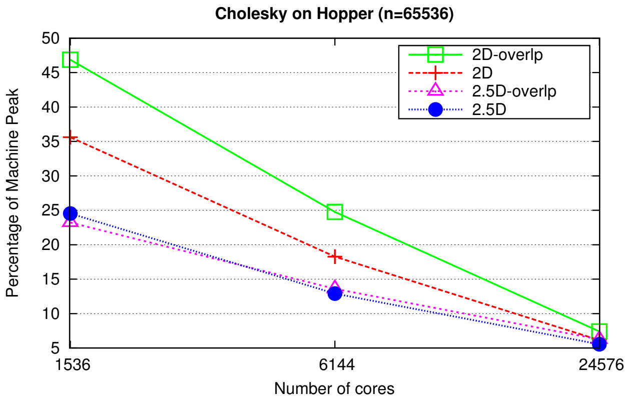
[McColl and Tiskin 99], [Solomonik and Demmel 11]



1. All layers jointly factorize a "fat" panel
2. Broadcast different subpanels within each layer & update trailing matrices
3. All-reduce the next "fat" panel to accumulate the updates

- At step 2 we can overlap computation and communication similarly to the 2D version

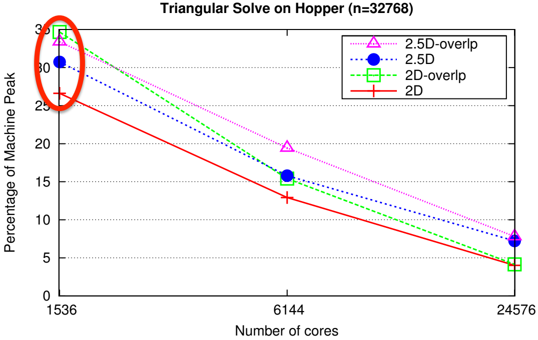
Performance results on Hopper (Cray XE6)



- CO helps more at the smallest scale (1,536 cores)
- Have not reached yet the cross-point of CA and 2D
- Future work: Aggregate updates to improve performance

Triangular Solve (TRSM)

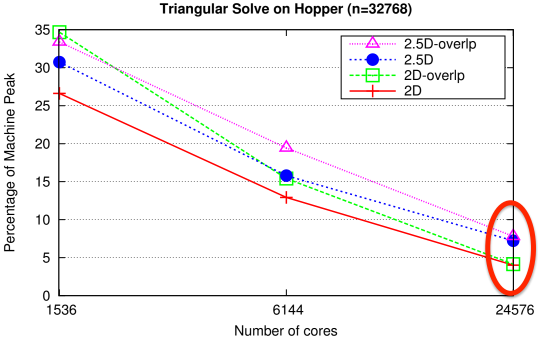
- Computes X, such that $X \cdot U = B$ with upper-triangular U
- Similar parallelization to Cholesky



- At small scale overlapping outperforms other versions

Triangular Solve (TRSM)

- Computes X, such that $X \cdot U = B$ with upper-triangular U
- Similar parallelization to Cholesky



- At small scale overlapping outperforms other versions
- At large scale the combining optimizations is the best option

Table of contents

1. Introduction
   - Communication is Expensive
   - Communication avoidance vs Overlapping
   - CA and CO in Linear Algebra
2. Linear algebra algorithms
   - Matrix Multiplication
   - Cholesky factorization
   - Triangular Solve (TRSM)
3. Performance modeling
   - Motivation
   - Methodology
4. Conclusions
   - Conclusions

Can we explain the behavior of CO and CA?

- Communication avoiding and overlapping:

Can we explain the behavior of CO and CA?

- Communication avoiding and overlapping:
  - Introduce four different algorithmic variants for each routine

Can we explain the behavior of CO and CA?

- Communication avoiding and overlapping:
  - Introduce four different algorithmic variants for each routine
  - Have different memory requirements

Can we explain the behavior of CO and CA?

- Communication avoiding and overlapping:
  - Introduce four different algorithmic variants for each routine
  - Have different memory requirements
  - Lead to performance gains dependent on the problem size, the algorithm and the machine configuration

Can we explain the behavior of CO and CA?

- Communication avoiding and overlapping:
  - Introduce four different algorithmic variants for each routine
  - Have different memory requirements
  - Lead to performance gains dependent on the problem size, the algorithm and the machine configuration
  - Tunable parameters with complicated interactions

Can we explain the behavior of CO and CA?

- Communication avoiding and overlapping:
  - Introduce four different algorithmic variants for each routine
  - Have different memory requirements
  - Lead to performance gains dependent on the problem size, the algorithm and the machine configuration
  - Tunable parameters with complicated interactions

Parameter \ Effect	# messages	load balance	computation efficiency
block size ↑	↓	↓	↑
replication ↑	↑/↓	↑/↓	NA
fat panel size ↑	↓	↓	NA

Can we explain the behavior of CO and CA?

- Communication avoiding and overlapping:
  - Introduce four different algorithmic variants for each routine
  - Have different memory requirements
  - Lead to performance gains dependent on the problem size, the algorithm and the machine configuration
  - Tunable parameters with complicated interactions

Effect / Parameter	# messages	load balance	computation efficiency
block size ↑	↓	↓	↑
replication ↑	↑/↓	↑/↓	NA
fat panel size ↑	↓	↓	NA

- Communication performance depends on number of processors

Can we explain the behavior of CO and CA?

- Communication avoiding and overlapping:
  - Introduce four different algorithmic variants for each routine
  - Have different memory requirements
  - Lead to performance gains dependent on the problem size, the algorithm and the machine configuration
  - Tunable parameters with complicated interactions

Effect / Parameter	# messages	load balance	computation efficiency
block size ↑	↓	↓	↑
replication ↑	↑/↓	↑/↓	NA
fat panel size ↑	↓	↓	NA

- Communication performance depends on number of processors
- **Given a problem instance and a machine configuration would like to predict the optimal variant**

Methodology for constructing performance models

- We construct detailed performance models

Methodology for constructing performance models

- We construct detailed performance models
  - Inputs: matrix size, # of processors, BLAS efficiency, LogGP parameters, block sizes, replication factors, fat panel sizes

Methodology for constructing performance models

- We construct detailed performance models
  - Inputs: matrix size, # of processors, BLAS efficiency, LogGP parameters, block sizes, replication factors, fat panel sizes
  - Output: Estimate for execution time of algorithm

Methodology for constructing performance models

- We construct detailed performance models
  - Inputs: matrix size, # of processors, BLAS efficiency, LogGP parameters, block sizes, replication factors, fat panel sizes
  - Output: Estimate for execution time of algorithm
- We track the execution flow of each algorithm and estimate completion time for encountered operation

Methodology for constructing performance models

- We construct detailed performance models
  - Inputs: matrix size, # of processors, BLAS efficiency, LogGP parameters, block sizes, replication factors, fat panel sizes
  - Output: Estimate for execution time of algorithm
- We track the execution flow of each algorithm and estimate completion time for encountered operation
  - Estimate computation times through BLAS microbenchmarks

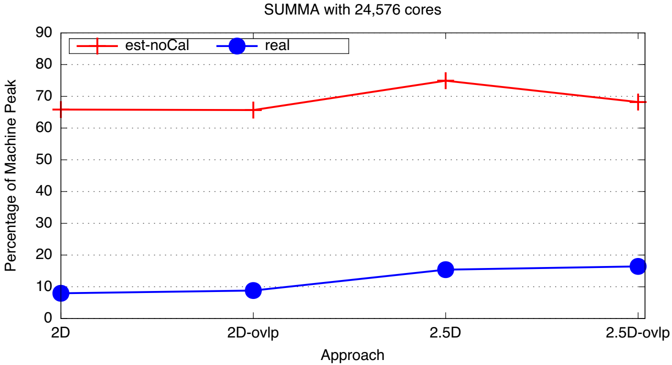
Methodology for constructing performance models

- We construct detailed performance models
  - Inputs: matrix size, # of processors, BLAS efficiency, LogGP parameters, block sizes, replication factors, fat panel sizes
  - Output: Estimate for execution time of algorithm
- We track the execution flow of each algorithm and estimate completion time for encountered operation
  - Estimate computation times through BLAS microbenchmarks
  - Estimate communication times through LogGP model (collective models from [Thakur, Rabenseifner, Gropp 2005])

Methodology for constructing performance models

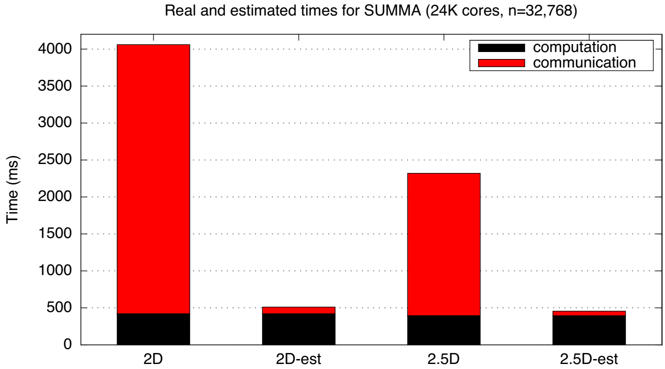
- We construct detailed performance models
  - Inputs: matrix size, # of processors, BLAS efficiency, LogGP parameters, block sizes, replication factors, fat panel sizes
  - Output: Estimate for execution time of algorithm
- We track the execution flow of each algorithm and estimate completion time for encountered operation
  - Estimate computation times through BLAS microbenchmarks
  - Estimate communication times through LogGP model (collective models from [Thakur, Rabenseifner, Gropp 2005])
- We take into account possible idle times

First approach: Ignore network congestion



- We predict correctly the relative performance of the CA algorithms
- Prediction of absolute performance is inaccurate

First approach: Ignore network congestion



- We predict accurately the computation time
- We ignore congestion → optimistic communication time

Quantify degradation due to congestion

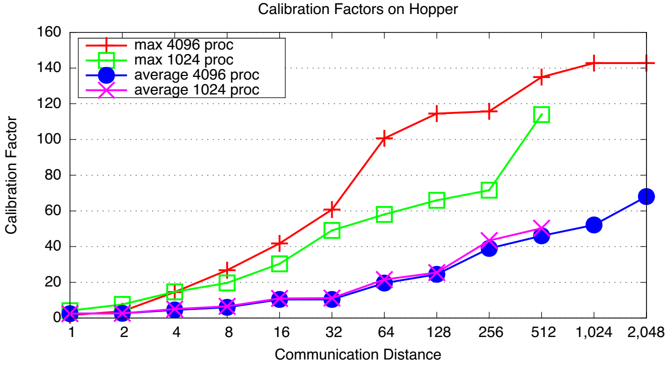
- **Calibration factor**: $\frac{ideal\ BW}{real\ BW}$ when several processes use the network simultaneously

Quantify degradation due to congestion

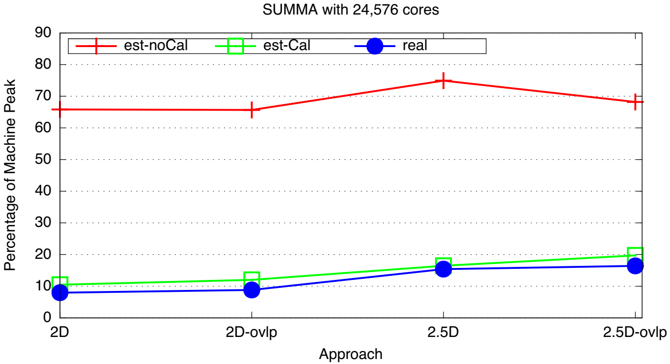
- **Calibration factor**: $\frac{ideal\ \ BW}{real\ \ BW}$ when several processes use the network simultaneously
- Extract calibration factors via microbenchmarks

Quantify degradation due to congestion

- **Calibration factor**: $\frac{ideal\ \ BW}{real\ \ BW}$ when several processes use the network simultaneously
- Extract calibration factors via microbenchmarks

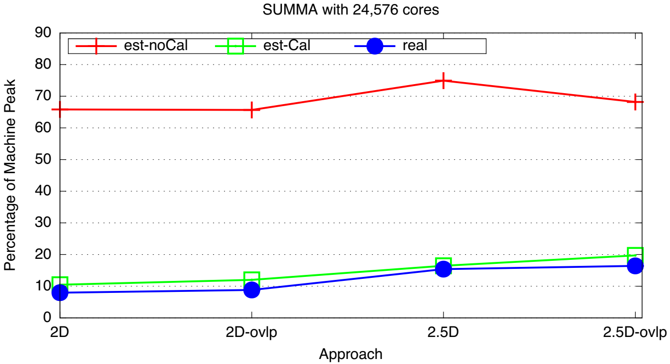


Including calibration factors in the models



- **We predict accurately the absolute performance**

Including calibration factors in the models



- **We predict accurately the absolute performance**
- Similar results for the rest algorithms

Table of contents

Contents:

1 Introduction
- Communication is Expensive
- Communication avoidance vs Overlapping
- CA and CO in Linear Algebra

2 Linear algebra algorithms
- Matrix Multiplication
- Cholesky factorization
- Triangular Solve (TRSM)

3 Performance modeling
- Motivation
- Methodology

4 Conclusions
- Conclusions

Conclusions

1. Which is more important? CA or CO?

Conclusions

1. Which is more important? CA or CO? It depends

Conclusions

1. Which is more important? CA or CO? It depends
   - For core counts where communication and computation are balanced CO helps more (up to 1.82× speedup)
   - For larger core counts CA is more beneficial (up to 2.08× speedup)

Conclusions

1. Which is more important? CA or CO? It depends
   - For core counts where communication and computation are balanced CO helps more (up to 1.82$\times$ speedup)
   - For larger core counts CA is more beneficial (up to 2.08$\times$ speedup)
2. Is there a benefit from combining both techniques?

Conclusions

1. Which is more important? CA or CO? It depends
   - For core counts where communication and computation are balanced CO helps more (up to 1.82× speedup)
   - For larger core counts CA is more beneficial (up to 2.08× speedup)
2. Is there a benefit from combining both techniques? Yes

Conclusions

1. Which is more important? CA or CO? It depends
   - For core counts where communication and computation are balanced CO helps more (up to 1.82$\times$ speedup)
   - For larger core counts CA is more beneficial (up to 2.08$\times$ speedup)
2. Is there a benefit from combining both techniques? Yes
   - CO helps further hide the minimized communication by CA (up to 2.33$\times$ speedup)
3. Can we explain the behavior of CA and CO under different situations?

Conclusions

1. Which is more important? CA or CO? It depends
   - For core counts where communication and computation are balanced CO helps more (up to 1.82$\times$ speedup)
   - For larger core counts CA is more beneficial (up to 2.08$\times$ speedup)
2. Is there a benefit from combining both techniques? Yes
   - CO helps further hide the minimized communication by CA (up to 2.33$\times$ speedup)
3. Can we explain the behavior of CA and CO under different situations? Yes - Performance models

Conclusions

1. Which is more important? CA or CO? It depends
   - For core counts where communication and computation are balanced CO helps more (up to 1.82× speedup)
   - For larger core counts CA is more beneficial (up to 2.08× speedup)
2. Is there a benefit from combining both techniques? Yes
   - CO helps further hide the minimized communication by CA (up to 2.33× speedup)
3. Can we explain the behavior of CA and CO under different situations? Yes - Performance models
   - We developed detailed performance models
   - They encapsulate complicated interactions between parameters
   - They predict correctly the performance

Thank you!