Parallel $\mathcal{H}$-Arithmetic for Multi- and Many-Core Systems

Ronald Kriemann
MPI MIS

Scalable Hierarchical Algorithms for eXtreme Computing

KAUST

2016-05-09

Outline

1. Model Problem

2. $\mathcal{H}$-Matrix Multiplication

3. $\mathcal{H}$-LU Factorization

4. Matrix-Vector Multiplication

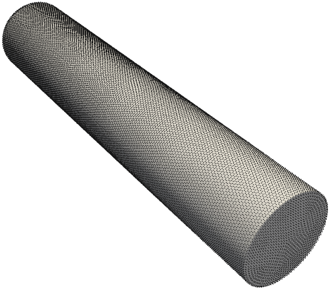
Model Problem

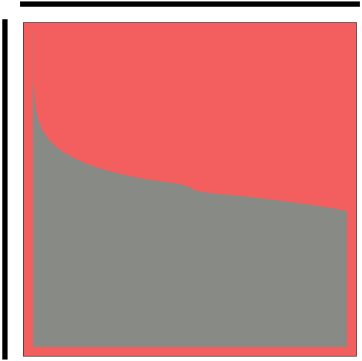
Model Problem

Let A be a matrix defined by the discretization of the Helmholtz SLP integral equation

$$\int_\Gamma \frac{e^{i\kappa|x-y|}}{\|x-y\|} u(x) dy = f(x), \quad x \in \Gamma$$

with wave number $\kappa \in \mathbb{C}$ over a domain $\Gamma = \partial\Omega \subset \mathbb{R}^3$.



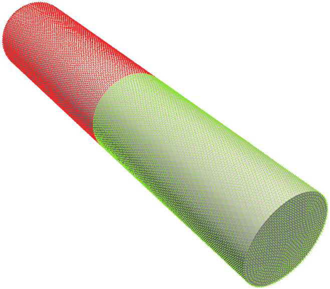


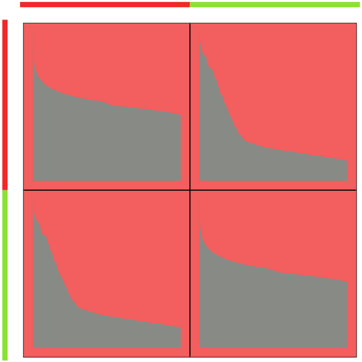
Model Problem

Let A be a matrix defined by the discretization of the Helmholtz SLP integral equation

$$\int_\Gamma \frac{e^{i\kappa|x-y|}}{\|x-y\|} u(x) dy = f(x), \quad x \in \Gamma$$

with wave number $\kappa \in \mathbb{C}$ over a domain $\Gamma = \partial\Omega \subset \mathbb{R}^3$.



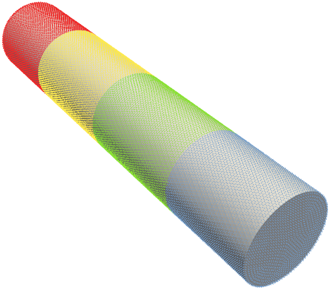


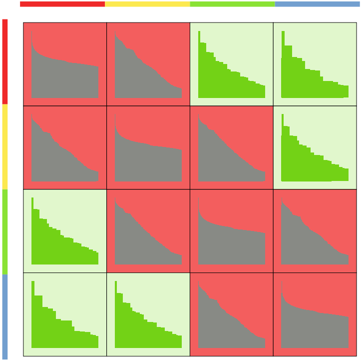
Model Problem

Let A be a matrix defined by the discretization of the Helmholtz SLP integral equation

$$\int_\Gamma \frac{e^{i\kappa|x-y|}}{\|x-y\|} u(x)dy = f(x), \quad x \in \Gamma$$

with wave number $\kappa \in \mathbb{C}$ over a domain $\Gamma = \partial\Omega \subset \mathbb{R}^3$.



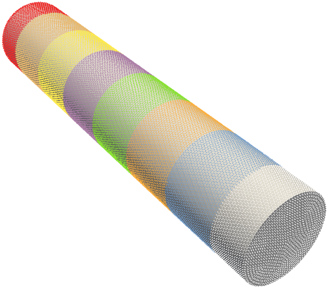


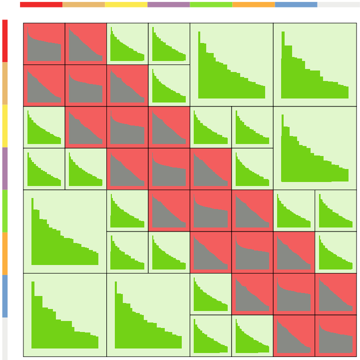
Model Problem

Let A be a matrix defined by the discretization of the Helmholtz SLP integral equation

$$\int_\Gamma \frac{e^{i\kappa|x-y|}}{\|x-y\|} u(x) dy = f(x), \quad x \in \Gamma$$

with wave number $\kappa \in \mathbb{C}$ over a domain $\Gamma = \partial\Omega \subset \mathbb{R}^3$.



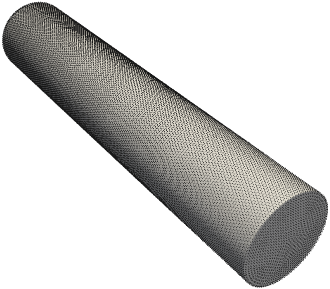


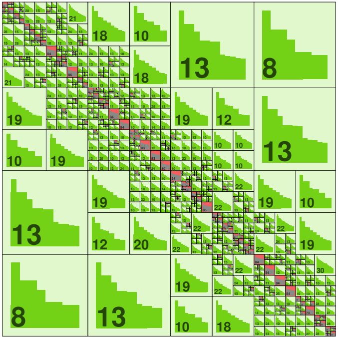
Model Problem

Let A be a matrix defined by the discretization of the Helmholtz SLP integral equation

$$\int_\Gamma \frac{e^{i\kappa|x-y|}}{\|x-y\|} u(x)dy = f(x), \quad x \in \Gamma$$

with wave number $\kappa \in \mathbb{C}$ over a domain $\Gamma = \partial\Omega \subset \mathbb{R}^3$.



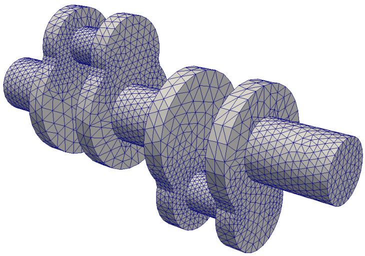


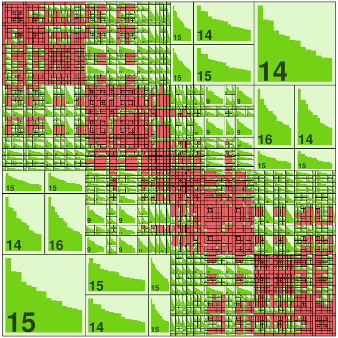
Model Problem

Let A be a matrix defined by the discretization of the Helmholtz SLP integral equation

$$\int_\Gamma \frac{e^{i\kappa|x-y|}}{\|x-y\|} u(x)dy = f(x), \quad x \in \Gamma$$

with wave number $\kappa \in \mathbb{C}$ over a domain $\Gamma = \partial\Omega \subset \mathbb{R}^3$.



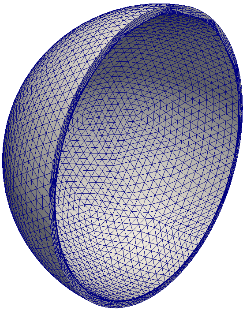


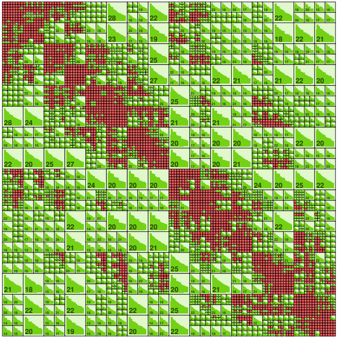
Model Problem

Let A be a matrix defined by the discretization of the Helmholtz SLP integral equation

$$\int_\Gamma \frac{e^{i\kappa|x-y|}}{\|x-y\|} u(x) dy = f(x), \quad x \in \Gamma$$

with wave number $\kappa \in \mathbb{C}$ over a domain $\Gamma = \partial\Omega \subset \mathbb{R}^3$.





$\mathcal{H}$-Matrix Construction

Let $I = \{0, \ldots, n-1\}$ be an index set, $T(I)$ a (binary) cluster tree over I and $T = T(I \times I)$ a block cluster tree over $T(I)$.

Complexity

Computational and memory complexity:

$$\mathcal{O}(n \log n)$$

Numerical Results (Sequential)

Construction of Helmholtz SLP operator with $\kappa = 8$ and fixed block wise accuracy of $\varepsilon = 10^{-4}$ using adaptive cross-approximation.

n	t in sec	$\frac{t}{n \log n}$	Mem in MB	$\frac{\text{Mem}}{n \log n}$
2,680	9.4	3.08	31	1.02
10,720	46.4	3.24	186	1.30
42,880	207.8	3.15	904	1.37
171,520	872.6	2.93	4,290	1.44
686,080	3689.4	2.77	19,810	1.49

(Xeon E7-8857)

$\mathcal{H}$-Matrix Construction

Numerical Results

	#Cores	Speedup
Xeon E5-2670	8	9.36
(2 threads/core)	16	18.49
Xeon E7-8857	12	10.08
	48	38.60
XeonPhi 5110P	60	74.36
(4 threads/core)		

$\mathcal{H}$-Matrix Construction

Numerical Results

	#Cores	Speedup	
		w/ Turbo	w/o Turbo
Xeon E5-2670	8	9.36	
(2 threads/core)	16	18.49	
Xeon E7-8857	12	10.08	*11.78*
	48	38.60	*44.57*
XeonPhi 5110P	60	74.36	
(4 threads/core)			

$\mathcal{H}$-Matrix Multiplication

$\mathcal{H}$-Matrix Multiplication

For an $\mathcal{H}$-matrix A each sub block $A|_{t\times s}$, $t \times s \in T$, of A is either a *low-rank matrix*, a *dense matrix* or a *block matrix* with subblocks

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} := \begin{pmatrix} A_{t_0\times s_0} & A_{t_0\times s_1} \\ A_{t_1\times s_0} & A_{t_1\times s_1} \end{pmatrix}$$

with son clusters $\mathcal{S}(t) = \{t_0, t_1\}$ and $\mathcal{S}(s) = \{s_0, s_1\}$.

$\mathcal{H}$-Matrix Multiplication

For an $\mathcal{H}$-matrix A each sub block $A|_{t \times s}$, $t \times s \in T$, of A is either a *low-rank matrix*, a *dense matrix* or a *block matrix* with subblocks

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} := \begin{pmatrix} A_{t_0 \times s_0} & A_{t_0 \times s_1} \\ A_{t_1 \times s_0} & A_{t_1 \times s_1} \end{pmatrix}$$

with son clusters $\mathcal{S}(t) = \{t_0, t_1\}$ and $\mathcal{S}(s) = \{s_0, s_1\}$.

Algorithm

For $\mathcal{H}$-matrices A, B, C the product $C := C + \alpha A \cdot B, \alpha \in \mathbb{C}$, is computed by:

procedure $\textsc{Multiply}(\alpha, A, B, C)$
  if A, B, C are block matrices **then**
    for $i \in \{0, 1\}$ **do**
      for $j \in \{0, 1\}$ **do**
        for $\ell \in \{0, 1\}$ **do**
          $\textsc{Multiply}(\alpha, A_{ij}, B_{i\ell}, C_{\ell j})$;
  else
    $C := C + \alpha AB$;   *//specialized $\mathcal{H}$-functions*

$\mathcal{H}$-Matrix Multiplication

Complexity

$$\mathcal{O}\left(n \log^2 n\right)$$

Numerical Results (Sequential)

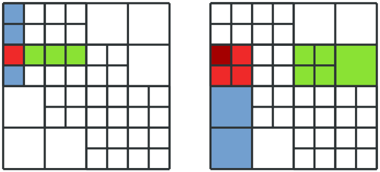
n	t in sec	$\frac{t}{n \log^2 n}$
2,680	22.6	6.49
10,720	155.6	8.10
42,880	800.3	7.88
171,520	4,421.6	8.53
686,080	22,286.4	8.64

(Xeon E7-8857)

$\mathcal{H}$-Matrix Multiplication

Parallelization

Only *disjoint* blocks in C can be computed independently:

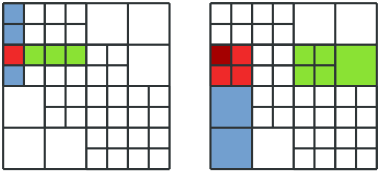


Synchronisation for matrix block updates are neccessary.

$\mathcal{H}$-Matrix Multiplication

Parallelization

Only *disjoint* blocks in C can be computed independently:



Synchronisation for matrix block updates are neccessary.

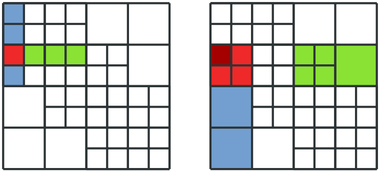
Define computational tasks and use dynamic task scheduling:

procedure $\textsc{Multiply}(\alpha, A, B, C)$
  if A, B, C are block matrices **then**
    parallel for $i, j, \ell \in \{0, 1\}$ **do**
      $\textsc{Multiply}(\alpha, A_{ij}, B_{i\ell}, C_{\ell j});$
  else
    task$(C := C + \alpha AB);$

$\mathcal{H}$-Matrix Multiplication

Parallelization

Only *disjoint* blocks in C can be computed independently:



Synchronisation for matrix block updates are neccessary.

Define computational tasks and use dynamic task scheduling:

procedure $\textsc{Multiply}(\alpha, A, B, C)$
  if A, B, C are block matrices **then**
    parallel for $i, j, \ell \in \{0, 1\}$ **do**
      task($\textsc{Multiply}(\alpha, A_{ij}, B_{i\ell}, C_{\ell j})$);
  else
    $C := C + \alpha AB$;

$\mathcal{H}$-Matrix Multiplication

Numerical Results (Parallel)

	#Cores	Speedup	
Xeon E5-2670	8	11.47	
	16	20.52	
Xeon E7-8857	12	10.68	*12.35*
	48	40.19	*46.33*
XeonPhi 5110P	60	111.62	

$\mathcal{H}$-LU Factorization

$\mathcal{H}$-LU Factorization

The LU factorisation $A = LU$ is defined by the block structure of A.
If $A|_{t \times t}, t \in T(I)$ is a block matrix, then we have:

$$A|_{t\times t} = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix},$$

which leads to the following equations:

$$A_{00} = L_{00}U_{00} \qquad \text{(Recursion)}$$

$$A_{01} = L_{00}U_{01} \qquad \text{(Matrix Solve)}$$

$$A_{10} = L_{10}U_{00} \qquad \text{(Matrix Solve)}$$

$$A_{11} = A_{11} - L_{10}U_{01} \qquad \text{(Multiplication)}$$

$$A_{11} = L_{11}U_{11} \qquad \text{(Recursion)}$$

$\mathcal{H}$-LU Factorization

The above equations directly translate into the following algorithms for $\mathcal{H}$-LU factorisation and matrix solve:

```
procedure LU(A, L, U)
   if A is block matrix then
      LU( A_00, L_00, U_00 );
      SolveLL( A_01, L_00, U_01 );
      SolveUR( A_10, L_10, U_00 );
      Multiply( −1, L_10, U_01, A_11 );
      LU( A_11, L_11, U_11 );
   else
      A = LU;
```

```
procedure SolveLL(A, L, B)
   if A, L, B are block matrices then
      SolveLL( A_00, L_00, B_00 );
      SolveLL( A_01, L_00, B_01 );
      Multiply( −1, L_10, B_00, A_11 );
      Multiply( −1, L_10, B_01, A_11 );
      SolveLL( A_10, L_11, B_10 );
      SolveLL( A_11, L_11, B_11 );
   else
      LB = A;
```

Complexity

$$\mathcal{O}\left(n \log^2 n\right)$$

$\mathcal{H}$-LU Factorization

Numerical Results (Sequential)

Factorization of Helmholtz SLP operator with $\kappa = 8$ and fixed block wise accuracy of $\varepsilon = 10^{-4}$.

n	t in sec	$\frac{t}{n \log^3 n}$	Mem in MB	$\frac{\text{Mem}}{n \log n}$
2,680	5.9	1.49	30	0.98
10,720	48.4	1.88	182	1.27
42,880	266.9	1.71	887	1.34
171,520	1636.2	1.81	4,220	1.41
686,080	8835.4	1.77	20,010	1.50

(Xeon E7-8857)

$\mathcal{H}$-LU Factorization

Numerical Results (Sequential)

Factorization of Helmholtz SLP operator with $\kappa = 8$ and fixed block wise accuracy of $\varepsilon = 10^{-4}$.

n	t in sec	$\frac{t}{n \log^3 n}$	Mem in MB	$\frac{\text{Mem}}{n \log n}$
2,680	5.9	1.49	30	0.98
10,720	48.4	1.88	182	1.27
42,880	266.9	1.71	887	1.34
171,520	1636.2	1.81	4,220	1.41
686,080	8835.4	1.77	20,010	1.50

(Xeon E7-8857)

Numerical Results (Parallel)

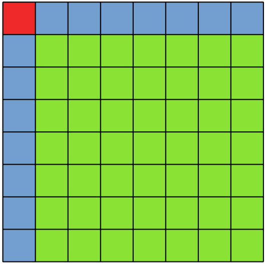
Only using internal parallelism and parallel matrix multiplication.

	#Cores	Speedup
XeonPhi 5110P	60	19.08

Dense LU Factorization

Let $A \in \mathbb{R}^{n \times n}$ be a dense matrix with block size $0 < N < n$.

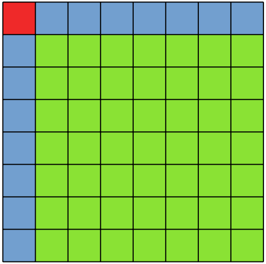
procedure DENSELU(A, L, U)
  for $0 \le i < n/N$ **do**
    $A_{ii} = L_{ii} U_{ii}$;
    for $i < j < n/N$ **do**
      SOLVELL(A_{ij}, L_{ii}, U_{ij});
      SOLVEUR(A_{ji}, L_{ji}, U_{ii});
    for $i < j < n/N$ **do**
      for $i < \ell < n/N$ **do**
        $A_{j\ell} := A_{j\ell} - L_{ji} U_{i\ell}$;



Dense LU Factorization

Let $A \in \mathbb{R}^{n \times n}$ be a dense matrix with block size $0 < N < n$.

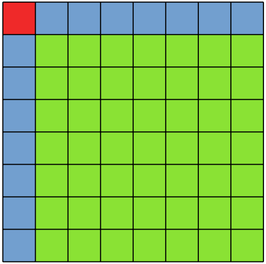
procedure $\textsc{DenseLU}(A, L, U)$
  for $0 \le i < n/N$ **do**
    $A_{ii} = L_{ii} U_{ii}$;
    parallel for $i < j < n/N$ **do**
      $\textsc{SolveLL}(\ A_{ij}, L_{ii}, U_{ij}\ )$;
      $\textsc{SolveUR}(\ A_{ji}, L_{ji}, U_{ii}\ )$;
    parallel for $i < j < n/N$ **do**
      parallel for $i < \ell < n/N$ **do**
        $A_{j\ell} := A_{j\ell} - L_{ji} U_{i\ell}$;



Dense LU Factorization

Let $A \in \mathbb{R}^{n \times n}$ be a dense matrix with block size $0 < N < n$.

procedure $\text{DenseLU}(A, L, U)$
  for $0 \leq i < n/N$ **do**
    task($A_{ii} = L_{ii} U_{ii}$);
    for $i < j < n/N$ **do**
      task($\text{SolveLL}(\ A_{ij}, L_{ii}, U_{ij}\ )$);
      task($\text{SolveUR}(\ A_{ji}, L_{ji}, U_{ii}\ )$);
    for $i < j < n/N$ **do**
      for $i < \ell < n/N$ **do**
        task($A_{j\ell} := A_{j\ell} - L_{ji} U_{i\ell}$);

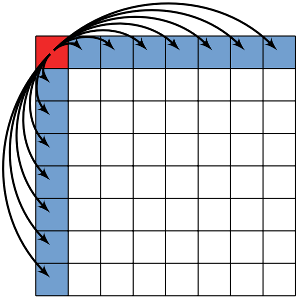


Define task for each block computation.

Dense LU Factorization

Let $A \in \mathbb{R}^{n \times n}$ be a dense matrix with block size $0 < N < n$.

procedure $\text{DenseLU}(A, L, U)$
  for $0 \leq i < n/N$ **do**
    task($A_{ii} = L_{ii} U_{ii}$);
    for $i < j < n/N$ **do**
      task($\text{SolveLL}(\ A_{ij}, L_{ii}, U_{ij}\ )$);
      task($\text{SolveUR}(\ A_{ji}, L_{ji}, U_{ii}\ )$);
    for $i < j < n/N$ **do**
      for $i < \ell < n/N$ **do**
        task($A_{j\ell} := A_{j\ell} - L_{ji} U_{i\ell}$);



Define task for each block computation.

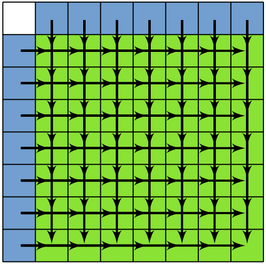
Task Dependencies

task($A_{ii} = L_{ii} U_{ii}$) → **task**($\text{SolveLL}(A_{ij}, L_{ii}, U_{ij})$)

Dense LU Factorization

Let $A \in \mathbb{R}^{n \times n}$ be a dense matrix with block size $0 < N < n$.

procedure $\text{DenseLU}(A, L, U)$
  for $0 \leq i < n/N$ **do**
    task$(A_{ii} = L_{ii} U_{ii})$;
    for $i < j < n/N$ **do**
      task$(\text{SolveLL}(\; A_{ij}, L_{ii}, U_{ij} \;))$;
      task$(\text{SolveUR}(\; A_{ji}, L_{ji}, U_{ii} \;))$;
    for $i < j < n/N$ **do**
      for $i < \ell < n/N$ **do**
        task$(A_{j\ell} := A_{j\ell} - L_{ji} U_{i\ell})$;



Define task for each block computation.

Task Dependencies

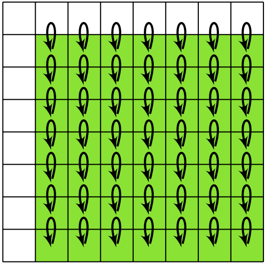
task$(A_{ii} = L_{ii} U_{ii})$ → **task**$(\text{SolveLL}(A_{ij}, L_{ii}, U_{ij}))$

task$(\text{SolveLL}(\; A_{i\ell}, L_{ii}, U_{i\ell} \;))$ → **task**$(A_{j\ell} := A_{j\ell} - L_{ji} U_{i\ell} \;)$

Dense LU Factorization

Let $A \in \mathbb{R}^{n \times n}$ be a dense matrix with block size $0 < N < n$.

procedure $\textsc{DenseLU}(A, L, U)$
  for $0 \le i < n/N$ **do**
    task($A_{ii} = L_{ii} U_{ii}$);
    for $i < j < n/N$ **do**
      task($\textsc{SolveLL}(\ A_{ij}, L_{ii}, U_{ij}\ )$);
      task($\textsc{SolveUR}(\ A_{ji}, L_{ji}, U_{ii}\ )$);
    for $i < j < n/N$ **do**
      for $i < \ell < n/N$ **do**
        task($A_{j\ell} := A_{j\ell} - L_{ji} U_{i\ell}$);



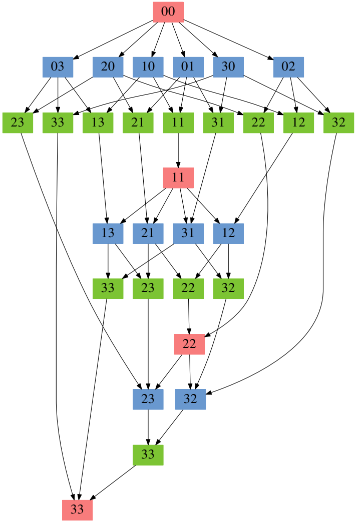
Define task for each block computation.

Task Dependencies

task($A_{ii} = L_{ii} U_{ii}$)	→	**task**($\textsc{SolveLL}(A_{ij}, L_{ii}, U_{ij})$)
task($\textsc{SolveLL}(\ A_{i\ell}, L_{ii}, U_{i\ell}\ )$)	→	**task**($A_{j\ell} := A_{j\ell} - L_{ji} U_{i\ell}$)
task($A_{j\ell} := A_{j\ell} - L_{ji} U_{i\ell}$)	→	**task**($\textsc{SolveLL}(\ A_{j\ell}, L_{jj}, U_{j\ell}\ )$)
	→	**task**($A_{jj} = L_{jj} U_{jj}$)

Dense LU Factorization

Tasks and dependencies form a *directed acyclic graph* (DAG).



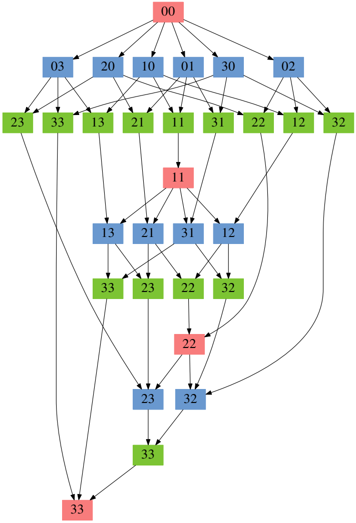
(DAG for a 4×4 matrix)

Dense LU Factorization

Tasks and dependencies form a *directed acyclic graph* (DAG).

DAG execution

- execute task after all its dependencies are met,
- avoids redundant synchronisations.



(DAG for a 4×4 matrix)

Dense LU Factorization

Tasks and dependencies form a *directed acyclic graph* (DAG).

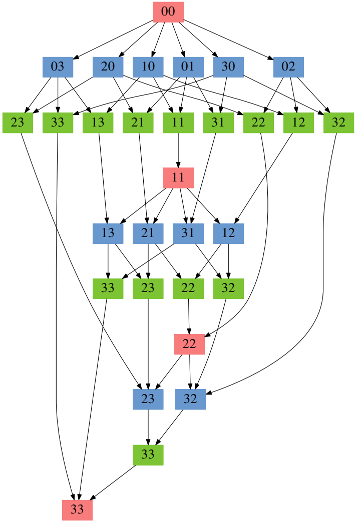
DAG execution

- execute task after all its dependencies are met,
- avoids redundant synchronisations.

DAG definition is hardware *independent*.

DAG execution (task scheduling) should be optimised for specific systems.

DAG defines parallel degree and minimal number of steps.

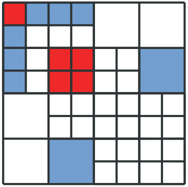


(DAG for a 4×4 matrix)

Task based $\mathcal{H}$-LU Factorization

A similar algorithm as for task-based dense LU can be formulated for a task-based $\mathcal{H}$-LU factorization:

```
procedure LU( A|_{t×t}, L|_{t×t}, U|_{t×t} )
  if A is block matrix then
    for i ∈ {0,1} do
      task(LU( A|_{t_i×t_i} )); ℓ := level(t_i);
      for s ∈ T^ℓ(I), s >_I t_i do
        if A|_{s×t_i} is not blocked then
          task(SolveUR( A|_{s×t_i}, L|_{s×t_i}, U|_{t_i×t_i} ));
        if A|_{t_i×s} is not blocked then
          task(SolveLL( A|_{t_i×s}, L|_{t_i×t_i}, U|_{t_i×s} ));
      for s, r ∈ T^ℓ(I), s, r >_I t_i do
        if L|_{r×t_i}, U_{t_i×s} or A|_{r×s} is not blocked then
          task(Multiply(−1, L_{r×t_i}, U_{t_i×s}, A|_{r×s}));
  else
    task(A := LU);
```



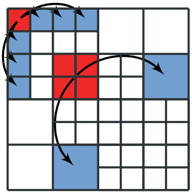
$$T^\ell(I) := \{t \in T(I) : \mathrm{level}(t) = \ell\} \text{ and } s >_I t :\Leftrightarrow \forall i \in s, j \in t : s > t.$$

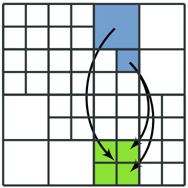
Task based $\mathcal{H}$-LU Factorization

A similar algorithm as for task-based dense LU can be formulated for a task-based $\mathcal{H}$-LU factorization:

procedure $\mathrm{LU}(\ A|_{t\times t},\ L|_{t\times t}, U|_{t\times t}\ )$
  if A is block matrix **then**
    for $i \in \{0,1\}$ **do**
      task($\mathrm{LU}(\ A|_{t_i\times t_i}\ )$); $\ell := \mathrm{level}(t_i)$;
      for $s \in T^\ell(I), s >_I t_i$ **do**
        if $A|_{s\times t_i}$ is not blocked **then**
          task($\mathrm{SolveUR}(\ A|_{s\times t_i}, L|_{s\times t_i}, U|_{t_i\times t_i}\ )$);
        if $A|_{t_i\times s}$ is not blocked **then**
          task($\mathrm{SolveLL}(\ A|_{t_i\times s}, L|_{t_i\times t_i}, U|_{t_i\times s}\ )$);
      for $s, r \in T^\ell(I), s, r >_I t_i$ **do**
        if $L|_{r\times t_i}$, $U_{t_i\times s}$ or $A|_{r\times s}$ is not blocked **then**
          task($\mathrm{Multiply}(-1, L_{r\times t_i}, U_{t_i\times s}, A|_{r\times s})$);
  else
    task($A := LU$);

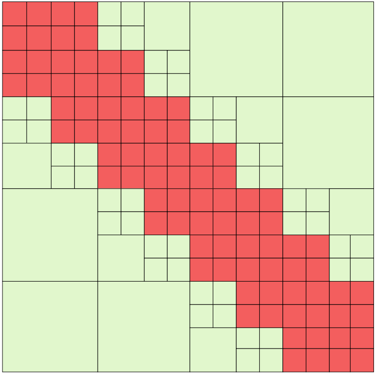
Dependencies:



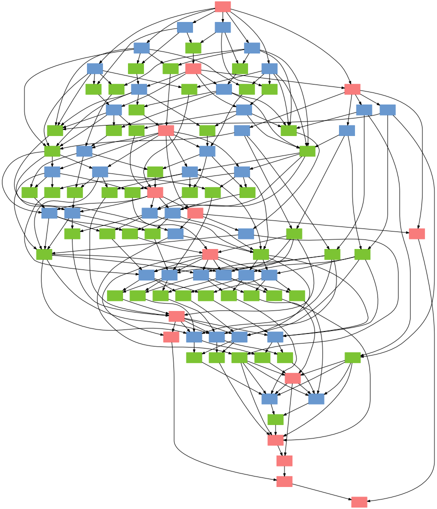


$$T^\ell(I) := \{t \in T(I) : \mathrm{level}(t) = \ell\} \text{ and } s >_I t :\Leftrightarrow \forall i \in s, j \in t : s > t.$$

Task based $\mathcal{H}$-LU Factorization



$\mathcal{H}$-matrix



$\mathcal{H}$-LU DAG

Task based $\mathcal{H}$-LU Factorization

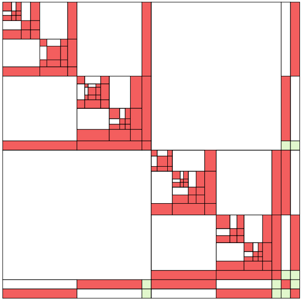
Numerical Results

	#Cores	Speedup	
Xeon E5-2670	8	9.09	
	16	18.27	
Xeon E7-8857	12	11.11	*12.38*
	48	39.76	*43.88*
XeonPhi 5110P	60	89.16	

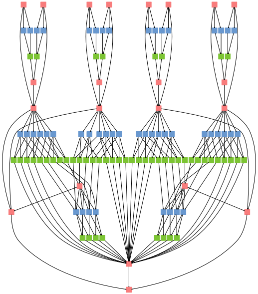
Task based $\mathcal{H}$-LU Factorization

The DAG-based $\mathcal{H}$-LU factorization can also be applied to $\mathcal{H}$-matrices based on sparse matrices.

When using *nested dissection*, all factorization tasks for diagonal domain-domain blocks are start tasks, i.e., without dependencies, for the DAG execution.



$\mathcal{H}$-matrix



$\mathcal{H}$-LU DAG

Matrix-Vector Multiplication

Matrix-Vector Multiplication

Depending on the precision of the $\mathcal{H}$-LU factorization $A = LU$, the system $Ax = b, x, b \in \mathbb{C}^I$, can be solved *directly*, i.e.,

$$x \quad \leftarrow \quad (LU)^{-1} b$$

or *iteratively*, e.g., via $\mathcal{H}$-LU iteration

$$x^{i+1} \quad \leftarrow \quad x^i - (LU)^{-1}(Ax^i - b)$$

For both, forward/backward solves with the triangular matrices L and U are needed:

$$\left(\begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix} \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \right)^{-1} \cdot \begin{pmatrix} v_0 \\ v_1 \end{pmatrix}$$

Numerical Results

Evaluation of

$$L \cdot Ux = b$$

via forward/backward solves:

Sequential		
n	t in msec	$\frac{t}{n \log n}$
2,680	6.6	2.16
10,720	48.1	3.35
42,880	221.9	3.36
171,520	1039.7	3.49
686,080	4613.2	3.47

(Xeon E7-8857)

Matrix-Vector Multiplication

Numerical Results

Evaluation of

$$L \cdot Ux = b$$

via forward/backward solves:

Sequential

n	t in msec	$\frac{t}{n \log n}$
2,680	6.6	2.16
10,720	48.1	3.35
42,880	221.9	3.36
171,520	1039.7	3.49
686,080	4613.2	3.47

(Xeon E7-8857)

Parallel

	#Cores	Speedup
Xeon E5-2670	8	2.02
	16	1.86
Xeon E7-8857	12	2.76
	48	0.97
XeonPhi 5110P	60	3.01

Matrix-Vector Multiplication

Numerical Results

Evaluation of

$$L \cdot Ux = b$$

via forward/backward solves:

Sequential

n	t in msec	$\frac{t}{n \log n}$
2,680	6.6	2.16
10,720	48.1	3.35
42,880	221.9	3.36
171,520	1039.7	3.49
686,080	4613.2	3.47

(Xeon E7-8857)

Parallel

	#Cores	Speedup
Xeon E5-2670	8	2.02
	16	1.86
Xeon E7-8857	12	2.76
	48	0.97
XeonPhi 5110P	60	3.01

Remark

Forward/backward solves are limited by sequential part.

Matrix-Vector Multiplication

By inverting L

$$\begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix}^{-1} = \begin{pmatrix} L_{00}^{-1} & \\ -L_{11}^{-1} L_{10} L_{00}^{-1} & L_{11}^{-1} \end{pmatrix} := W$$

and U

$$\begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix}^{-1} = \begin{pmatrix} U_{00}^{-1} & -U_{00}^{-1} U_{01} U_{11}^{-1} \\ & U_{11}^{-1} \end{pmatrix} := Z$$

the evaluation of $(LU)^{-1}x = ZWx = b$ is performed via matrix-vector multiplication instead of forward/backward solves.

Remark

$Z \cdot W$ is a matrix factorisation of A^{-1}.

Matrix-Vector Multiplication

Parallel Algorithm

Computing the matrix-vector multiplication

$$y := y + \alpha A \cdot x$$

with $x, y \in \mathbb{C}^I, \alpha \in \mathbb{C}$.

procedure $\text{MVM}(\alpha, A, x, y)$
  parallel for leaves $t \times s \in T$ **do**
    task($y' := \alpha A|_{t \times s} x|_s$);
    task($\textsc{Update}(y, y', t)$);

procedure $\textsc{Update}(y, y', t)$
  for chunks $t' \subset t$ **do**
    $\textsc{Lock}(\ t'\ )$;
    $y|_{t'} := y|_{t'} + y'|_{t'}$;
    $\textsc{UnLock}(\ t'\ )$;

With fine-grained chunk sizes for minimal CPU core blocking.

Matrix-Vector Multiplication

Numerical Results

Evaluation of

$$(L \cdot U)^{-1} x = Z \cdot W x = b$$

via matrix-vector multiplication:

	Sequential	
n	t in msec	$\frac{t}{n \log n}$
2,680	5.6	1.83
10,720	36.2	2.52
42,880	159.8	2.42
171,520	755.6	2.53
686,080	3249.3	2.44

(Xeon E7-8857)

Numerical Results

Evaluation of

$$(L \cdot U)^{-1} x = Z \cdot W x = b$$

via matrix-vector multiplication:

Sequential

n	t in msec	$\frac{t}{n \log n}$
2,680	5.6	1.83
10,720	36.2	2.52
42,880	159.8	2.42
171,520	755.6	2.53
686,080	3249.3	2.44

(Xeon E7-8857)

Parallel

	#Cores	Speedup	
Xeon E5-2670	8	4.85	
	16	8.15	
Xeon E7-8857	12	6.73	*7.13*
	48	10.21	*13.07*
XeonPhi 5110P	60	113.55	

Matrix-Vector Multiplication

Numerical Results

Evaluation of

$$(L \cdot U)^{-1} x = Z \cdot W x = b$$

via matrix-vector multiplication:

Sequential

n	t in msec	$\frac{t}{n \log n}$
2,680	5.6	1.83
10,720	36.2	2.52
42,880	159.8	2.42
171,520	755.6	2.53
686,080	3249.3	2.44

(Xeon E7-8857)

Parallel

	#Cores	Speedup	
Xeon E5-2670	8	4.85	
	16	8.15	
Xeon E7-8857	12	6.73	*7.13*
	48	10.21	*13.07*
XeonPhi 5110P	60	113.55	

Remark

Matrix-vector multiplication is limited by memory bandwith.

Literature

R. Kriemann, S. Le Borne,
H-FAINV: Hierarchically factored approximate inverse preconditioners,
Computing and Visualization in Science, 17, pp. 135-150, 2015.

R. Kriemann,
$\mathcal{H}$-LU Factorization on Many-Core Systems,
Computing and Visualization in Science, 16, pp. 105-117, 2013.

R. Kriemann,
Parallel $\mathcal{H}$-Matrix Arithmetics on Shared Memory Systems,
Computing, 74:273–297, 2005.

hlibpro.com