February 27th, 2019

SIAM Conference on Computational Science and Engineering
February 25-March 1, 2019
Spokane Convention Center
Spokane, Washington, USA

Exploiting Nested Task-based Parallelism in the Factorization of Hierarchical Matrices

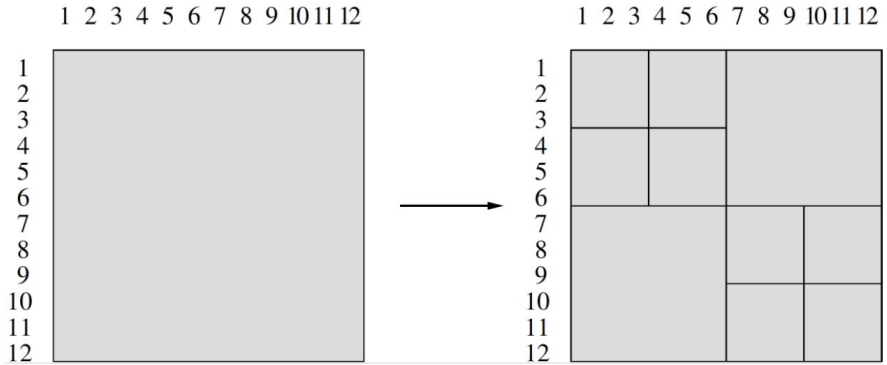
Rocío Carratalá-Sáez
Enrique S. Quintana-Ortí
Universitat Jaume I

Steffen Börm
Sven Christophersen
Christian-Albrechts-Universität zu Kiel

Vicenç Beltran
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

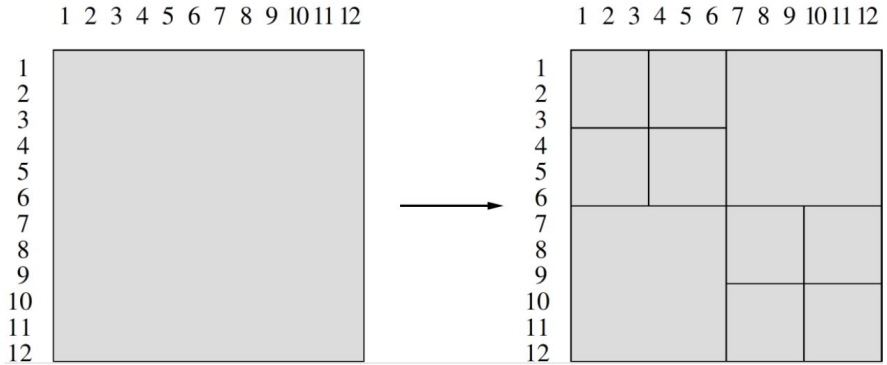
DENSE MATRICES | $\mathcal{H}$ - MATRICES | SPARSE MATRICES

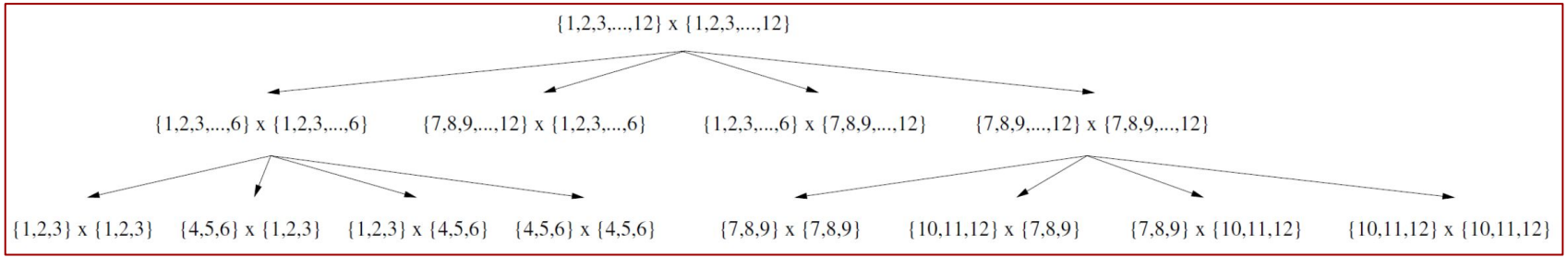
DENSE MATRICES | $\mathcal{H}$ - MATRICES | SPARSE MATRICES



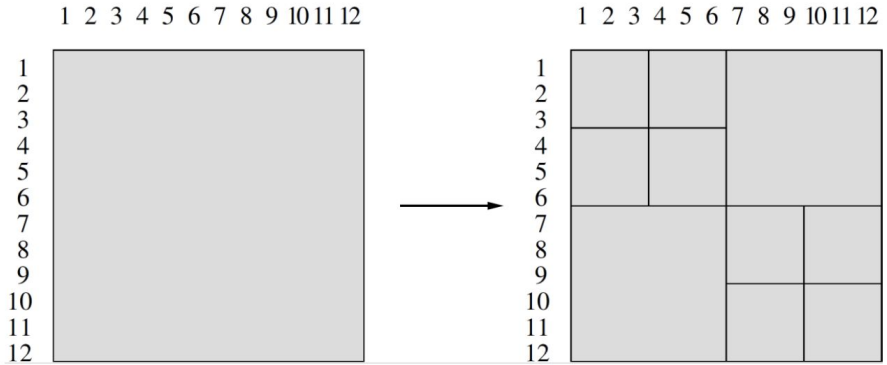
DENSE MATRICES

$\mathcal{H}$ - MATRICES





DENSE MATRICES → $\mathcal{H}$ - MATRICES



DENSE BLOCKS → **DENSE BLOCKS** + **LOW-RANK BLOCKS**

O(log) STORAGE [1]

O(log) COMPUTATIONS [1]

[1] W. Hackbusch. *A sparse matrix arithmetic based on H-matrices*. Computing (1999), 62:89-108

$\mathcal{H}$-LU factorization

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{pmatrix}$$

$\mathcal{H}$-LU factorization

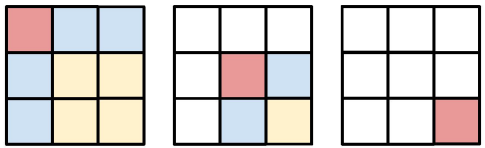
$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{pmatrix}$$

Algorithm 1 Blocked RL algorithm for the LU factorization.

Require: $A \in \mathbb{R}^{n \times n}$

1: **for** $k = 1, 2, \ldots, n_t$ **do**
2: $A_{kk} = L_{kk} U_{kk}$ — **LU**
3: **for** $j = k+1, k+2, \ldots, n_t$ **do**
4: $U_{kj} := L_{kk}^{-1} A_{kj}$
5: **end for** — **TRSM**
6: **for** $i = k+1, k+2, \ldots, n_t$ **do**
7: $L_{ik} := A_{ik} U_{kk}^{-1}$
8: **end for** — **TRSM**
9: **for** $i = k+1, k+2, \ldots, n_t$ **do**
10: **for** $j = k+1, k+2, \ldots, n_t$ **do**
11: $A_{ij} := A_{ij} - L_{ik} \cdot U_{kj}$
12: **end for**
13: **end for** — **GEMM**
14: **end for**

K = 1 K = 2 K = 3



$\mathcal{H}$-LU factorization

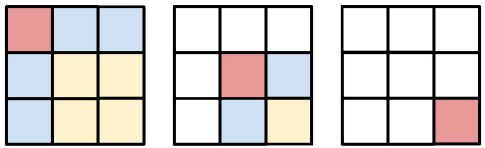
$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{pmatrix}$$

Algorithm 1 Blocked RL algorithm for the LU factorization.

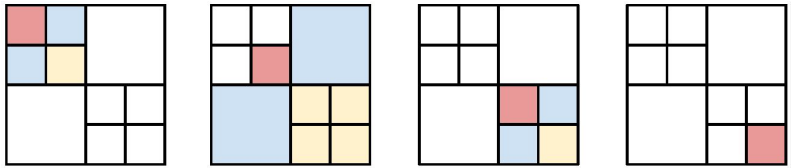
Require: $A \in \mathbb{R}^{n \times n}$

1: **for** $k = 1, 2, \ldots, n_t$ **do**
2: $A_{kk} = L_{kk} U_{kk}$ — **LU**
3: **for** $j = k+1, k+2, \ldots, n_t$ **do**
4: $U_{kj} := L_{kk}^{-1} A_{kj}$
5: **end for** — **TRSM**
6: **for** $i = k+1, k+2, \ldots, n_t$ **do**
7: $L_{ik} := A_{ik} U_{kk}^{-1}$
8: **end for** — **TRSM**
9: **for** $i = k+1, k+2, \ldots, n_t$ **do**
10: **for** $j = k+1, k+2, \ldots, n_t$ **do**
11: $A_{ij} := A_{ij} - L_{ik} \cdot U_{kj}$
12: **end for**
13: **end for** — **GEMM**
14: **end for**

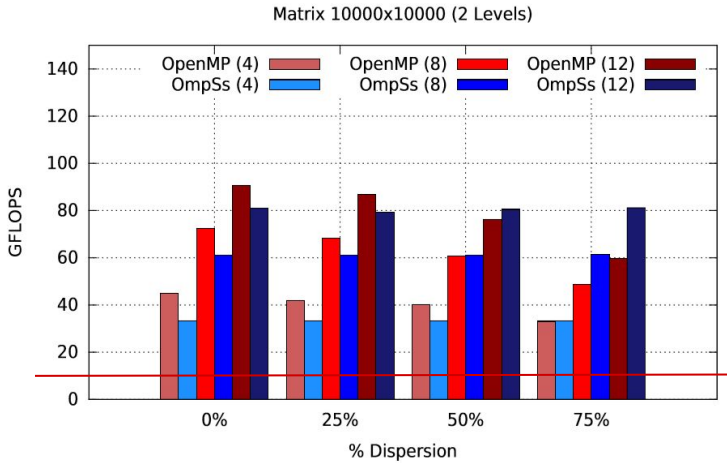
K = 1 K = 2 K = 3

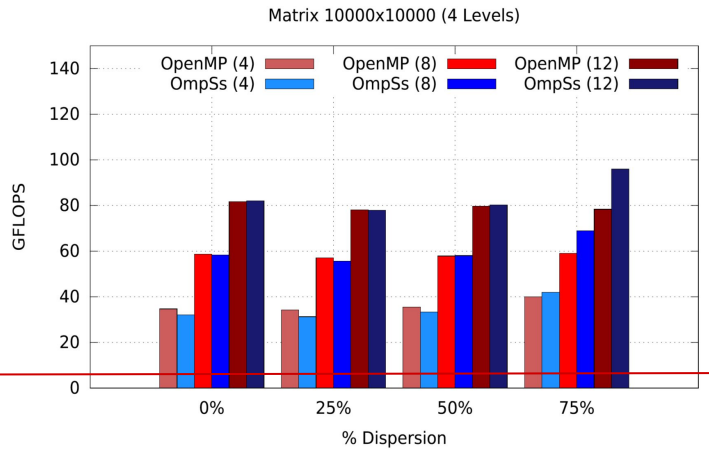


$\mathcal{H}$-Matrices



$\mathcal{H}$-LU factorization preliminar results [2]

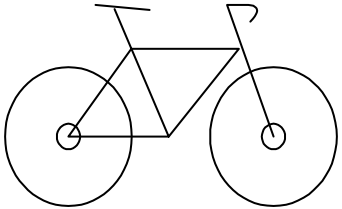




Sequential execution: 12 GFLOPS

[2] J. I. Aliaga, **R. Carratalá-Sáez**, R. Kriemann, E. S. Quintana-Ortí, *Task-parallel LU factorization of hierarchical matrices using OmpSs*, in: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 1148–1157. doi:10.1109/IPDPSW.2017.124.

“There's no need to reinvent the wheel”



“There's no need to reinvent the wheel”

http://www.h2lib.org
Prof. Dr. Steffen Börm
Christian-Albrechts-Universität zu Kiel

H2Lib

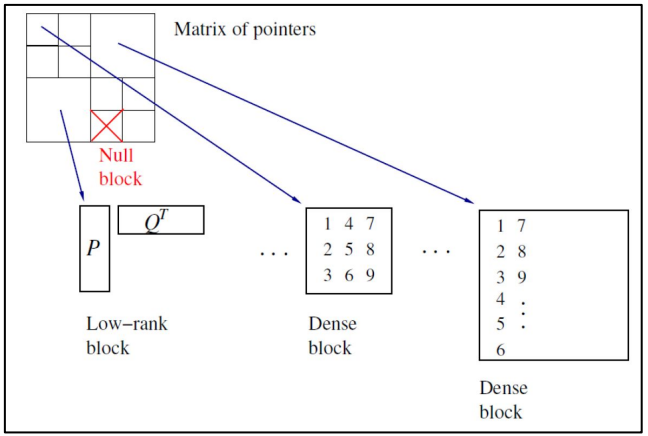
- Open Source (https://github.com/H2Lib)
- C language

H2Lib

- Open Source (https://github.com/H2Lib)
- C language
- $\mathcal{H}$-Matrices stored as a matrix of (nested) pointers

H2Lib

- Open Source (https://github.com/H2Lib)
- C language
- $\mathcal{H}$-Matrices stored as a matrix of (nested) pointers
  - 1st limitation: data location (not continuous in memory)



H2Lib

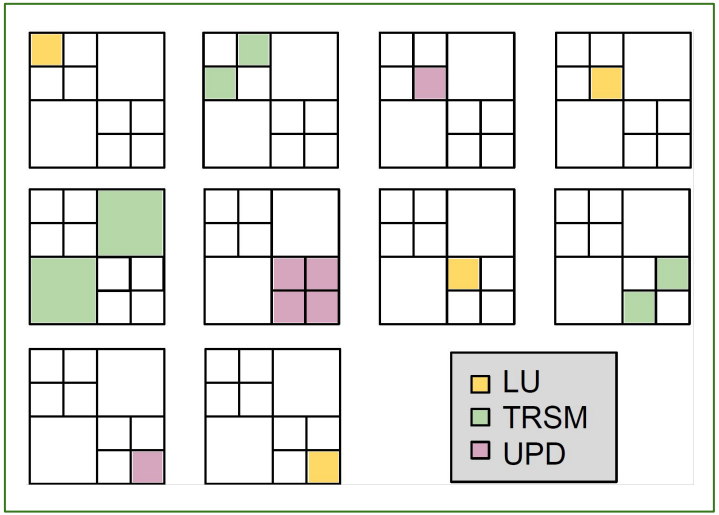
- Open Source (https://github.com/H2Lib)
- C language
- $\mathcal{H}$-Matrices stored as a matrix of (nested) pointers
  - 1st limitation: data location (not continuous in memory)
- Recursive algorithms

H2Lib

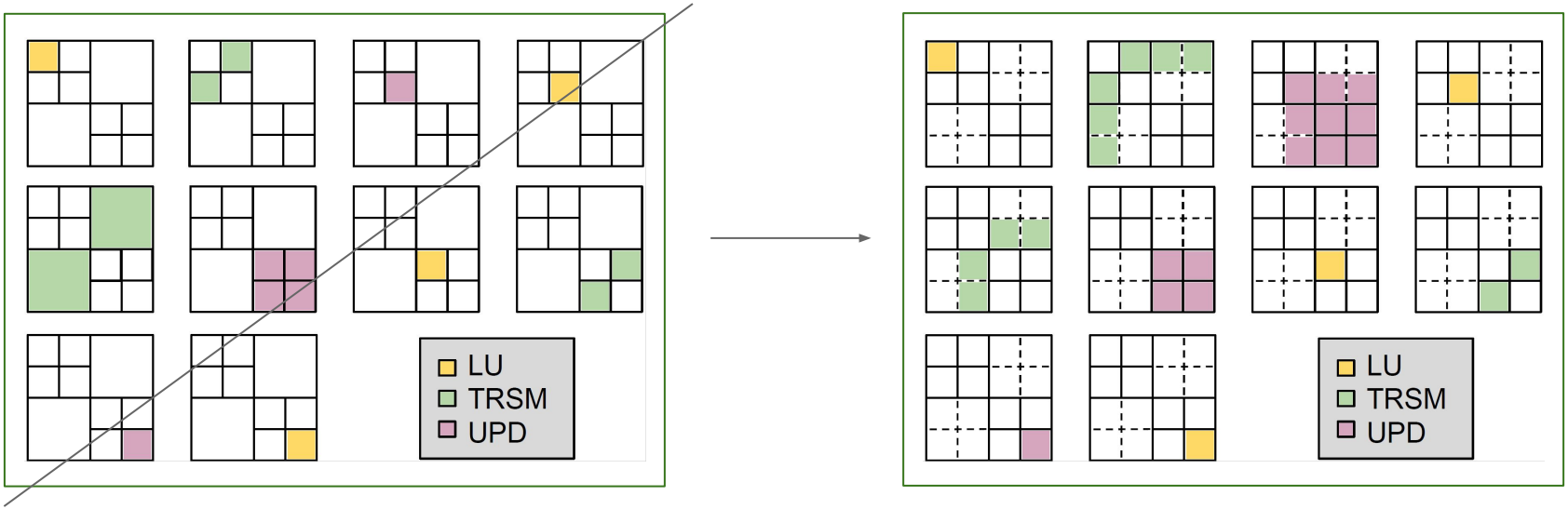
- Open Source (https://github.com/H2Lib)
- C language
- $\mathcal{H}$-Matrices stored as a matrix of (nested) p
  - 1st limitation: data location (not contir
- Recursive algorithms
  - 2nd limitation: nested dependencies

```
void RECURSIVE_LU( A )
{
     if ( isLeaf( A ) )
          LU( A );
     else
     {
          for ( i = 0 : n_row_sons )
          {
               RECURSIVE_LU( son_ii );
               for ( j = 0 : n_col_sons )
               {
                    TRSM( son_ij );
                    TRSM( son_ji );
               }
               for ( j = i+1 : n_row_sons )
                    for ( k = i+1 : n_col_sons )
                         UPD( son_jk );
          }
     }
}
```

Data location limitations



Data location limitations



Forces to partition every leaf block into blocks
that present the **SMALLEST** block size **INTERVENING** in the operation.

Recursion (nested dependencies) limitations

- H2Lib H-LU is **recursive**.
- **Nested** parallelism.

```
void RECURSIVE_LU( A )
{
     if ( isLeaf( A ) )
          LU( A );
     else
     {
          for ( i = 0 : n_row_sons )
          {
               RECURSIVE_LU( son_ii );
               for ( j = 0 : n_col_sons )
               {
                    TRSM( son_ij );
                    TRSM( son_ji );
               }
               for ( j = i+1 : n_row_sons )
                    for ( k = i+1 : n_col_sons )
                         UPD( son_jk );
          }
     }
}
```

Recursion (nested dependencies) limitations

- H2Lib H-LU is **recursive**.
- **Nested** parallelism.
- **Taskwait** is necessary.
- Parallelism is **limited**.

```
void RECURSIVE_LU( A )
{
     if ( isLeaf( A ) )
          LU( A );
     else
     {
          for ( i = 0 : n_row_sons )
          {
               RECURSIVE_LU( son_ii );
               for ( j = 0 : n_col_sons )
               {
                    TRSM( son_ij );
                    TRSM( son_ji );
               }
               for ( j = i+1 : n_row_sons )
                    for ( k = i+1 : n_col_sons )
                         UPD( son_jk );
          }
     }
}
```

TASKWAIT

The OmpSs * Programming Model & Runtime

- Task-based programming model
- Data-dependences among tasks expressed with clauses
- Discovers the data-flow parallelism at execution time

(*) https://pm.bsc.es/ompss

The OmpSs * Programming Model & Runtime

- Task-based programming model
- Data-dependences among tasks expressed with clauses
- Discovers the data-flow parallelism at execution time
- OmpSs-2 & Nanos6: new features
  - Nested regions dependencies
  - Early release
  - Weak dependencies

(*) https://pm.bsc.es/ompss

OmpSs-2 & Nanos6 novelties: nested dependencies

- Solved OmpSs overlapping PO and regions troubles

- H2Lib limitation: data location
  - Without nested dependencies: partition needed to smallest block size in the matrix.

OmpSs-2 & Nanos6 novelties: weak dependencies & early release

- H2Lib limitation: recursion
- With weak deps & early release:
  - No taskwait needed.
  - Anticipation of children tasks.

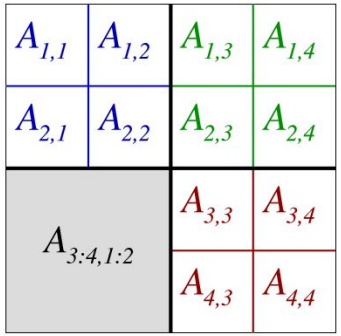
```
#pragma oss task inout(A)
void RECURSIVE_LU( A )
{
     if ( isLeaf( A ) )
          #pragma oss task inout(A)
          LU( A );
     else
     {
          for ( k = 0 : n_row_sons )
          {
               RECURSIVE_LU( son_kk );
               for ( j = 0 : n_col_sons )
               {
                    #pragma oss task in(A) inout(son_kj)
                    TRSM( son_kj );
                    #pragma oss task in(A) inout(son_jk)
                    TRSM( son_jk );
               }
               for ( j = k+1 : n_row_sons )
                    for ( i = k+1 : n_col_sons )
                         #pragma oss task in(son_ik, son_kj) inout(son_ij)
                         UPD( son_jk );
          }
     }
     #pragma oss taskwait
}
```

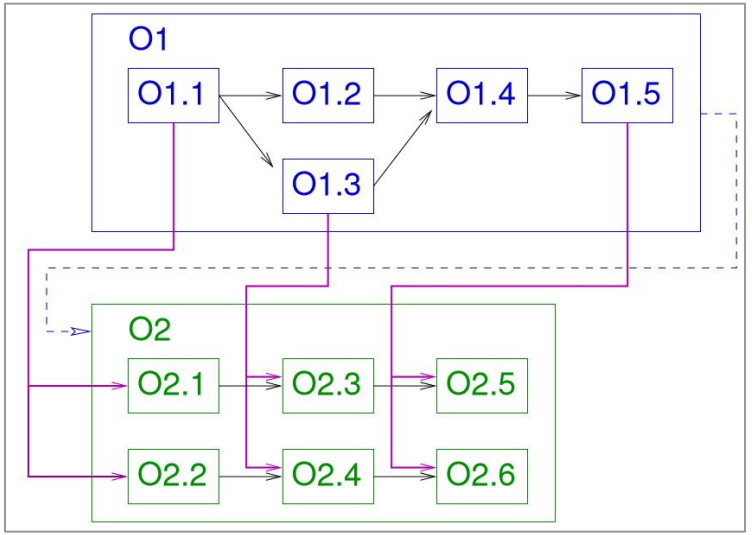
OmpSs-2 & Nanos6 novelties: weak dependencies & early release

- H2Lib limitation: recursion
- With weak deps & early release:
  - No taskwait needed.
  - Anticipation of children tasks.

```
#pragma oss task weakinout(A)
```

```
#pragma oss task inout(A)
void RECURSIVE_LU( A )
{
     if ( isLeaf( A ) )
          #pragma oss task inout(A)
          LU( A );
     else
     {
          for ( k = 0 : n_row_sons )
          {
               RECURSIVE_LU( son_kk );
               for ( j = 0 : n_col_sons )
               {
                    #pragma oss task in(A) inout(son_kj)
                    TRSM( son_kj );
                    #pragma oss task in(A) inout(son_jk)
                    TRSM( son_jk );
               }
               for ( j = k+1 : n_row_sons )
                    for ( i = k+1 : n_col_sons )
                         #pragma oss task in(son_ik, son_kj) inout(son_ij)
                         UPD( son_jk );
          }
     }
     #pragma oss taskwait
}
```





Sequence of operations for the $\mathcal{H}$-LU factorization of A:

O1.1 :	$A_{1,1}$	$=$	$L_{1,1}U_{1,1}$
O1.2 :	$U_{1,2}$	$:=$	$L_{1,1}^{-1}A_{1,2}$
O1.3 :	$L_{2,1}$	$:=$	$A_{2,1}U_{1,1}^{-1}$
O1.4 :	$A_{2,2}$	$:=$	$A_{2,2} - L_{2,1} \cdot U_{1,2}$
O1.5 :	$A_{2,2}$	$=$	$L_{2,2}U_{2,2}$
O2 :	$U_{1:2,3:4}$	$:=$	$L_{1:2,1:2}^{-1}A_{1:2,3:4}$
O3 :	$L_{3:4,1:2}$	$:=$	$A_{3:4,1:2}U_{1:2,1:2}^{-1}$
O4 :	$A_{3:4,3:4}$	$:=$	$A_{3:4,3:4} - L_{3:4,1:2} \cdot U_{1:2,3:4}$
O5.1 :	$A_{3,3}$	$=$	$L_{3,3}U_{3,3}$
O5.2 :	$U_{3,4}$	$:=$	$L_{3,3}^{-1}A_{3,4}$
O5.3 :	$L_{4,3}$	$:=$	$A_{4,3}U_{3,3}^{-1}$
O5.4 :	$A_{4,4}$	$:=$	$A_{4,4} - L_{4,3} \cdot U_{3,4}$
O5.5 :	$A_{4,4}$	$=$	$L_{4,4}U_{4,4}$

O2.1 : $U_{1,3} := L_{1,1}^{-1}A_{1,3}$,

O2.2 : $U_{1,4} := L_{1,1}^{-1}A_{1,4}$,

O2.3 : $A_{2,3} := A_{2,3} - L_{21} \cdot U_{1,3}$,

O2.4 : $A_{2,4} := A_{2,4} - L_{21} \cdot U_{1,4}$

O2.5 : $U_{2,3} := L_{2,2}^{-1}A_{2,3}$, and

O2.6 : $U_{2,4} := L_{2,2}^{-1}A_{2,4}$.

Tests & results

- Integral equations, BEM
- Underlying kernel function: Laplace equation

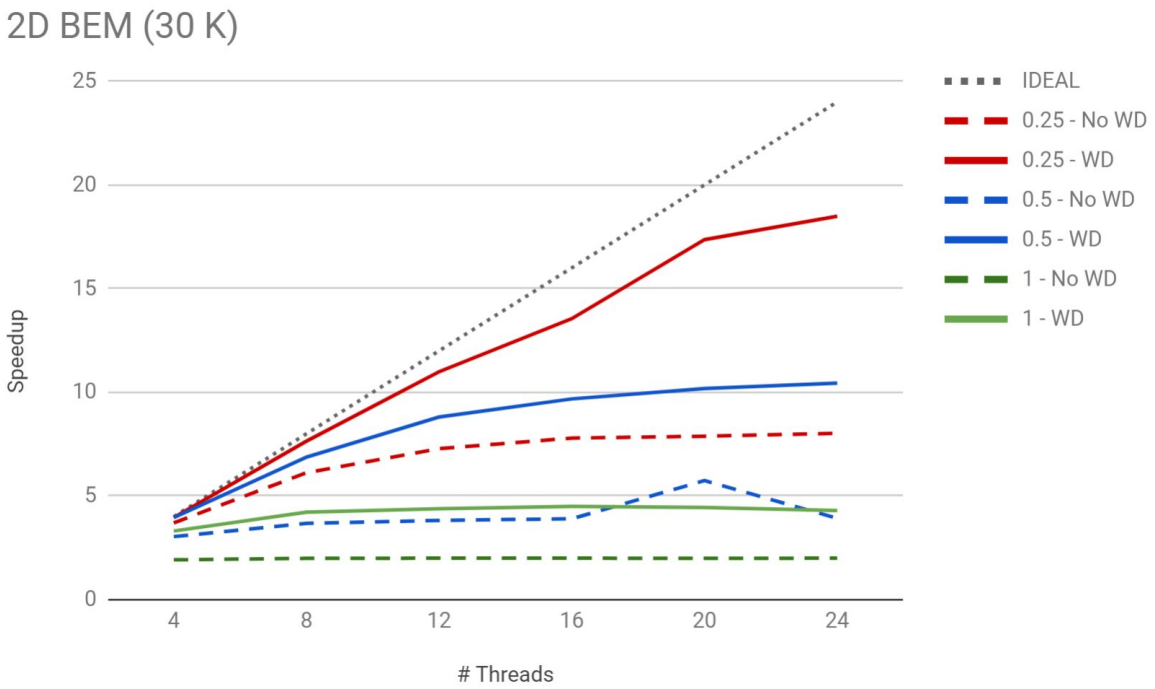
$$g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \quad g(x,y) = \begin{cases} -\log|x-y| & : d = 1, \\ -\frac{1}{2\pi}\log\|x-y\|_2 & : d = 2, \\ \frac{1}{4\pi}\|x-y\|_2^{-1} & : d = 3. \end{cases}$$

- Compression with SVD.
- Admissibility condition $\max\{\mathrm{diam}(\mathcal{B}_t), \mathrm{diam}(\mathcal{B}_s)\} \leq \eta\, \mathrm{dist}(\mathcal{B}_t, \mathcal{B}_s)$ *where* $\eta \in \mathbb{R}_{>0}$

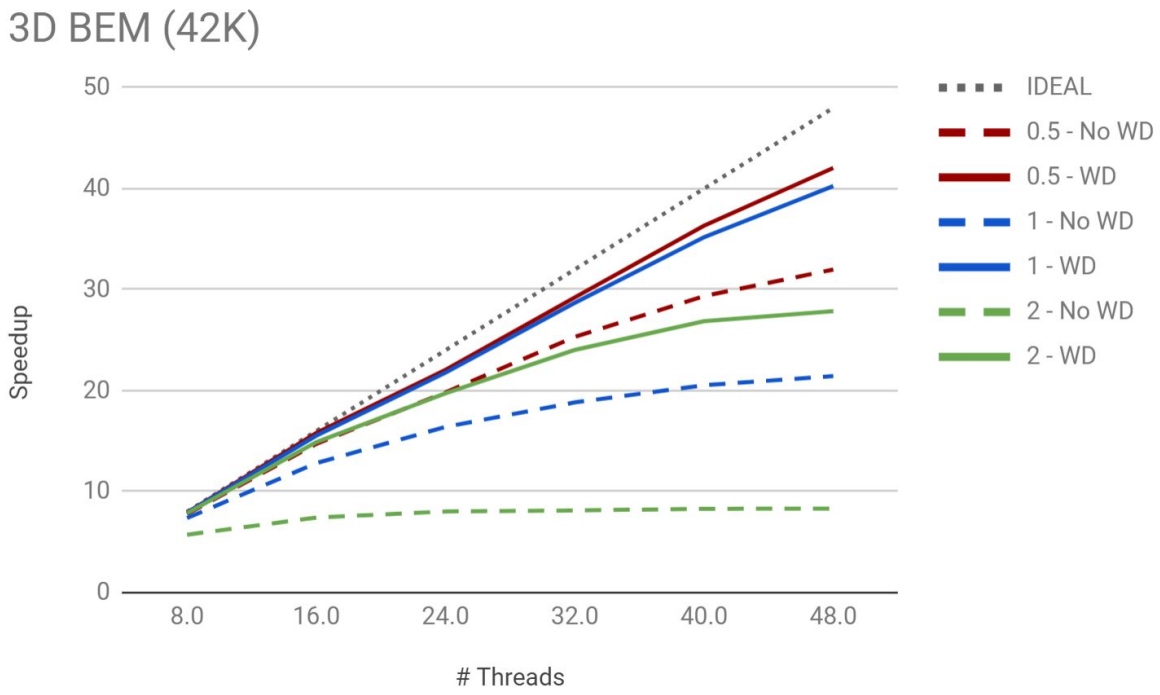
SETUP:

- **IEEE 754 double precision arithmetic.**
- **Single node of the MareNostrum 4 system at BSC, with 2 Intel Xeon Platinum 8160 sockets, 24 cores per socket, 96 Gbytes of DDR4 RAM.**
- **GCC 4.8.5, Intel MKL 2017.4 (AVX2 instructions enabled), OmpSs-2 (mcxx 2.1.0).**

Tests & results



Tests & results



Conclusions

From **previous** [2] work...

- **Task-based parallelism** seems to offer good performance for H-structures.

From **using OmpSs 2** [3] on H2Lib H-LU...

- Good parallel performance with OmpSs 2 **task-based parallelism**.
- **New features (weak dependencies & early release)** avoid detected limitations.

[2] R. Carratalá-Sáez et al. *Task-parallel LU factorization of hierarchical matrices using OmpSs.* Proceedings of the 19th IEEE Workshop on Parallel and Distributed Scientific and Engineering Computing, PDSEC 2017, 2017.

[3] R. Carratalá-Sáez et al. *Exploiting Nested Task-Parallelism in the H-LU Factorization.* Journal of Computational Science (Elsevier), February 2018. Accepted, pending of publication.

Future work

- Related to H2Lib:
  - Maybe go deeper in parallelism.
  - Study how to improve parallel efficiency in 2D.
  - More test cases (different applications).
  - Other operations.

Future work

- Related to H2Lib:
  - Maybe go deeper in parallelism.
  - Study how to improve parallel efficiency in 2D.
  - More test cases (different applications).
  - Other operations.

- Implement distributed memory H-LU.
- Chameleon extension (low-rank support).

Inria

Exploiting Nested Task-based Parallelism in the Factorization of Hierarchical Matrices

SIAM Conference on Computational Science and Engineering
February 25-March 1, 2019
Spokane Convention Center
Spokane, Washington, USA

Thanks for your attention!

UNIVERSITAT JAUME I

C A U
Christian-Albrechts-Universität zu Kiel

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

You are more than welcome to contact us:

Rocío Carratalá-Sáez	**rcarrata@uji.es**
Enrique S. Quintana-Ortí	quintana@uji.es
Steffen Börm	boerm@math.uni-kiel.de
Sven Christophersen	christophersen@math.uni-kiel.de
Vicenç Beltrán	vbeltran@bsc.es