Parallel Algorithms for $\mathcal{H}$-matrices

- Definitions
- Load balancing
- Matrix-Vector-Multiplication
- Matrix-Multiplication
- Matrix-Inversion

Definitions

- Let I be an indexset,
- T_I is a clustertree over I,
- $T_{I\times I}$ is a block-clustertree defined by T_I and some adm. criterion.
- for all nodes $t \in T_{I\times I}$ let $S(t)$ be the sons of t,
- $L(T_{I\times I})$ denotes the set of leafs of $T_{I\times I}$ with $N = |L(T_{I\times I})|$
- $p \geq 1$ is the number of processors with $p \ll N$

Goal

- parallelise building of $\mathcal{H}$-matrix and $\mathcal{H}$-matrix-arithmetics
- memory and computation should be balanced among processors
- algorithms should support wide range of machines (shared- and distributed-memory)

Load-Balancing

- data-based balancing:
  - let $c : T_{I\times I} \rightarrow \mathbb{R}_+$ be a cost-function,
  - we are looking for a function

$$d : T_{I\times I} \rightarrow \{-1, 0, \ldots, p-1\}$$

    such that

$$\max_{i<p} \sum_{t \in L(T_{I\times I})\,:\,d(t)=i} c(t)$$

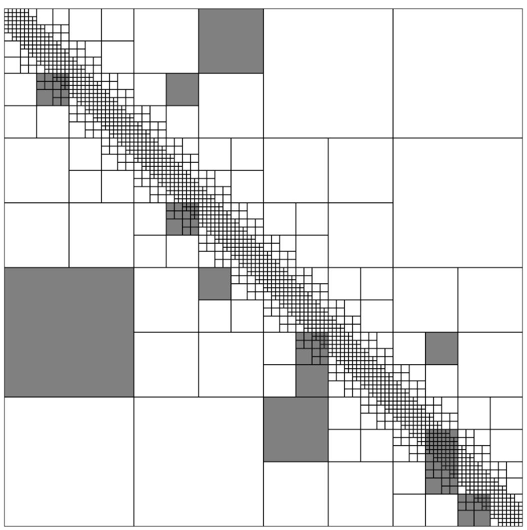
    is minimised.
  - restriction: $\forall\, t \in T_{I\times I}, t' \in S(t) : d(t) \neq -1 \implies d(t') = d(t)$
- computation-based balancing:
  - cost-function c defined over operations
  - same min-max-problem
  - problem: definition of cost-function

Scheduling-Algorithms

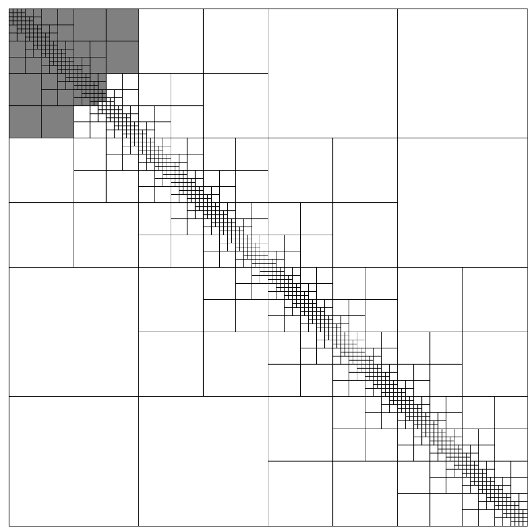
- LPT-list-scheduling :
  - works on *set* of items (matrix-blocks, operations),
  - order items according to costs
  - schedule items in ordered list to processor with minimal costs,
  - $\left(\frac{4}{3} - \frac{1}{3p}\right)$-approximation scheme,
  - complexity: $\mathcal{O}(n \log n + np)$, where n is the number of items
- Sequence-partitioning :
  - works on *sequence* (array) of items,
  - partition sequence into sub-sequences such that costs of most expensive sub-sequence is minimised,
  - problem can be solved in $\mathcal{O}(np)$
  - can be approximated by greedy-algorithm in less time
  - in data-based balancing: with correct ordering of blocks, we have data-locality

Examples for Data-Based-Balancing

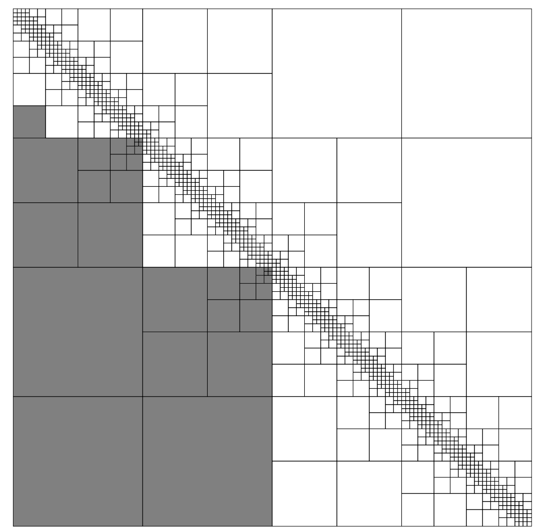
LPT



Sequence-part. 1



Sequence-part. 2



Matrix-Vector-Multiplication

- want to compute $y = Ax$, where A is a $\mathcal{H}$-matrix,
- for all leafs $T \in L(T_{I\times I})$ let $M_A(t)$ denote the corresponding matrix-block in A,
- Algorithm for processor i:

> each proc has copy of x and y;
> **for all** $t \in L(T_{I\times I})$ **do**
>   **if** $d(t) = i$ **then** multiply $M_A(t)$ with x;
> scatter local copy to all processors;
> add copies from other processors to local copy;

- not perfect scaling because whole vector must be transferred and added on each processor
- solution: with data-locality, only store part of vector on each processor

Matrix-Matrix-Multiplication

- Want to compute $C = AB$, where A, B and C are $\mathcal{H}$-matrices.

Shared-Memory

1. data-based balancing:
   - distribute destination-matrix C
   - use serial algorithm on each processor but only work on local part of block-clustertree
2. computation-based balancing:
   - simulate multiplication (store list of block-multiplications)
   - distribute list of block-multiplications with the restriction that each dest-block is only on one processor
   - do block-multiplications assigned to each processor
   - problems: definition of cost-function, how to handle blocked destination-blocks

Numerical Results

- partitioning from 2D-FEM on unit-square, constant rank 5, constant filling
- times in seconds

dof / p	1	2	3	4	5	6	7	8
4096	249.6	123.8 123.3 (2.0/2.0)	86.0 83.5 (2.9/3.0)	66.7 63.6 (3.7/3.9)	54.3 50.5 (4.6/4.9)	44.9 43.0 (5.5/5.8)	39.0 36.1 (6.4/6.9)	37.1 28.7 (6.7/8.7)
16384	2175.5	1465.4 1106.4 (1.5/2.0)	751.6 725.2 (2.9/3.0)	576.3 573.0 (3.8/3.8)	468.6 444.2 (4.6/4.9)	410.9 375.1 (5.3/5.8)	348.7 314.2 (6.2/6.9)	309.5 265.1 (7.0/8.2)

Distributed Memory

- Problem: how to handle communication and when to synchronise
- *BSP*-computation:
  - one step of a BSP-computation:
    - local computation
    - communication
    - synchronisation
- BSP-algorithm for matrix-multiplication:
  - use computation-based balancing
  - simulate multiplication
  - partition local operations into steps (balanced among processors)
  - step:
    - transfer matrix-blocks needed for local operations
    - do multiplications
    - transfer results
- algorithm allows to control amount of communication by choosing number of steps

Matrix-Inversion

Assume following structure of $\mathcal{H}$-matrices:

$$A = \left(\begin{array}{c|c} A_0 & A_1 \\ \hline A_2 & A_3 \end{array} \right).$$

Using the Schur-complement the inverse of A is:

$$A^{-1} = \left(\begin{array}{c|c} A_0^{-1} + A_0^{-1} A_1 S^{-1} A_2 A_0^{-1} & -A_0^{-1} A_1 S^{-1} \\ \hline -S^{-1} A_2 A_0^{-1} & S^{-1} \end{array} \right)$$

with

$$S = A_3 - A_2 A_0^{-1} A_1.$$

The algorithm for inversion is:

procedure invert(A, C, T)
  if A is dense **then** $C = A^{-1}$;
  else
    invert(A_0, C_0, T_0);
    $T_1 = C_0 A_1; T_2 = A_2 C_0$; { *mult. in parallel* }
    $A_3 = A_3 - A_2 T_1$;
    invert(A_3, C_3, T_3); { *build Schur-compl.* }
    $C_1 = -T_1 C_3; C_2 = -C_3 T_2$; { *mult. in parallel* }
    $C_0 = C_0 - T_1 C_2$;
endif; **end**;

- only minimal internal parallelity
- therefore only multiplications can be parallelised
- ***needs fast online-scheduling***
- problem: cost-function for matrix-multiplication is expansive