Parallel Programming

Linear Algebra – Matrix Operations

Jan Schönherr
Technische Universität Berlin
School IV – Electrical Engineering and Computer Sciences
Communication and Operating Systems (KBS)
Einsteinufer 17, Sekr. EN6, 10587 Berlin

Matrix-Vector Multiplication

- Given
  - Matrix *A* with dimensions m x n
  - Vector *b* with dimensions n x 1
- Calculate Vector *c* with dimensions m x 1

$$A \cdot b = c$$

- Used in …
  - Solving systems of linear equations
  - Neural networks
  - …

Pseudo Code

- Each component of *c* is defined by

$$c_i = \sum_{j=1}^{n} a_{i,j} \cdot b_j$$

- This leads to:

```
for (i = 0; i < m; i++) {
    c[i] = 0;
    for (j = 0; j < n; j++)
        c[i] += a[i][j] * b[j];
}
```

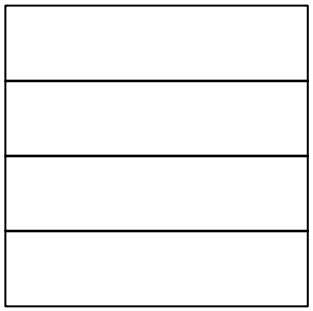
Task Decomposition

- What is a primitive task?
- Calculation of a single component of c?
  - No dependencies/communication between tasks
  - Upper bound of m tasks, while there are $\Theta(m \cdot n)$ operations
- Calculation of each $a_{i,j} \cdot b_j$, and adding them up?
  - Many tasks, multiplications independent
  - Dependencies between tasks

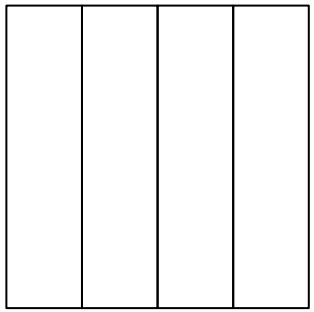
Data Decomposition

- Which layout of *A* to choose?

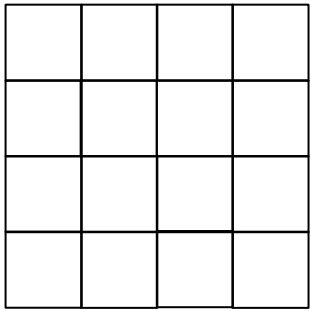
Row blocked



Column blocked



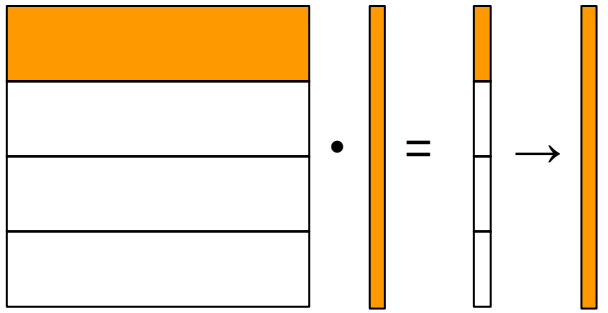
Row and column blocked



- Focusing on *c* or *b* is also possible
- Results in different parallel algorithms with different properties

Analysis: Row blocked layout of *A*

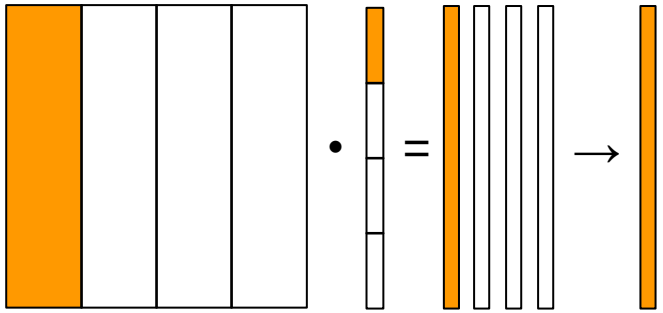
- All components of vector *b* are used by all processors
- Result vector *c* has a blocked layout, may need to take care of that (all-gather or multi-broadcast)



- Properties (assuming *m=n* and large *n*)
  - Computation: $\Theta(n^2/p)$
  - Communication: $\Theta(\log p + n) \approx \Theta(n)$ (all-gather)
  - Isoefficiency: $n = \Theta(p)$

Analysis: Column blocked layout of *A*

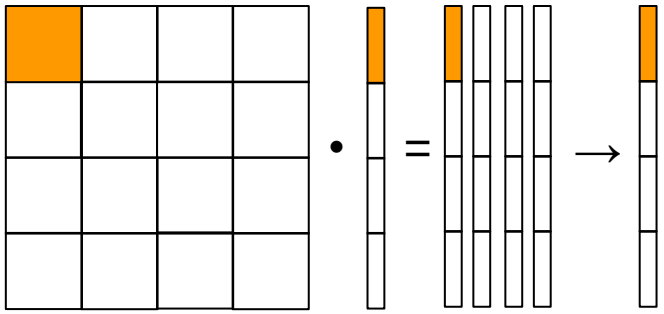
- Each processor needs a different part of *b*
- Each processor computes only a part of each component of *c*, needs a reduction afterwards to end either with a blocked or replicated *c*



- Properties (assuming *m=n* and large *n*)
  - Computation: $\Theta(n^2/p)$
  - Communication: $\Theta(\log p + n \cdot \log p) \approx \Theta(n \cdot \log p)$
    or $\Theta(p + n) \approx \Theta(n)$ (all-reduce)
  - Isoefficiency: $n = \Theta(p)$

Analysis: Row and column blocked

- Cannot utilize arbitrary numbers of processors (without getting more complicated)
- Each column needs a different part of *b*
- Each processor computes only a part of some components of *c*, needs a reduction afterwards
- Result vector *c* has a blocked layout (across rows)



- Properties (assuming *m=n* and large *n*)
  - Computation: $\Theta(n^2/p)$
  - Communication: $\Theta(\log \text{sqrt}(p) + n/\text{sqrt}(p) \cdot \log p)$
    $\approx \Theta(n/\text{sqrt}(p) \cdot \log p)$
    (broadcasting and reducing vectors along one dimension)
  - Isoefficiency: $n = \Theta(\text{sqrt}(p) \cdot \log p)$

Matrix-Matrix Multiplication

- Given
  - Matrix *A* with dimensions l x m
  - Matrix *B* with dimensions m x n
- Calculate matrix *C* with dimensions l x n

$$A \cdot B = C$$

- Used in …
  - Chemical systems
  - Signal processing
  - …

Pseudo Code

- Elements of *C* are defined by

$$c_{i,j} = \sum_{k=1}^{m} a_{i,k} \cdot b_{k,j}$$

- This leads to:

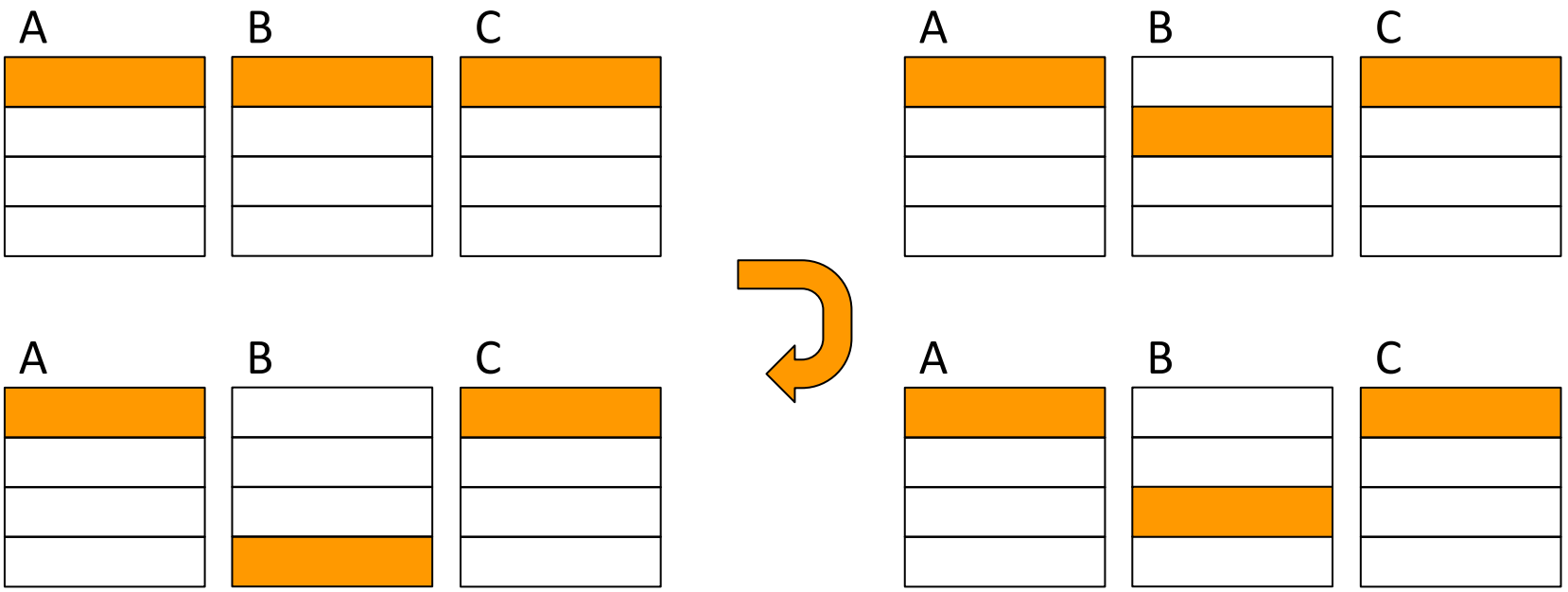
```
for (i = 0; i < l; i++) {
    for (j = 0; j < n; j++) {
        c[i][j] = 0;
        for (k = 0; k < m; k++)
            c[i][j] += a[i][k] * b[k][j];
    }
}
```

Task Decomposition

- Straightforward: Calculate one element $c_{i,j}$
  - (More tasks this time, but still corner cases.)
- Data dependencies
  - Calculation of $c_{i,j}$ needs row i of *A* and column j of *B*
- How to group these primitive tasks?
  - Rows of *C*? Columns of *C*?
    - Minimizes dependencies on *A* or *B*, respectively
    - But: needs to access all elements of the other matrix
  - Blocks of *C*?
    - Better, but still high memory demand per processor

Data Decomposition

- Consider a row blocked layout of *A*, *B* and *C*
  - (Same layout so that the output can be used as input.)
- Allows only to compute partial results of $c_{i,j}$
- Now shift blocks of *B* in a ring
- Needs *p* steps to complete



Analysis

- (Assume l=m=n)

- Computation: $\Theta(n^3/p)$
- Communication: $\Theta(n^2)$
- Isoefficiency: $n = \Theta(p)$

- Observation: If n is large enough, it should be possible to overlap communication with computations nearly completely

- Can we do better?

Cannon's Algorithm (i)

- Based on a 2D block decomposition
- Each processor is responsible for a submatrix of *A*, *B* and *C*
- To calculate submatrix $C_{i,j}$, processor $P_{i,j}$ needs access to sqrt(p) pairs $A_{i,k}$ and $B_{k,j}$
  - After an initial skew, sqrt(p) steps are performed
  - Blocks of *A* are shifted left
  - Blocks of *B* are shifted up

A0,0 B0,0	A0,1 B0,1	A0,2 B0,2	A0,3 B0,3
A1,0 B1,0	A1,1 B1,1	A1,2 B1,2	A1,3 B1,3
A2,0 B2,0	A2,1 B2,1	A2,2 B2,2	A2,3 B2,3
A3,0 B3,0	A3,1 B3,1	A3,2 B3,2	A3,3 B3,3

Initial skew:
Row i of A is shifted i columns left
Column i of B is shifted i rows up

A0,0 B0,0	A0,1 B1,1	A0,2 B2,2	A0,3 B3,3
A1,1 B1,0	A1,2 B2,1	A1,3 B3,2	A1,0 B0,3
A2,2 B2,0	A2,3 B3,1	A2,0 B0,2	A2,1 B1,3
A3,3 B3,0	A3,0 B0,1	A3,1 B1,2	A3,2 B2,3

Cannon's Algorithm (ii)

- After each step
  - Shift *A* one column leftwards
  - Shift *B* one row upwards

A0,0 B0,0	A0,1 B1,1	A0,2 B2,2	A0,3 B3,3
A1,1 B1,0	A1,2 B2,1	A1,3 B3,2	A1,0 B0,3
A2,2 B2,0	A2,3 B3,1	A2,0 B0,2	A2,1 B1,3
A3,3 B3,0	A3,0 B0,1	A3,1 B1,2	A3,2 B2,3

Step 1

A0,1 B1,0	A0,2 B2,1	A0,3 B3,2	A0,0 B0,3
A1,2 B2,0	A1,3 B3,1	A1,0 B0,2	A1,1 B1,3
A2,3 B3,0	A2,0 B0,1	A2,1 B1,2	A2,2 B2,3
A3,0 B0,0	A3,1 B1,1	A3,2 B2,2	A3,3 B3,3

Step 2

A0,2 B2,0	A0,3 B3,1	A0,0 B0,2	A0,1 B1,3
A1,3 B3,0	A1,0 B0,1	A1,1 B1,2	A1,2 B2,3
A2,0 B0,0	A2,1 B1,1	A2,2 B2,2	A2,3 B3,3
A3,1 B1,0	A3,2 B2,1	A3,3 B3,2	A3,0 B0,3

Step 3

A0,3 B3,0	A0,0 B0,1	A0,1 B1,2	A0,2 B2,3
A1,0 B0,0	A1,1 B1,1	A1,2 B2,2	A1,3 B3,3
A2,1 B1,0	A2,2 B2,1	A2,3 B3,2	A2,0 B0,3
A3,2 B2,0	A3,3 B3,1	A3,0 B0,2	A3,1 B1,3

Step 4

Analysis

- (Assume l=m=n)

- Computation: $\Theta(n^3/p)$
- Communication: $\Theta(n^2/\mathrm{sqrt}(p))$
- Isoefficiency: $n = \Theta(\mathrm{sqrt}(p))$

- Again: overlapping of communication and computations is possible

Further options not covered

- Other ways to parallelize matrix multiplication
  - E. g. 3D decomposition (e. g. DNS algorithm)
- Not using an $\Theta(n^3)$ algorithm as a basis
  - E. g. Strassen's algorithm
- Operations on sparse (or structured) matrices
  - For e. g. finite element methods, graphs
- Implications of shared memory architectures