Computação Paralela / Computação Avançada

cap6 - 2022-11-16

Matrix Multiplication

Parallelizing with NumPy

Helmut Wolters
LIP / UC

$$C = A \cdot B = \begin{pmatrix} 38 & 92 & 92 & 52 & 15 & 76 & 66 & 45 \\ 44 & 11 & 95 & 35 & 53 & 50 & 68 & 68 \\ 65 & 54 & 44 & 37 & 32 & 42 & 20 & 30 \\ 68 & 36 & 85 & 60 & 81 & 65 & 58 & 18 \\ 95 & 71 & 55 & 31 & 23 & 15 & 64 & 40 \\ 11 & 19 & 65 & 66 & 59 & 58 & 70 & 11 \\ 55 & 29 & 32 & 71 & 96 & 59 & 54 & 87 \\ 52 & 34 & 34 & 41 & 48 & 64 & 44 & 78 \end{pmatrix} \cdot \begin{pmatrix} 55 & 17 & 94 & 40 & 74 & 63 & 63 & 89 \\ 77 & 89 & 69 & 19 & 30 & 73 & 23 & 12 \\ 40 & 59 & 53 & 29 & 28 & 57 & 29 & 22 \\ 61 & 20 & 20 & 78 & 41 & 90 & 21 & 66 \\ 46 & 45 & 79 & 12 & 85 & 76 & 89 & 23 \\ 50 & 44 & 17 & 39 & 54 & 73 & 93 & 51 \\ 59 & 57 & 94 & 65 & 27 & 79 & 81 & 31 \\ 91 & 88 & 35 & 78 & 43 & 73 & 45 & 47 \end{pmatrix}$$

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} & c_{18} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} & c_{28} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} & c_{37} & c_{38} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} & c_{47} & c_{48} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} & c_{57} & c_{58} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} & c_{67} & c_{68} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & c_{77} & c_{78} \\ c_{81} & c_{82} & c_{83} & c_{84} & c_{85} & c_{86} & c_{87} & c_{88} \end{pmatrix}$$

$$C = A \cdot B = \begin{pmatrix} 38 & 92 & 92 & 52 & 15 & 76 & 66 & 45 \\ 44 & 11 & 95 & 35 & 53 & 50 & 68 & 68 \\ 65 & 54 & 44 & 37 & 32 & 42 & 20 & 39 \\ 68 & 36 & 85 & 60 & 81 & 65 & 58 & 18 \\ 95 & 71 & 55 & 31 & 23 & 15 & 64 & 40 \\ 11 & 19 & 65 & 66 & 59 & 58 & 70 & 11 \\ 55 & 29 & 32 & 71 & 96 & 59 & 54 & 87 \\ 52 & 34 & 34 & 41 & 48 & 64 & 44 & 78 \end{pmatrix} \cdot \begin{pmatrix} 55 & 17 & 94 & 40 & 74 & 63 & 63 & 89 \\ 77 & 89 & 69 & 19 & 30 & 73 & 23 & 12 \\ 40 & 59 & 53 & 29 & 28 & 57 & 29 & 22 \\ 61 & 20 & 20 & 78 & 41 & 90 & 21 & 66 \\ 46 & 45 & 79 & 12 & 85 & 76 & 89 & 23 \\ 50 & 44 & 17 & 39 & 54 & 73 & 93 & 51 \\ 59 & 57 & 94 & 65 & 27 & 79 & 81 & 31 \\ 91 & 88 & 35 & 78 & 45 & 73 & 45 & 47 \end{pmatrix}$$

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} & c_{18} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} & c_{28} \\ c_{31} & c_{32} & c_{33} & c_{34} & \boxed{c_{35}} & c_{36} & c_{37} & c_{38} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} & c_{47} & c_{48} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} & c_{57} & c_{58} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} & c_{67} & c_{68} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & c_{77} & c_{78} \\ c_{81} & c_{82} & c_{83} & c_{84} & c_{85} & c_{86} & c_{87} & c_{88} \end{pmatrix}$$

$$C = A \cdot B = \begin{pmatrix} 38 & 92 & 92 & 52 & 15 & 76 & 66 & 45 \\ 44 & 11 & 95 & 35 & 53 & 50 & 68 & 68 \\ 65 & 54 & 44 & 37 & 32 & 42 & 20 & 30 \\ 68 & 36 & 85 & 60 & 81 & 65 & 58 & 18 \\ 95 & 71 & 55 & 31 & 23 & 15 & 64 & 40 \\ 11 & 19 & 65 & 66 & 59 & 58 & 70 & 11 \\ 55 & 29 & 32 & 71 & 96 & 59 & 54 & 87 \\ 52 & 34 & 34 & 41 & 48 & 64 & 44 & 78 \end{pmatrix} \cdot \begin{pmatrix} 55 & 17 & 94 & 40 & 74 & 63 & 63 & 89 \\ 77 & 89 & 69 & 19 & 30 & 73 & 23 & 12 \\ 40 & 59 & 53 & 29 & 28 & 57 & 29 & 22 \\ 61 & 20 & 20 & 78 & 41 & 90 & 21 & 66 \\ 46 & 45 & 79 & 12 & 85 & 76 & 89 & 23 \\ 50 & 44 & 17 & 39 & 54 & 73 & 93 & 51 \\ 59 & 57 & 94 & 65 & 27 & 79 & 81 & 31 \\ 91 & 88 & 35 & 78 & 43 & 73 & 45 & 47 \end{pmatrix}$$

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} & c_{18} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} & c_{28} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} & c_{37} & c_{38} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} & c_{47} & c_{48} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} & c_{57} & c_{58} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} & c_{67} & c_{68} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & c_{77} & c_{78} \\ c_{81} & c_{82} & c_{83} & c_{84} & c_{85} & c_{86} & c_{87} & c_{88} \end{pmatrix}$$

Example:

- n lines and n rows of floating numbers with n=1024
- Floating number occupies 8 bytes
- Matrix: 1024 ×1024 × 8 = 8 MegaBytes
- np = 128 processes
- Master has to send 1/128 of A and the whole matrix B to each process
  - A: 127 × **Send** ≈ 8 MegaBytes over the network

    1 **Scatter** ≈ 8 MegaBytes over the network

    (no change, but much faster due to optimization by MPI)
  - B: 127 × **Send** = 127 × 8 Megabytes ≈ 1 GigaByte over the network (!)

    1 **Broadcast**: only ≈ 8 MegaBytes over the network !
- Master receives 1/128 of C from each process
  - 127 **Recv** ≈ 8 MegaBytes over the network
  - 1 **Gather** ≈ 8 MegaBytes over the network

    (no change, but **very** much faster due to optimization by MPI)

$$C = A \cdot B = \begin{pmatrix} \boxed{\begin{matrix} 38 & 92 & 92 & 52 & 15 & 76 & 66 & 45 \\ 44 & 11 & 95 & 35 & 53 & 50 & 68 & 68 \end{matrix}} \\ \boxed{\begin{matrix} 65 & 54 & 44 & 37 & 32 & 42 & 20 & 30 \\ 68 & 36 & 85 & 60 & 81 & 65 & 58 & 18 \end{matrix}} \\ \boxed{\begin{matrix} 95 & 71 & 55 & 31 & 23 & 15 & 64 & 40 \\ 11 & 19 & 65 & 66 & 59 & 58 & 70 & 11 \end{matrix}} \\ \boxed{\begin{matrix} 55 & 29 & 32 & 71 & 96 & 59 & 54 & 87 \\ 52 & 34 & 34 & 41 & 48 & 64 & 44 & 78 \end{matrix}} \end{pmatrix} \cdot \begin{pmatrix} 55 & 17 & 94 & 40 & 74 & 63 & 63 & 89 \\ 77 & 89 & 69 & 19 & 30 & 73 & 23 & 12 \\ 40 & 59 & 53 & 29 & 28 & 57 & 29 & 22 \\ 61 & 20 & 20 & 78 & 41 & 90 & 21 & 66 \\ 46 & 45 & 79 & 12 & 85 & 76 & 89 & 23 \\ 50 & 44 & 17 & 39 & 54 & 73 & 93 & 51 \\ 59 & 57 & 94 & 65 & 27 & 79 & 81 & 31 \\ 91 & 88 & 35 & 78 & 43 & 73 & 45 & 47 \end{pmatrix}$$

Scatter

Broadcast

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

Gather

$$C = \begin{pmatrix} \boxed{\begin{matrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \\ c_{41} & c_{42} \\ c_{51} & c_{52} \\ c_{61} & c_{62} \\ c_{71} & c_{72} \\ c_{81} & c_{82} \end{matrix}} & \boxed{\begin{matrix} c_{13} & c_{14} \\ c_{23} & c_{24} \\ c_{33} & c_{34} \\ c_{43} & c_{44} \\ c_{53} & c_{54} \\ c_{63} & c_{64} \\ c_{73} & c_{74} \\ c_{83} & c_{84} \end{matrix}} & \boxed{\begin{matrix} c_{15} & c_{16} \\ c_{25} & c_{26} \\ c_{35} & c_{36} \\ c_{45} & c_{46} \\ c_{55} & c_{56} \\ c_{65} & c_{66} \\ c_{75} & c_{76} \\ c_{85} & c_{86} \end{matrix}} & \boxed{\begin{matrix} c_{17} & c_{18} \\ c_{27} & c_{28} \\ c_{37} & c_{38} \\ c_{47} & c_{48} \\ c_{57} & c_{58} \\ c_{67} & c_{68} \\ c_{77} & c_{78} \\ c_{87} & c_{88} \end{matrix}} \end{pmatrix}$$

$$C = A \cdot B = \begin{pmatrix} 38 & 92 & 92 & 52 & 15 & 76 & 66 & 45 \\ 44 & 11 & 95 & 35 & 53 & 50 & 68 & 68 \\ 65 & 54 & 44 & 37 & 32 & 42 & 20 & 30 \\ 68 & 36 & 85 & 60 & 81 & 65 & 58 & 18 \\ 95 & 71 & 55 & 31 & 23 & 15 & 64 & 40 \\ 11 & 19 & 65 & 66 & 59 & 58 & 70 & 11 \\ 55 & 29 & 32 & 71 & 96 & 59 & 54 & 87 \\ 52 & 34 & 34 & 41 & 48 & 64 & 44 & 78 \end{pmatrix} \cdot \begin{pmatrix} 55 & 17 & 94 & 40 & 74 & 63 & 63 & 89 \\ 77 & 89 & 69 & 19 & 30 & 73 & 23 & 12 \\ 40 & 59 & 53 & 29 & 28 & 57 & 29 & 22 \\ 61 & 20 & 20 & 78 & 41 & 90 & 21 & 66 \\ 46 & 45 & 79 & 12 & 85 & 76 & 89 & 23 \\ 50 & 44 & 17 & 39 & 54 & 73 & 93 & 51 \\ 59 & 57 & 94 & 65 & 27 & 79 & 81 & 31 \\ 91 & 88 & 35 & 78 & 43 & 73 & 45 & 47 \end{pmatrix}$$

Send/Recv

Send/Recv

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

Gather

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} & c_{18} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} & c_{28} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} & c_{37} & c_{38} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} & c_{47} & c_{48} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} & c_{57} & c_{58} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} & c_{67} & c_{68} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & c_{77} & c_{78} \\ c_{81} & c_{82} & c_{83} & c_{84} & c_{85} & c_{86} & c_{87} & c_{88} \end{pmatrix}$$

Example:

- $n=1024$, Matrix: $1024 \times 1024 \times 8$ = 8 MegaBytes, $np = 128$ processes
- Master has to send 1/64 of A and 1/64 of B to each process
  - A: 127 × Send ≈ 8 MegaBytes over the network
  - B: 127 × Send ≈ 8 MegaBytes over the network
- Master receives 1/128 of C from each process
  - 127 Recv ≈ 8 MegaBytes over the network
  - 1 Gather ≈ 8 MegaBytes over the network

    (no change, but much faster due to optimization by MPI)
- **And more**: each process needs much less memory:
  - Instead of 1/128 of A and whole B: 1/64 of A and 1/64 of B
  - Instead of 64 kiloBytes + 8 MegaBytes: 2×128 kiloBytes = 256 kiloBytes
- Doesn’t sound much,
  - but now go from np=128 processes to 1024 processes
  - and from $n=1024$ to $n = 1024 \times 1024 \approx 10^6$
  - **This grows exponentially !**

$$C = A \cdot B = \begin{pmatrix} 38 & 92 & 92 & 52 & 15 & 76 & 66 & 45 \\ 44 & 11 & 95 & 35 & 53 & 50 & 68 & 68 \\ 65 & 54 & 44 & 37 & 32 & 42 & 20 & 30 \\ 68 & 36 & 85 & 60 & 81 & 65 & 58 & 18 \\ 95 & 71 & 55 & 31 & 23 & 15 & 64 & 40 \\ 11 & 19 & 65 & 66 & 59 & 58 & 70 & 11 \\ 55 & 29 & 32 & 71 & 96 & 59 & 54 & 87 \\ 52 & 34 & 34 & 41 & 48 & 64 & 44 & 78 \end{pmatrix} \cdot \begin{pmatrix} 55 & 17 & 94 & 40 & 74 & 63 & 63 & 89 \\ 77 & 89 & 69 & 19 & 30 & 73 & 23 & 12 \\ 40 & 59 & 53 & 29 & 28 & 57 & 29 & 22 \\ 61 & 20 & 20 & 78 & 41 & 90 & 21 & 66 \\ 46 & 45 & 79 & 12 & 85 & 76 & 89 & 23 \\ 50 & 44 & 17 & 39 & 54 & 73 & 93 & 51 \\ 59 & 57 & 94 & 65 & 27 & 79 & 81 & 31 \\ 91 & 88 & 35 & 78 & 43 & 73 & 45 & 47 \end{pmatrix}$$

Scatter

Scatter

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

Gather

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} & c_{18} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} & c_{28} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} & c_{37} & c_{38} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} & c_{47} & c_{48} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} & c_{57} & c_{58} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} & c_{67} & c_{68} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & c_{77} & c_{78} \\ c_{81} & c_{82} & c_{83} & c_{84} & c_{85} & c_{86} & c_{87} & c_{88} \end{pmatrix}$$

$$\begin{pmatrix} 55 & 17 & 94 & 40 & 74 & 63 & 63 & 89 \\ 77 & 89 & 69 & 19 & 30 & 73 & 23 & 12 \\ 40 & 59 & 53 & 29 & 28 & 57 & 29 & 22 \\ 61 & 20 & 20 & 78 & 41 & 90 & 21 & 66 \\ 46 & 45 & 79 & 12 & 85 & 76 & 89 & 23 \\ 50 & 44 & 17 & 39 & 54 & 73 & 93 & 51 \\ 59 & 57 & 94 & 65 & 27 & 79 & 81 & 31 \\ 91 & 88 & 35 & 78 & 43 & 73 & 45 & 47 \end{pmatrix}$$

Scatter

This is not as trivial as it looks like!
The Scatter method does not send a matrix,
it just sends a stream of bytes,
following the numbers line by line:

$$55\ 17\ 94\ 40\ 74\ 63\ 63\ 89\ 77\ 89\ 69\ 19\ 30\ 73\ 23\ 12\ 40\ 59\ 53\ 29\ 28\ 57\ 29\ 22\ 61\ 20\ 20\ 78\ 41\ 90\ 21\ 66\ 46\ 45\ 79\ 12\ 85\ 76\ \ldots$$

- We could use “scatter” (lowercase s), but this destroys all the efficiency when coming to large matrizes
- Solution:
  Transpose matrix B
- This is a costly procedure when B is large, so
  in production environment, you would store B transposed from the beginning
- But in our exercises, we will just transpose it, so we are sure we know what we are doing...

Gather

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} & c_{18} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} & c_{28} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} & c_{37} & c_{38} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} & c_{47} & c_{48} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} & c_{57} & c_{58} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} & c_{67} & c_{68} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & c_{77} & c_{78} \\ c_{81} & c_{82} & c_{83} & c_{84} & c_{85} & c_{86} & c_{87} & c_{88} \end{pmatrix}$$

This is even worse!

The Gather method does not send/receive pieces of a matrix, every process just sends a stream of bytes for each chunk, and the master receives all the chunks in the sequence of the processes:

$$c_{11}\; c_{12}\; c_{13}\; c_{14}\; c_{21}\; c_{22}\; c_{23}\; c_{24}\; c_{31}\; c_{32}\; c_{33}\; c_{34}\; c_{41}\; c_{42}\; c_{43}\; c_{44}\; c_{15}\; c_{16}\; c_{17}\; c_{18}\; c_{25}\; c_{26}\; c_{27}\; c_{28}\; c_{35}\; c_{36}\; c_{37}\; c_{38}\; c_{45}\; c_{46}\; c_{47}\; \ldots$$

- Solution: numpy is your friend - it has all the methods you need for this type of problems
- Concrete solution for reorganizing the chunks from the incoming linear stream: np.block

Numpy methods

```
x = np.linspace(1,n**2,n**2)
A = x.reshape((n,n))
B = np.linspace(10,10*n**2,n**2).reshape((n,n))
C = np.matmul(A, B)
```

Strategy:

- Start with n=8, single core, just a straight python program with numpy
- Create two matrizes A and B, and determine C = np.matmul(A, B) as reference
- Transform both A and B in linear arrays with reshape and play with np.reshape and np.block to understand the details of how it works:
    - Create the chunks you will need for the parallel process
    - Do the matmul on each chunk
    - Join the result back into one data stream
    - Transform the stream into the correct resulting matrix
- Develop your parallel code with the exact same example and make it work
- That is: run it, check it, repair it, until you got the chunks right
- Make it work for any n for which n^2 is a multiple of np.

MPI methods for exchanging numpy arrays in messages:

- Same names, but starting with a capital letter:
  - comm.Send (numpy_array, dest=p)
  - comm.Recv (numpy_array, source=0)
  - etc
- Note that the numpy_array, on the receiving side,
  - **Must exist and have been created with the necessary space to hold the array it will receive**
  - Typically with something like: numpy_array = np.empty (numData, dtype=float)
- Have a look at the example programs called *numpy?.py*
  for detailed examples of all the methods
- These methods send a bit stream of the object that then needs to be interpreted correctly on the receiving end
- The advantage is that this is much faster than sending a generic python object that contains a lot of overhead of metadata, and for array calculations we usually have just big sequences of floating point numbers that can easily be reordered through numpy methods.
- Have a look at the example programs called *matrix?.py*

Numpy methods

- a good starting point for an introduction to the *Numpy* package is the tutorial at NumPy quickstart — NumPy v1.24.dev0 Manual