Parallelisation of $\mathcal{H}$-Matrix Computations

Ronald Kriemann

Max Planck Institute for Mathematics
in the
Sciences Leipzig

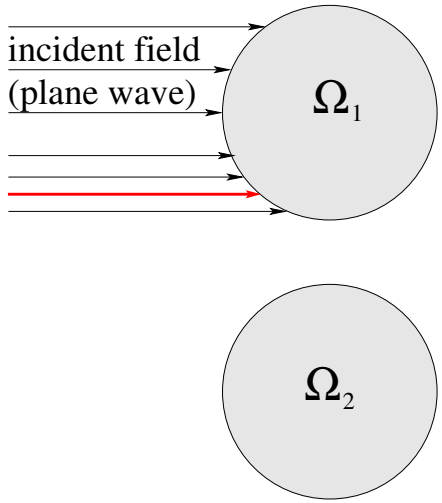
Winterschool on $\mathcal{H}$-Matrices
2007

Overview

Overview

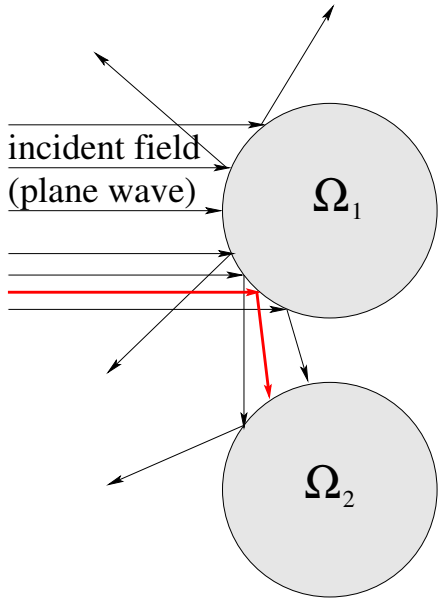
Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.



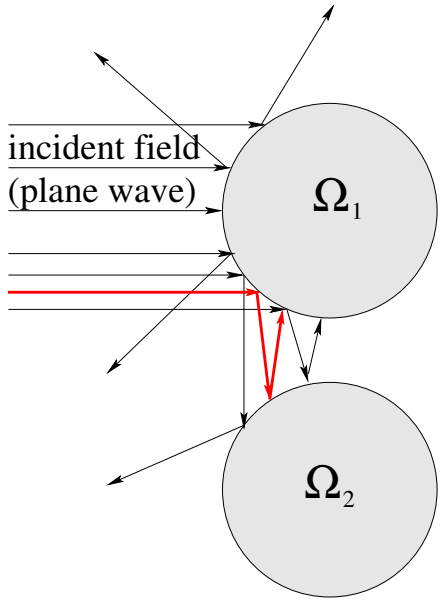
Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.



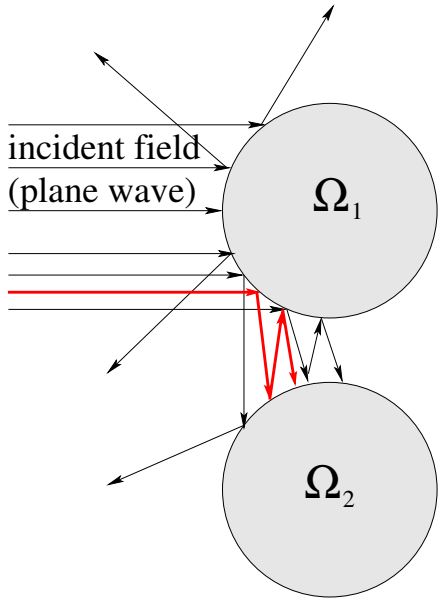
Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.



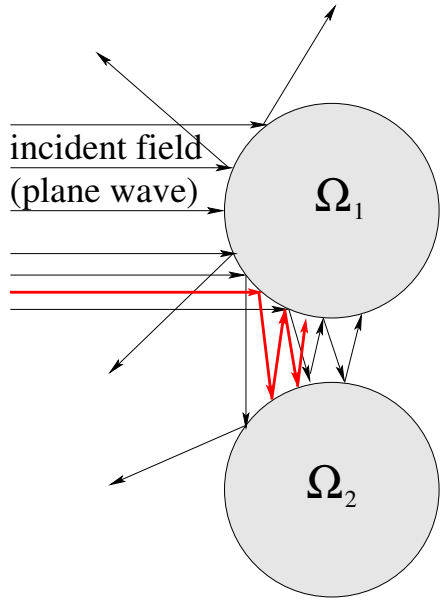
Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.



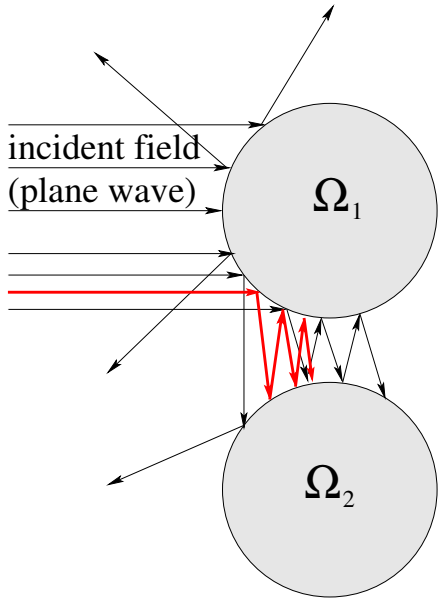
Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.



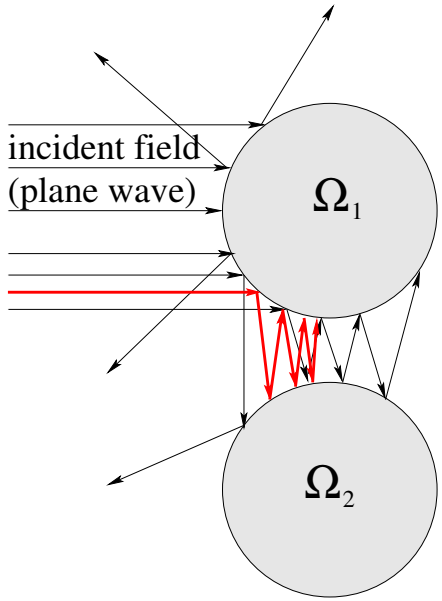
Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.



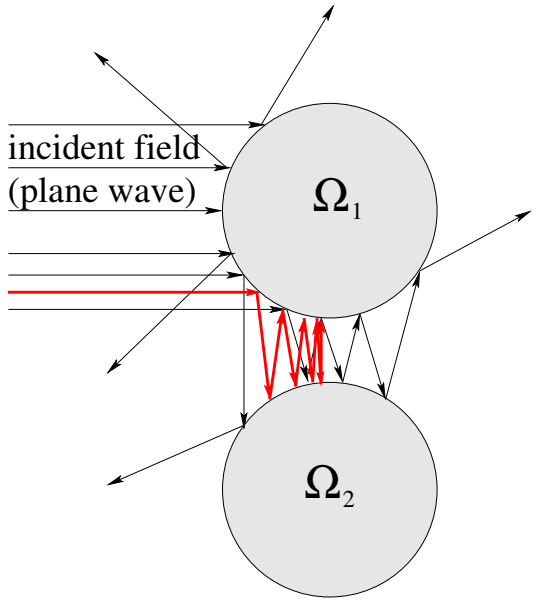
Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.



Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.

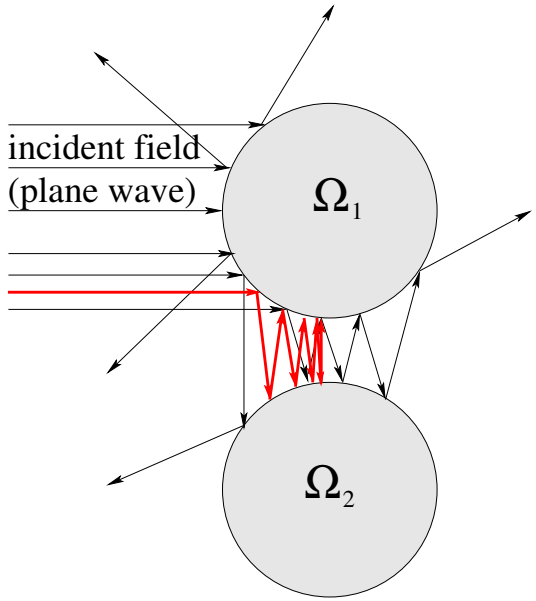


Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.

Let $\nu = \nu_1, \nu_2, \ldots$ with $1 \leq \nu_i \leq r$ be a sequence, such that $\nu_i \neq \nu_{i+1}$ for all i.

To compute: densities η_{ν_i} on Γ_{ν_i} defined by



$$\eta_{\nu_i} - \mathcal{K}_{\nu_i}\eta_{\nu_i} = g_{\nu_i},$$

with

$$(\mathcal{K}_{\nu_i}\eta_{\nu_i})(x) := \int_{\Gamma_{\nu_i}} \frac{\partial G(x,y)}{\partial n(x)} \eta_{\nu_i}(y) dy, \quad x \in \Gamma_{\nu_i}$$

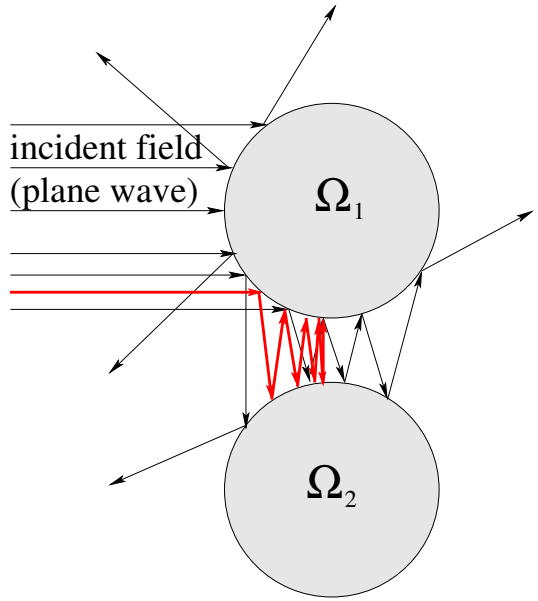
with $G(x,y) = -\frac{1}{2\pi}\frac{e^{ik|x-y|}}{|x-y|}$, $k > 0$

Model Problem: Acoustic Scattering

Let $\Omega_i \in \mathbb{R}^3, i = 1, \ldots, r$ be convex sets with smooth boundaries $\Gamma_i = \partial\Omega_i$.

Let $\nu = \nu_1, \nu_2, \ldots$ with $1 \leq \nu_i \leq r$ be a sequence, such that $\nu_i \neq \nu_{i+1}$ for all i.

To compute: densities η_{ν_i} on Γ_{ν_i} defined by



$$\eta_{\nu_i} - \mathcal{K}_{\nu_i}\eta_{\nu_i} = g_{\nu_i},$$

with

$$(\mathcal{K}_{\nu_i}\eta_{\nu_i})(x) := \int_{\Gamma_{\nu_i}} \frac{\partial G(x,y)}{\partial n(x)} \eta_{\nu_i}(y) dy, \quad x \in \Gamma_{\nu_i}$$

with $G(x,y) = -\frac{1}{2\pi}\frac{e^{ik|x-y|}}{|x-y|}$, $k > 0$, and with $d \in \mathbb{R}^3$, $|d| = 1$,

$$g_{\nu_i}(x) := \begin{cases} 2\frac{\partial e^{ik\langle d,x\rangle}}{\partial n(x)}, & i = 1 \\ (\mathcal{K}_{\nu_{i-1}}\eta_{\nu_{i-1}})(x), \ x \in \Gamma_{\nu_i} & i > 1 \end{cases}$$

Model Problem: Acoustic Scattering

Discrete System

With a standard Galerkin discretisation using constant ansatz functions, one gets the following matrices

$$K_{ij}^{\nu_i,\nu_i} := \frac{1}{2\pi} \int_{\Gamma_{\nu_i}} \int_{\Gamma_{\nu_i}} e^{ik|x-y|} \frac{1 - ik|x-y|}{|x-y|^3} \langle x - y, n(x) \rangle dy dx$$

and

$$K_{ij}^{\nu_i,\nu_{i-1}} := \frac{1}{2\pi} \int_{\Gamma_{\nu_i}} \int_{\Gamma_{\nu_{i-1}}} e^{ik|x-y|} \frac{1 - ik|x-y|}{|x-y|^3} \langle x - y, n(x) \rangle dy dx$$

Model Problem: Acoustic Scattering

Discrete System

With a standard Galerkin discretisation using constant ansatz functions, one gets the following matrices

$$K_{ij}^{\nu_i,\nu_i} := \frac{1}{2\pi} \int_{\Gamma_{\nu_i}} \int_{\Gamma_{\nu_i}} e^{ik|x-y|} \frac{1 - ik|x-y|}{|x-y|^3} \langle x - y, n(x) \rangle dy dx$$

and

$$K_{ij}^{\nu_i,\nu_{i-1}} := \frac{1}{2\pi} \int_{\Gamma_{\nu_i}} \int_{\Gamma_{\nu_{i-1}}} e^{ik|x-y|} \frac{1 - ik|x-y|}{|x-y|^3} \langle x - y, n(x) \rangle dy dx$$

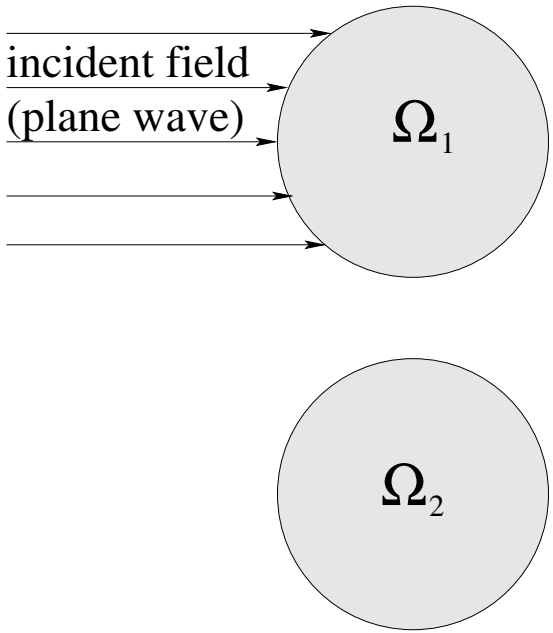
Properties

- expensive integrals to compute,
- complex valued arithmetic even more costly

Model Problem: Acoustic Scattering

Algorithm for two Objects ($\nu = 1, 2, 1, 2, 1, 2, \ldots$)

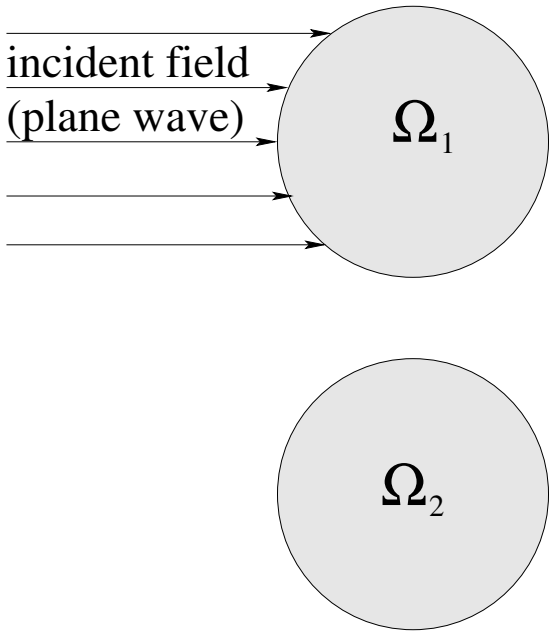
build $K^{1,1}, K^{2,2}, K^{2,1}$ and $K^{1,2}$;
build initial right hand side $b_{1,1} := g_1$;
for $i = 1, \cdots$ **do**
  solve $(I - K^{1,1})\eta_{i,1} = b_{i,1}$;
  if $\eta_{i,1}/\eta_{i-1,1}$ converged **then stop**
  $b_{i,2} := K^{2,1}\eta_{i,1}$;
  solve $(I - K^{2,2})\eta_{i,2} = b_{i,2}$;
  $b_{i,1} := K^{1,2}\eta_{i,2}$;
endfor



Model Problem: Acoustic Scattering

Algorithm for two Objects ($\nu = 1, 2, 1, 2, 1, 2, \ldots$)

build $K^{1,1}, K^{2,2}, K^{2,1}$ and $K^{1,2}$;
build initial right hand side $b_{1,1} := g_1$;
for $i = 1, \cdots$ **do**
  solve $(I - K^{1,1})\eta_{i,1} = b_{i,1}$;
  if $\eta_{i,1}/\eta_{i-1,1}$ converged **then stop**
  $b_{i,2} := K^{2,1}\eta_{i,1}$;
  solve $(I - K^{2,2})\eta_{i,2} = b_{i,2}$;
  $b_{i,1} := K^{1,2}\eta_{i,2}$;
endfor

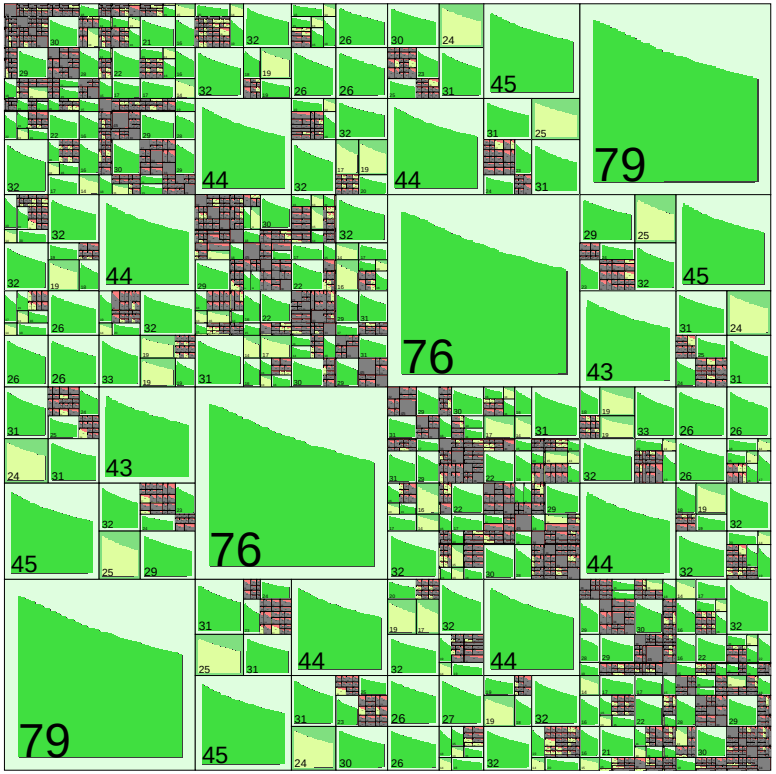


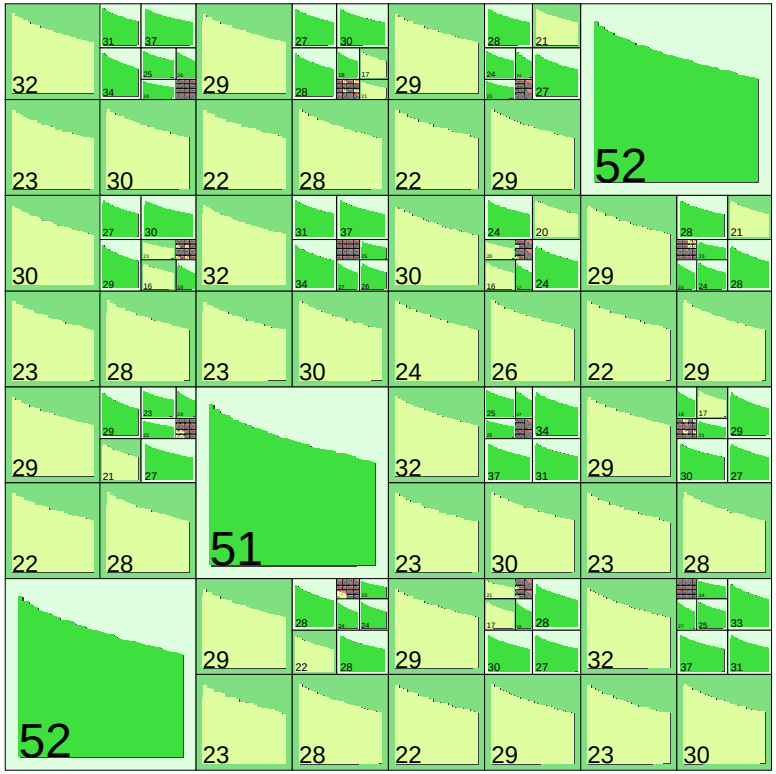
Involved $\mathcal{H}$-Arithmetics

- Matrix-Construction,
- solving linear equation systems with $\mathcal{H}$-Preconditioner and
- Matrix-Vector Multiplication

Model Problem: Acoustic Scattering

Example





Model Problem: Acoustic Scattering

Problem

- Matrix-Construction is very expensive, especially for large k
- $\mathcal{H}$-Preconditioner via $\mathcal{H}$-Inversion or $\mathcal{H}$-LU-Factorisation can be expensive

Model Problem: Acoustic Scattering

Problem

- Matrix-Construction is very expensive, especially for large k
- $\mathcal{H}$-Preconditioner via $\mathcal{H}$-Inversion or $\mathcal{H}$-LU-Factorisation can be expensive

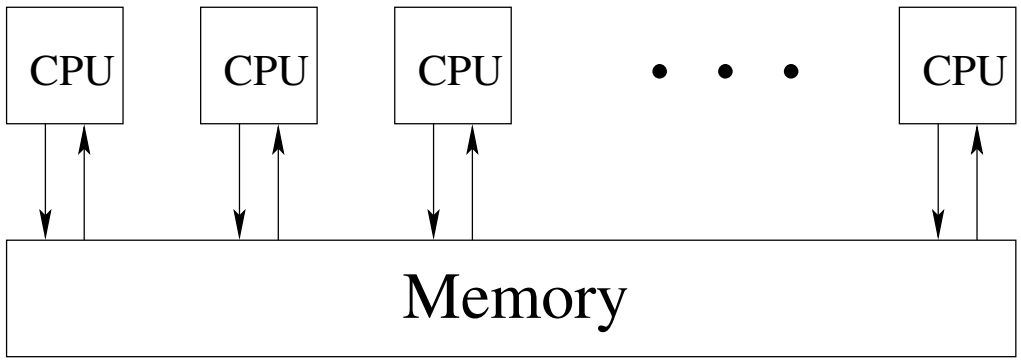
Goal

Use parallel algorithms for involved $\mathcal{H}$-Arithmetic with small programming effort but high efficiency on workstations or small compute servers.

Overview

System Architecture

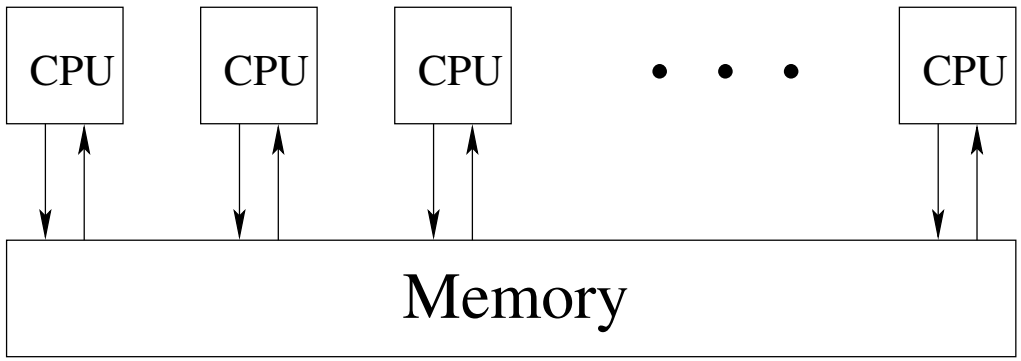
Workstations and small compute servers are usually systems with a shared memory, e.g. all processors have direct access to the same memory:



Technical Prerequisites

System Architecture

Workstations and small compute servers are usually systems with a shared memory, e.g. all processors have direct access to the same memory:



Consequences

- simplified programming because no communications involved,
- but protection of critical resources necessary, e.g. simultaneous access to the same memory position

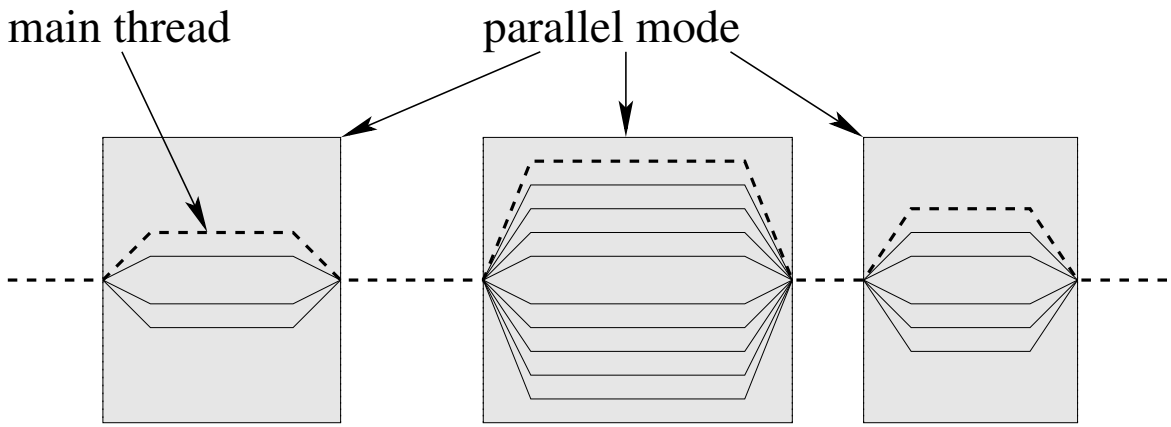
Technical Prerequisites

Threads

The standard parallelisation tool on shared memory systems are threads, i.e. parallel computation paths in a program.

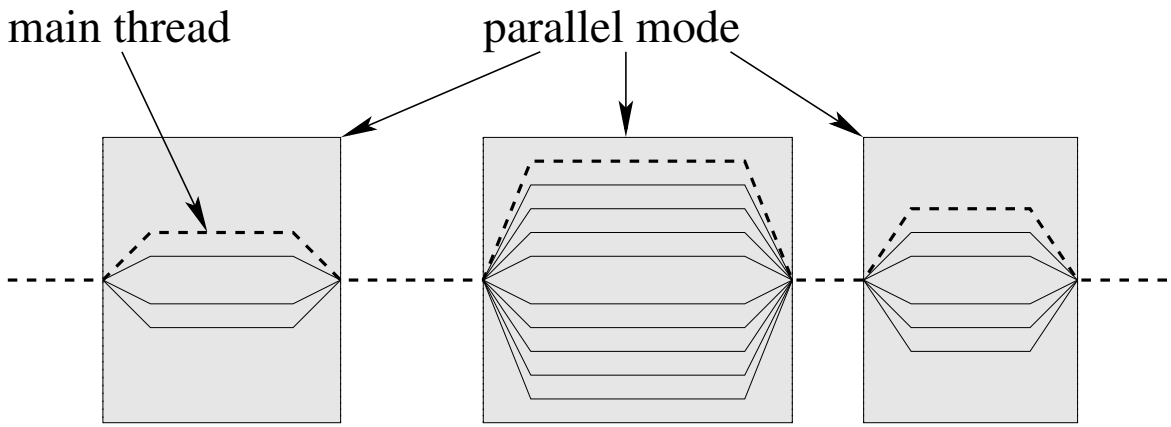
Threads

The standard parallelisation tool on shared memory systems are threads, i.e. parallel computation paths in a program.
Each program has a main thread, e.g. the `main` function in a C program. Afterwards, new threads can be started, e.g:



Threads

The standard parallelisation tool on shared memory systems are threads, i.e. parallel computation paths in a program.
Each program has a main thread, e.g. the `main` function in a C program. Afterwards, new threads can be started, e.g:



Functions for threads:

- `t = tcreate( f )`: create thread `t` and start function `f`,
- `tsync( t )`: synchronise with thread `t`, e.g. wait until `t` has finished.

Mutices

A mutex is a tool for mutual exclusion of critical sections in a program. It either is `LOCKED` or `UNLOCKED`.
Locking an already locked mutex blocks the thread until the mutex is unlocked by another thread.

Mutices

A mutex is a tool for mutual exclusion of critical sections in a program. It either is `LOCKED` or `UNLOCKED`.
Locking an already locked mutex blocks the thread until the mutex is unlocked by another thread.

Example: compute $A := A + \sum_{i=1}^{4} A_i$, with matrices A and $A_i, i = 1, \ldots, 4$

On thread 1:

$$T_1 := A_1 + A_2;$$

$$A := A + T_1;$$

On thread 2:

$$T_2 := A_3 + A_4;$$

$$A := A + T_2;$$

Mutices

A mutex is a tool for mutual exclusion of critical sections in a program. It either is `LOCKED` or `UNLOCKED`.
Locking an already locked mutex blocks the thread until the mutex is unlocked by another thread.

Example: compute $A := A + \sum_{i=1}^{4} A_i$, with matrices A and $A_i, i = 1, \ldots, 4$

On thread 1:

$T_1 := A_1 + A_2;$
LOCK(m);
$A := A + T_1;$
UNLOCK(m);

On thread 2:

$T_2 := A_3 + A_4;$
LOCK(m);
$A := A + T_2;$
UNLOCK(m);

with common mutex m

Overview

Matrix Construction

Let I be an index set with $|I| = n$, $T(I)$ a cluster tree over I and $T(I \times I)$ a block cluster tree over $I \times I$.
Let $p > 0$ be the number of processors.

Sequential Algorithm without Hierarchy

Build matrix blocks only for leaves in block cluster tree:

for $(t, s) \in \mathcal{L}(T(I \times I))$ **do**
    if (t, s) is admissible **then** build low rank matrix;
    else build dense matrix;
endfor;

Matrix Construction

Let I be an index set with $|I| = n$, $T(I)$ a cluster tree over I and $T(I \times I)$ a block cluster tree over $I \times I$.
Let $p > 0$ be the number of processors.

Sequential Algorithm without Hierarchy

Build matrix blocks only for leaves in block cluster tree:

for $(t, s) \in \mathcal{L}(T(I \times I))$ **do**
    if (t, s) is admissible **then** build low rank matrix;
    else build dense matrix;
endfor;

Properties

- $\mathcal{L}(T(I \times I)) \gg p$, e.g. enough to do for each processor,
- all matrix blocks can be built independently.

Matrix Construction

Parallel Algorithm without Hierarchy

Compute each matrix block in own thread.

procedure build ((t, s))
    if (t, s) is admissible **then** build low rank matrix;
    else build dense matrix;
end;

for $(t, s) \in \mathcal{L}(T(I \times I))$ **do**
    $\mathcal{T} := \mathcal{T} \cup \{$ `tcreate`(build((t, s)) $\}$;
for `t` $\in \mathcal{T}$ **do**
    `tsync`(`t`);

Matrix Construction

Parallel Algorithm without Hierarchy

Compute each matrix block in own thread.

procedure build ((t, s))
    if (t, s) is admissible **then** build low rank matrix;
    else build dense matrix;
end;

for $(t, s) \in \mathcal{L}(T(I \times I))$ **do**
    $\mathcal{T} := \mathcal{T} \cup \{$ `tcreate`(build((t, s)) $\}$;
for `t` $\in \mathcal{T}$ **do**
    `tsync`(`t`);

Complexity

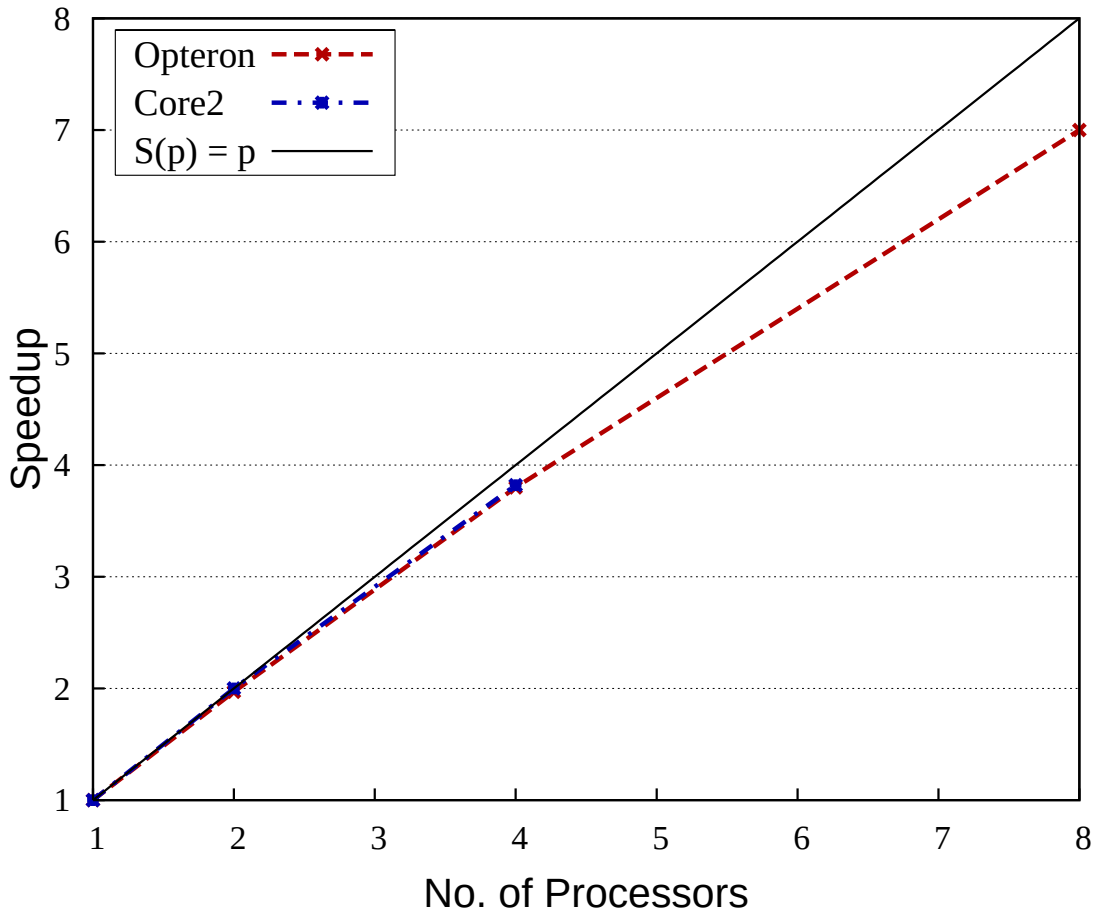
Simple, automatic load balancing by thread implementation leads to optimal parallel complexity:

$$\mathcal{W}_{\text{Con}}(n, p) = \mathcal{O}\left(\frac{n \log n}{p}\right)$$

Matrix Construction

Numerical Experiments ($K^{1,1}, n = 8192, k = 8$)

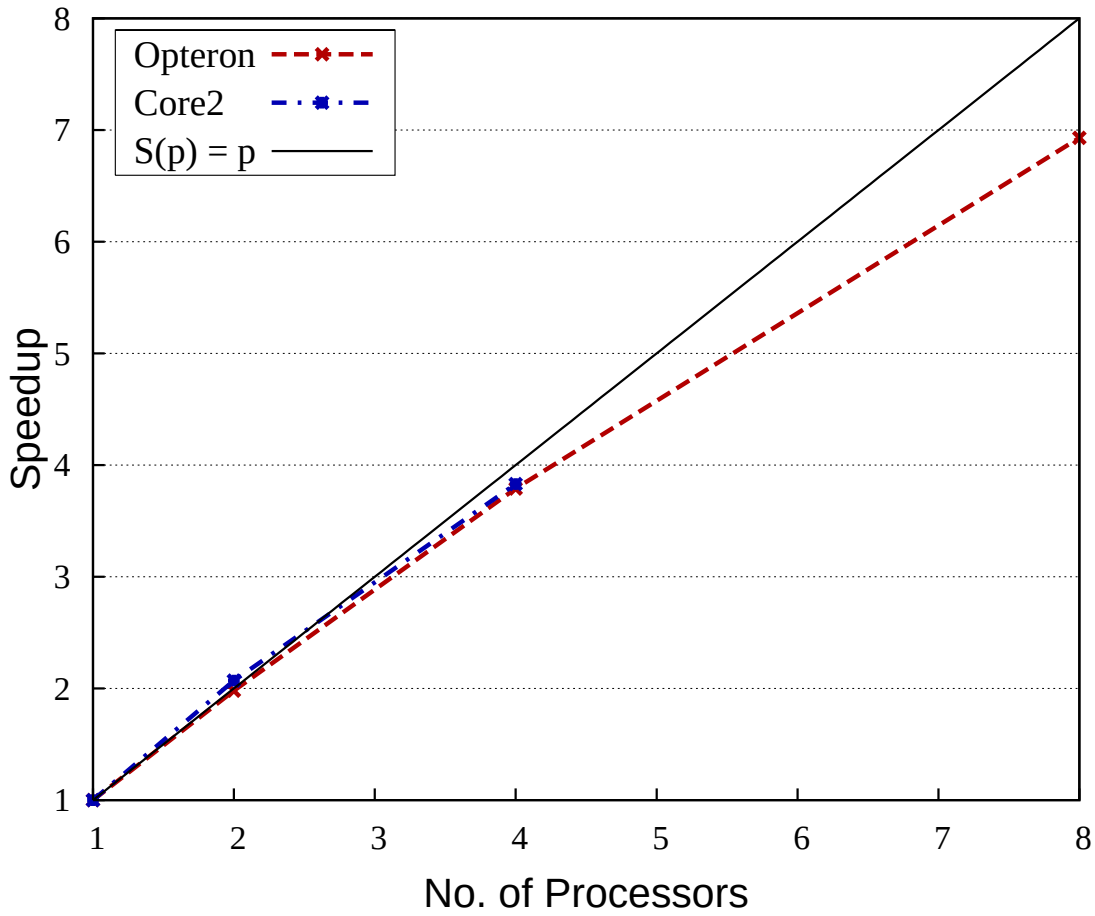
p	Opteron	Core2
1	899.1 s	895.8 s
2	455.5 s	448.0 s
4	236.3 s	234.0 s
8	128.4 s	



Matrix Construction

Numerical Experiments ($K^{1,2}, n = 8192, k = 8$)

p	**Opteron**	**Core2**
1	349.8 s	346.0 s
2	176.9 s	167.5 s
4	92.2 s	88.2 s
8	50.5 s	



Overview

Matrix Inversion

Let $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ be an $\mathcal{H}$-Matrix. The inverse of A can be computed as

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1} A_{12} S^{-1} A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} S^{-1} \\ -S^{-1} A_{21} A_{11}^{-1} & S^{-1} \end{pmatrix},$$

with the *Schur-complement*

$$S = A_{22} - A_{21} A_{11}^{-1} A_{12}.$$

Matrix Inversion

Let $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ be an $\mathcal{H}$-Matrix. The inverse of A can be computed as

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1} A_{12} S^{-1} A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} S^{-1} \\ -S^{-1} A_{21} A_{11}^{-1} & S^{-1} \end{pmatrix},$$

with the *Schur-complement*

$$S = A_{22} - A_{21} A_{11}^{-1} A_{12}.$$

Sequential Algorithm

procedure invert (A, C)
    invert(A_{00}, C_{00});
    $T_{12} := C_{11} \cdot A_{12}$; $T_{21} := A_{21} \cdot C_{11}$;
    $A_{22} := A_{22} - A_{21} \cdot T_{12}$;
    invert(A_{22}, C_{22});
    $C_{12} := -T_{12} \cdot C_{22}$; $C_{21} := -C_{22} \cdot T_{21}$;
    $C_{11} := C_{11} - T_{12} \cdot C_{21}$;
end;

Matrix Inversion

Let $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ be an $\mathcal{H}$-Matrix. The inverse of A can be computed as

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1} A_{12} S^{-1} A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} S^{-1} \\ -S^{-1} A_{21} A_{11}^{-1} & S^{-1} \end{pmatrix},$$

with the *Schur-complement*

$$S = A_{22} - A_{21} A_{11}^{-1} A_{12}.$$

Sequential Algorithm

procedure invert (A, C)
  invert(A_{00}, C_{00});
  $T_{12} := C_{11} \cdot A_{12}$; $T_{21} := A_{21} \cdot C_{11}$;
  $A_{22} := A_{22} - A_{21} \cdot T_{12}$;
  invert(A_{22}, C_{22});
  $C_{12} := -T_{12} \cdot C_{22}$; $C_{21} := -C_{22} \cdot T_{21}$;
  $C_{11} := C_{11} - T_{12} \cdot C_{21}$;
end;

Matrix Inversion

Let $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ be an $\mathcal{H}$-Matrix. The inverse of A can be computed as

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1} A_{12} S^{-1} A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} S^{-1} \\ -S^{-1} A_{21} A_{11}^{-1} & S^{-1} \end{pmatrix},$$

with the *Schur-complement*

$$S = A_{22} - A_{21} A_{11}^{-1} A_{12}.$$

Sequential Algorithm

procedure invert (A, C)
    invert(A_{00}, C_{00});
    $T_{12} := C_{11} \cdot A_{12}$; $T_{21} := A_{21} \cdot C_{11}$;
    $A_{22} := A_{22} - A_{21} \cdot T_{12}$;
    invert(A_{22}, C_{22});
    $C_{12} := -T_{12} \cdot C_{22}$; $C_{21} := -C_{22} \cdot T_{21}$;
    $C_{11} := C_{11} - T_{12} \cdot C_{21}$;
end;

Matrix Inversion

Complexity

Assumption: complexity of parallel matrix multiplication on p processors is

$$\mathcal{W}_{\text{Mul}}(n, p) = \mathcal{O}\left(\frac{n \log^2 n}{p}\right)$$

Matrix Inversion

Complexity

Assumption: complexity of parallel matrix multiplication on p processors is

$$\mathcal{W}_{\text{Mul}}(n, p) = \mathcal{O}\left(\frac{n \log^2 n}{p}\right)$$

Then, by using the parallel matrix multiplication for all products in the inversion algorithm, for the complexity of the $\mathcal{H}$-matrix inversion one gets

$$\mathcal{W}_{\text{Inv}}(n, p) = \mathcal{O}\left(n + \frac{n \log^2 n}{p}\right)$$

Matrix Inversion

Complexity

Assumption: complexity of parallel matrix multiplication on p processors is

$$\mathcal{W}_{\text{Mul}}(n, p) = \mathcal{O}\left(\frac{n \log^2 n}{p}\right)$$

Then, by using the parallel matrix multiplication for all products in the inversion algorithm, for the complexity of the $\mathcal{H}$-matrix inversion one gets

$$\mathcal{W}_{\text{Inv}}(n, p) = \mathcal{O}\left(n + \frac{n \log^2 n}{p}\right)$$

Remark: sequential part due to diagonal decreases for larger n.

Parallel Matrix Multiplication

To compute:

$$C := \alpha AB + \beta C$$

Parallel Matrix Multiplication

To compute:

$$C := \alpha AB + \beta C$$

Sequential Algorithm for a $m \times m$ block matrix:

```
procedure mul( α, A, B, β, C )
    if A, B and C are block matrices then
        for i := 1, ..., m do
            for j := 1, ..., m do
                for l := 1, ..., m do
                    mul( α, A_il, B_lj, β, C_ij );
    else
8:      C := αAB + βC;
end;
```

Matrix Inversion

Parallel Matrix Multiplication

To compute:

$$C := \alpha AB + \beta C$$

Sequential Algorithm for a $m \times m$ block matrix:

```
    procedure mul( α, A, B, β, C )
        if A, B and C are block matrices then
            for i := 1, ..., m do
                for j := 1, ..., m do
                    for l := 1, ..., m do
                        mul( α, A_il, B_lj, β, C_ij );
        else
8:          tcreate( C := αAB + βC );
    end;
```

First parallelisation ansatz: execute line 8 in a thread

Matrix Inversion

Parallel Matrix Multiplication

Consider

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} := \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} + \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

The parallel execution of

$$C_{11} = C_{11} + A_{11}B_{11} \quad \text{and} \quad C_{11} = C_{11} + A_{12}B_{21}.$$

leads to a collision

Matrix Inversion

Parallel Matrix Multiplication

Consider

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} := \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} + \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

The parallel execution of

$$C_{11} = C_{11} + A_{11}B_{11} \quad \text{and} \quad C_{11} = C_{11} + A_{12}B_{21}.$$

leads to a collision and has to be guarded by a mutex:

LOCK(m); $C_{11} := C_{11} + A_{11}B_{11}$; UNLOCK(m);	LOCK(m); $C_{11} := C_{11} + A_{12}B_{21}$; UNLOCK(m);

This blocks one of the threads and therefore enforces sequential execution.

Matrix Multiplication (2nd Try)

Simulate matrix multiplication to collect all products $A \cdot B$ for a destination block C and execute list of products for each C in a different thread.

Matrix Multiplication (2nd Try)

Simulate matrix multiplication to collect all products $A \cdot B$ for a destination block C and execute list of products for each C in a different thread.

procedure $\mathrm{sim\_mul}(\ A, B, C\ )$
  if A, B and C are block matrices **then**
    for $i, j, l := 1, \ldots, m$ **do**
      sim_mul(A_{il}, B_{lj}, C_{ij});
  else
    $\mathcal{P}(C) := \mathcal{P}(C) \cup \{(A, B)\}$; $\mathcal{L}_{\mathrm{MM}} := \mathcal{L}_{\mathrm{MM}} \cup \{C\}$;
end;

Matrix Multiplication (2nd Try)

Simulate matrix multiplication to collect all products $A \cdot B$ for a destination block C and execute list of products for each C in a different thread.

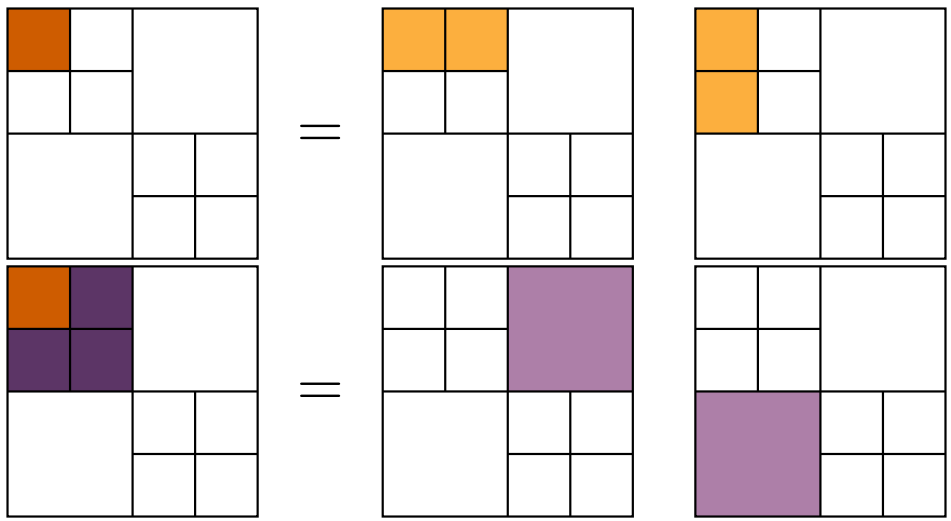
procedure sim_mul(A, B, C)
  if A, B and C are block matrices **then**
    for $i, j, l := 1, \ldots, m$ **do**
      sim_mul(A_{il}, B_{lj}, C_{ij});
  else
    $\mathcal{P}(C) := \mathcal{P}(C) \cup \{(A, B)\}$; $\mathcal{L}_{\text{MM}} := \mathcal{L}_{\text{MM}} \cup \{C\}$;
end;

procedure mul_block(C)
  for all $(A, B) \in \mathcal{P}(C)$ **do** $C := C + \alpha AB$;

procedure mul($\alpha, \beta, \mathcal{L}_{\text{MM}}$)
  for all $C \in \mathcal{L}_{\text{MM}}$ **do** $\mathcal{T} := \mathcal{T} \cup \{$ `tcreate`(mul_block(C)) $\}$;
  for $\texttt{t} \in \mathcal{T}$ **do** tsync(t);

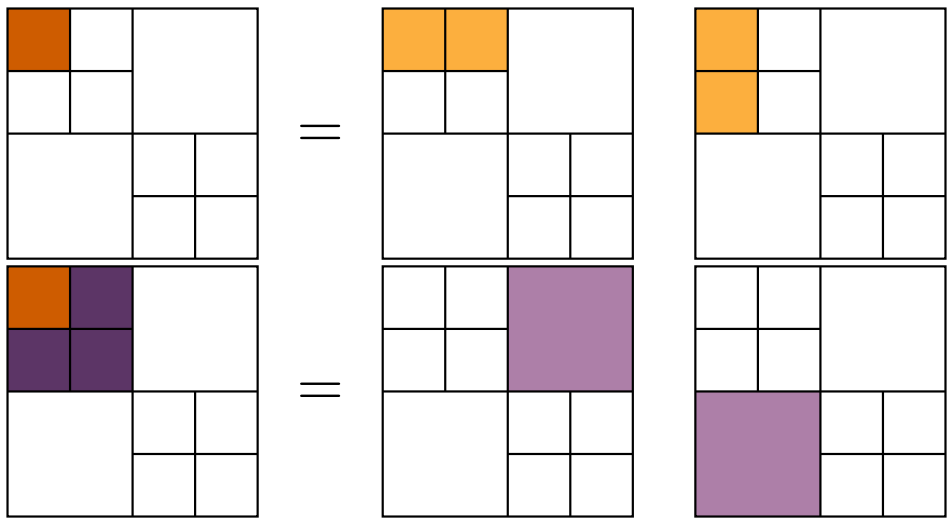
Matrix Multiplication (2nd Try)

Even in the second algorithm collisions can occur, e.g.



Matrix Multiplication (2nd Try)

Even in the second algorithm collisions can occur, e.g.



Therefore, all matrix blocks have to guarded by a mutex, e.g.

procedure mul_block(C)
  for all $(A, B) \in \mathcal{P}(C)$ **do**
    LOCK(m(C)); $C := C + \alpha AB$; UNLOCK(m(C));
end;

Matrix Multiplication

For moderate p, collisions are very seldom or non-existent at all, hence:

$$\mathcal{W}_{\text{Mul}}(n, p) = \mathcal{O}\left(\frac{n \log^2 n}{p}\right)$$

Matrix Inversion

Matrix Multiplication

For moderate p, collisions are very seldom or non-existent at all, hence:

$$\mathcal{W}_{\text{Mul}}(n, p) = \mathcal{O}\left(\frac{n \log^2 n}{p}\right)$$

Complexity of Matrix Inversion

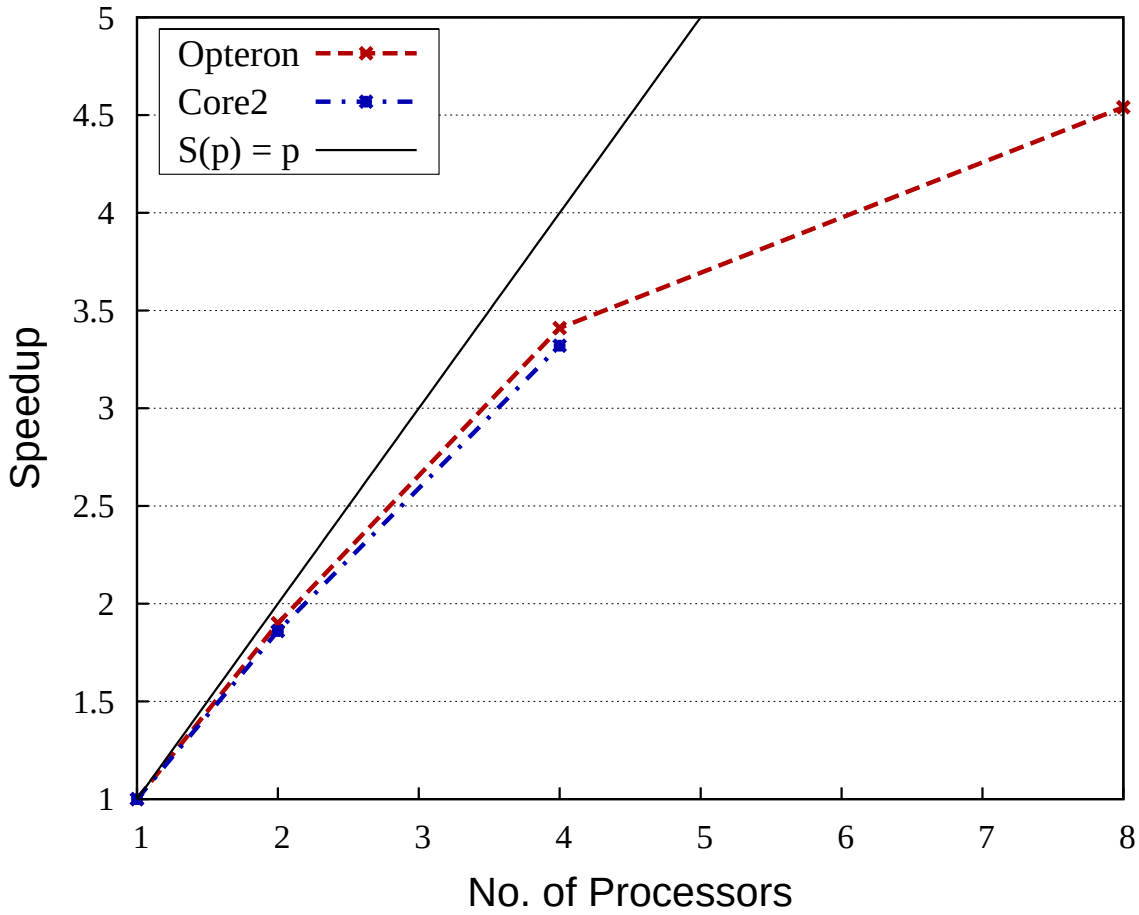
Since previous assumption about complexity of matrix multiplication was justified, the complexity of matrix inversion is truly

$$\mathcal{W}_{\text{Inv}}(n, p) = \mathcal{O}\left(n + \frac{n \log^2 n}{p}\right)$$

Matrix Inversion

Numerical Experiments ($K^{1,1}, n = 8192, k = 8$)

p	**Opteron**	**Core2**
1	221.2 s	217.1 s
2	116.6 s	116.9 s
4	64.8 s	65.4 s
8	48.7 s	



Remark about LU Factorisation

The same technique can be applied for the LU factorisation of an $\mathcal{H}$-Matrix, but in contrast to inversion, the recursion also takes place in the off-diagonal blocks (solving L and U). This increases the sequential part of the algorithm and one gets:

$$\mathcal{W}_{\mathrm{LU}}(n,p) = \mathcal{O}\left(n\log^2 n + \frac{n\log^2 n}{p}\right)$$

Remark about LU Factorisation

The same technique can be applied for the LU factorisation of an $\mathcal{H}$-Matrix, but in contrast to inversion, the recursion also takes place in the off-diagonal blocks (solving L and U). This increases the sequential part of the algorithm and one gets:

$$\mathcal{W}_{\mathrm{LU}}(n,p) = \mathcal{O}\left(n\log^2 n + \frac{n\log^2 n}{p}\right)$$

But for small p, the parallel speedup of the algorithm is nevertheless acceptable:

System	$p=1$	$p=2$	$p=4$	$p=8$
Opteron	84.9 sec	45.5 sec	37.2 sec	34.6 sec
Core2	85.6 sec	49.2 sec	38.0 sec	

Overview

Matrix-Vector Multiplication

For $\mathcal{H}$-Matrix A and vectors x and y compute

$$y := Ax$$

Matrix-Vector Multiplication

For $\mathcal{H}$-Matrix A and vectors x and y compute

$$y := Ax$$

Sequential Algorithm

Compute matrix-vector product for matrix blocks corresponding to leaves in block cluster tree:

for $(t, s) \in \mathcal{L}(T(I \times I))$ **do**
    $y|_t := y|_t + A|_{(t,s)} \cdot x|_s$;
endfor;

Matrix-Vector Multiplication

For $\mathcal{H}$-Matrix A and vectors x and y compute

$$y := Ax$$

Sequential Algorithm

Compute matrix-vector product for matrix blocks corresponding to leaves in block cluster tree:

for $(t, s) \in \mathcal{L}(T(I \times I))$ **do**
    $y|_t := y|_t + A|_{(t,s)} \cdot x|_s$;
endfor;

Parallelisation

Use same idea as in matrix construction: execute local matrix-vector multiplication in own thread.

procedure mulvec($A, x, y, (t, s)$)
    $y|_t := y|_t + A|_{(t,s)} \cdot x|_s$;

for $(t, s) \in \mathcal{L}(T(I \times I))$ **do** $\mathcal{T} := \mathcal{T} \cup \{$`tcreate`($A, x, y, (t, s)$)$\}$;
for `t` $\in \mathcal{T}$ **do** `tsync( t )`;

Matrix-Vector Multiplication

Collisions

If two threads update y, the access to y has to be guarded:

procedure $\mathrm{mulvec}(\ A, x, y, (t, s)\ )$
    LOCK(m); $y|_t := y|_t + A|_{(t,s)} \cdot x|_s$; UNLOCK(m);
end;

which prohibits parallel execution.

Matrix-Vector Multiplication

Collisions

If two threads update y, the access to y has to be guarded:

procedure $\mathrm{mulvec}(\ A, x, y, (t, s)\ )$
    LOCK(m); $y|_t := y|_t + A|_{(t,s)} \cdot x|_s$; UNLOCK(m);
end;

which prohibits parallel execution.

Parallelisation (2nd Try)

Use local result vectors y^i per processor $1 \leq i \leq p$:

procedure $\mathrm{mulvec}(\ A, x, y, (t, s)\ )$
    Let i be index of local processor;
    $y^i|_t := y^i|_t + A|_{(t,s)} \cdot x|_s$;
end;

for $(t, s) \in \mathcal{L}(T(I \times I))$ **do** $\mathcal{T} := \mathcal{T} \cup \{$`tcreate(` $A, x, y, (t, s)$ `)`$\}$;
for `t` $\in \mathcal{T}$ **do** `tsync( t )`;
$y := y + \sum_{i=1}^{p} y_i$;

Matrix-Vector Multiplication

Complexity

Again, automatic load balancing due to threads gives optimal parallel speedup. Together with sequential summation of p vectors one gets:

$$\mathcal{W}_{\text{Vec}}(n, p) = \mathcal{O}\left(\frac{n \log n}{p} + pn\right)$$

Matrix-Vector Multiplication

Complexity

Again, automatic load balancing due to threads gives optimal parallel speedup. Together with parallel summation of p vectors one gets:

$$\mathcal{W}_{\text{Vec}}(n, p) = \mathcal{O}\left(\frac{n \log n}{p} + n\right)$$

Matrix-Vector Multiplication

Complexity

Again, automatic load balancing due to threads gives optimal parallel speedup. Together with parallel summation of p vectors one gets:

$$\mathcal{W}_{\text{Vec}}(n,p) = \mathcal{O}\left(\frac{n \log n}{p} + n\right)$$

Remark

With better load balancing based on space-filling curves one can achieve

$$\mathcal{W}_{\text{Vec}}(n,p) = \mathcal{O}\left(\frac{n \log n}{p} + \frac{n}{\sqrt{p}}\right)$$

Matrix-Vector Multiplication

Complexity

Again, automatic load balancing due to threads gives optimal parallel speedup. Together with parallel summation of p vectors one gets:

$$\mathcal{W}_{\text{Vec}}(n,p) = \mathcal{O}\left(\frac{n \log n}{p} + n\right)$$

Remark

With better load balancing based on space-filling curves one can achieve

$$\mathcal{W}_{\text{Vec}}(n,p) = \mathcal{O}\left(\frac{n \log n}{p} + \frac{n}{\sqrt{p}}\right)$$

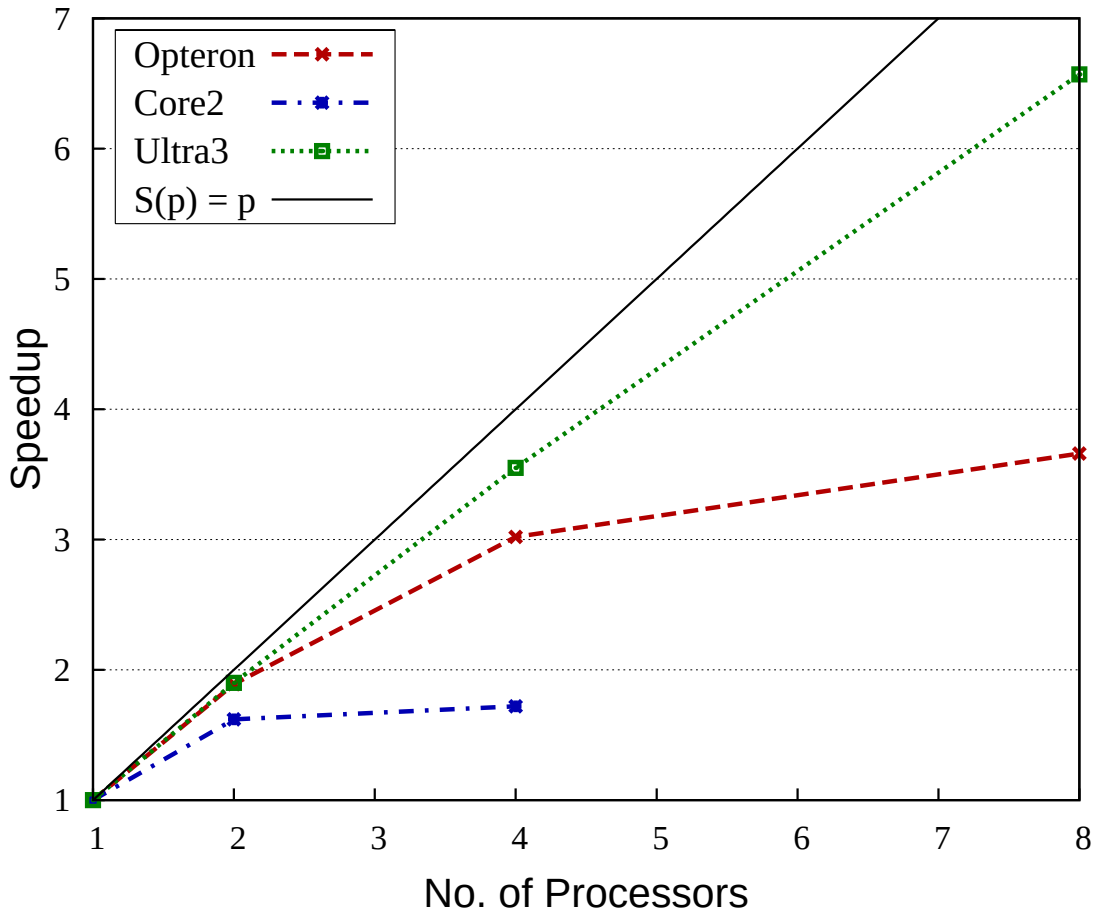
and with alternative data distribution

$$\mathcal{W}_{\text{Vec}}(n,p) = \mathcal{O}\left(\frac{n \log n}{p}\right)$$

Matrix-Vector Multiplication

Numerical Experiments ($n = 131072$)

p	Opteron	Core2
1	1.13 s	0.45 s
2	0.60 s	0.28 s
4	0.48 s	0.26 s
8	0.31 s	



Literature

R. Kriemann,

Parallele Algorithmen für $\mathcal{H}$-Matrizen,

Ph.D. Thesis, University of Kiel, 2005.

M. Bebendorf and R. Kriemann,

Fast Parallel Solution of Boundary Integral Equations and Related Problems,

Computing and Visualization in Science, 8(3–4):121–135, 2005.

R. Kriemann,

Parallel $\mathcal{H}$-Matrix Arithmetics on Shared Memory Systems,

Computing, 74:273–297, 2005.

L. Grasedyck, R. Kriemann and S. Le Borne,

Parallel Black Box Domain Decomposition Based $\mathcal{H}$-LU Preconditioning,

Preprint Nr. 115/2003, MPI MIS (submitted to "Numerische Mathematik").