COSC 462

Parallel Algorithms

Matrix-Matrix Multiplication

Piotr Luszczek

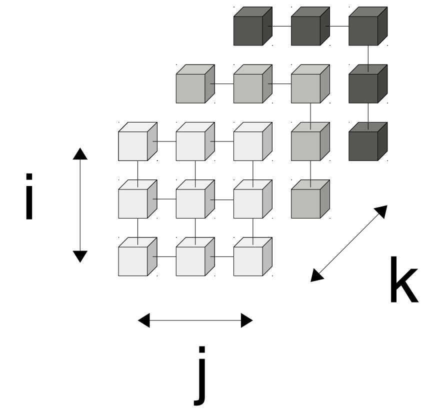
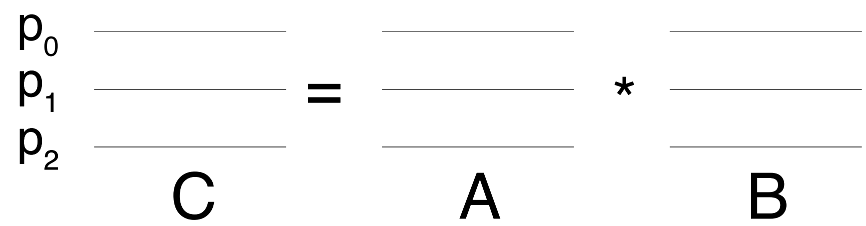
Remarks on Divisibility

- In practice, matrix dimensions and processor counts do not divide each other
  - N is not multiple of P
    - Solution: padding with 0's
      - New dimension N'=N+b
    - Especially useful for matrix-matrix multiply because 0's don't contribute to the result
    - CPUs often take shortcut when 0's are encountered in floating-point unit
    - If adding 0's is not an option, for example, small memory then the cleanup code has to be provided to deal with
  - P is not a square of an integer
    - Factor P into shape closest to a square
      - For example: P=128=8*16
    - Advantage:
      - Math equations for algorithm scaling will work
  - P is a prime number
    - Remove one process from computing and try to factor again

Why Study Matrix-Matrix Multiplication?

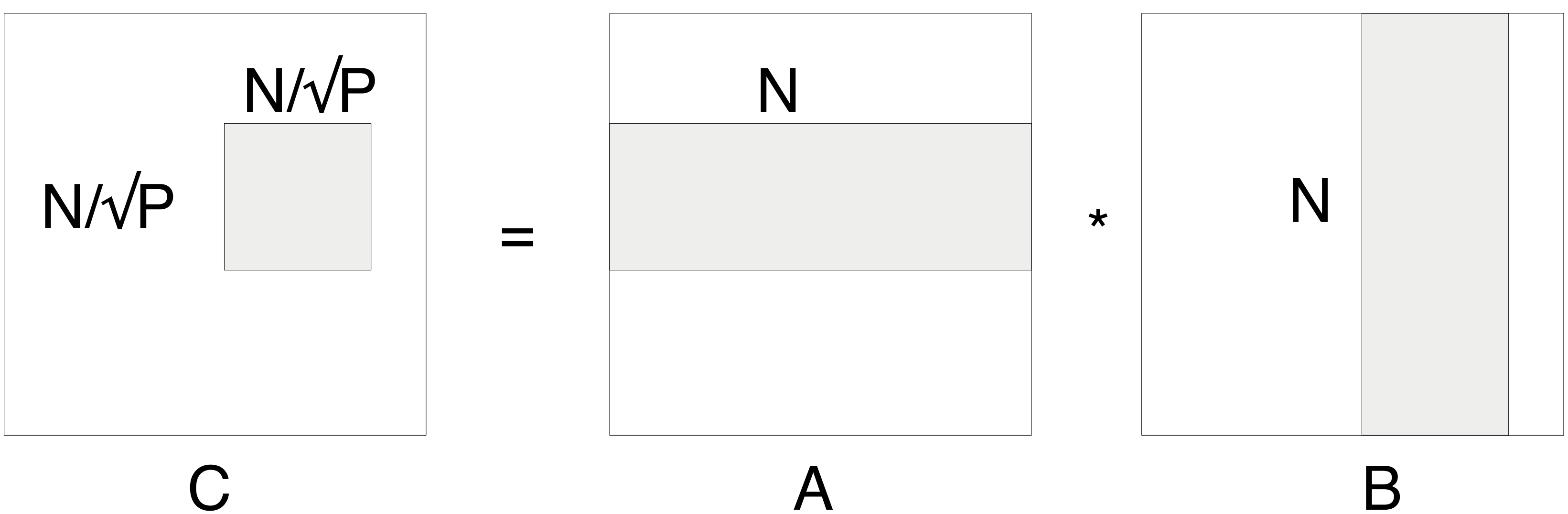
- Perfectly parallel yet contains reductions
- Various data distributions possible
- Plenty of examples and written material available
  - Algorithms: Cannon (systolic), SUMMA, PUMMA, 3D, 2.5D
- Separate algorithms can be developed for network topologies
  - Hypercube (SGI)
  - Fat-tree (Infiniband)
  - Dragonfly (Cray)
  - Torus (Tofu, K computer)
- Applications
  - Computational chemistry (change of basis for Hamiltonian)
  - Signal processing
  - Plasma containment physics
    - Tokamak design
    - github.com/ORNL-Fusion/aorsa2d

Definition and Observations

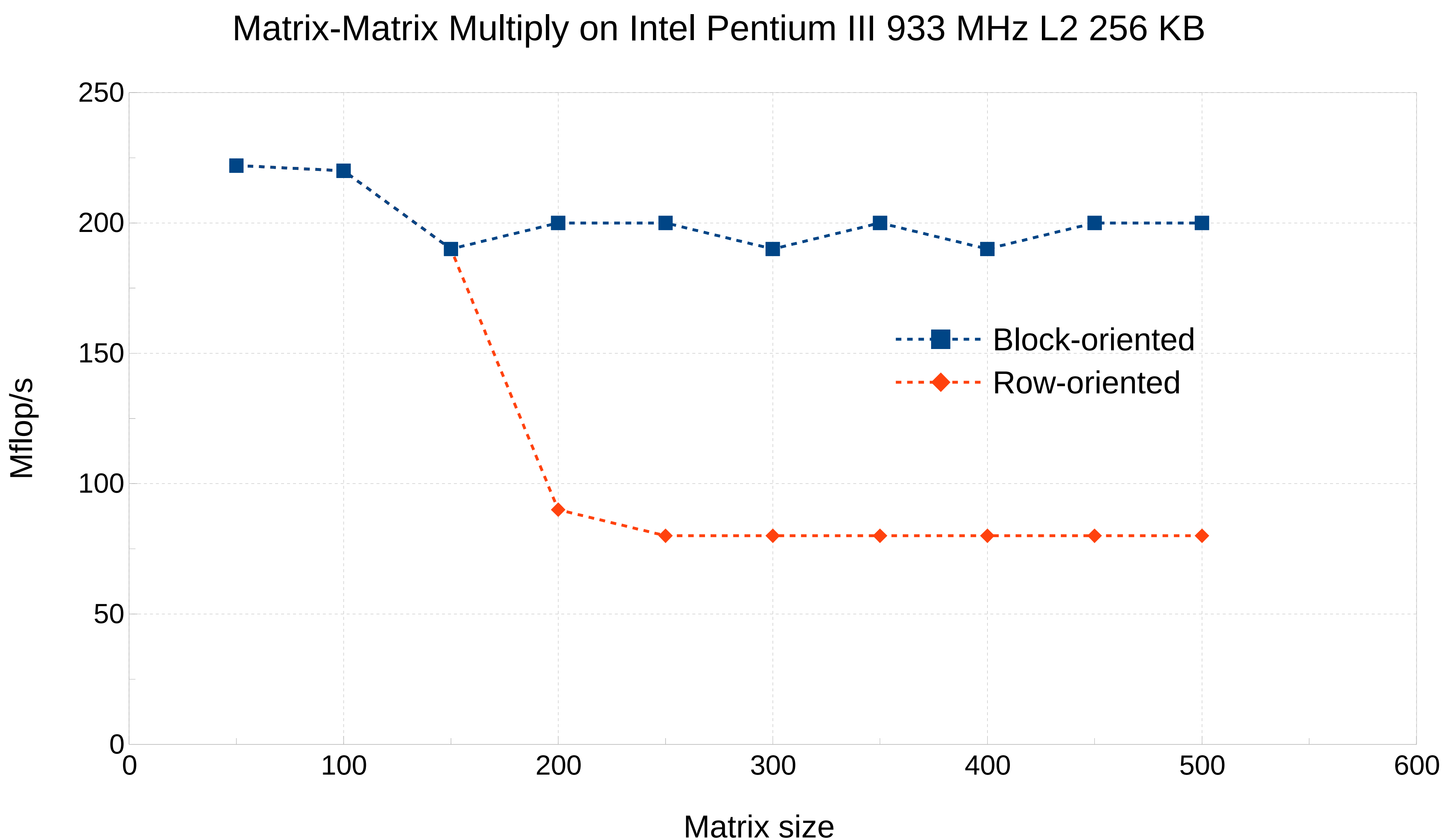
- Matrix notation
  - $C = A * B$ $\quad A,B,C \in \mathbf{R}^{N*N}$
- Element-wise
  - $c_{ij} = \sum a_{ik} b_{kj}$
- Code
  - ```
    for (i = 0; i < N; ++i)
      for (j = 0; j < N; ++j)
        for (k = 0; k < N; ++k)
          c[i][j] += a[i][k] * b[k][j]
    ```
- Observations:
  - A lot of work, little input/output data
    - Complexity(N) = $2N^3 + \Theta(N^2)$
    - Data(N) = $3N^2 + \Theta(N)$
    - We call it: surface to volume effect
  - Parallelism abounds
    - The loops can be interchanged
    - Summation can use any variant of efficient reduction
    ```

Attempt 1: Single-element Tasks



- Tasks
  - N^3 compute tasks $t_{i,j,k}$: $c_{i,j}^{(k)} = a_{i,k}\ b_{k,j}$
  - N^2 reduction tasks $r_{i,j}$: $\sum_k c_{i,j}^{(k)}$
- Data partitioning
  - Elements of C don't need to be replicated
  - Elements of A and/or B must be replicated or communicated
    - a1,1 is needed by c1,* and c*,1 (N+N tasks)
  - Must agglomerate to decrease message count

Attempt 2: Rowwise Agglomeration



- Rows of B have to be communicated:
  - ```
    for (i = 0; i < N; ++i)
      sendrecv((self+1) % P, (self-1+P) % P, B[i][:])
    ```
- Each processor must exchange N messages
  - Total : N*N → $\Theta(N^2)$
- Computation to communication ratio
  - $2N^3$ computations on P processors: $2N^3/P$
  - Entire B is communicated: N^2
  - Ratio: 2N/P
    - The ratio is very small (bad)
    - The problem needs to grow linearly with number of processors
    ```

Attempt 3: Cannon's Algorithm

- Main idea:
  - Use 2D processor grid and 2D partitioning of the matrix
- Computation to communication ratio
  - Computation for a single processor: $2 * N/\sqrt{P} * N/\sqrt{P} * N$
  - Communication to send the data to a processor: $2*N/\sqrt{P}*N$
  - Ratio: $N/\sqrt{P}$
    - Compare to N/P for rowwise agglomeration

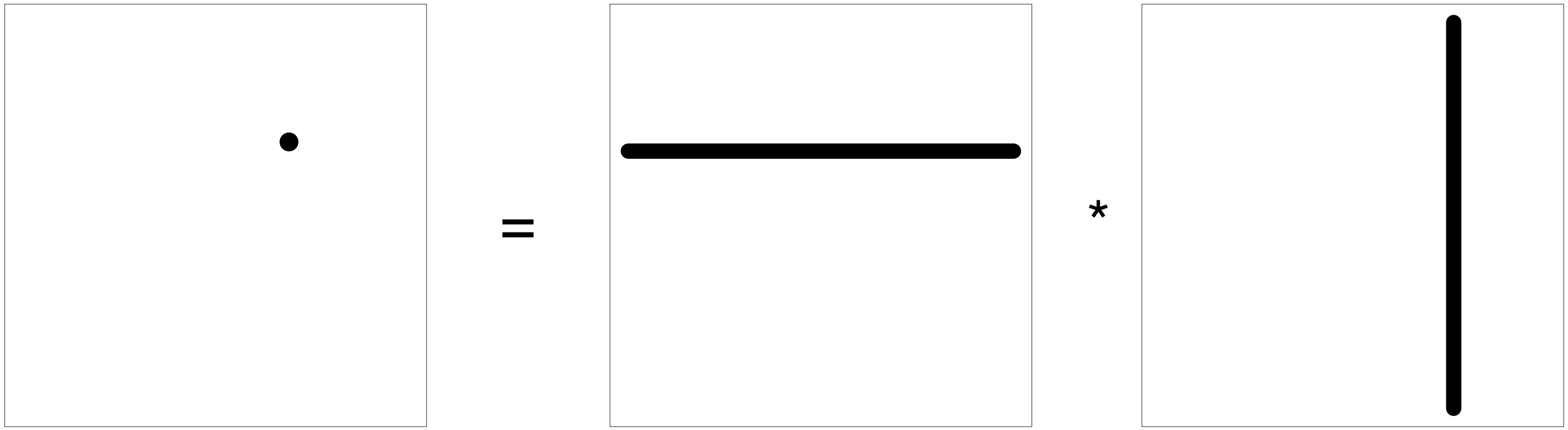


Beware of Sequential Performance



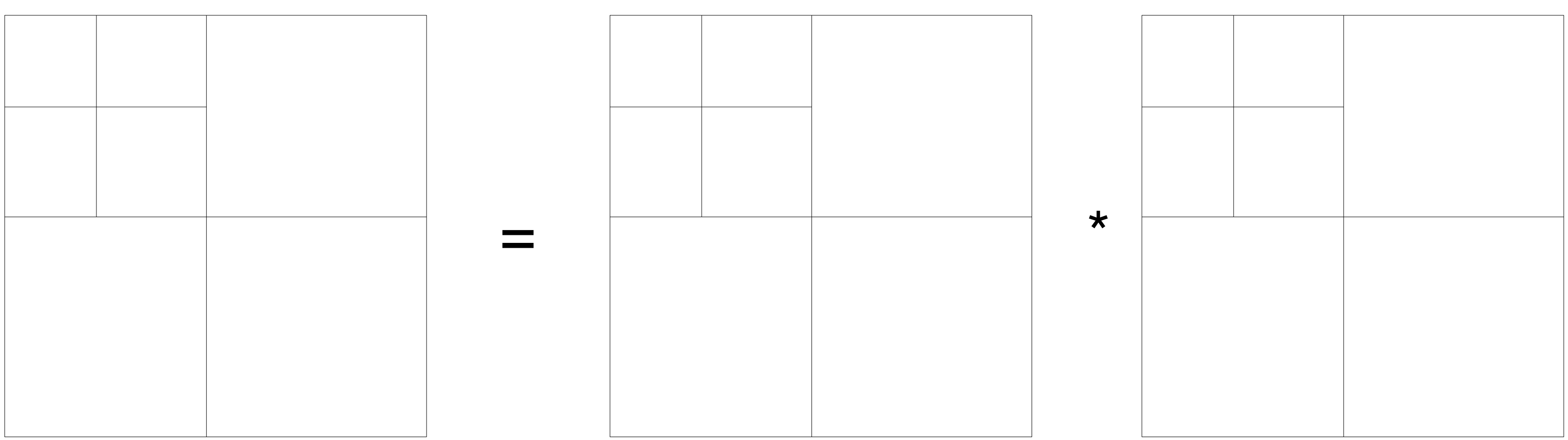
Element-wise vs. Block-wise vs Recursive

Dot-product formulation, size of “a” and “b” vectors grows with the problem size N



Submatrix formulation, block size can be limited to fit in cache

Recursive formulation is “cache oblivious” - will perform well without selecting block size explicitly



Amdahl Law: Sequential Performance Matters

- The reference (sequential) performance for computing Amdahl fraction must be optimized
- Slow sequential performance is bad because
  - Gives a false sense of scalability
  - Makes communication look slow compared to computation
  - Creates superlinear scalability when there is none
  - Gives the wrong basis for comparing between different hardware and (sequential/parallel) algorithms
    - My machine is better because scaling is better
    - My algorithm is better because it scales better