Fast Parallel Solution of Boundary Integral Equations and Related Problems

Mario Bebendorf
University of Leipzig

bebendorf@math.uni-leipzig.de

joint work with R. Kriemann, MPI MIS, Leipzig

Overview

- ▷ *What are $\mathcal{H}$-matrices ?*
- ▷ *The ACA method*
- ▷ *parallel version of ACA* $\longrightarrow$ *building* $\mathcal{O}(n/p \log^* n)$
- ▷ *parallel matrix-vector multiplication* $\longrightarrow$ $\mathcal{O}(n/p \log^* n)$
- ▷ *numerical examples*

What are $\mathcal{H}$-matrices ?

Partition of the coefficient matrix into blocks

$$P = \{b = (t_1, t_2),\ t_1, t_2 \subset I\}, \quad I := \{1, \ldots, N\}$$

with pairwise disjoint P and

$$I \times I = \bigcup_{(t_1,t_2)\in P} t_1 \times t_2.$$

What are $\mathcal{H}$-matrices ?

Partition of the coefficient matrix into blocks

$$P = \{b = (t_1, t_2),\ t_1, t_2 \subset I\}, \quad I := \{1, \ldots, N\}$$

with pairwise disjoint P and

$$I \times I = \bigcup_{(t_1,t_2)\in P} t_1 \times t_2.$$

Blockwise low-rank approximation (M *has full rank !*)

$$\mathcal{H}(P,k) := \{M \in \mathbb{R}^{N\times N} : \operatorname{rank} M|_b \le k \text{ for all } b \in P\}$$

What are $\mathcal{H}$-matrices ?

Partition of the coefficient matrix into blocks

$$P = \{b = (t_1, t_2),\ t_1, t_2 \subset I\}, \quad I := \{1, \dots, N\}$$

with pairwise disjoint P and

$$I \times I = \bigcup_{(t_1,t_2)\in P} t_1 \times t_2.$$

Blockwise low-rank approximation (M *has full rank !*)

$$\mathcal{H}(P, k) := \{M \in \mathbb{R}^{N\times N} : \operatorname{rank} M|_b \le k \text{ for all } b \in P\}$$

Admissibility condition on a block $b = (t_1, t_2)$:

$$\min\{\operatorname{diam} X_{t_1}, \operatorname{diam} X_{t_2}\} \le \eta \operatorname{dist}(X_{t_1}, X_{t_2}), \quad 0 < \eta < 1$$

or $\min\{\#t_1, \#t_2\} = 1$, where $X_t := \bigcup_{i\in t} \operatorname{supp} \varphi_i$.
Number of generated blocks is $\mathcal{O}(\eta^{-2(d-1)} N \log N)$.

The ACA-Algorithm (Building)

Focus on a single admissible block $A \in \mathbb{R}^{m\times n}$:

```
Let $k = 1$;   $Z = \varnothing$
repeat
    if $k > 1$ then $i_k := \operatorname{argmax}_{i\notin Z} |(u_{k-1})_i|$
    else $i_k := \min\{1,\ldots,m\} \setminus Z$
    $\tilde{v}_k := a_{i_k,1:n} - \sum_{\ell=1}^{k-1} (u_\ell)_{i_k} v_\ell$
    $Z := Z \cup \{i_k\}$
    if $\tilde{v}_k$ does not vanish then
        $j_k := \operatorname{argmax}_{j=1,\ldots,n} |(\tilde{v}_k)_j|$;     $v_k := (\tilde{v}_k)_{j_k}^{-1} \tilde{v}_k$
        $u_k := a_{1:m,j_k} - \sum_{\ell=1}^{k-1} (v_\ell)_{j_k} u_\ell$.
        $k := k + 1$
    endif
until the following stopping criterion is fulfilled
```

$$\|u_k\|_2 \, \|v_k\|_2 < \varepsilon \, \| \sum_{\ell=1}^{k-1} u_\ell v_\ell^T \|_F .$$

Convergence proof exists (Beb. ’99 / ’00, Beb. & Rjasanow ’03) for

- ▷ *Nyström matrices:*

$$a_{ij} = \kappa(y_i, y_j)$$

- ▷ *collocation matrices:*

$$a_{ij} = \int_\Gamma \kappa(x, y_i)\varphi_j(x)\,\mathrm{d}s_x.$$

- ▷ *Radiation heat transfer:*

$$a_{ij} = \int_{\Gamma_i}\int_{\Gamma_j} s(x,y)\frac{(n_x, x-y)\,(n_y, y-x)}{|x-y|^4}\,\mathrm{d}s_x\,\mathrm{d}s_y.$$

Convergence proof exists (Beb. ’99 / ’00, Beb. & Rjasanow ’03) for

- ▷ *Nyström matrices:*

$$a_{ij} = \kappa(y_i, y_j)$$

- ▷ *collocation matrices:*

$$a_{ij} = \int_\Gamma \kappa(x, y_i)\varphi_j(x)\,\mathrm{d}s_x.$$

- ▷ *Radiation heat transfer:*

$$a_{ij} = \int_{\Gamma_i}\int_{\Gamma_j} s(x,y)\frac{(n_x, x-y)\,(n_y, y-x)}{|x-y|^4}\,\mathrm{d}s_x\,\mathrm{d}s_y.$$

Theorem: Let (X_s, X_t) be an admissible pair of domains and κ be an asymptotically smooth kernel. In the case of Galerkin matrices

$$a_{ij} = \int_\Gamma\int_\Gamma \kappa(x,y)\varphi_j(x)\varphi_i(y)\,\mathrm{d}s_x\,\mathrm{d}s_y, \quad i = 1,\dots,m,\ j = 1,\dots,n$$

for $|Z| \ge n_p$ it holds that

$$|(R_k)_{ij}| \le c\,\mathrm{dist}^g(X_s, X_t)\,\|\varphi_i\|_{L^1}\,\|\varphi_j\|_{L^1}\,\eta^p, \quad 0 < \eta < \frac{1}{d}.$$

Generating the $\mathcal{H}$-matrix approximant

Sequential computation of an $\mathcal{H}$-matrix approximant:

```
for all $b \in P$ do
    if $b$ is admissible then
        create low-rank matrix using ACA
    else
        create a dense matrix
    endif
endfor
```

Generating the $\mathcal{H}$-matrix approximant

Sequential computation of an $\mathcal{H}$-matrix approximant:

```
for all $b \in P$ do
    if $b$ is admissible then
        create low-rank matrix using ACA
    else
        create a dense matrix
    endif
endfor
```

- ✓ *Computation in both cases is fully independent → can be done in parallel*
- ✗ *for load balancing, prior knowledge of the amount of work per block is needed*
- ✗ *ACA is adaptive → no apriori info about cost for block.*

Alternative to cost-related load balancing: *list scheduling*.

```
for all $b \in P$ do
    let $0 \leq i < p$ be the number of the first idle processor
    if $b$ is admissible then
        create a low-rank matrix using ACA on processor $i$
    else
        create a dense matrix on processor $i$
    endif
endfor
```

Alternative to cost-related load balancing: *list scheduling*.

```
for all $b \in P$ do
    let $0 \leq i < p$ be the number of the first idle processor
    if $b$ is admissible then
        create a low-rank matrix using ACA on processor $i$
    else
        create a dense matrix on processor $i$
    endif
endfor
```

Guaranteed parallel efficiency:

Let

$t(p)$	time for n jobs on p processors using list scheduling
$t_{\min}(p)$	minimal time needed for n jobs on p processors,

then

$$t(p) \leq \left(2 - \frac{1}{p}\right) t_{\min}(p).$$

Shared Memory Systems

Widely used on shared memory systems: *threads*.

- ▷ *share same address space → no communication between processors*
- ▷ *distribution of threads among processors by operating system*

Shared Memory Systems

Widely used on shared memory systems: *threads*.

- ▷ *share same address space → no communication between processors*
- ▷ *distribution of threads among processors by operating system*

Standard interface: *POSIX*-threads

- ✘ *complicated*
- ✘ *creation of Pthreads expensive → only a pool of p threads started*
- ✘ *user interface: C++ class*

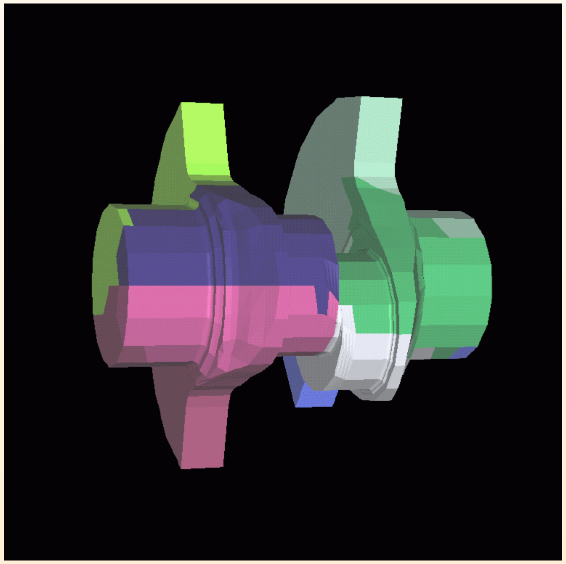
```
class ThreadPool {
    init ( p ∈ ℕ );
    run ( Job j );
    sync ( Job j );
    sync_all ();
}
```

```
procedure build_block($b$)
    if $b$ is admissible then
        build low-rank matrix using ACA
    else
        build a dense matrix
    endif
end

ThreadPool->init(p)
for all $b \in P$ do
    ThreadPool->run( build_block($b$) )
endfor
ThreadPool->sync_all()
```

Results on SunFire 6800 (24 Proc, 96 GB):

n	$p=1$	$p=4$	$p=8$	$p=12$	$p=16$
$4\,416$	54.4 s	13.7 s	6.9 s	4.6 s	3.6 s
$16\,128$	177.0 s	44.6 s	22.5 s	15.3 s	11.8 s
$89\,412$	2097.9 s	528.7 s	271.6 s	180.7 s	139.3 s



Parallel efficiency

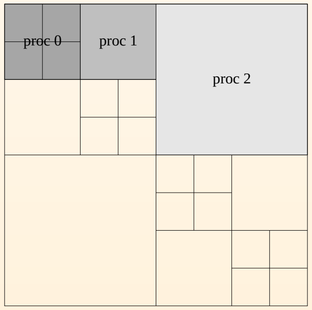
$$E_{\text{par}} = \frac{t(1)}{p \cdot t(p)}$$

n	$p=4$	$p=8$	$p=12$	$p=16$
$4\,416$	99.3%	98.6%	98.6%	94.4%
$16\,128$	99.2%	98.3%	96.4%	93.8%
$89\,412$	99.2%	96.6%	96.7%	94.1%

Parallel Matrix-Vector Multiplication

Aim: Calculate $y := Ax$ on p processors, where A is an $\mathcal{H}$-matrix, with $\mathcal{O}(n/p \log^* n)$ complexity.

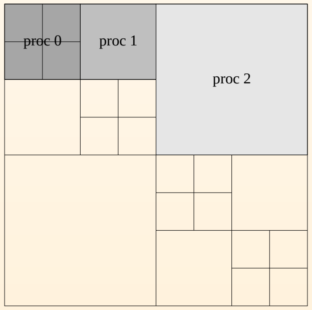
Naive approach: distribute the blocks among the processors
Problem: processors write to the same part of y
$\rightarrow$ p temporary vectors of length $\geq n/p \Rightarrow \mathcal{O}(n)$ complexity



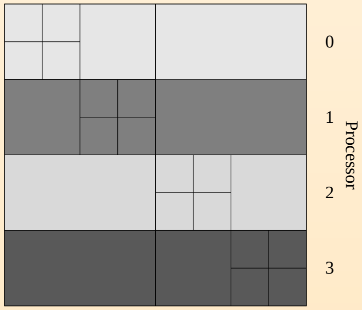
Parallel Matrix-Vector Multiplication

Aim: Calculate $y := Ax$ on p processors, where A is an $\mathcal{H}$-matrix, with $\mathcal{O}(n/p \log^* n)$ complexity.

Naive approach: distribute the blocks among the processors
Problem: processors write to the same part of y
$\rightarrow$ p temporary vectors of length $\geq n/p \Rightarrow \mathcal{O}(n)$ complexity



Solution: each processor writes to a private part of y
large blocks UV^T are split among different processors
$z = V^T x$ does not have to be calculated on all processors sharing a block (calculate beforehand)



MV-Algorithm

▷ *partition* y *(use sequence partitioning)*

▷ *calculate* $z := V^T x$ *for all shared blocks (use LPT-scheduling)*

▷ *matrix-vector multiplications in each stripe*

How to partition y ?

Partition y so that cost is minimal.
Let $c_P(b)$ be given and let $i \in I$. Define

$$\tilde{c}(i) = \sum_{i \in t : (s,t) \in P} c_P((s,t))|_i$$

and for $t \in T \setminus \mathcal{L}(T)$

$$c_I(t) = \sum_{t' \in S(t)} c_I(t')$$

where $c_I(t) = \sum_{i \in t} \tilde{c}(i)$ for $t \in \mathcal{L}(T)$.
For $S \subset T_I$ let $c_I(S) = \sum_{t \in S} c_I(t)$.

How to partition y ?

Partition y so that cost is minimal.
Let $c_P(b)$ be given and let $i \in I$. Define

$$\tilde{c}(i) = \sum_{i \in t : (s,t) \in P} c_P((s,t))|_i$$

and for $t \in T \setminus \mathcal{L}(T)$

$$c_I(t) = \sum_{t' \in S(t)} c_I(t')$$

where $c_I(t) = \sum_{i \in t} \tilde{c}(i)$ for $t \in \mathcal{L}(T)$.
For $S \subset T_I$ let $c_I(S) = \sum_{t \in S} c_I(t)$.

Sequence partitioning:

$$\{1, \ldots, N\} = \bigcup_{i=1}^{p} S_i, \qquad S_i := \{r_{i-1}, \ldots, r_i\}$$

where $1 = r_0 \leq r_1 \leq \cdots \leq r_p = N$.

Sequence partitioning optimal if $\max_{0 \leq i < p} c_I(S_i)$ minimal.

Shared blocks

Longest-Process-Time (LPT) scheduling for shared low-rank blocks: job with maximal cost to the processor with lowest load

Guaranteed:

$$t(p) \leq \left(\frac{4}{3} - \frac{1}{3p}\right) t_{\min}(p).$$

Results: Time for 100 MV multiplications

n	$p = 1$	$p = 4$	$p = 8$	$p = 12$	$p = 16$
$4\,416$	18.8s	5.1s	2.7s	1.8s	1.4s
$16\,128$	62.2s	17.3s	8.9s	6.1s	4.6s
$89\,412$	664.0s	183.1s	93.5s	64.7s	49.4s

n	$p = 4$	$p = 8$	$p = 12$	$p = 16$
$4\,416$	92.1%	87.0%	85.0%	84.0%
$16\,128$	89.9%	87.4%	85.2%	84.1%
$89\,412$	90.7%	88.8%	85.5%	84.0%