

# Parallel Numerics

## 1 Setting up a MPI Environment

Setting up a MPI environment on your own linux machine for use as a personal playground should be pretty straightforward.

- i) Install the necessary packages (`openmpi` on Arch, `openmpi-bin` on Debian/Ubuntu, maybe `-dev` packages as well)
- ii) Compile using `mpicc`, `mpic++` or for whatever language you prefer
- iii) Run your binary with `mpirun -np N ./your_app` using  $N$  processes.

## 2 MPI Application Code

The MPI programs that are shown in this tutorial are available for download at Github: <https://github.com/floli/ParNum>. Download them and get your hands dirty. For questions use the forum which I will monitor.

## 3 Data Dependency Graphs

- i) Draw the data dependency graph for the evaluation of the following expression. Follow the given parenthesis. How many parallel steps are needed to compute the result?

$$\frac{((((x_1 + x_2) - x_3) + 5x_4) - 7x_5) - x_6}{2}$$

- ii) Design a faster algorithm that evaluates this expression by rearranging the parenthesis. How many parallel steps are needed now?

iii) Draw the data dependency graph of the following system of equations:

$$\begin{aligned}a &= f_a(a) \\ b &= f_b(a, b, d) \\ c &= f_c(a, b, e) \\ d &= f_d(b, e) \\ e &= f_e(c, d)\end{aligned}$$

Derive a suitable coloring that allows for maximal parallelism in a Gauss-Seidel type iteration over the given system.

## 4 BLAS — Basic Linear Algebra Subroutines

Basic Linear Algebra Subprograms (BLAS) is a specification that prescribes a set of low-level routines for performing common linear algebra operations such as vector addition, scalar multiplication, dot products, linear combinations, and matrix multiplication. They are the de facto standard low-level routines for linear algebra libraries; the routines have bindings for both C and Fortran. Although the BLAS specification is general, BLAS implementations are often optimized for speed on a particular machine, so using them can bring substantial performance benefits. BLAS implementations will take advantage of special floating point hardware such as vector registers or SIMD instructions.

[https://en.wikipedia.org/wiki/Basic\\_Linear\\_Algebra\\_Subprograms](https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms)

blas level	description	complexity	example
1	Vector-vector operations Scalar-vector operations	$\mathcal{O}(n)$ $\mathcal{O}(n)$	$\alpha x + y, \alpha \in \mathbb{R}, x, y \in \mathbb{R}^n$
2	Matrix-vector operations	$\mathcal{O}(n^2)$	$\alpha Ax + \beta y, A \in \mathbb{R}^{n \times n}, \beta \in \mathbb{R}$
3	Matrix-matrix operations	$\mathcal{O}(n^3)$	$\alpha AB + \beta C, B, C \in \mathbb{R}^{n \times n}$

BLAS routines are preinstalled and, usually, not linked statically because each vendor could provide an optimized version.

## 5 Matrix–Matrix Multiplication

The product  $A \cdot B$  of two  $N \times N$ -matrices  $A$  and  $B$  is calculated as follows:

$$C = A \cdot B \quad \text{where} \quad c_{ij} = \sum_{k=1}^N a_{ik} b_{kj}.$$

A program that calculates this product has to use three independent loops (over  $i, j, k$ ). The execution speed of the program depends on the arrangement of these loops.

- i) Give a sketch of the data dependency and the data execution graph for one  $c_{ij}$  of this problem.

- ii) Assume the dimension  $N$  is a multiple of  $\sqrt{p}$ , with  $p$  being the number of processors. The result matrix is split up into  $p$  quadratic submatrices. Every node has to compute one complete submatrix of the result matrix  $C$ .

$A \cdot B = C$  is computed by  $p$  nodes. E.g. node #0 computes block  $C_{11}$ , node #1 computes block  $C_{12}$ , etc.:

$$\left( \begin{array}{c|c|c} A_{11} & A_{12} & A_{13} \\ \hline A_{21} & A_{22} & A_{23} \\ \hline A_{31} & A_{32} & A_{33} \end{array} \right) \cdot \left( \begin{array}{c|c|c} B_{11} & B_{12} & B_{13} \\ \hline B_{21} & B_{22} & B_{23} \\ \hline B_{31} & B_{32} & B_{33} \end{array} \right) = \left( \begin{array}{c|c|c} \#0 & \#1 & \#2 \\ \hline \#3 & \#4 & \#5 \\ \hline \#6 & \#7 & \#8 \end{array} \right)$$

What parts of  $A$  and  $B$  are required to compute the result submatrix on one node? The first few nodes have to compute the following blocks:

node #0 :  $C_{11} = A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31}$

node #1 :  $C_{12} = A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{32}$

node #2 :  $C_{13} = A_{11}B_{13} + A_{12}B_{23} + A_{13}B_{33}$

node #4 :  $C_{20} = A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32}$

Interpret the multiplication formula  $C_{ij} = \sum_{k=1}^{\sqrt{p}} A_{ik} B_{kj}$  block-wise. Let's assume every summand of the result is computed in one time step. What parts of  $A$  and  $B$  are required in which order? How many time steps are required?

The entries from the respective row and the column are required on each processor. There are  $\sqrt{p}$  time steps required.

- iii) Every node is allowed to communicate with the node that computes the left, right, top or bottom submatrix of  $C$  only. Assume this cartesian topology is cyclic (toroidal). Furthermore, every node is allowed to hold only one submatrix of  $A$  and  $B$  at one time. Derive a communication scheme. Implement it using MPI and a language of your choice.