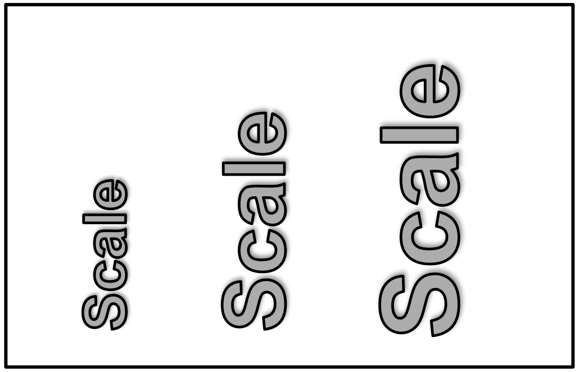
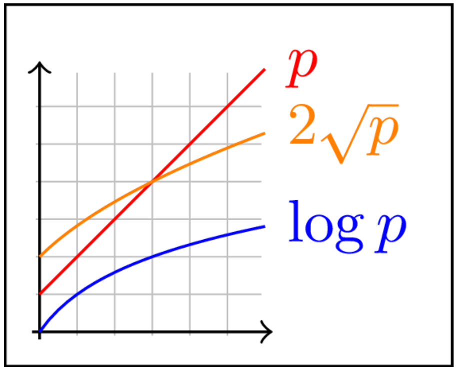
SPP (WiSe 17/18): MPI Laboratory

Sergei Shudler , Sebastian Rinke





Matrix-matrix multiplication – overview

- Lab purpose: Implement two parallel matrix-matrix multiplication algorithms and analyze the performance of the first one
- Input: Matrices A, B with size *n* x *n*

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix} \quad B = \begin{pmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-1,0} & b_{n-1,1} & \cdots & b_{n-1,n-1} \end{pmatrix}$$

- Output: Matrix C of size *n* x *n* such that:

$$C = \begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,n-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & \cdots & c_{n-1,n-1} \end{pmatrix} \qquad c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} \cdot b_{k,j}$$

General remarks

- We provide header files and source templates
  - You need to implement specific functions

- Makefile and job script is provided

- Implementation has to be in C

Tasks

- Task 1 – Simple (parallel) matrix-matrix multiplication
- Task 2 – Distributed block-based matrix-matrix multiplication
- Task 3 – Performance analysis and modeling

Task 1 – Simple multiplication (1)

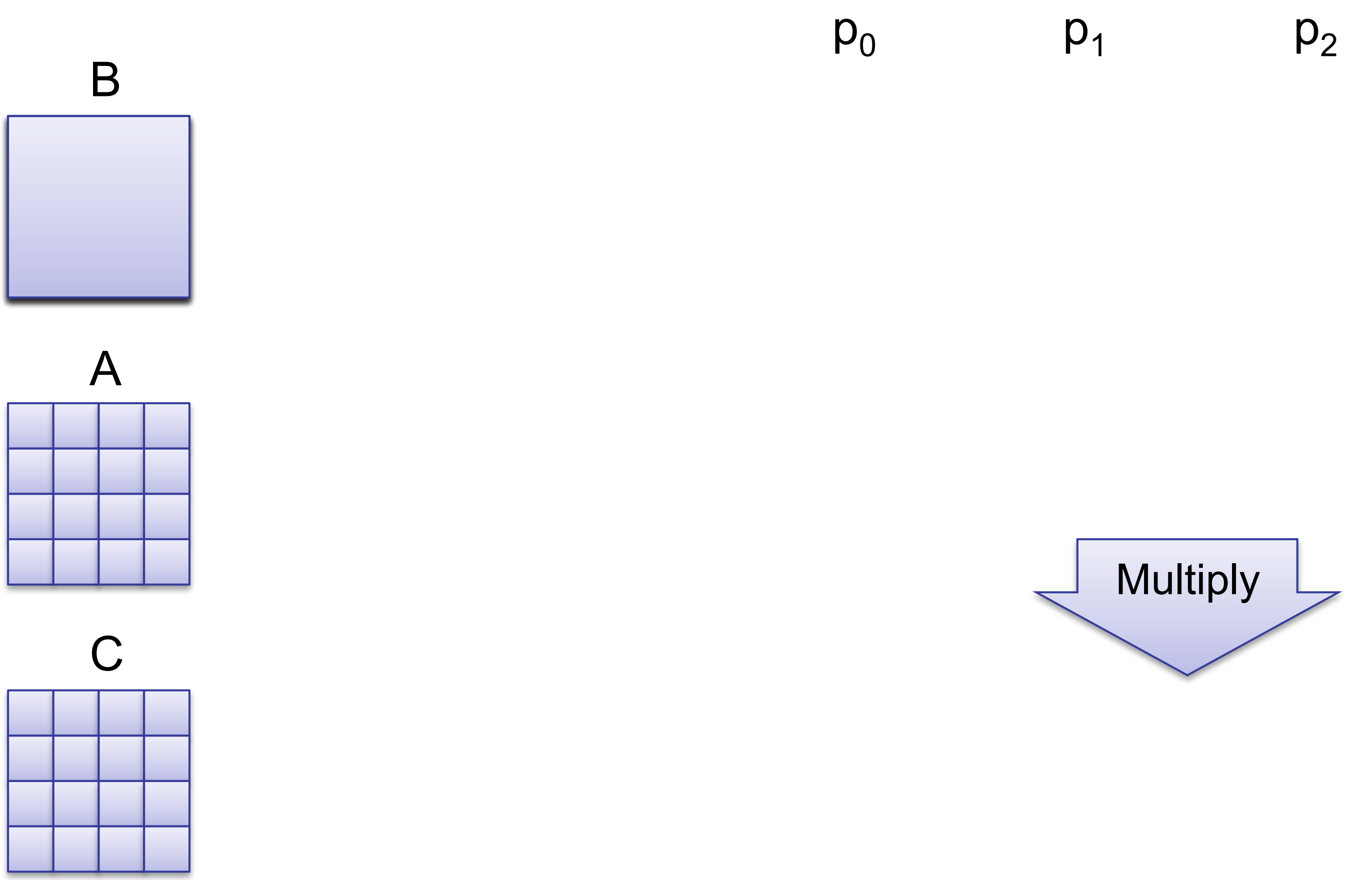
- Assume you have *p* process and *n* divisible by *p*
- Only two matrices can fit into the memory of a single process

- Observation:

$$\begin{pmatrix} a_{i,0} & a_{i,1} & \cdots & a_{i,n-1} \\ a_{i+1,0} & a_{i+1,1} & \cdots & a_{i+1,n-1} \\ a_{i+2,0} & a_{i+2,1} & \cdots & a_{i+2,n-1} \end{pmatrix} \cdot \begin{pmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-1,0} & b_{n-1,1} & \cdots & b_{n-1,n-1} \end{pmatrix} = \begin{pmatrix} c_{i,0} & c_{i,1} & \cdots & c_{i,n-1} \\ c_{i+1,0} & c_{i+1,1} & \cdots & c_{i+1,n-1} \\ c_{i+2,0} & c_{i+2,1} & \cdots & c_{i+2,n-1} \end{pmatrix}$$

Task 1 – Simple multiplication (2)

- Solution strategy:



Task 1 – Simple multiplication (3)

- Implement results verification:
  - Multiply matrices locally at the root process and compare with the result of the solution
  - Ignore the memory limitation in the verification

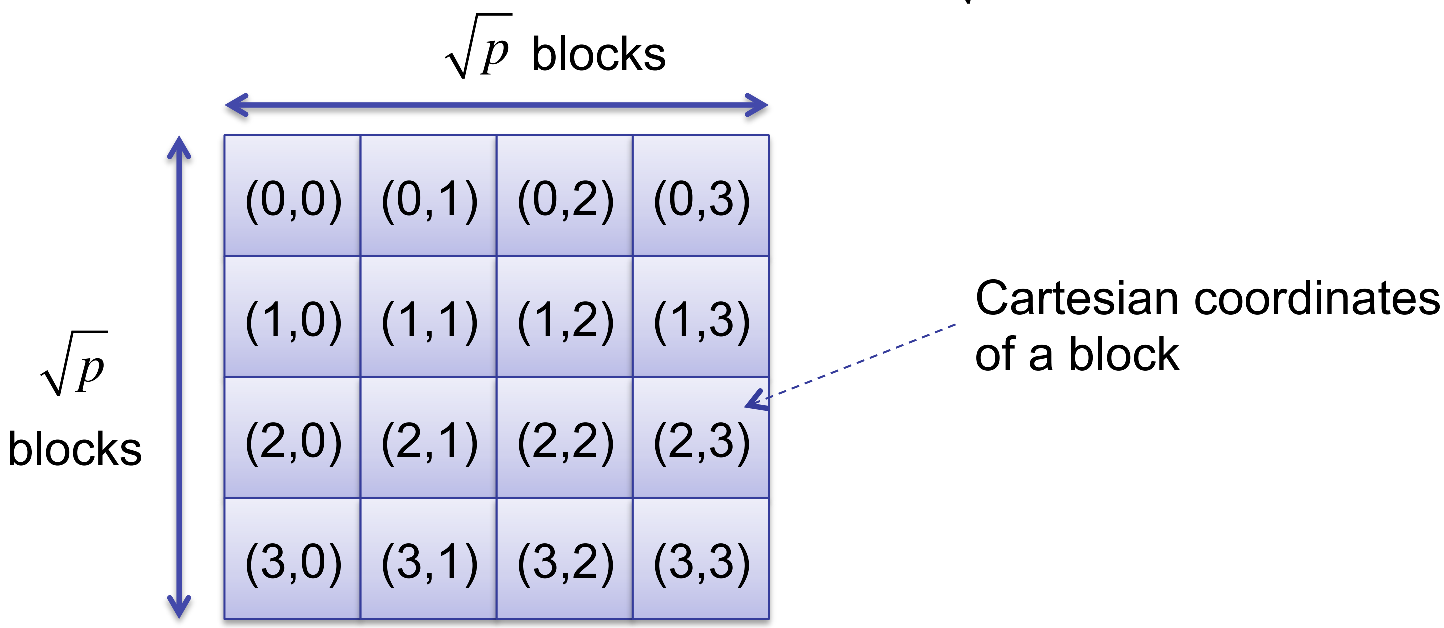
- Template: task1.c

Tasks

- Task 1 – Simple (parallel) matrix-matrix multiplication
- Task 2 – Distributed block-based matrix-matrix multiplication
- Task 3 – Performance analysis and modeling

Task 2 – Distributed block-based multiplication (1)

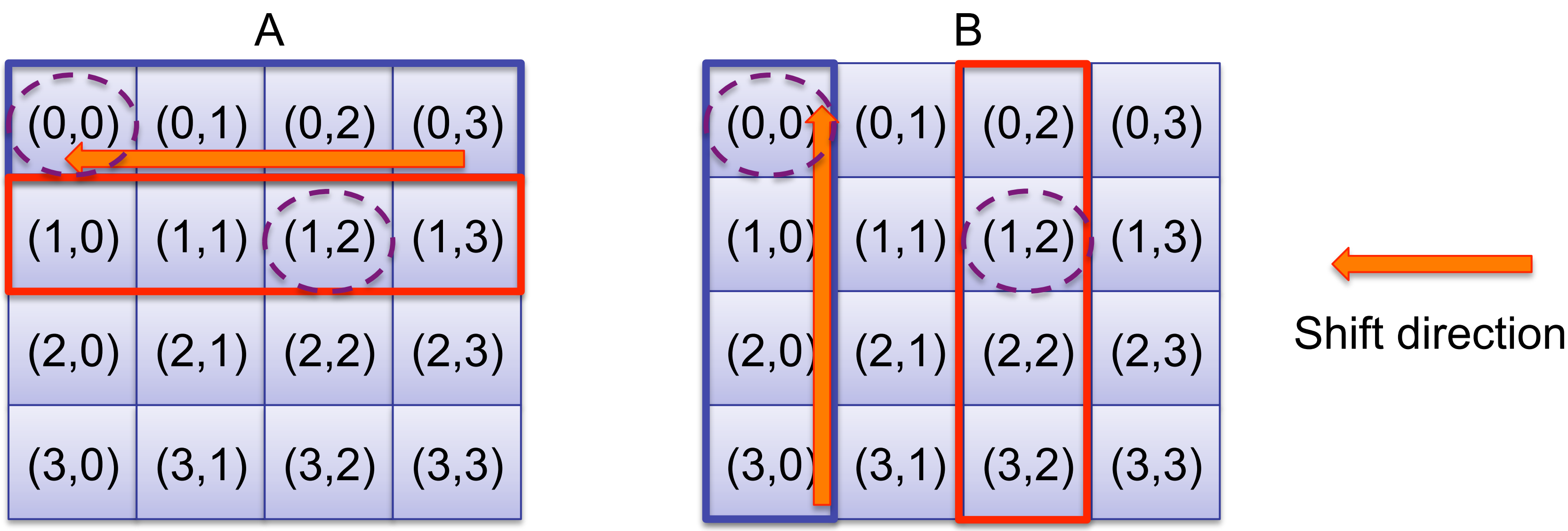
- Matrices too big to fit in memory, so each process has just one block of A, B, and C
- Assume n is divisible by $\sqrt{p}$, size of a block $q = \frac{n}{\sqrt{p}}$



Task 2 – Distributed block-based multiplication (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Strategy: shift blocks in circular manner, multiply, and accumulate the result in a C block



C(0,0) = A(0,0)*B(0,0) + A(0,1)*B(1,0) + A(0,2)*B(2,0) + A(0,3)*B(3,0)

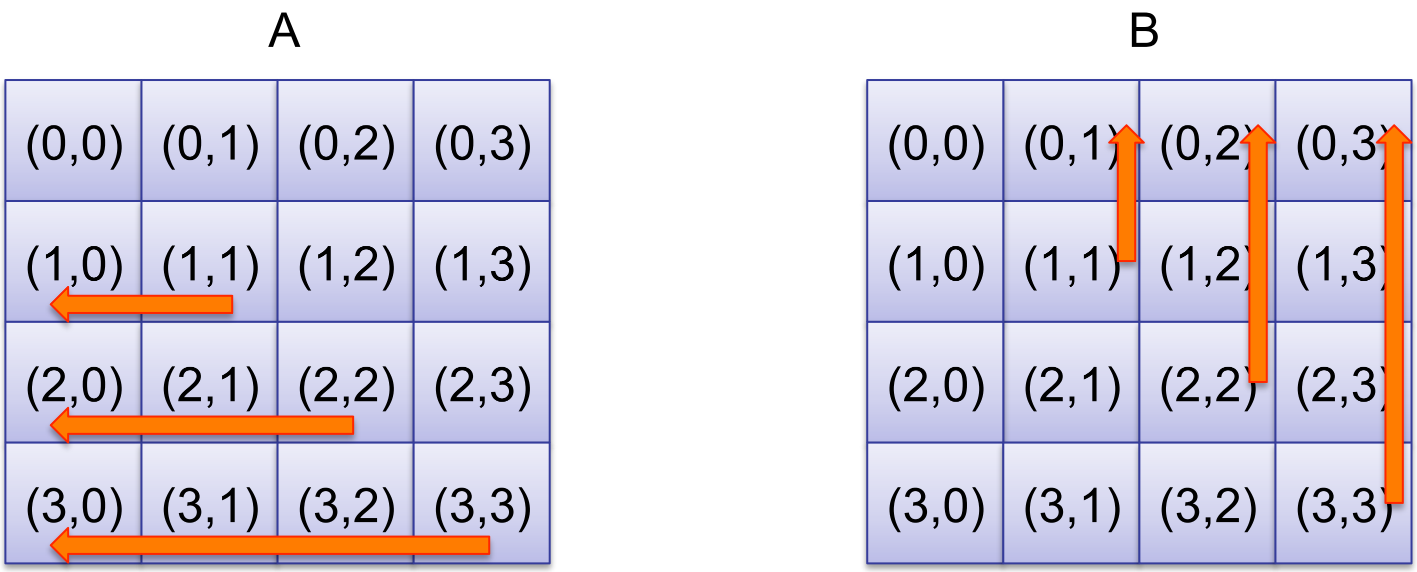
C(1,2) = A(1,2)*B(1,2) + A(1,3)*B(2,2) + A(1,0)*B(3,2) + A(1,1)*B(0,2) Incorrect

C(1,2) = A(1,0)*B(0,2) + A(1,1)*B(1,2) + A(1,2)*B(2,2) + A(1,3)*B(3,2) Correct

Task 2 – Distributed block-based multiplication (3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Solution: matrices A and B should be first skewed



Task 2 – Distributed block-based multiplication (4)

TECHNISCHE UNIVERSITÄT DARMSTADT

- The result:

A

(0,0)	(0,1)	(0,2)	(0,3)
(1,1)	(1,2)	(1,3)	(1,0)
(2,2)	(2,3)	(2,0)	(2,1)
(3,3)	(3,0)	(3,1)	(3,2)

B

(0,0)	(1,1)	(2,2)	(3,3)
(1,0)	(2,1)	(3,2)	(0,3)
(2,0)	(3,1)	(0,2)	(1,3)
(3,0)	(0,1)	(1,2)	(2,3)

C(1,2) = A(1,0)*B(0,2) + A(1,1)*B(1,2) + A(1,2)*B(2,2) + A(1,3)*B(3,2) **Correct (original)**

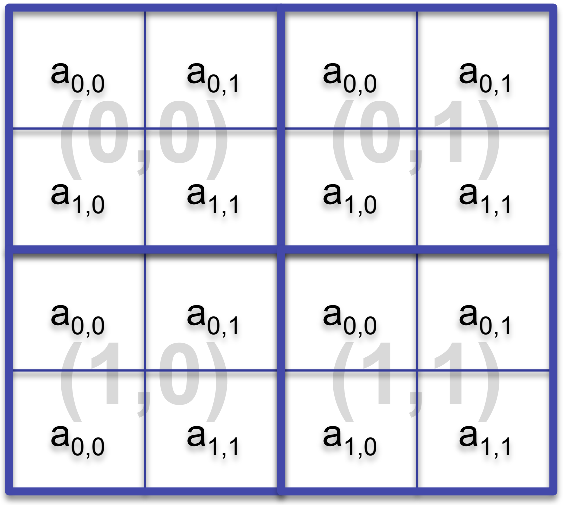
C(1,2) = A(1,3)*B(3,2) + A(1,0)*B(0,2) + A(1,1)*B(1,2) + A(1,2)*B(2,2) **Correct (shifted)**

- Now it works for every block!

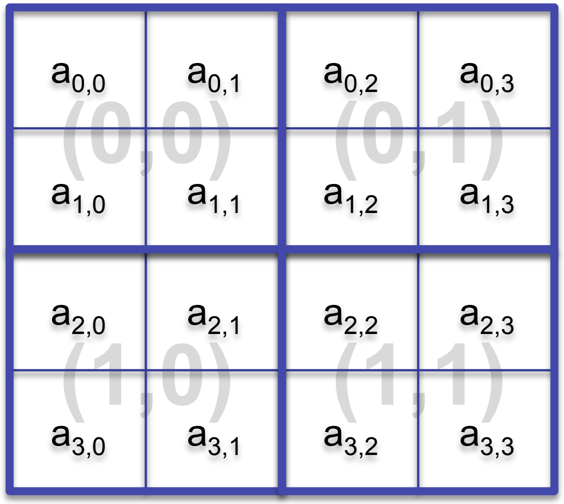
Task 2 – Distributed block-based multiplication (5)

- Matrix A and B should be initialized using global coordinates:

Local coordinates



Global coordinates



- Using global coordinates, each entry should be:

  $a_{I,J} = b_{I,J} = I * n + J$

Task 2 – Distributed block-based multiplication (6)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Initial value in A and B: $a_{I,J} = b_{I,J} = I * n + J$
- Multiplication result: $c_{I,J} = \sum_{k=0}^{n-1} (I \cdot n + k) \cdot (k \cdot n + J)$
- Solve the equation and get the expected value

Sum of arithmetic series

$$c_{I,J} = \sum_{k=0}^{n-1} (In + k)(kn + J) = \sum_{k=0}^{n-1} (In^2 k + IJn + nk^2 + Jk) = In^2 \sum_{k=0}^{n-1} k + IJn^2 + n \sum_{k=0}^{n-1} k^2 + J \sum_{k=0}^{n-1} k$$

Sum of sequence of squares

- Validation – compare result to the expected value without using additional memory
- Due to floating point errors, works for at most n = 1024

Task 2 – Distributed block-based multiplication (7)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Details are in the exercise description:
  - MPI functions to work with Cartesian topologies
  - Function for local matrix-matrix multiplication provided

- Template: task2.c

Tasks

- Task 1 – Simple (parallel) matrix-matrix multiplication
- Task 2 – Distributed block-based matrix-matrix multiplication
- Task 3 – Performance analysis and modeling

Task 3 – Performance analysis and modeling

TECHNISCHE UNIVERSITÄT DARMSTADT

- Purpose: use Extra-P to create a performance model for the runtime of the algorithm in Task 1
- You are provided with the job script *perf_analysis.sh*:
  - Runs the algorithm on 6 different values of n (on 16 processes)
  - Writes the output to *input.res*
- The template for Task 1 (task1.c) has timing calls – do not remove them!
- Run Extra-P on the *input.res*: `extrap_cmd input.res`
- Output is a line of comma-separated values
- Compare model to the expectation $O(n^3)$