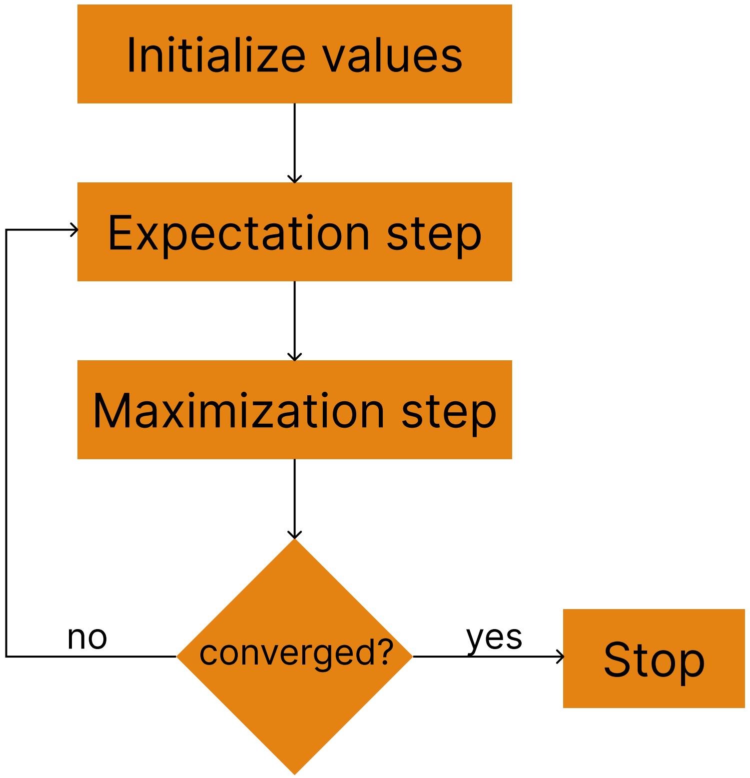
Parallel Expectation Maximization

Problem formalization

Cluster a set of points in their respective Gaussian distribution utilizing the Expectation Maximization algorithm

Sequential approach

Sequential approach



- E-step: estimate cluster assignment given the current hypothesis
- M-step: estimate a new ML hypothesis given current cluster assgnment

Expectation step

Iterate over the whole training set and for each training example estimate the probability of belonging to each cluster

$$p_{ij} = \frac{w_j p(x_i|\mu_j, \Sigma_j)}{\sum_{n=1}^{k} w_n p(x_i|\mu_n, \Sigma_n)}$$

Maximization step

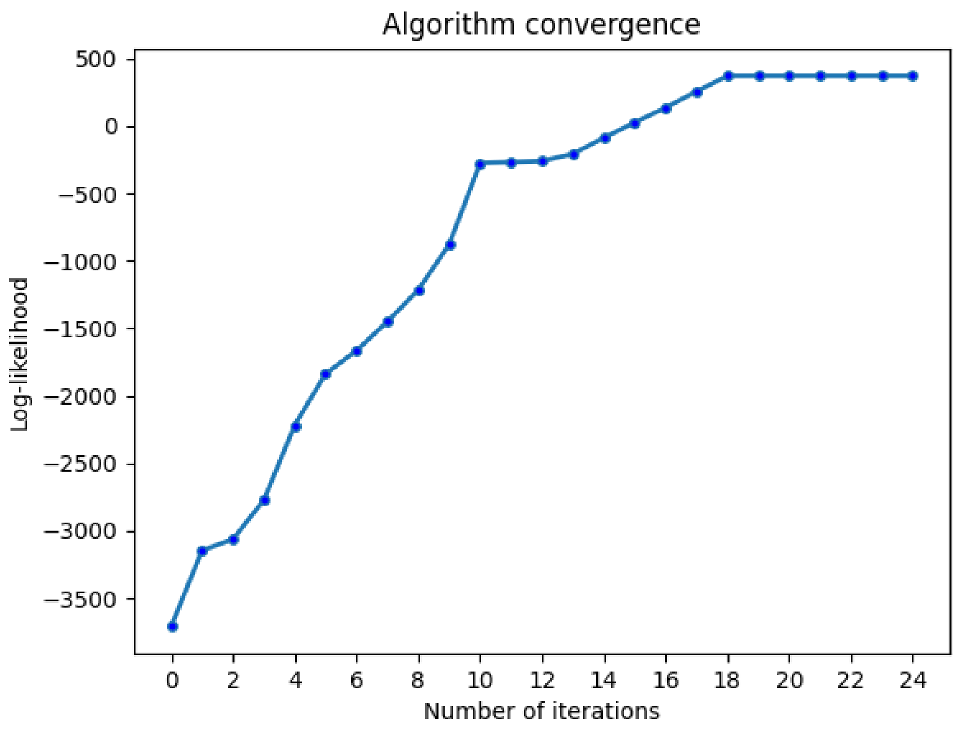
Update the values for mean, covariance and weights

$$\mu_j = \frac{\sum_{i=1}^{n} p_{ij} x_i}{\sum_{i=1}^{n} p_{ij}}$$

$$\sigma_{j,r,c} = \frac{\sum_{i=1}^{n} p_{ij}(x_{ir} - \mu_{i,r})(x_{ic} - \mu_{i,c})}{\sum_{i=1}^{n} p_{ij}}$$

$$w_j = \frac{\sum_{i=1}^{n} p_{ij}}{\sum_{l=1}^{k} \sum_{i=1}^{n} p_{li}}$$

Convergence check

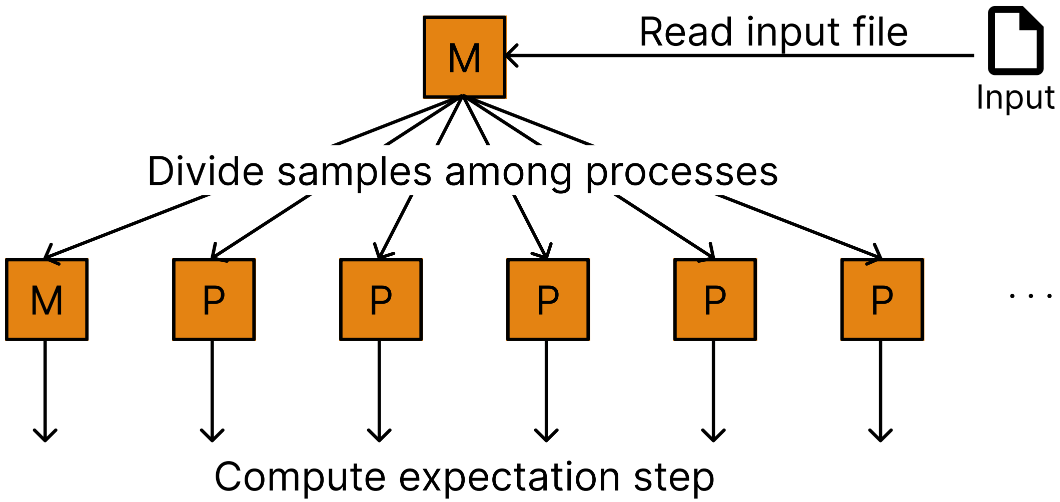


Check if the log-likelihood of the data is improving

If it does not change for 5 consecutive iterations – stop

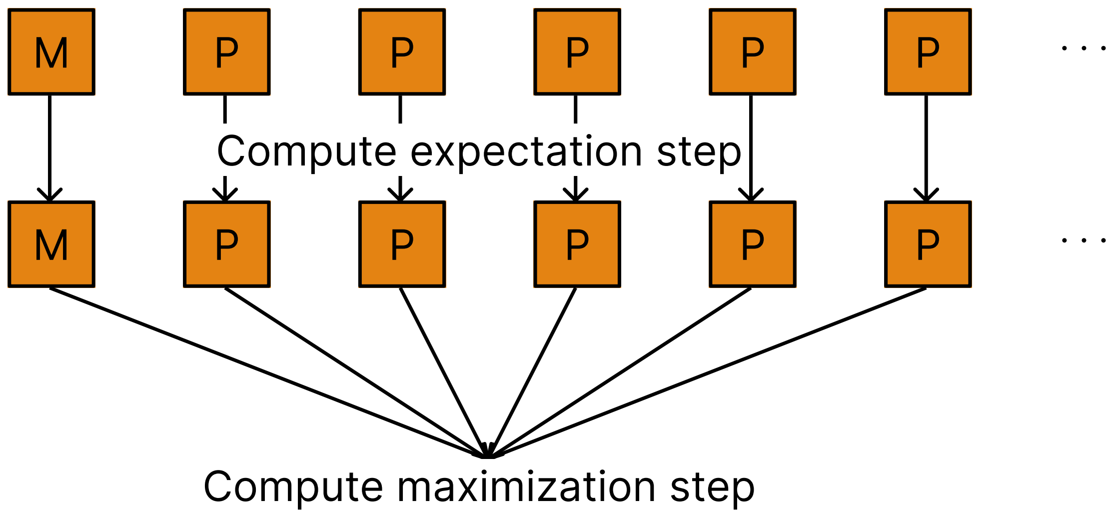
Parallel approach 1

Initialization



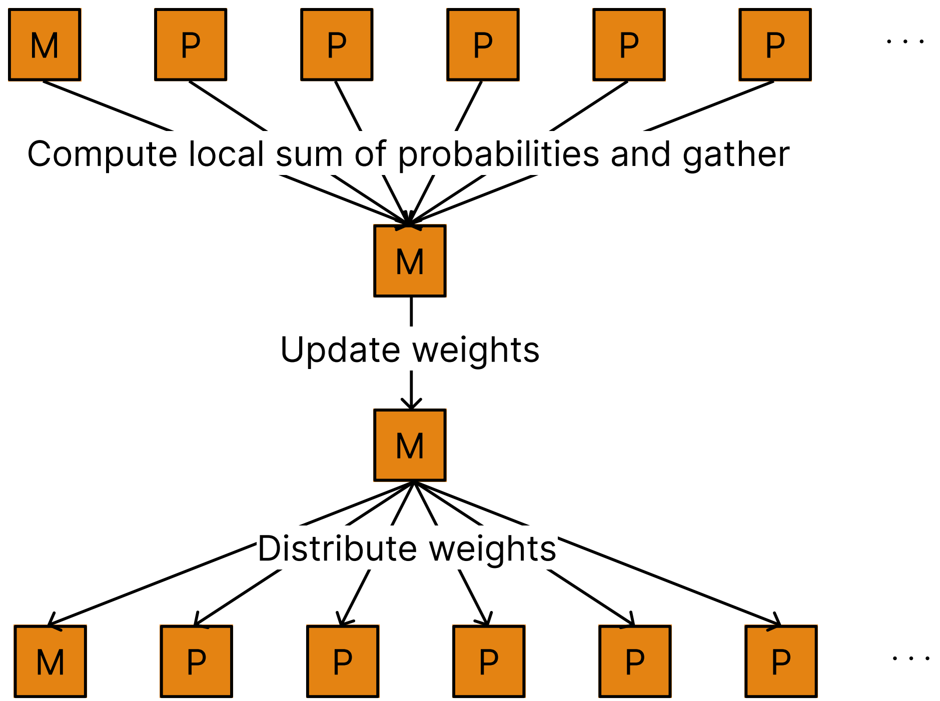
Master process reads input data, standardize it, divides it with *MPI_Scatterv* and distributes the weights, mean and covariance with *MPI_Bcast*.

Expectation step



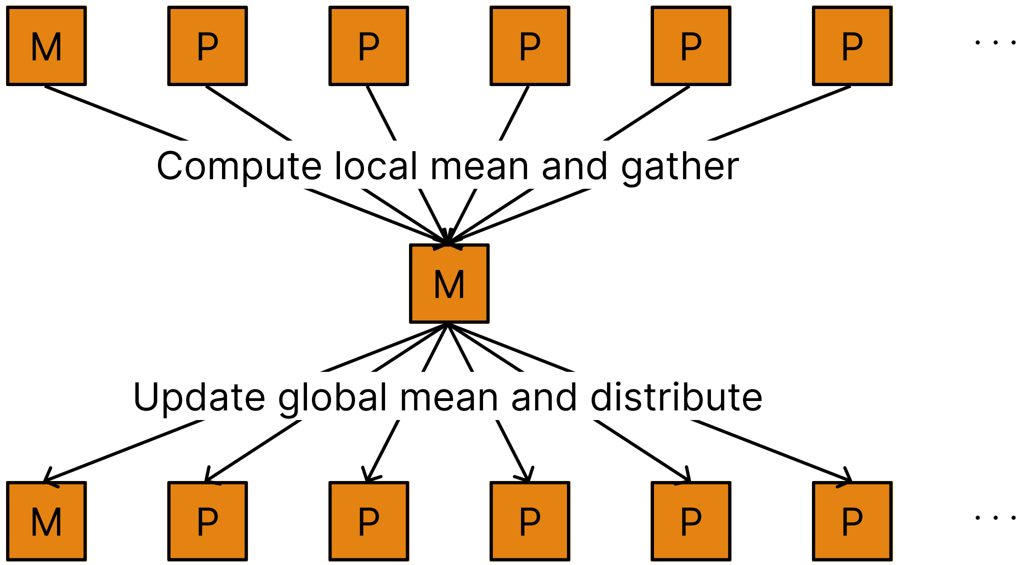
Each process estimates the cluster assignments for each example in its local submatrix.

Maximization step



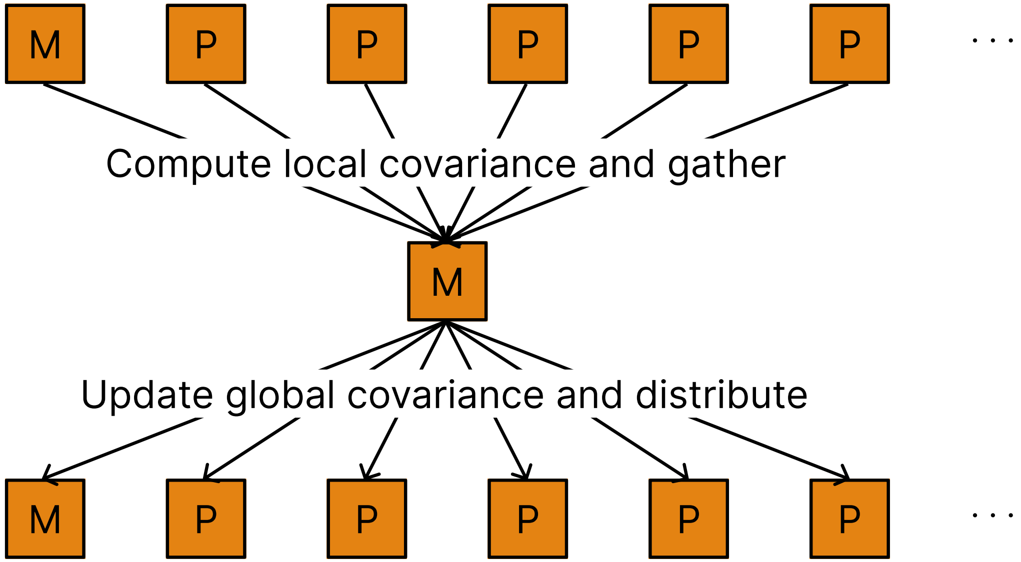
- Each process computes the local sum of probabilities on its sub-matrix.
- *MPI_Reduce* is called to calculate the total sum and the result is sent to the master process.
- The master process updates the weights values and distributes the result with *MPI_Bcast*.

Maximization step



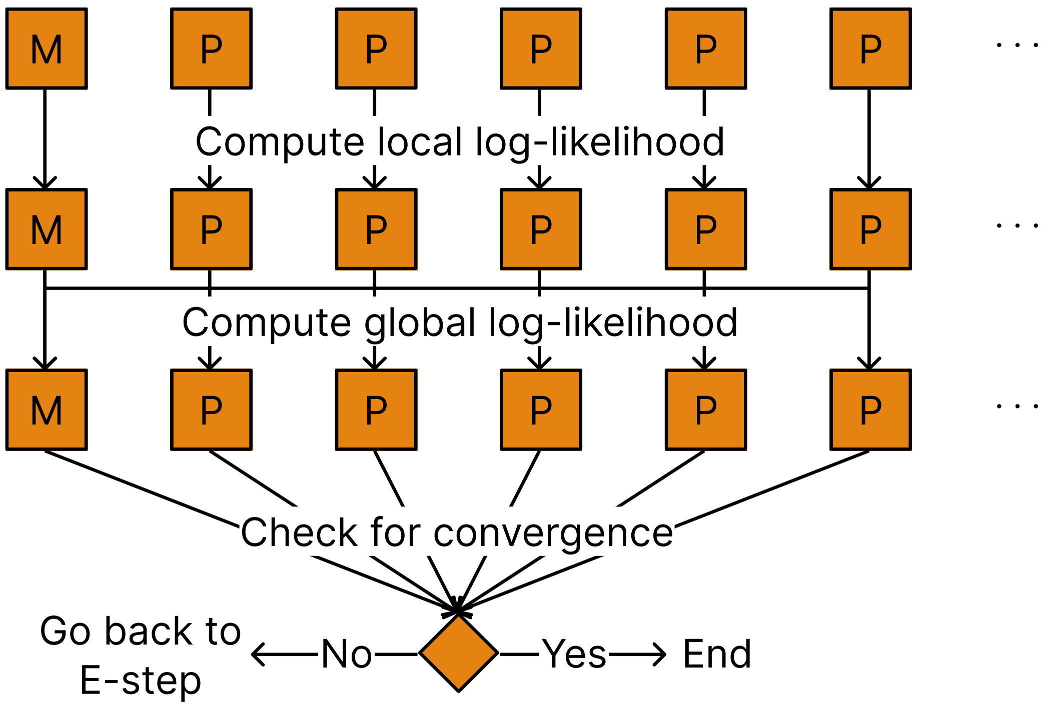
- Each process computes the numerator of the mean on its sub-matrix.
- *MPI_Reduce* is called to calculate the total sum and the result is sent to the master process.
- The master process updates the mean values and distributes the result with *MPI_Bcast*.

Maximization step



- Each process computes the numerator of the covariance on its sub-matrix.
- *MPI_Reduce* is called to calculate the total sum and the result is sent to the master process.
- The master process updates the covariance values and distributes the result with *MPI_Bcast*.

Convergence check



- Each process estimates the new log-likelihood of the data in its sub-matrix.
- *MPI_Allreduce* is used to estimate the total log-likelihood and send the result to all the processes.
- If the log-likelihood does not change for more than 5 consecutive iterations, all processes stop.

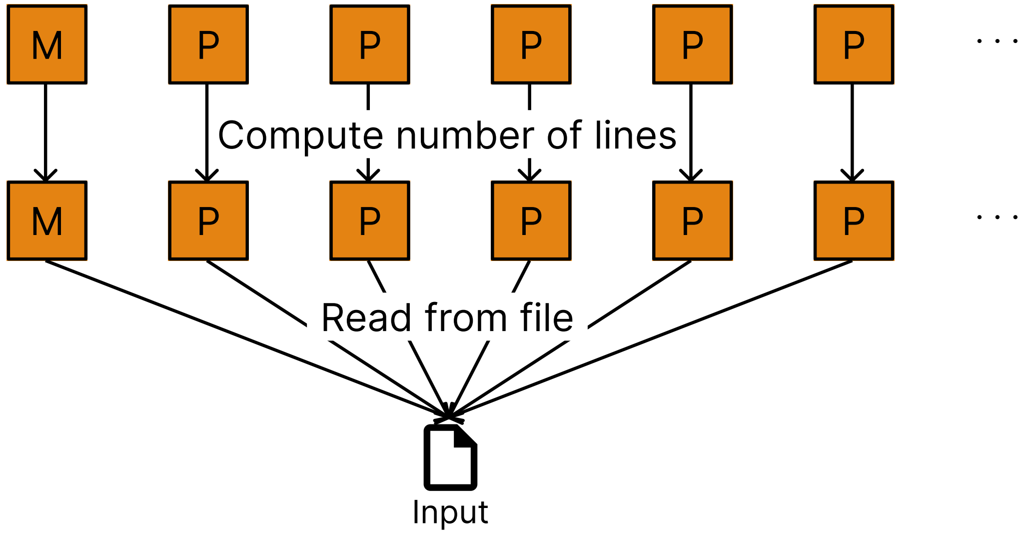
Parallel approach 2

Parallel approach 2

We focused on:

- Parallel file reading
- Parallel determinant

Parallel file reading



- Each process computes its number of samples
- The reading procedure is performed parallelly by every process at the same time

Parallel determinant

We implemented OpenMP to parallelize part of the computation of the determinant

```
#pragma omp parallel for reduction(* : det) num_threads(2) schedule(static, 1)
for (int i = 0; i < size; i++)
{
  int ind = i * size + i;
  double value = matrix[ind];
  det *= value;
}
```

Performance evaluation

Tests

Each test was conducted on the HPC cluster with different configurations:

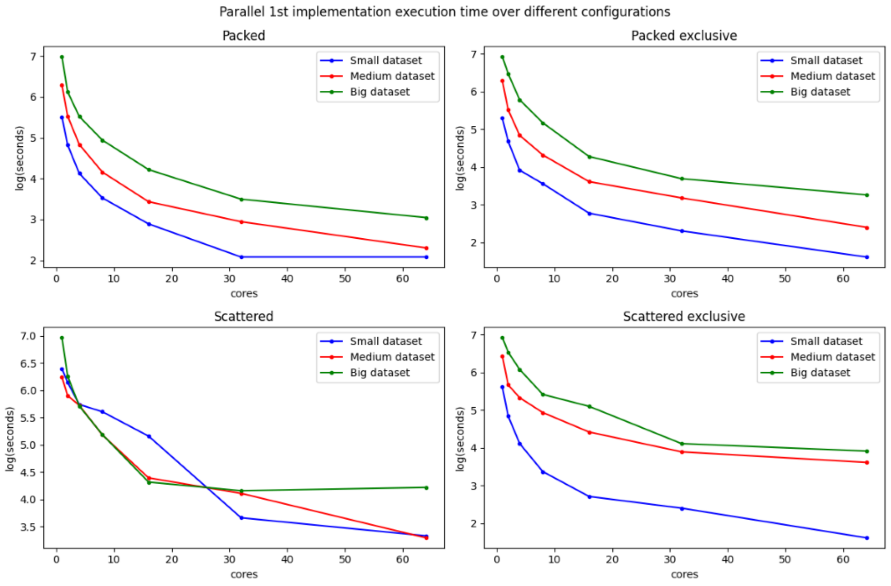
- Packed
- Packed exclusive
- Scattered
- Scattered exclusive

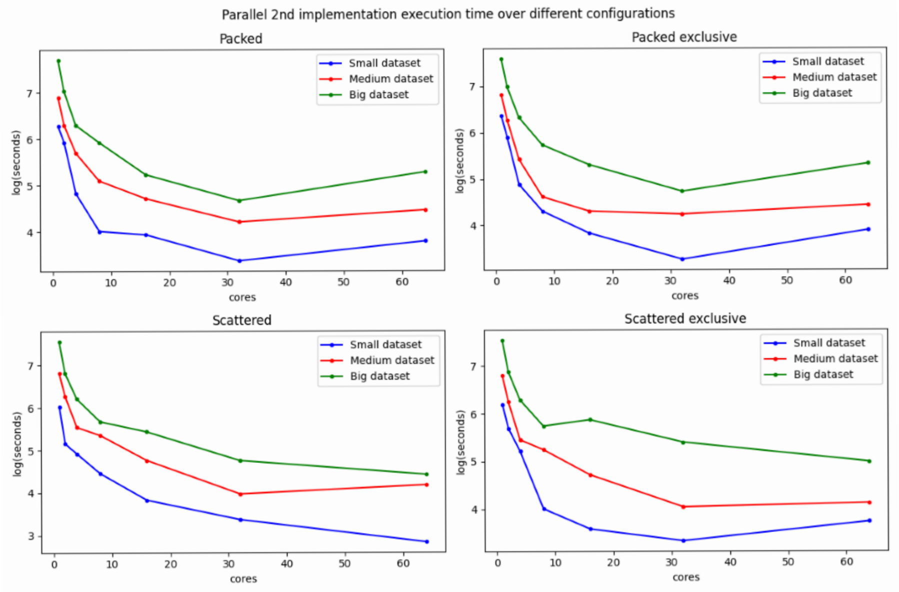
Datasets

The tests were conducted on the following datasets:

- Small: 250000 samples, 5 Gaussians and 4 dimensions
- Medium: 625000 samples, 5 Gaussians and 4 dimensions
- Big: 1250000 samples, 5 Gaussians and 4 dimensions
- 6-dim: 20000 samples, 4 Gaussians and 6 dimensions
- 8-dim: 4000 samples, 4 Gaussians and 8 dimensions

Execution time

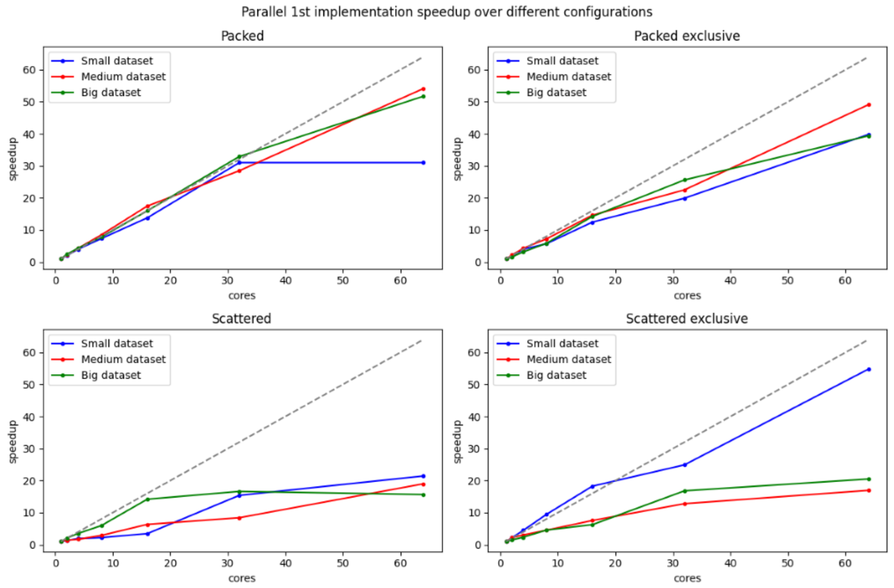


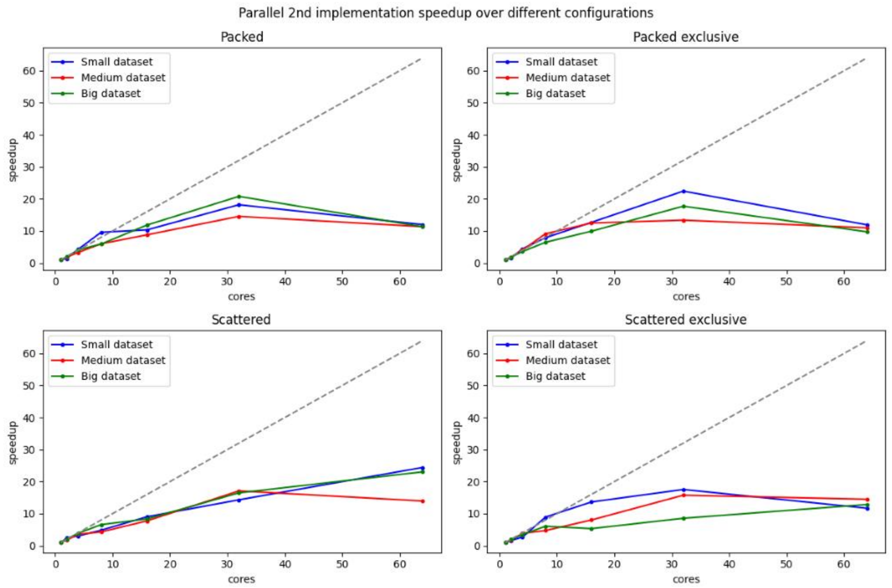


- Effective execution time reduction
- Variations in *Scattered* configurations

- Effective execution time reduction
- Slower than first counterpart

Speedup

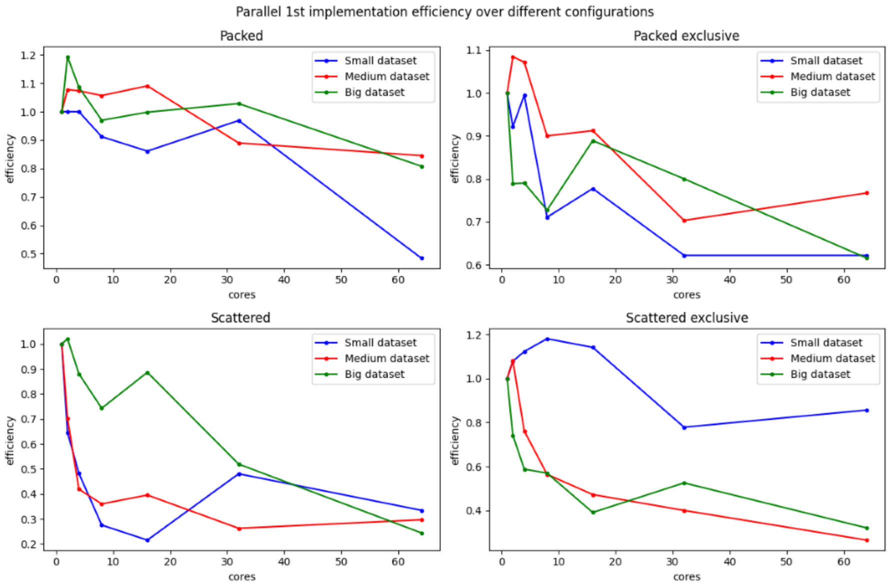


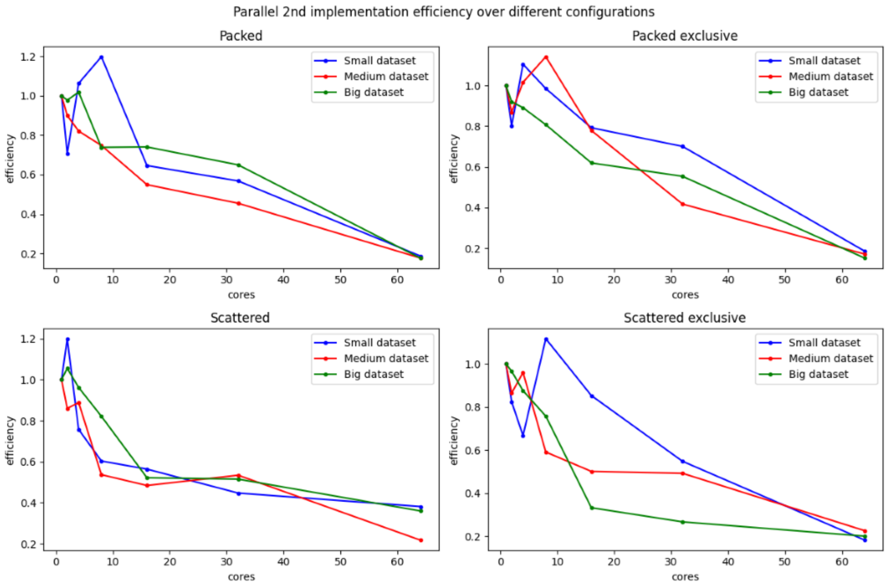


- Almost linear speedup with some exceptions
- Variations in *Scattered* configurations

- Sub-linear speedup
- Worse speedup than first counterpart

Efficiency

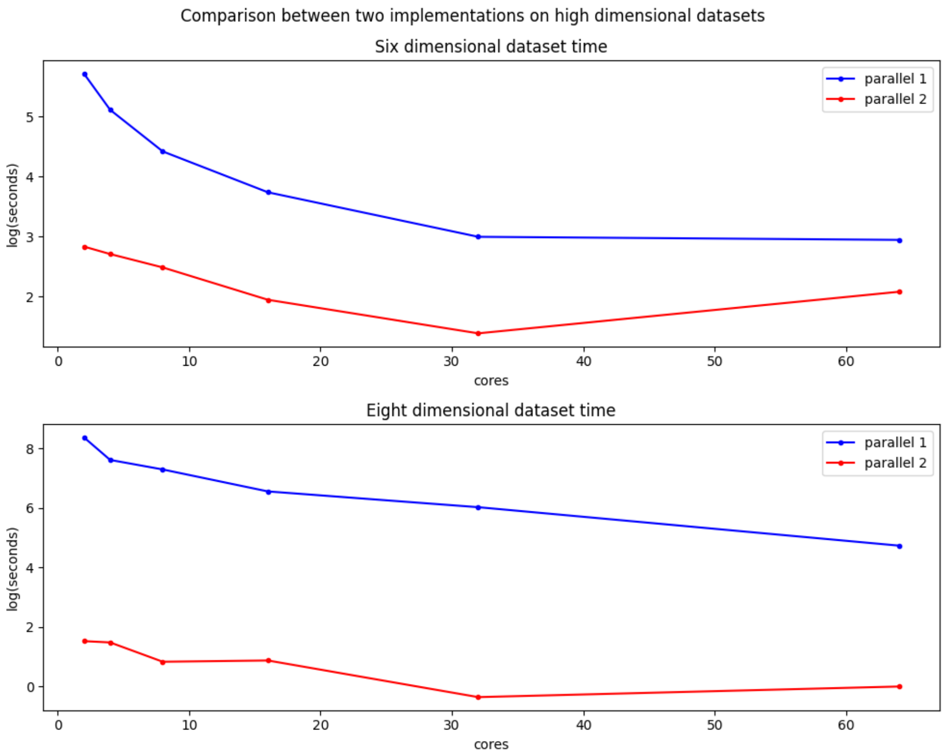




- Good efficiency in *Packed* configurations
- Reduction in efficiency in *Scattered* configurations

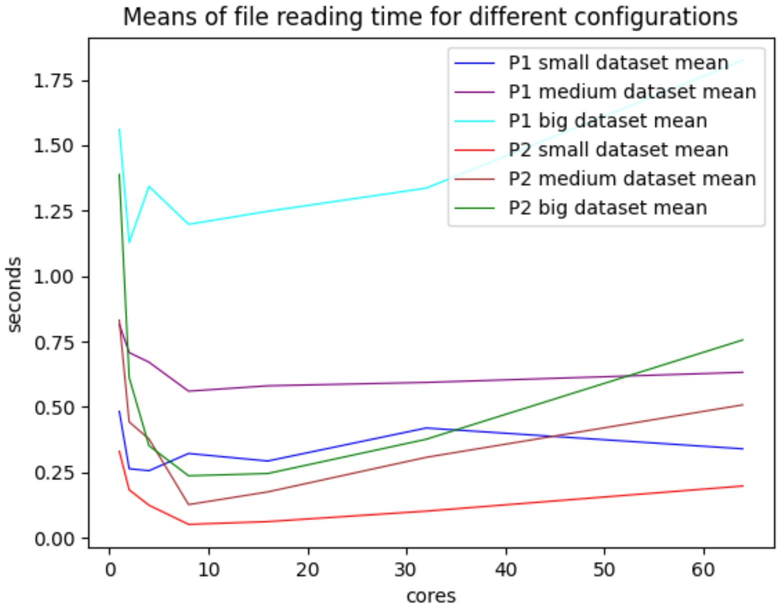
- Overall bad efficiency
- Worse efficiency than first counterpart

High-dimensional datasets



- Effective at reducing execution time in high-dimensional datasets
- Poor performance on low-dimensional datasets

File reading time



- Effective at reducing file reading time
- Compared to the algorithm the time reduction is minimal

Thank you for your kind attention