Algorithmic aspects of hierarchical matrices

Steffen Börm, joint work with Sven Christophersen

Christian-Albrechts-Universität zu Kiel

Bergen-Kiel Workshop, 2016

Overview

1. Introduction
2. Algorithms
3. Parallelization

Overview

1. Introduction
2. Algorithms
3. Parallelization

Goal

Dense matrices are required by important applications, e.g.,

- when discretizing integral equations,
- when solving partial differential equations, or
- when approximating matrix functions like $\exp(A)$.

Problem: Standard representation requires n^2 units of storage.

Goal

Dense matrices are required by important applications, e.g.,

- when discretizing integral equations,
- when solving partial differential equations, or
- when approximating matrix functions like $\exp(A)$.

Problem: Standard representation requires n^2 units of storage.

Approach: Use data-sparse approximation of G.

- Storage $\mathcal{O}(nk \log(n))$,
- matrix-vector product $\mathcal{O}(nk \log(n))$,
- approximated matrix-matrix product $\mathcal{O}(nk^2 \log^2(n))$,
- approximated inverse $\mathcal{O}(nk^2 \log^2(n))$.

Goal

Dense matrices are required by important applications, e.g.,

- when discretizing integral equations,
- when solving partial differential equations, or
- when approximating matrix functions like $\exp(A)$.

Problem: Standard representation requires n^2 units of storage.

Approach: Use data-sparse approximation of G.

- Storage $\mathcal{O}(nk \log(n))$,
- matrix-vector product $\mathcal{O}(nk \log(n))$,
- approximated matrix-matrix product $\mathcal{O}(nk^2 \log^2(n))$,
- approximated inverse $\mathcal{O}(nk^2 \log^2(n))$.

Challenge: Still very time-consuming, has to be parallelized.

Model problem

Integral operator approximated by Nystrøm's method leads to matrix $G \in \mathbb{R}^{n \times n}$ with

$$g_{ij} = \gamma(x_i, x_j), \qquad \gamma : [0,1] \times [0,1] \to \mathbb{R} \qquad \text{for all } i, j \in [1:n],$$

where $x_1, \ldots, x_n \in [0,1]$.

Challenge: Dense matrix, have to store n^2 coefficients.

Low-rank approximation

Idea: Given intervals $t, s \subseteq [0,1]$ with $\operatorname{diam}(t) \leq \operatorname{dist}(t,s)$, we have

$$\gamma(x,y) \approx \sum_{\nu=0}^{k-1} \frac{(x-x_t)^\nu}{\nu!} \frac{\partial^\nu \gamma}{\partial x^\nu}(x_t,y) \text{ for all } x \in t,\ y \in s.$$

Low-rank approximation

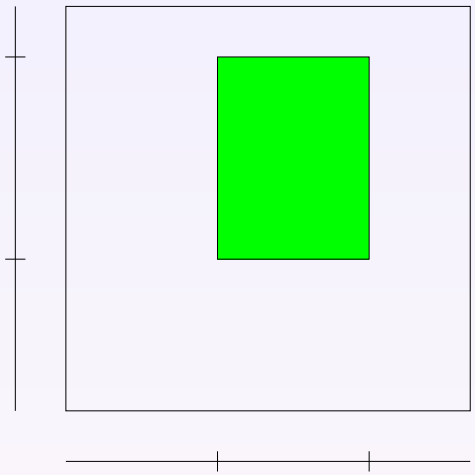
Idea: Given intervals $t, s \subseteq [0,1]$ with $\operatorname{diam}(t) \leq \operatorname{dist}(t,s)$, we have

$$\gamma(x,y) \approx \sum_{\nu=0}^{k-1} \frac{(x-x_t)^\nu}{\nu!} \frac{\partial^\nu \gamma}{\partial x^\nu}(x_t,y) \text{ for all } x \in t,\ y \in s.$$

Low-rank factorization given by

$$G|_{\hat{t}\times\hat{s}} \approx A_{ts} B_{ts}^*$$

with $\hat{t} := \{i \in [1:n] \ : \ x_i \in t\}$ and



$$a_{ts,i\nu} := \frac{(x_i - x_t)^\nu}{\nu!}, \qquad b_{ts,j\nu} := \frac{\partial^{\nu+\mu}\gamma}{\partial x^\nu \partial y^\mu}(x_t, y_j).$$

Low-rank approximation

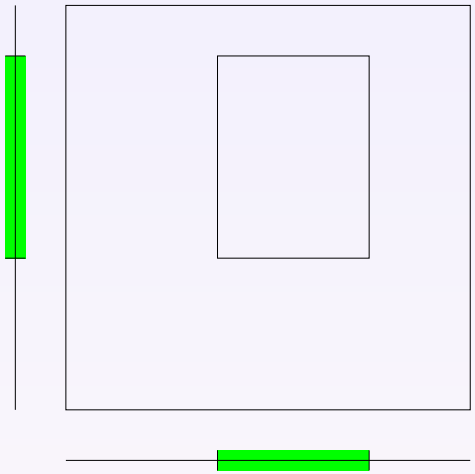
Idea: Given intervals $t, s \subseteq [0,1]$ with $\operatorname{diam}(t) \leq \operatorname{dist}(t,s)$, we have

$$\gamma(x,y) \approx \sum_{\nu=0}^{k-1} \frac{(x-x_t)^\nu}{\nu!} \frac{\partial^\nu \gamma}{\partial x^\nu}(x_t,y) \text{ for all } x \in t,\ y \in s.$$

Low-rank factorization given by

$$G|_{\hat{t}\times\hat{s}} \approx A_{ts} B_{ts}^*$$

with $\hat{t} := \{i \in [1:n] \ : \ x_i \in t\}$ and



$$a_{ts,i\nu} := \frac{(x_i - x_t)^\nu}{\nu!}, \qquad b_{ts,j\nu} := \frac{\partial^{\nu+\mu}\gamma}{\partial x^\nu \partial y^\mu}(x_t, y_j).$$

Low-rank approximation

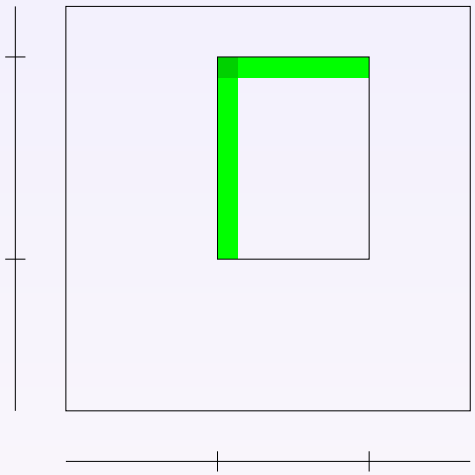
Idea: Given intervals $t, s \subseteq [0,1]$ with $\operatorname{diam}(t) \leq \operatorname{dist}(t,s)$, we have

$$\gamma(x,y) \approx \sum_{\nu=0}^{k-1} \frac{(x-x_t)^\nu}{\nu!} \frac{\partial^\nu \gamma}{\partial x^\nu}(x_t,y) \text{ for all } x \in t,\ y \in s.$$

Low-rank factorization given by

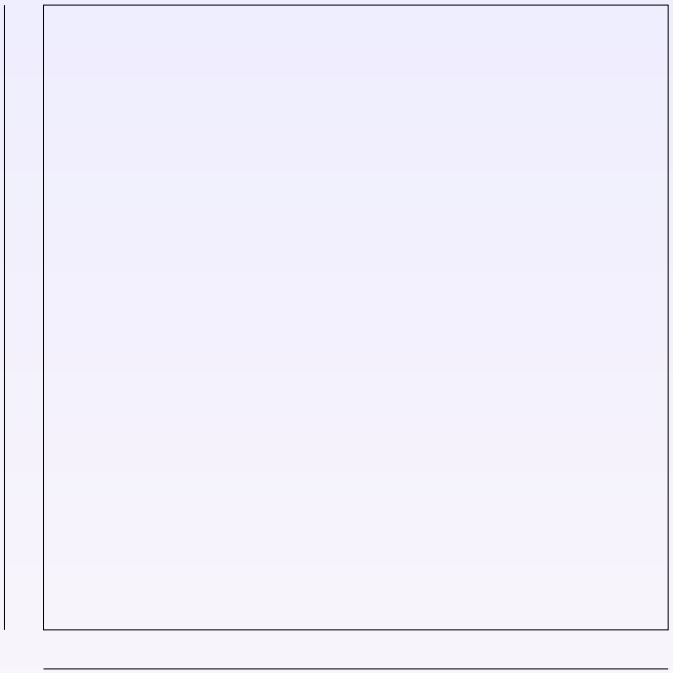
$$G|_{\hat{t}\times\hat{s}} \approx A_{ts} B_{ts}^*$$

with $\hat{t} := \{i \in [1:n] \ : \ x_i \in t\}$ and



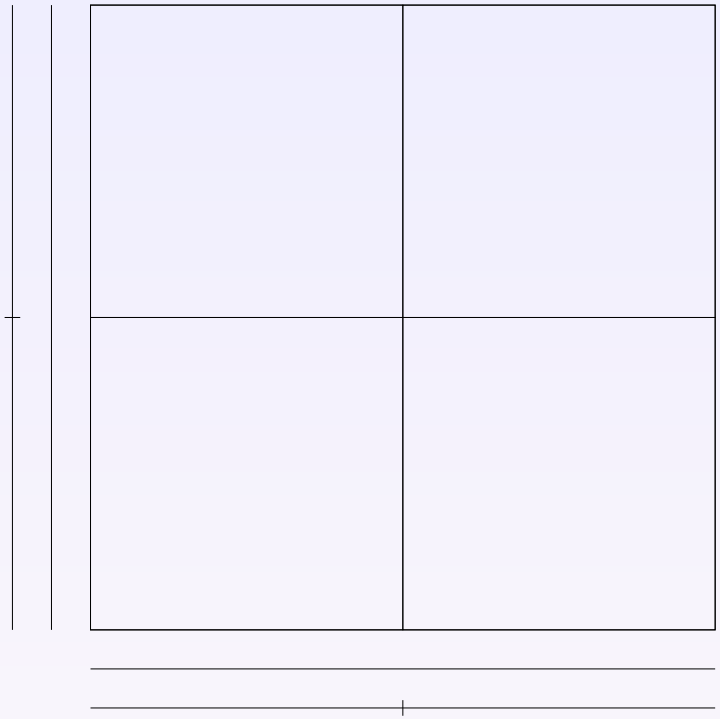
$$a_{ts,i\nu} := \frac{(x_i - x_t)^\nu}{\nu!}, \qquad b_{ts,j\nu} := \frac{\partial^{\nu+\mu}\gamma}{\partial x^\nu \partial y^\mu}(x_t, y_j).$$

Hierarchical matrix



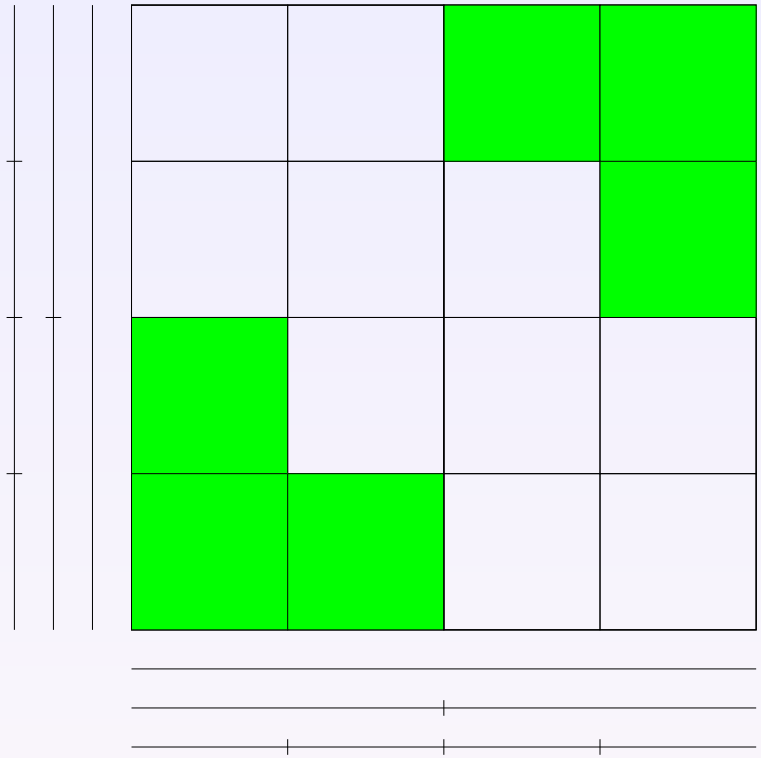
Goal: Split matrix into submatrices that can be approximated

Hierarchical matrix



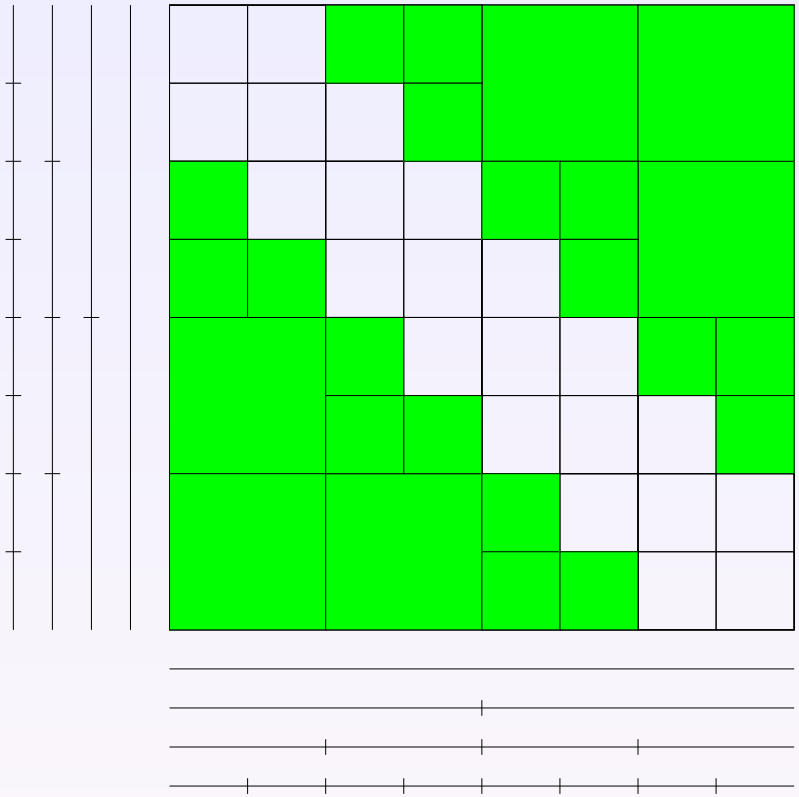
Goal: Split matrix into submatrices that can be approximated

Hierarchical matrix



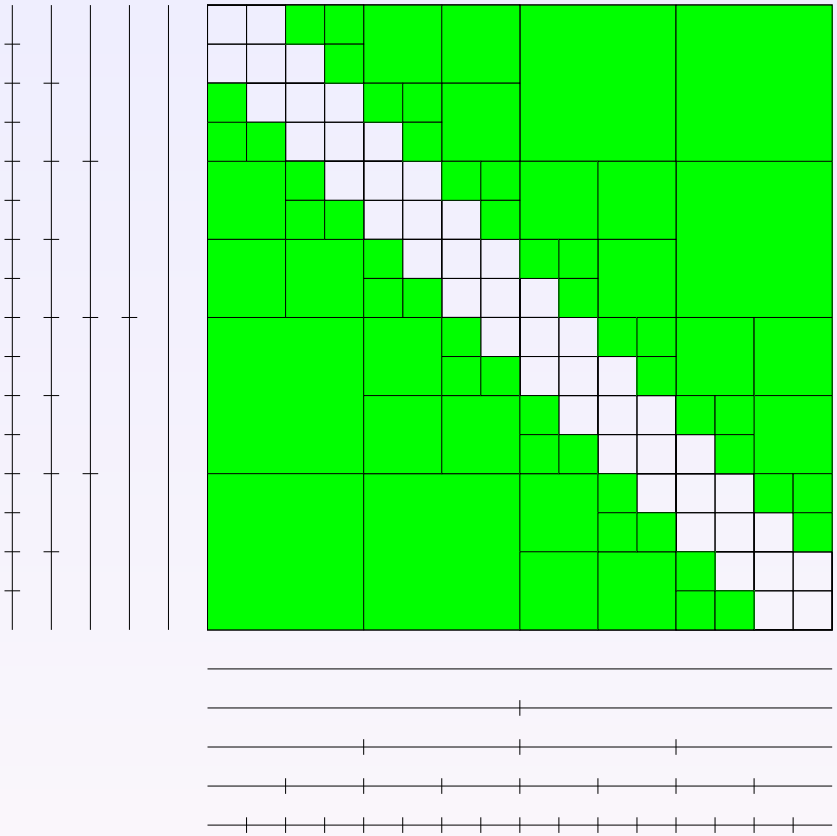
Goal: Split matrix into submatrices that can be approximated

Hierarchical matrix



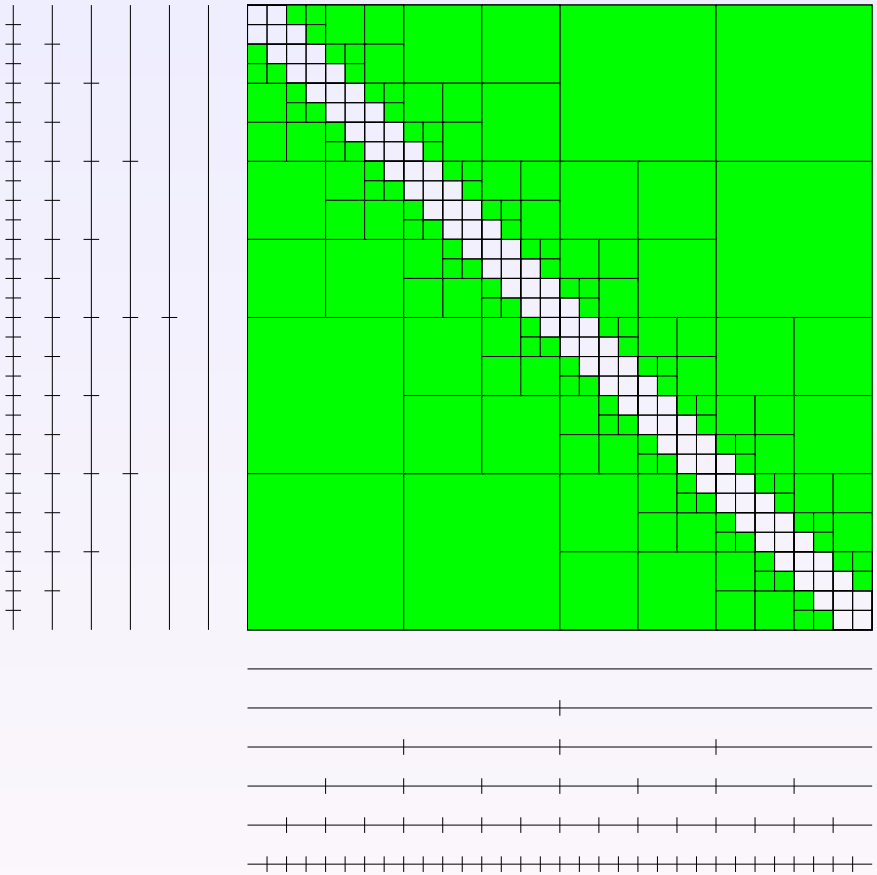
Goal: Split matrix into submatrices that can be approximated

Hierarchical matrix



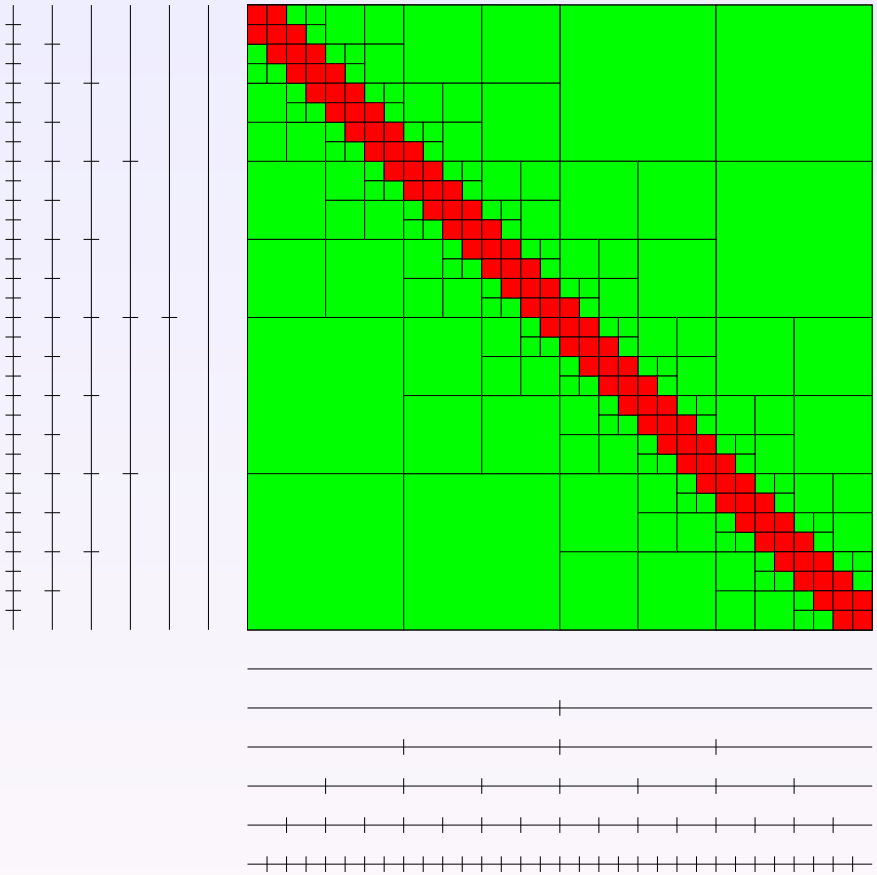
Goal: Split matrix into submatrices that can be approximated

Hierarchical matrix



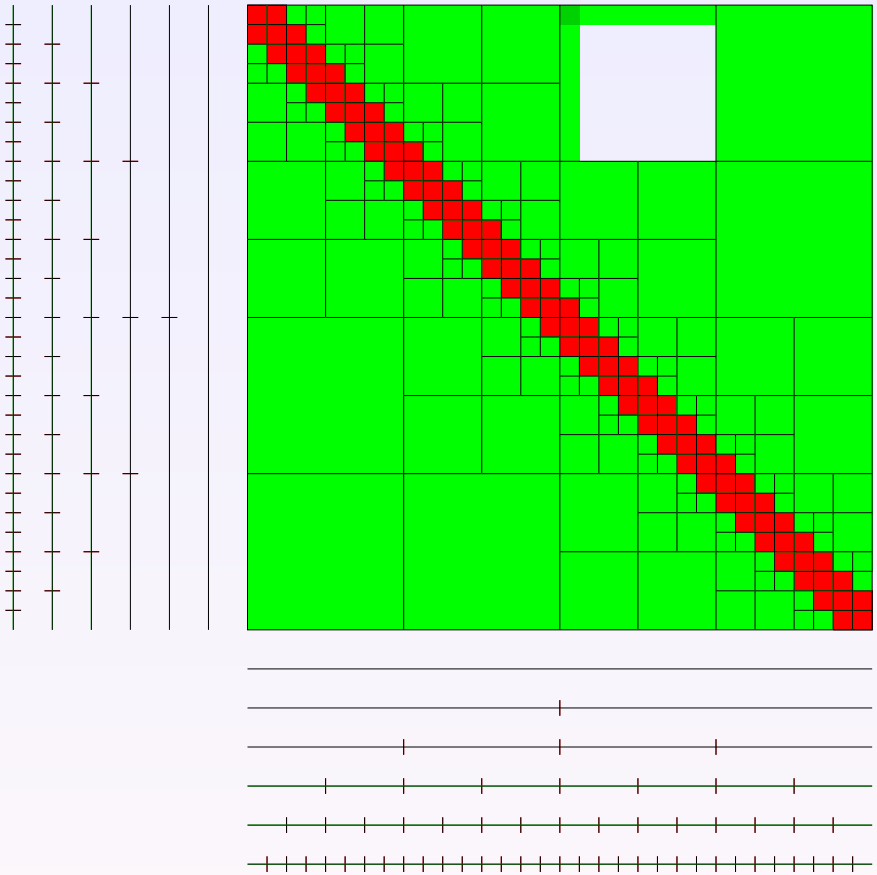
Goal: Split matrix into submatrices that can be approximated

Hierarchical matrix



Goal: Split matrix into submatrices that can be approximated and a sparse remainder matrix.

Hierarchical matrix

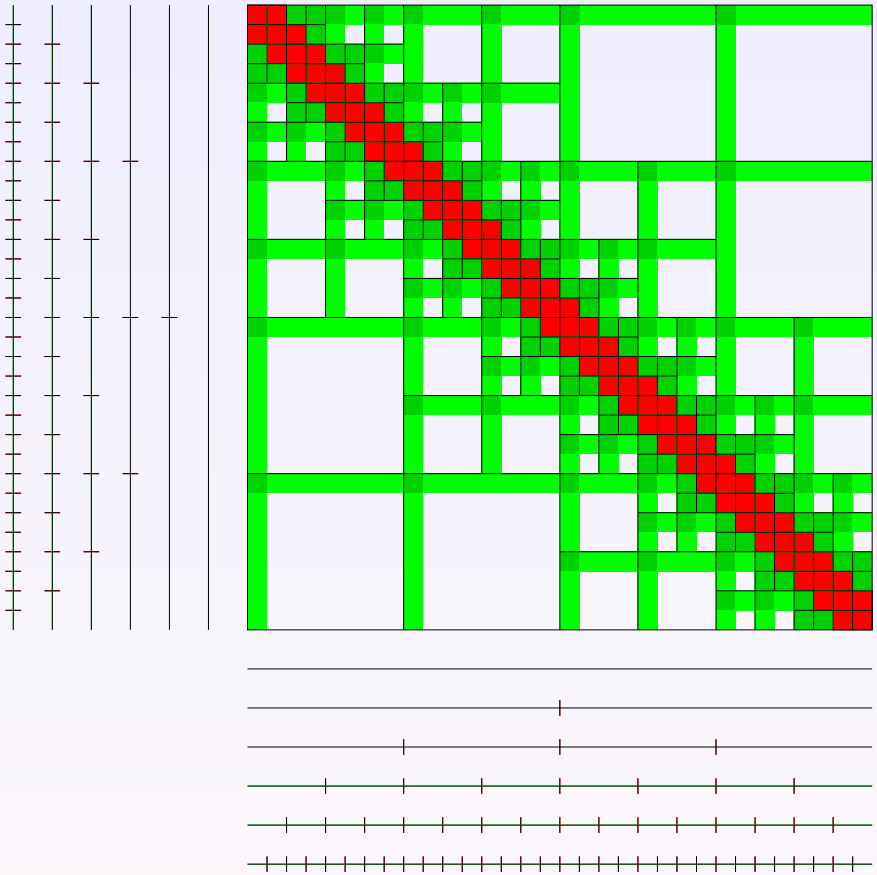


Goal: Split matrix into submatrices that can be approximated and a sparse remainder matrix.

Approximate submatrices by factorization

$$G|_{\hat{t}\times\hat{s}} \approx A_{ts} B_{ts}^*.$$

Hierarchical matrix



Goal: Split matrix into submatrices that can be approximated and a sparse remainder matrix.

Approximate submatrices by factorization

$$G|_{\hat{t}\times\hat{s}} \approx A_{ts} B_{ts}^*.$$

Result: Storage $\mathcal{O}(nk\log(n))$.

Overview

1 Introduction

2 Algorithms

3 Parallelization

Fundamental operations

Matrix-vector multiplication: Split into two products

$$y = A_{ts} B_{ts}^* x \qquad \Longleftrightarrow \qquad z = B_{ts}^* x \wedge y = A_{ts} z.$$

Complexity $\mathcal{O}(k(\#\hat{t} + \#\hat{s}))$ for one submatrix, $\mathcal{O}(nk \log(n))$ for entire hierarchical matrix.

Fundamental operations

Matrix-vector multiplication: Split into two products

$$y = A_{ts} B_{ts}^* x \quad \Longleftrightarrow \quad z = B_{ts}^* x \wedge y = A_{ts} z.$$

Complexity $\mathcal{O}(k(\#\hat{t} + \#\hat{s}))$ for one submatrix, $\mathcal{O}(nk \log(n))$ for entire hierarchical matrix.

Truncated update: Best approximation of

$$A_{ts} B_{ts}^* + C_{ts} D_{ts}^* = \begin{pmatrix} A_{ts} & C_{ts} \end{pmatrix} \begin{pmatrix} B_{ts} & D_{ts} \end{pmatrix}^*$$

can be found in $\mathcal{O}(k^2(\#\hat{t} + \#\hat{s}))$ for one submatrix, $\mathcal{O}(nk^2 \log(n))$ for entire hierarchical matrix.

Matrix multiplication

Goal: Compute update

$$Z|_{\hat{t}\times\hat{r}} \leftarrow Z|_{\hat{t}\times\hat{r}} + \alpha X|_{\hat{t}\times\hat{s}} Y|_{\hat{s}\times\hat{r}}$$

Recursion can be used as long as $X|_{\hat{t}\times\hat{s}}$ and $Y|_{\hat{s}\times\hat{r}}$ are subdivided.

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} X_{11}Y_{11} + X_{12}Y_{21} & \\ & \end{pmatrix}$$

Matrix multiplication

Goal: Compute update

$$Z|_{\hat{t}\times\hat{r}} \leftarrow Z|_{\hat{t}\times\hat{r}} + \alpha X|_{\hat{t}\times\hat{s}} Y|_{\hat{s}\times\hat{r}}$$

Recursion can be used as long as $X|_{\hat{t}\times\hat{s}}$ and $Y|_{\hat{s}\times\hat{r}}$ are subdivided.

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ & \end{pmatrix}$$

Matrix multiplication

Goal: Compute update

$$Z|_{\hat{t}\times\hat{r}} \leftarrow Z|_{\hat{t}\times\hat{r}} + \alpha X|_{\hat{t}\times\hat{s}} Y|_{\hat{s}\times\hat{r}}$$

Recursion can be used as long as $X|_{\hat{t}\times\hat{s}}$ and $Y|_{\hat{s}\times\hat{r}}$ are subdivided.

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & \end{pmatrix}$$

Matrix multiplication

Goal: Compute update

$$Z|_{\hat{t}\times\hat{r}} \leftarrow Z|_{\hat{t}\times\hat{r}} + \alpha X|_{\hat{t}\times\hat{s}} Y|_{\hat{s}\times\hat{r}}$$

Recursion can be used as long as $X|_{\hat{t}\times\hat{s}}$ and $Y|_{\hat{s}\times\hat{r}}$ are subdivided.

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{pmatrix}$$

Matrix multiplication

Goal: Compute update

$$Z|_{\hat{t}\times\hat{r}} \leftarrow Z|_{\hat{t}\times\hat{r}} + \alpha X|_{\hat{t}\times\hat{s}} Y|_{\hat{s}\times\hat{r}}$$

Recursion can be used as long as $X|_{\hat{t}\times\hat{s}}$ and $Y|_{\hat{s}\times\hat{r}}$ are subdivided.

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{pmatrix}$$

Factorization is crucial if one factor has low rank.

$$Z|_{\hat{t}\times\hat{r}} \leftarrow Z|_{\hat{t}\times\hat{r}} + \alpha\ X|_{\hat{t}\times\hat{s}} A_{sr}\ B_{sr}^*.$$

Matrix multiplication

Goal: Compute update

$$Z|_{\hat{t}\times\hat{r}} \leftarrow Z|_{\hat{t}\times\hat{r}} + \alpha X|_{\hat{t}\times\hat{s}} Y|_{\hat{s}\times\hat{r}}$$

Recursion can be used as long as $X|_{\hat{t}\times\hat{s}}$ and $Y|_{\hat{s}\times\hat{r}}$ are subdivided.

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{pmatrix}$$

Factorization is crucial if one factor has low rank.

$$Z|_{\hat{t}\times\hat{r}} \leftarrow Z|_{\hat{t}\times\hat{r}} + \alpha(X|_{\hat{t}\times\hat{s}} A_{sr}) B_{sr}^*.$$

Reduces to k matrix-vector multiplications and a low-rank update.

Complexity $\mathcal{O}(k^2(\#\hat{t} + \#\hat{s} + \#\hat{r})\log(n))$ for one update,
$\mathcal{O}(nk^2\log^2(n))$ for entire multiplication.

LU factorization

Goal: Find $G|_{\hat{t}\times\hat{t}} = LU$ with lower triangular L and upper triangular U.

Approach: Block representation

$$\begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix}$$

yields recursive algorithm

$$G_{11} = L_{11} U_{11},$$

LU factorization

Goal: Find $G|_{\hat{t}\times\hat{t}} = LU$ with lower triangular L and upper triangular U.

Approach: Block representation

$$\begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix}$$

yields recursive algorithm

$$G_{11} = L_{11} U_{11},$$
$$G_{12} = L_{11} U_{12}, \qquad G_{21} = L_{21} U_{11},$$

LU factorization

Goal: Find $G|_{\hat{t}\times\hat{t}} = LU$ with lower triangular L and upper triangular U.

Approach: Block representation

$$\begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix}$$

yields recursive algorithm

$$G_{11} = L_{11} U_{11},$$
$$G_{12} = L_{11} U_{12}, \qquad G_{21} = L_{21} U_{11},$$
$$G_{22} - L_{21} U_{12} = L_{22} U_{22}.$$

LU factorization

Goal: Find $G|_{\hat{t}\times\hat{t}} = LU$ with lower triangular L and upper triangular U.

Approach: Block representation

$$\begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix}$$

yields recursive algorithm

$$\begin{aligned} G_{11} &= L_{11} U_{11}, \\ G_{12} = L_{11} U_{12}, &\qquad G_{21} = L_{21} U_{11}, \\ G_{22} - L_{21} U_{12} &= L_{22} U_{22}. \end{aligned}$$

Similar: Inversion, Cholesky factorization.

Overview

1. Introduction
2. Algorithms
3. Parallelization

Task graph

Goal: Parallelize matrix operations like the LU factorization.

Our approach: Construct a task graph representing the dependencies between operations.

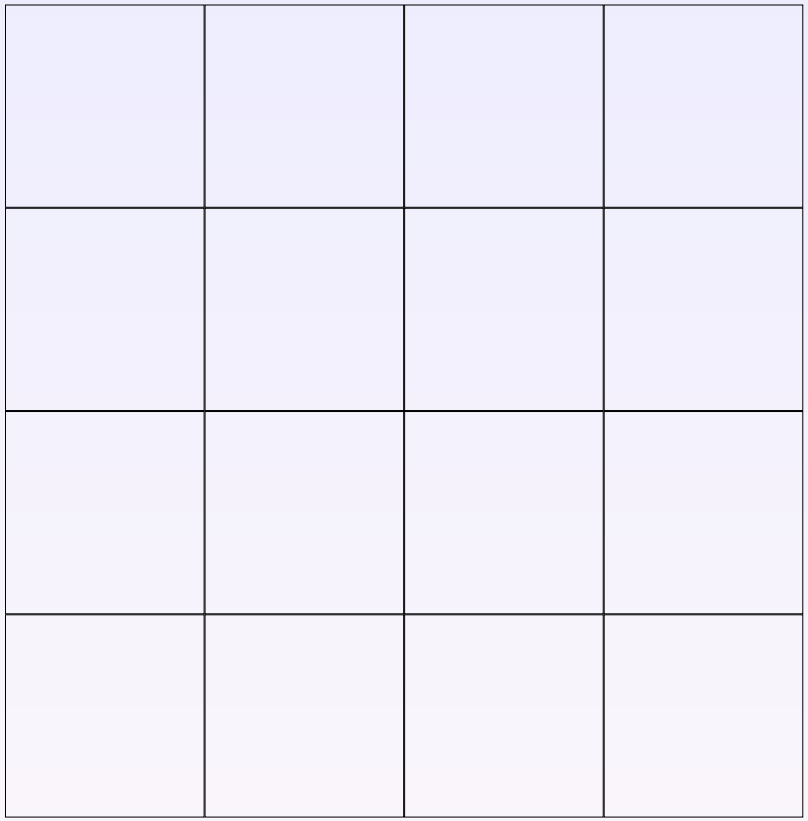
$$G_{11} \xrightarrow{\text{decomp}} L_{11}, U_{11}$$

$$G_{12}, L_{11} \xrightarrow{\text{lsolve}} U_{12}, \qquad G_{21}, R_{11} \xrightarrow{\text{usolve}} L_{21}$$

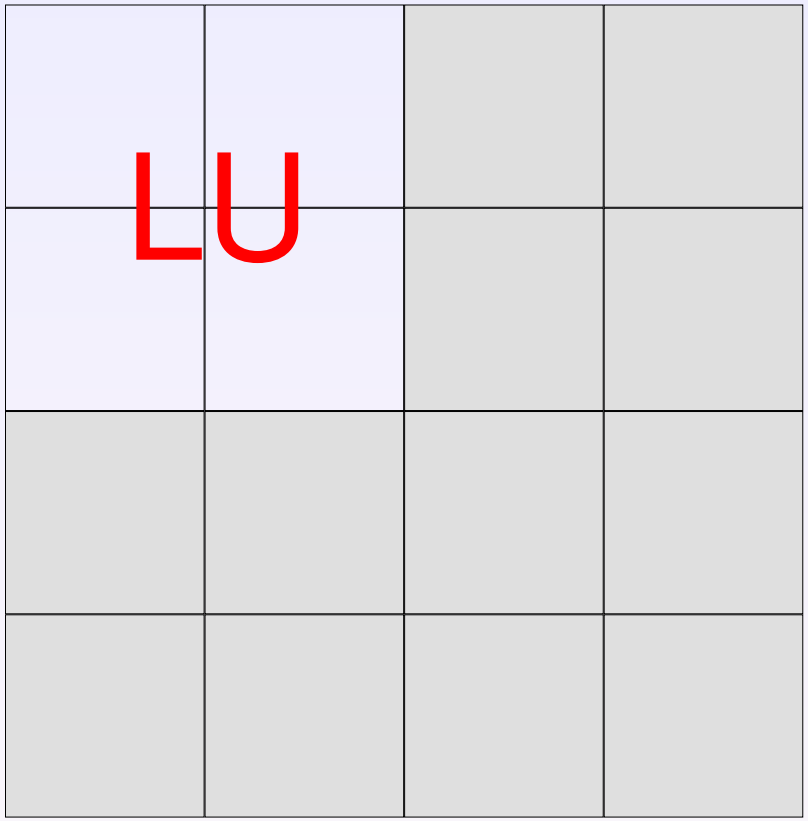
$$G_{22}, L_{21}, R_{12} \xrightarrow{\text{decomp}} L_{22}, U_{22}.$$

Problem: Straightforward approach offers only very limited potential for parallelization.

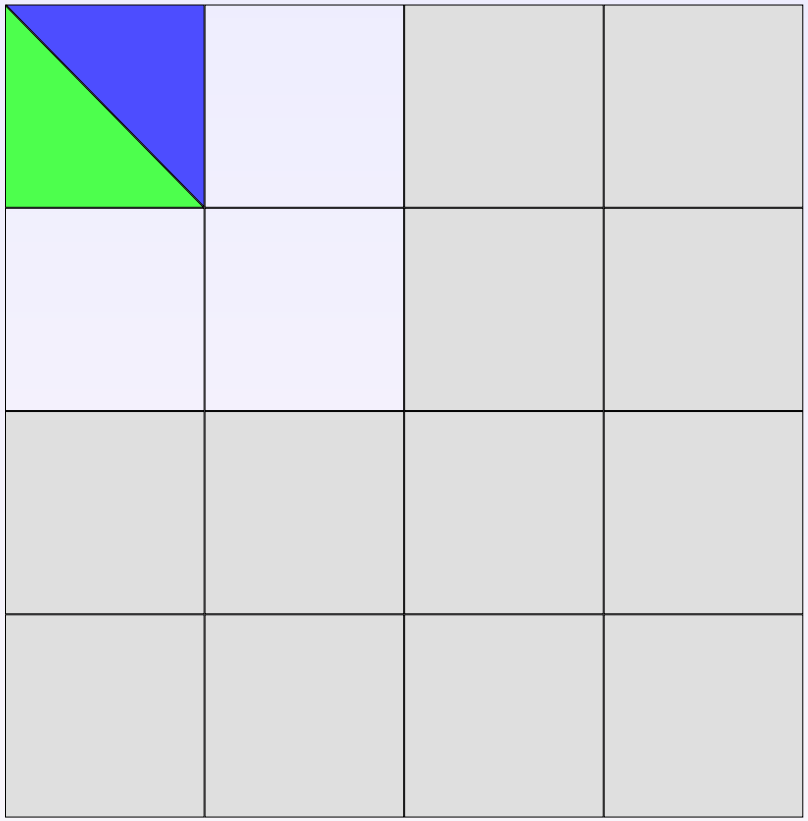
Recursive LU decomposition



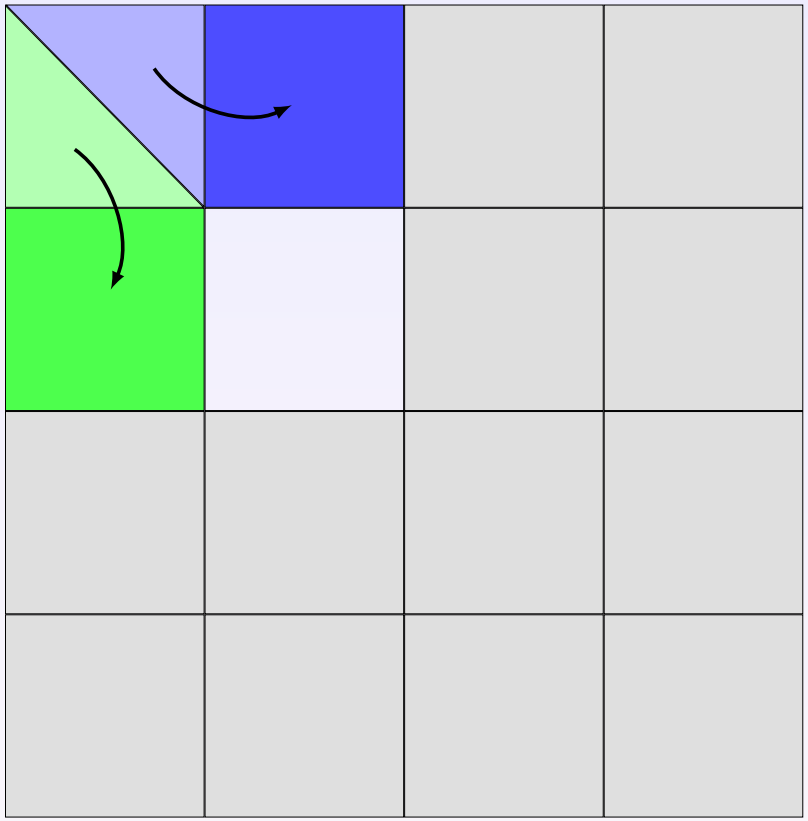
Recursive LU decomposition



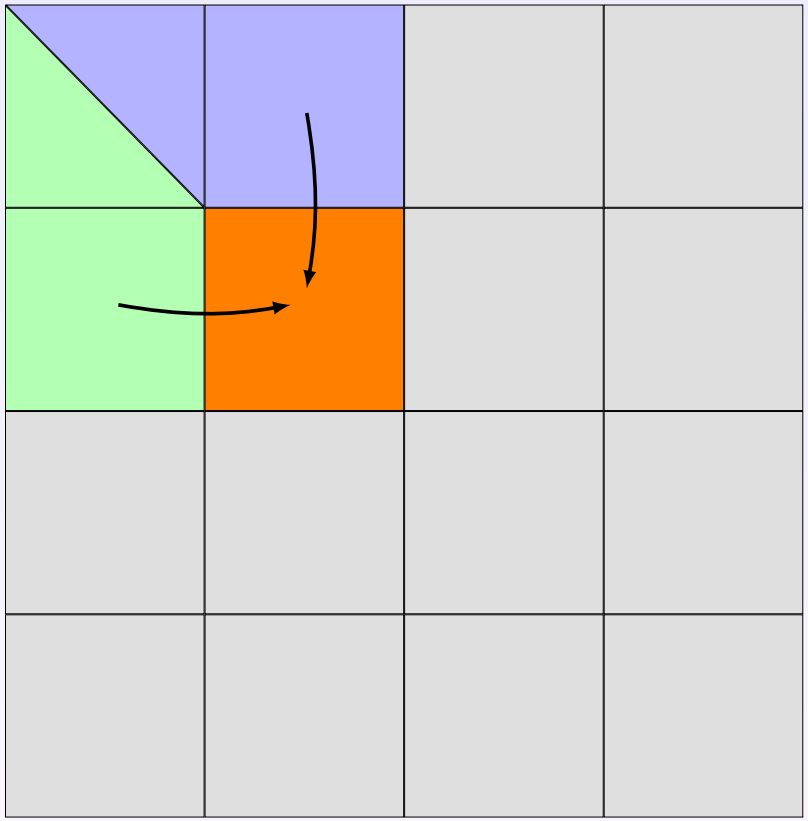
Recursive LU decomposition



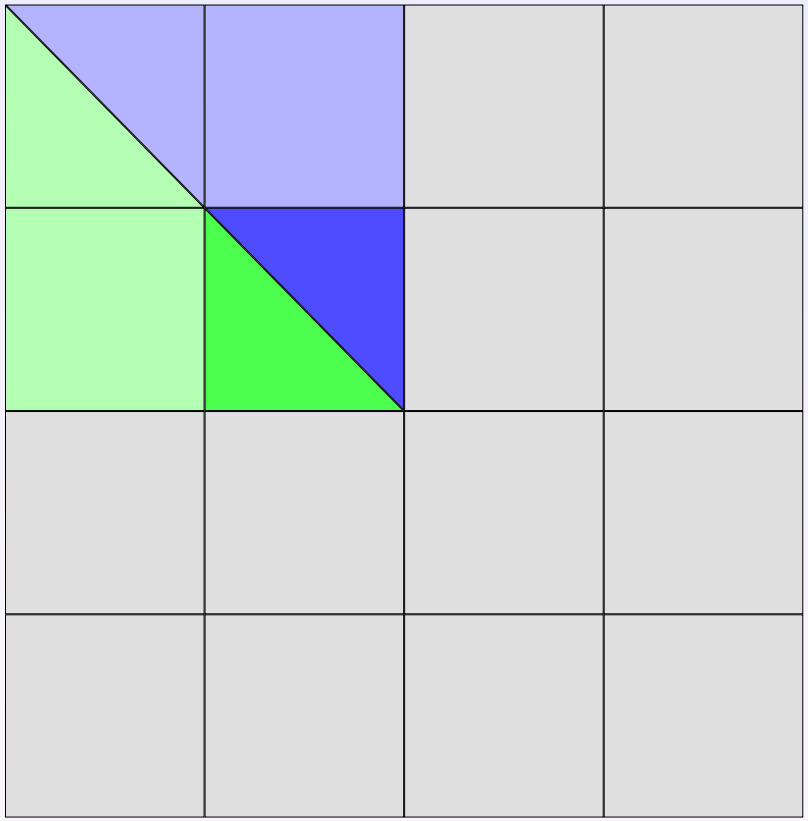
Recursive LU decomposition



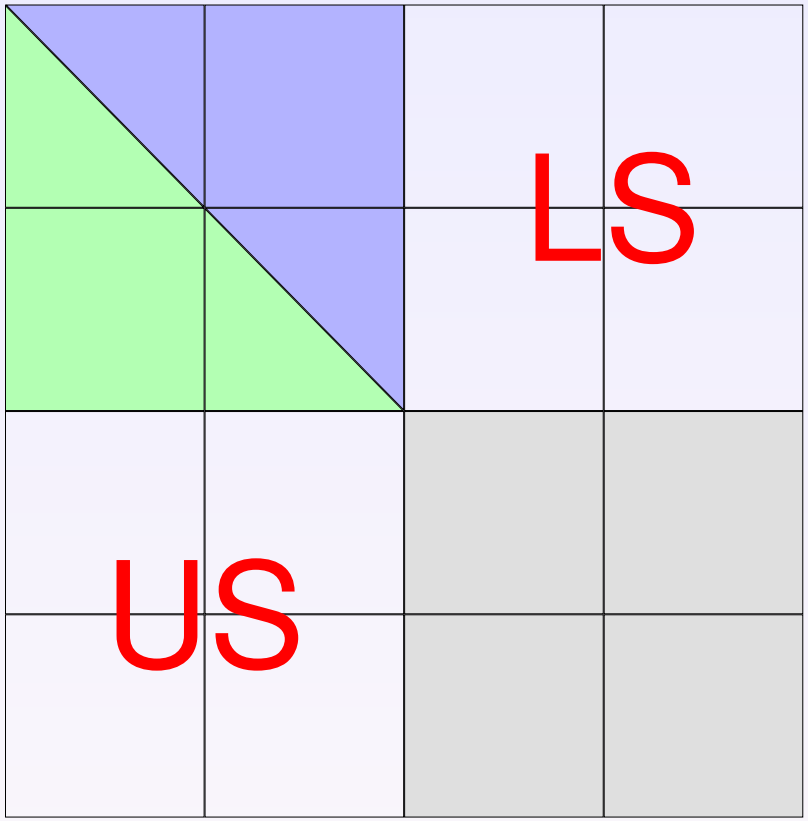
Recursive LU decomposition



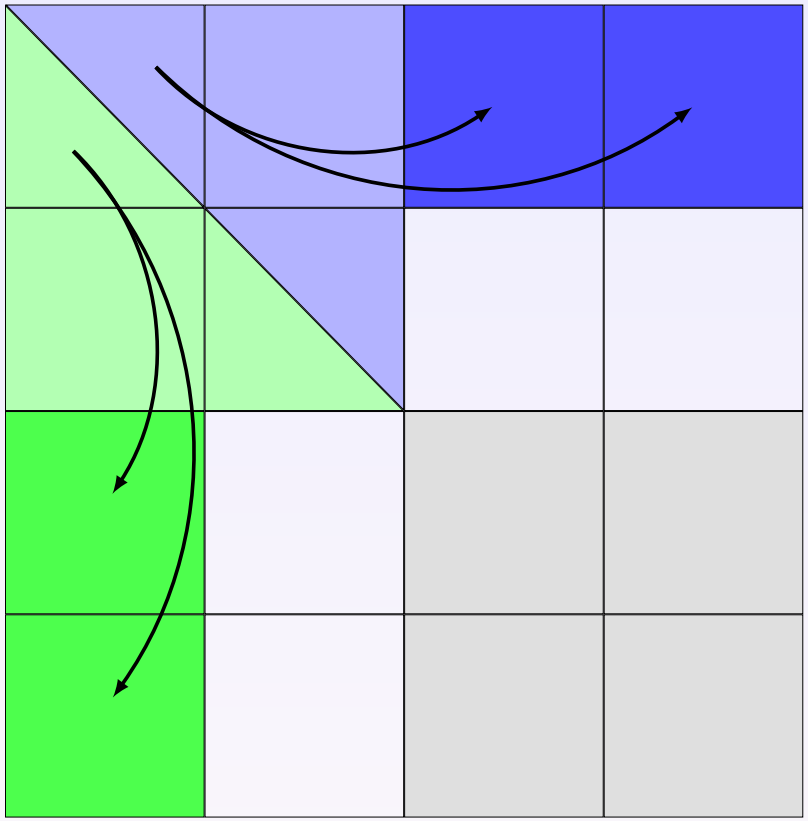
Recursive LU decomposition



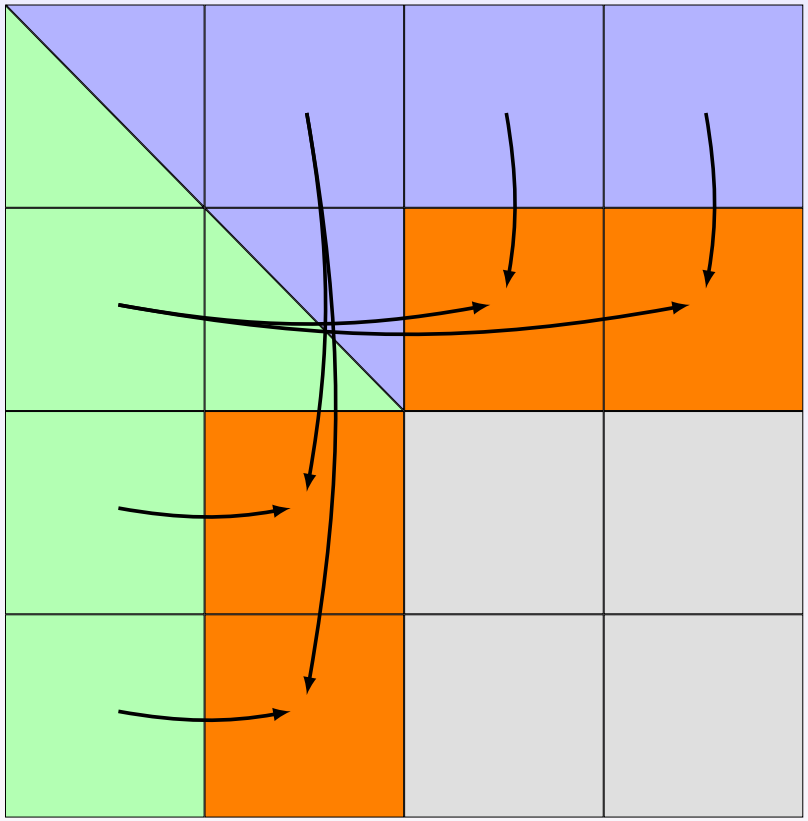
Recursive LU decomposition



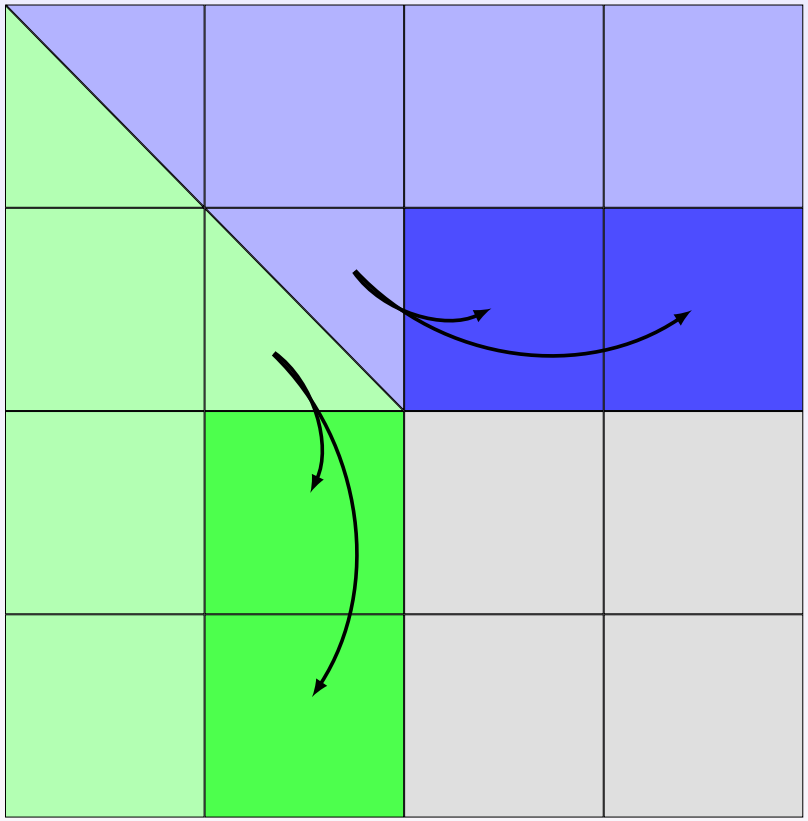
Recursive LU decomposition



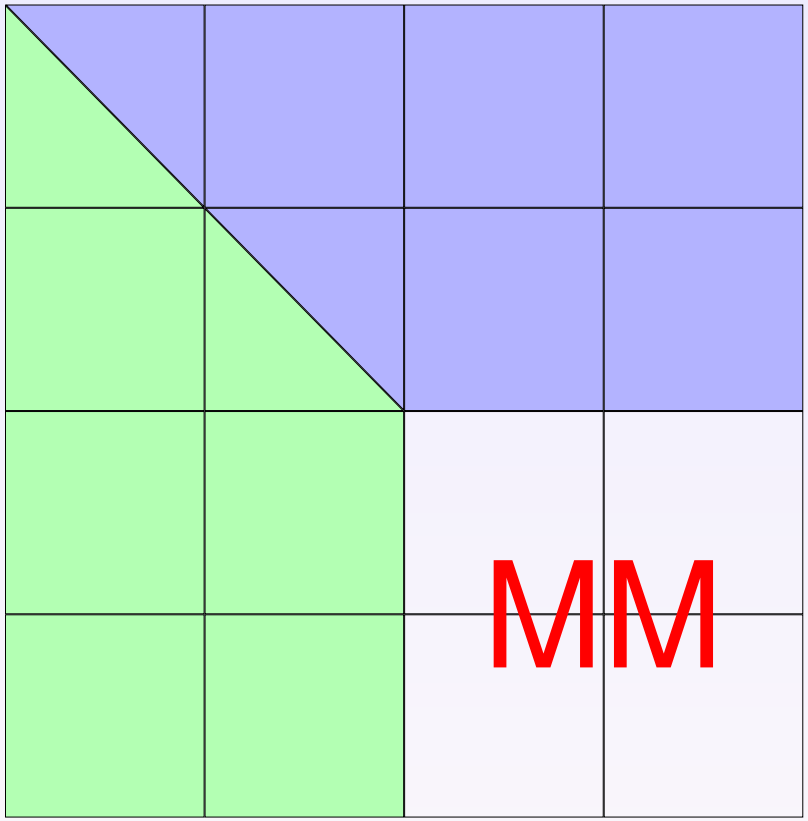
Recursive LU decomposition



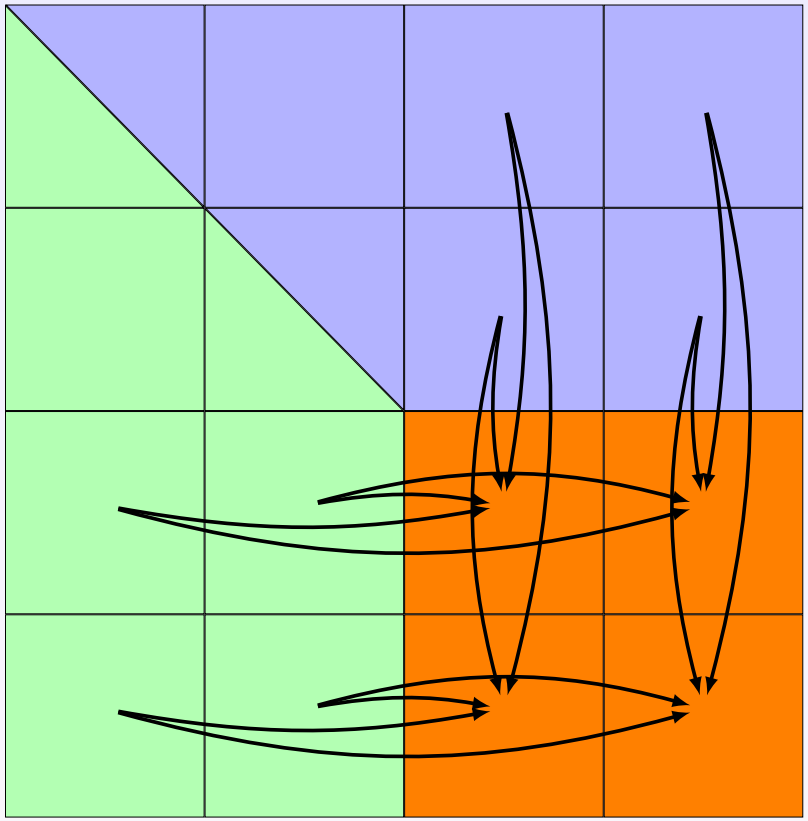
Recursive LU decomposition



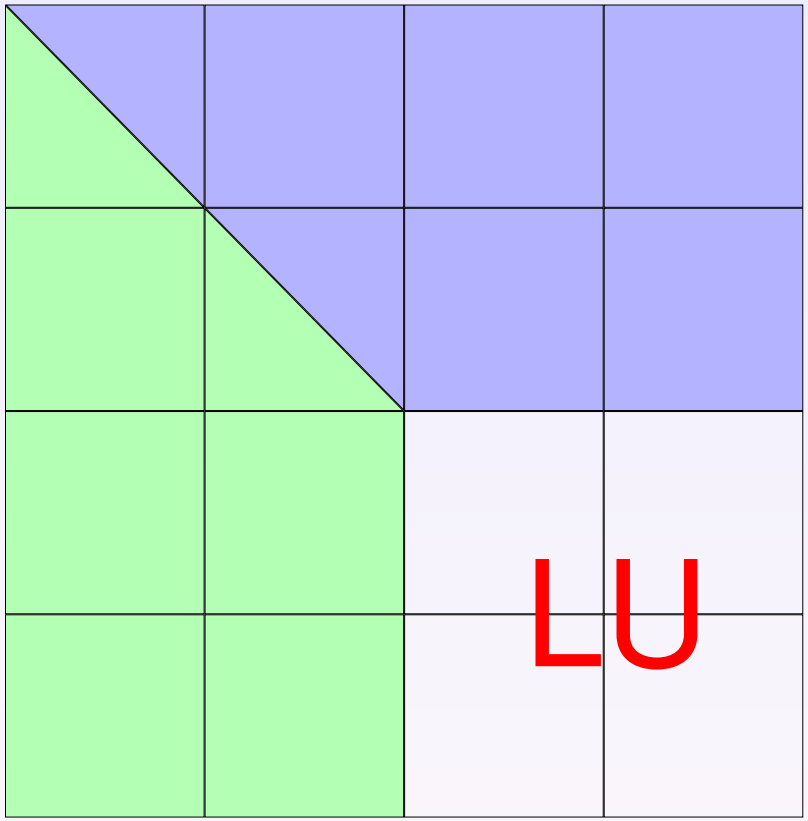
Recursive LU decomposition



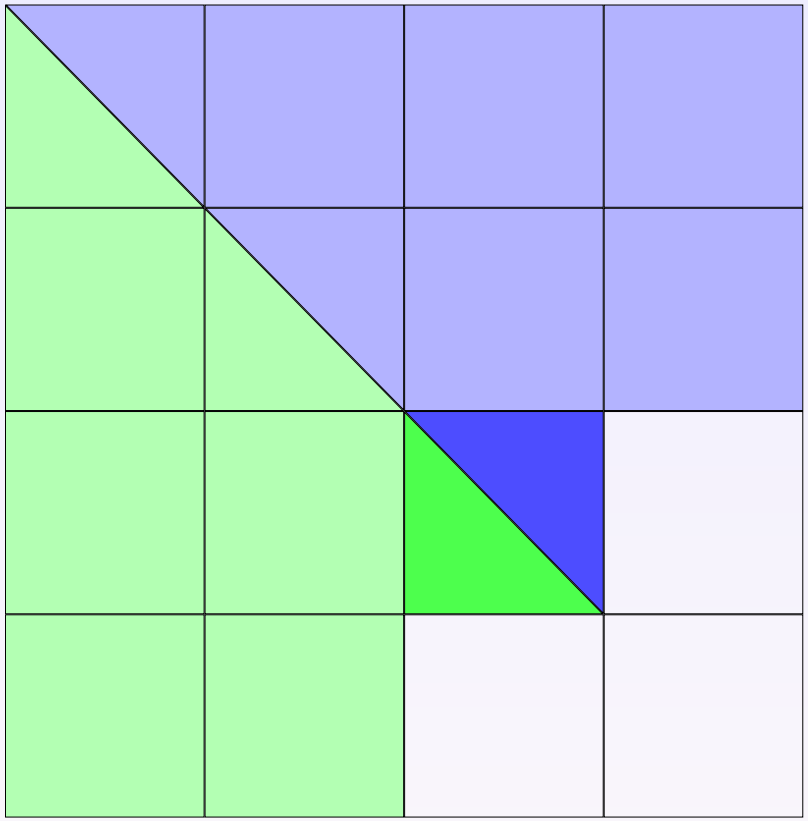
Recursive LU decomposition



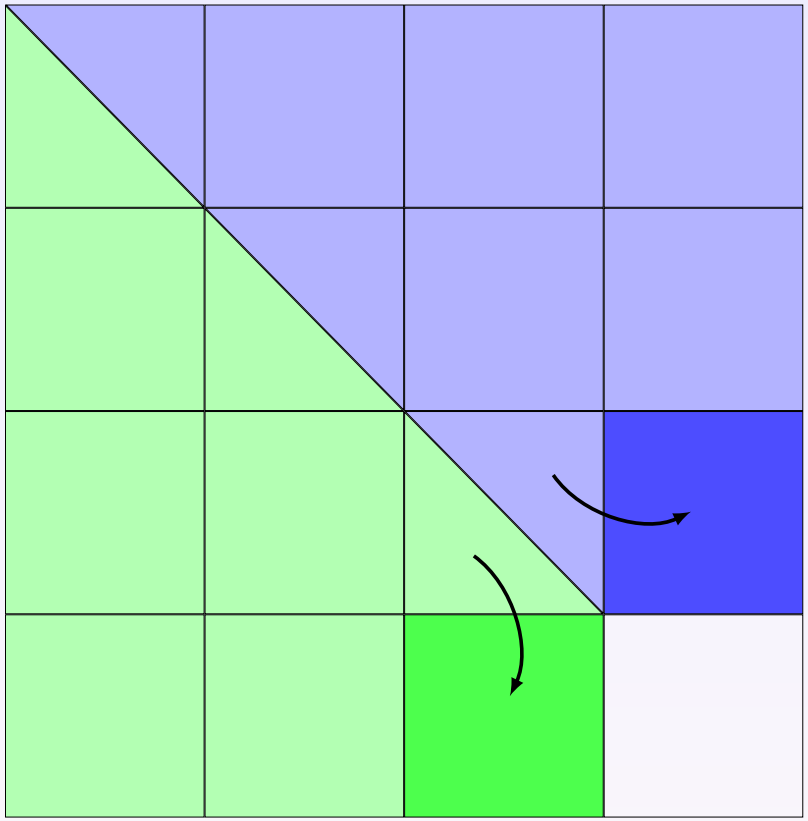
Recursive LU decomposition



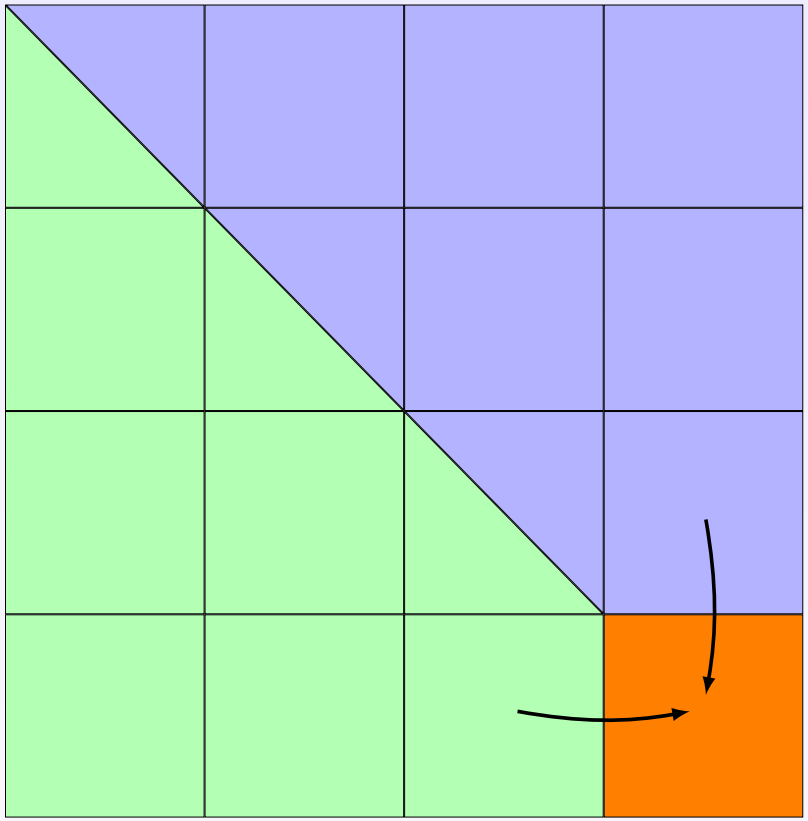
Recursive LU decomposition



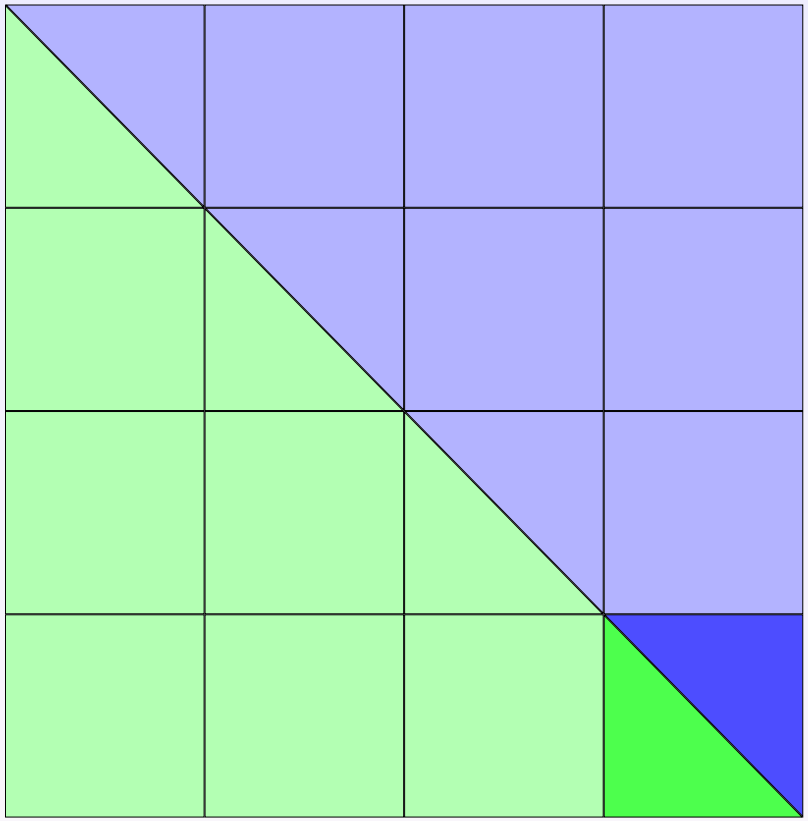
Recursive LU decomposition



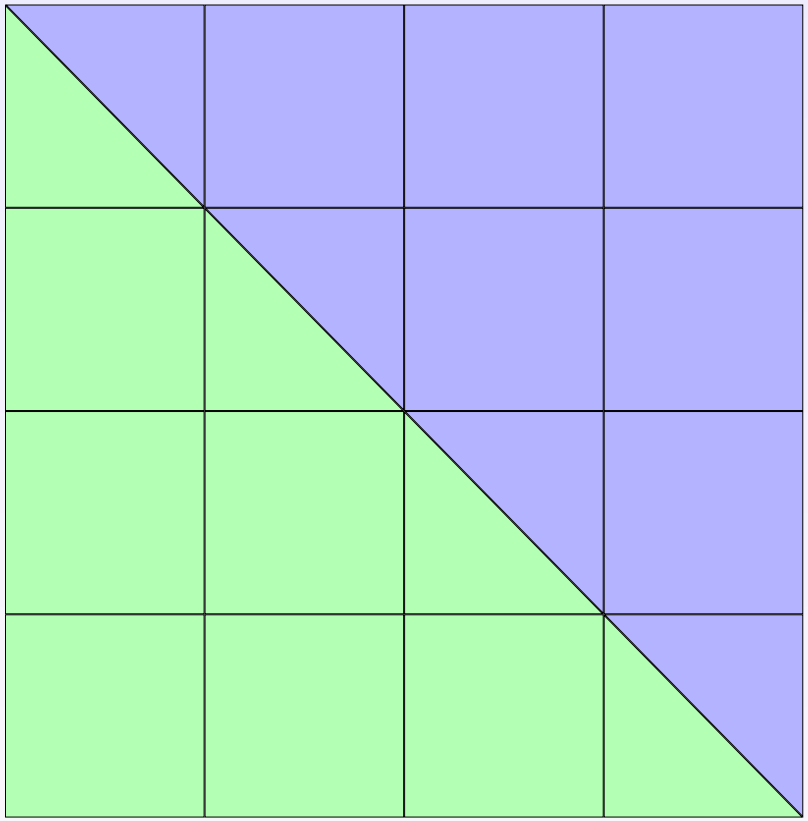
Recursive LU decomposition



Recursive LU decomposition



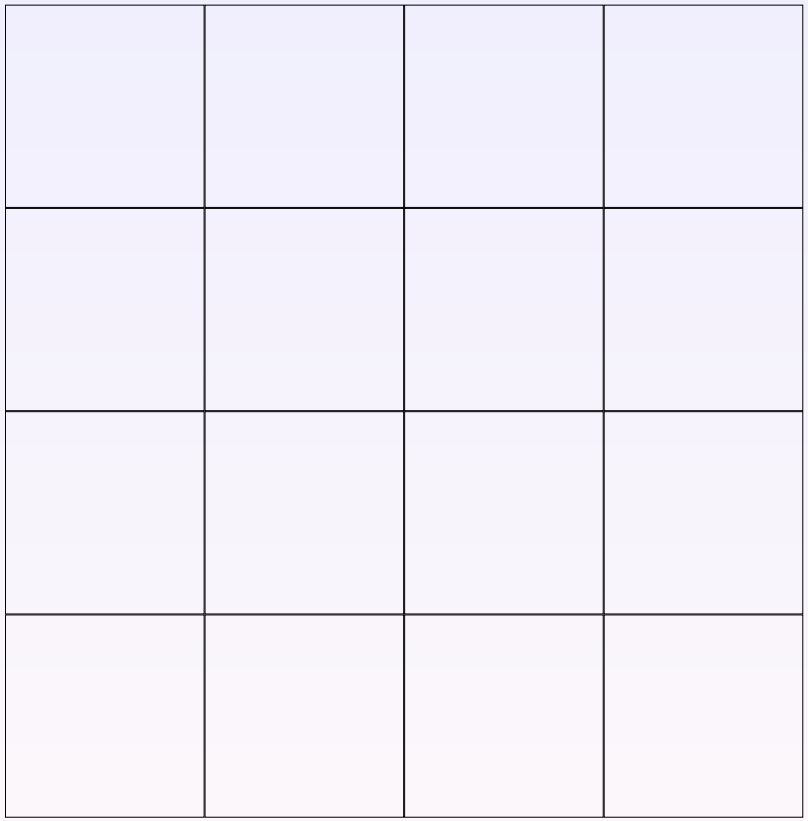
Recursive LU decomposition



Problem: Recursive structure introduces artificial dependencies between “super-tasks” that are unnecessary and harm parallelization.

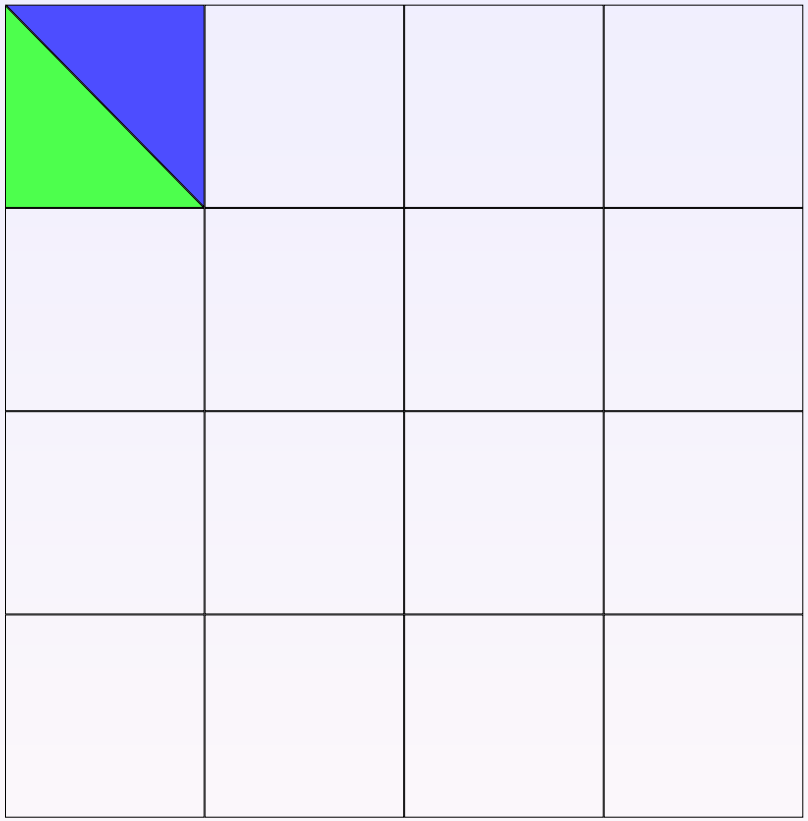
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



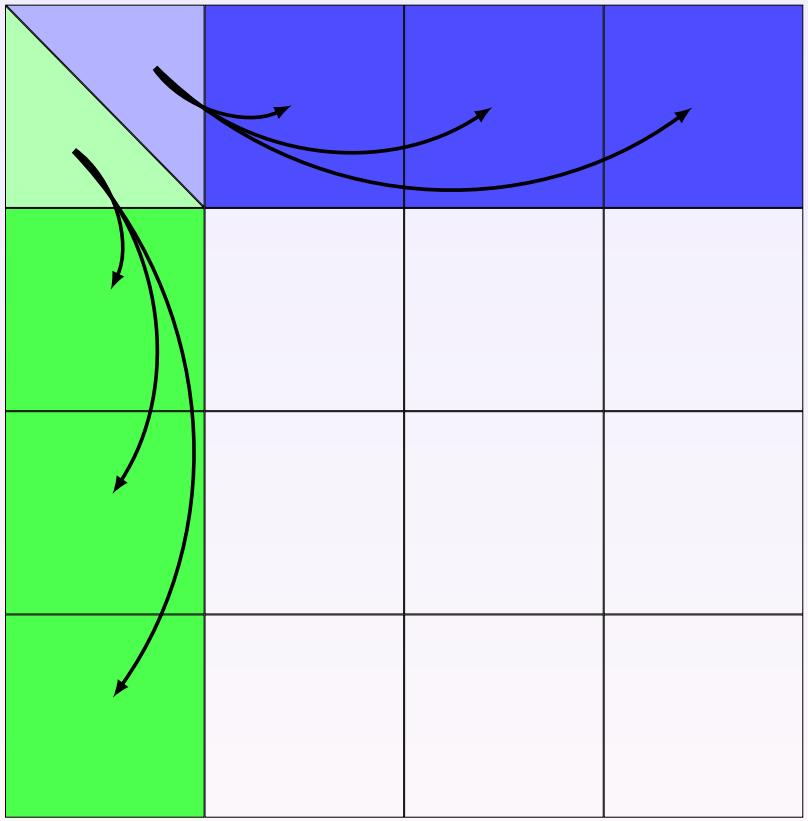
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



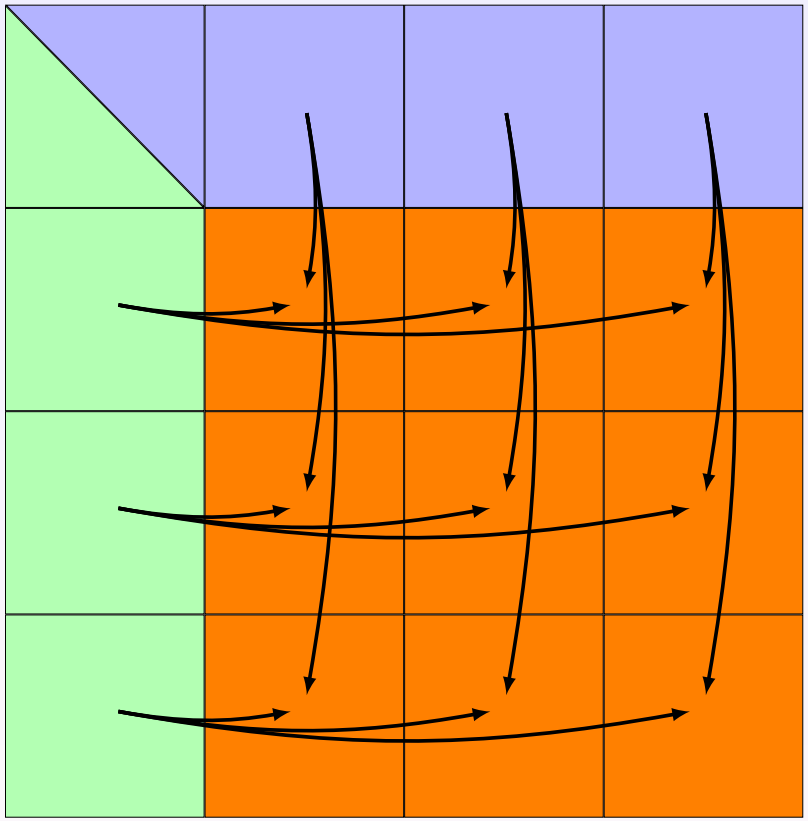
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



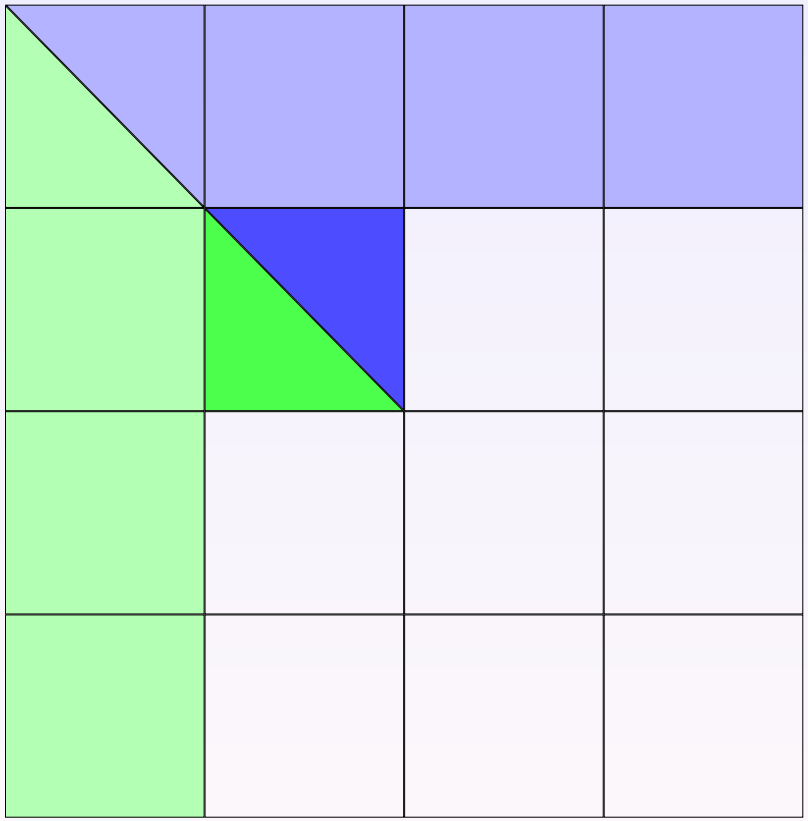
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



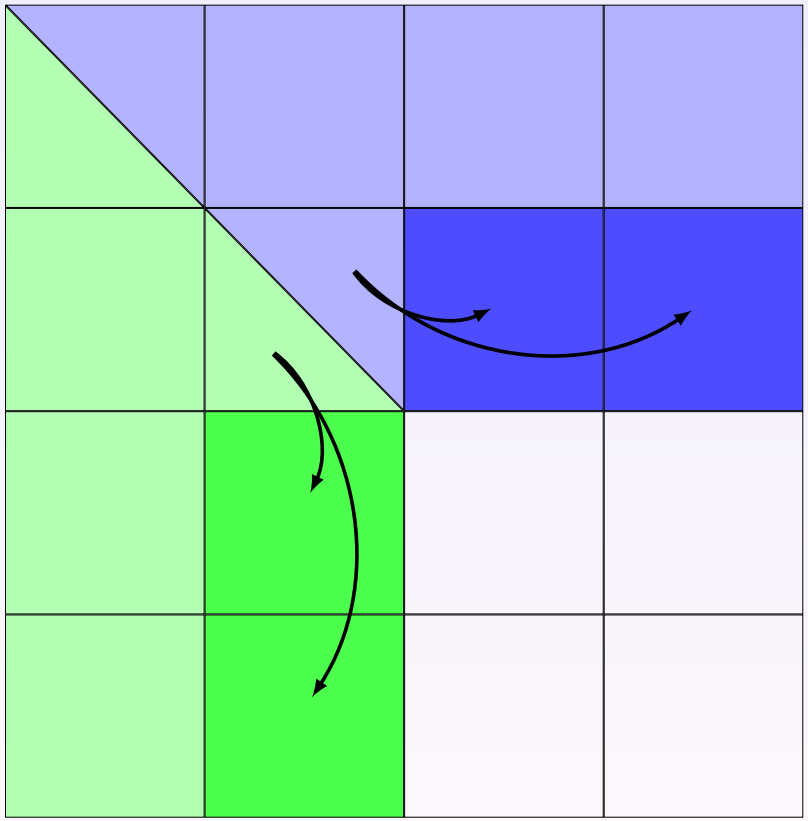
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



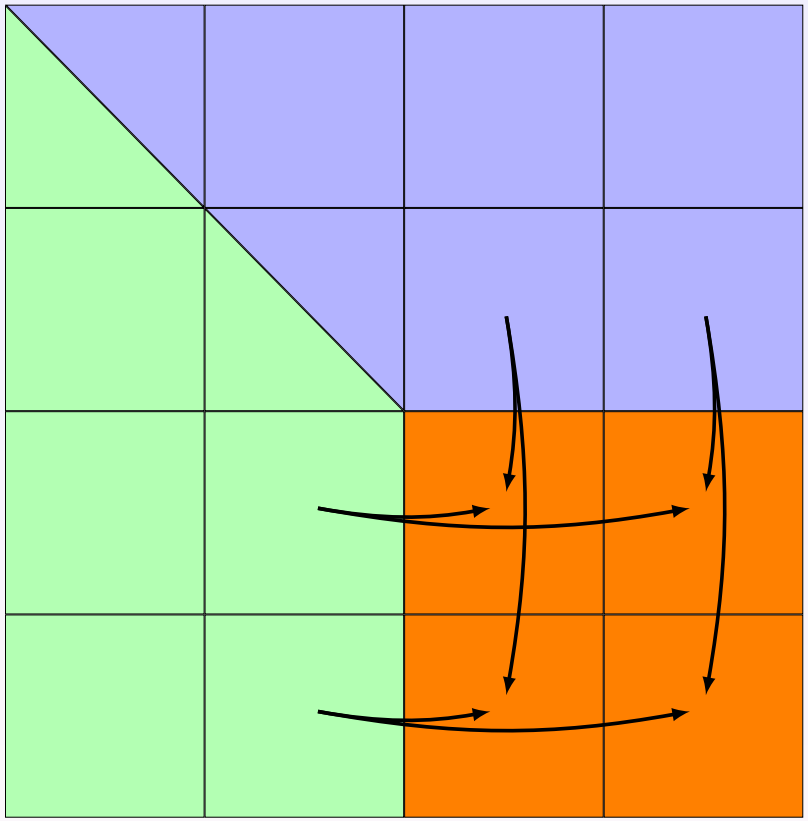
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



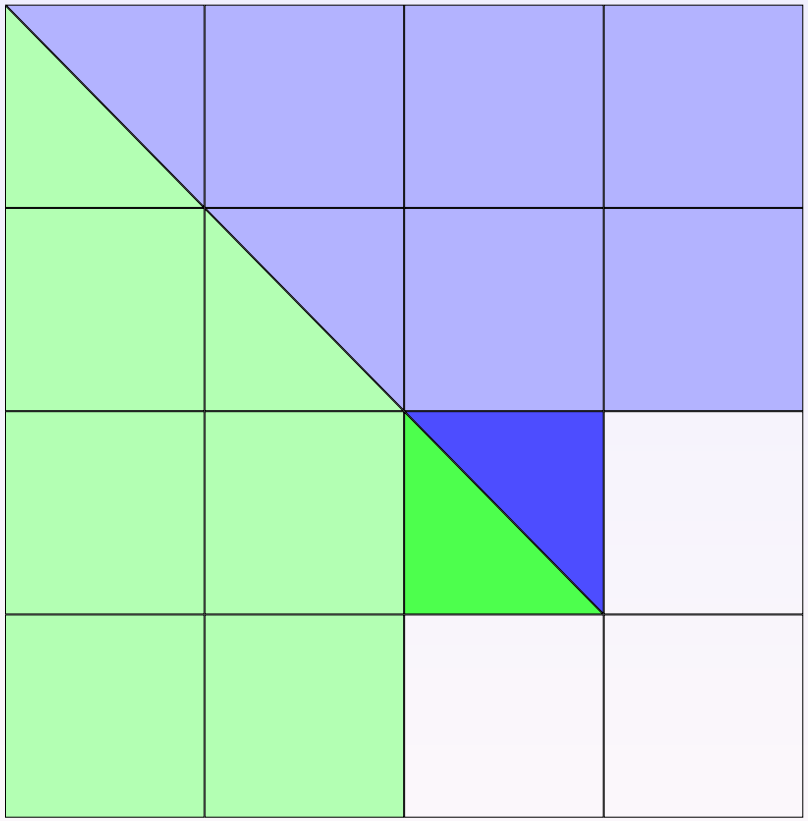
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



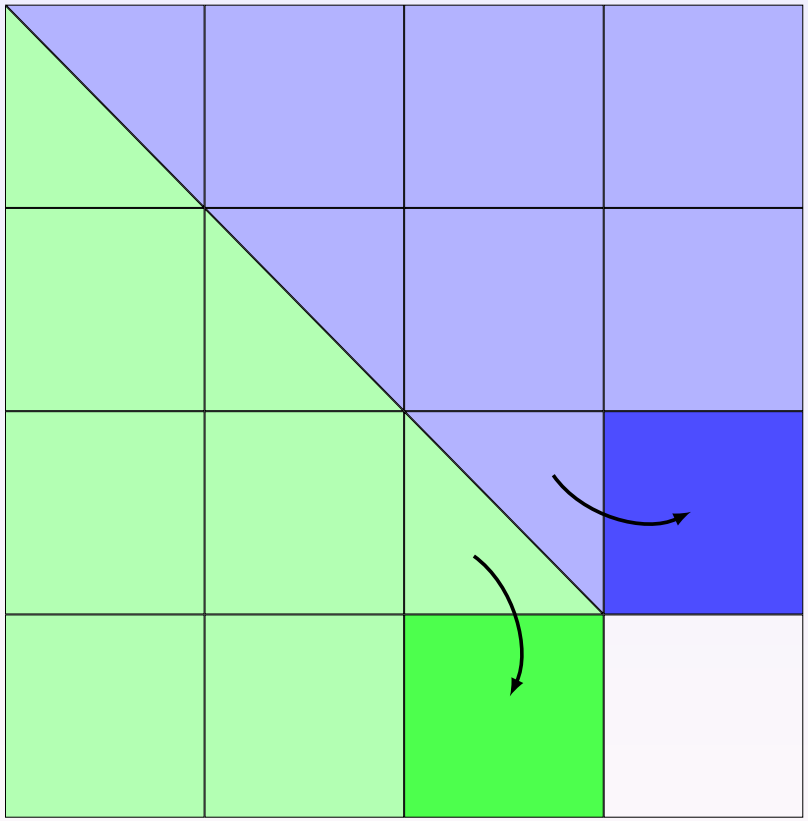
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



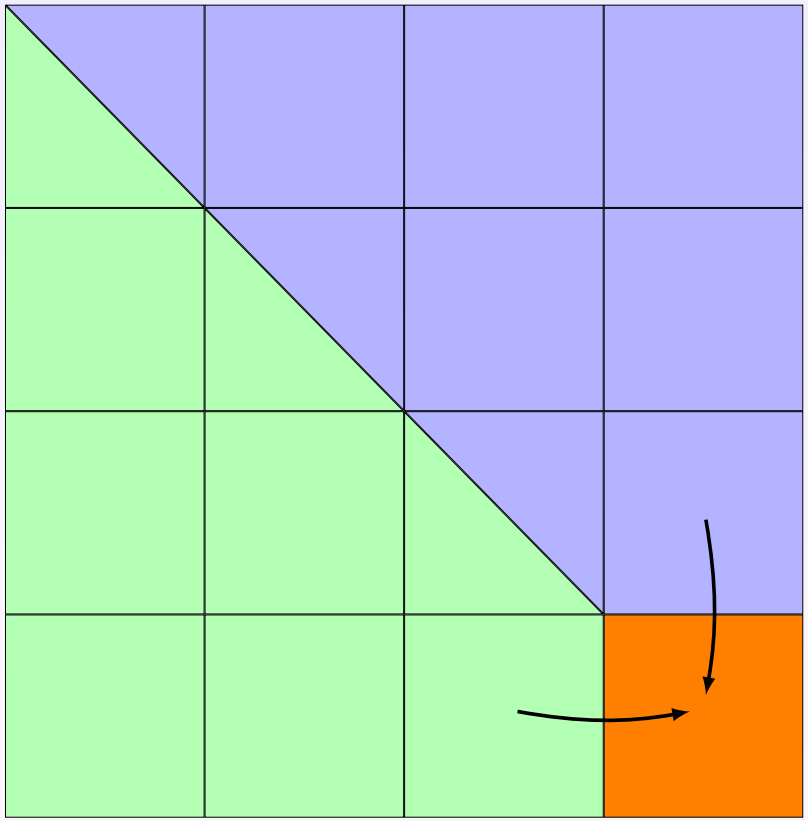
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



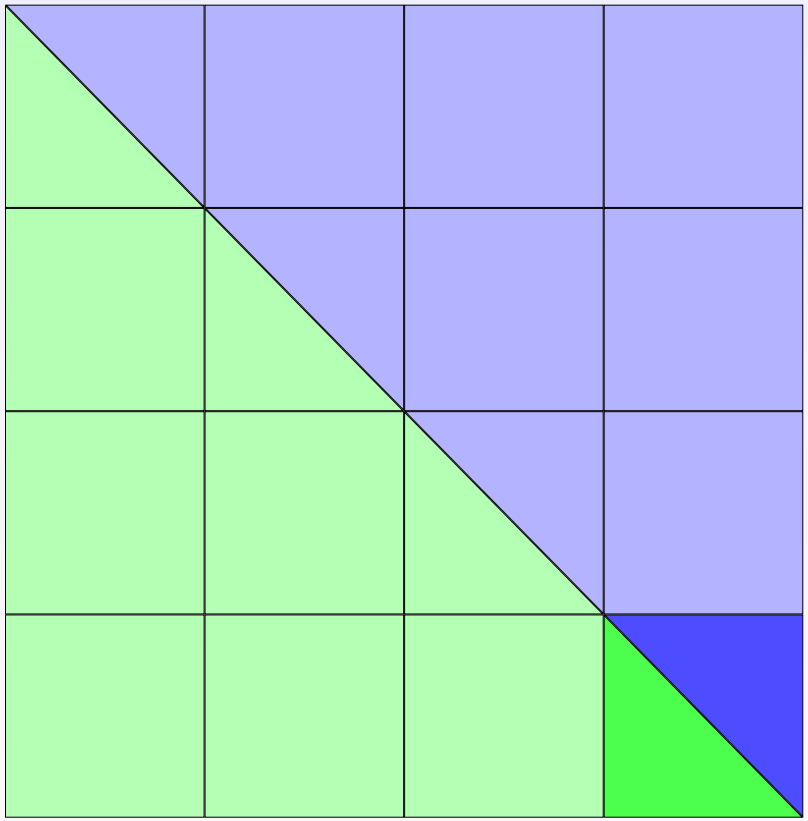
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



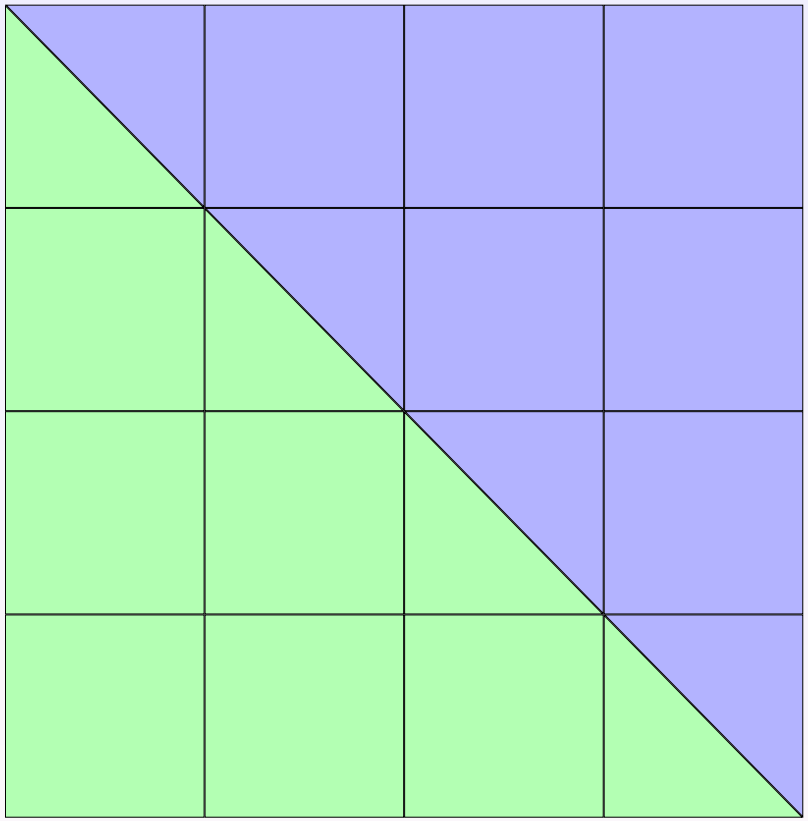
Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



Flattened task graph

Idea: Eliminate super-tasks and expose only the necessary dependencies between sub-tasks.



Open Questions

Scheduling: How can we employ multiple processors?

Data locality: How can we reduce data transfers between remote memory segments? How can we take advantage of caches?

Dynamic scheduling: Can we avoid setting up the entire (large) task graph prior to scheduling?

`http://www.h2lib.org`