Parallel programming

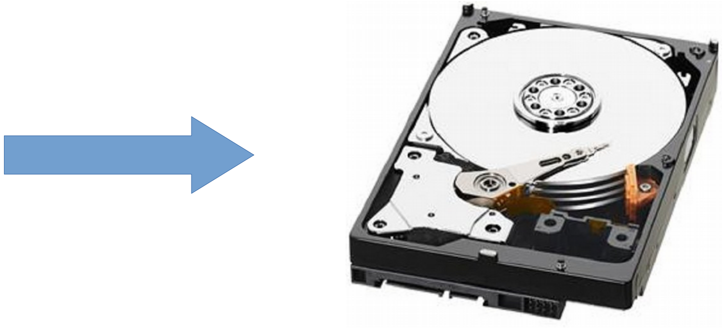
Semester project
Data storage optimization

FAKULTA
ELEKTROTECHNICKÁ
ČVUT V PRAZE

Storing lot of data

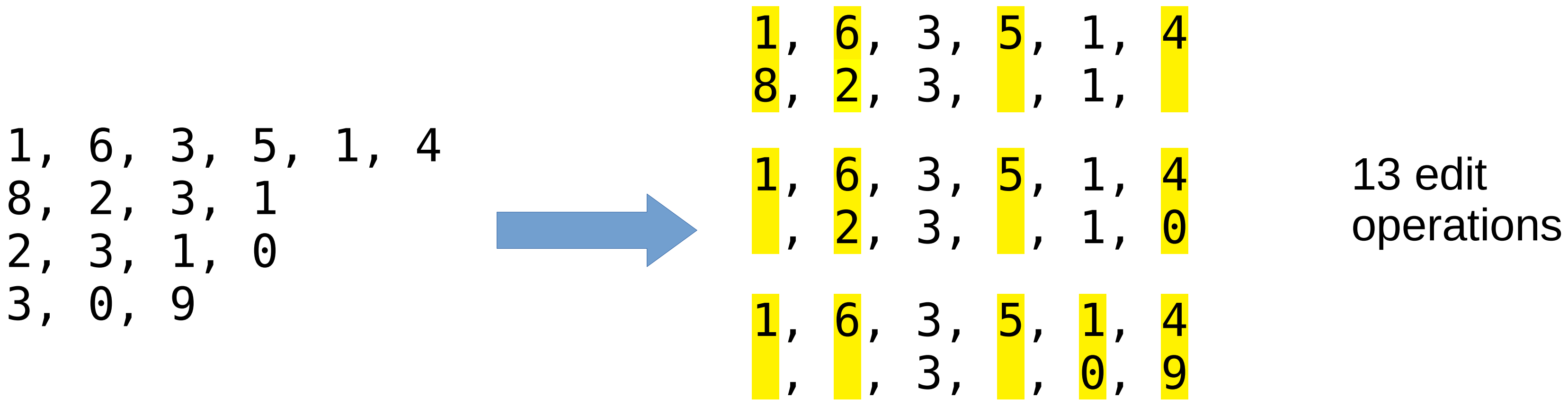
- We are given *n* records
  - A *record* is a sequence of integers, e.g.,
    - `1, 6, 3, 5, 1, 4`
- Our goal is to store them to disk in the most memory-efficient way possible.

```
1, 6, 3, 5, 1, 4
8, 2, 3, 1
2, 3, 1, 0
3, 0, 9
```



Storing records efficiently

- How to store the records in memory-efficient way?
  - Find the *edit difference* between the records and store only the differences
    - Edit difference = Levenshtein distance
  - The original records then can be restored by re-applying the differences



```
1, 6, 3, 5, 1, 4
8, 2, 3, 1
2, 3, 1, 0
3, 0, 9
```

Original records

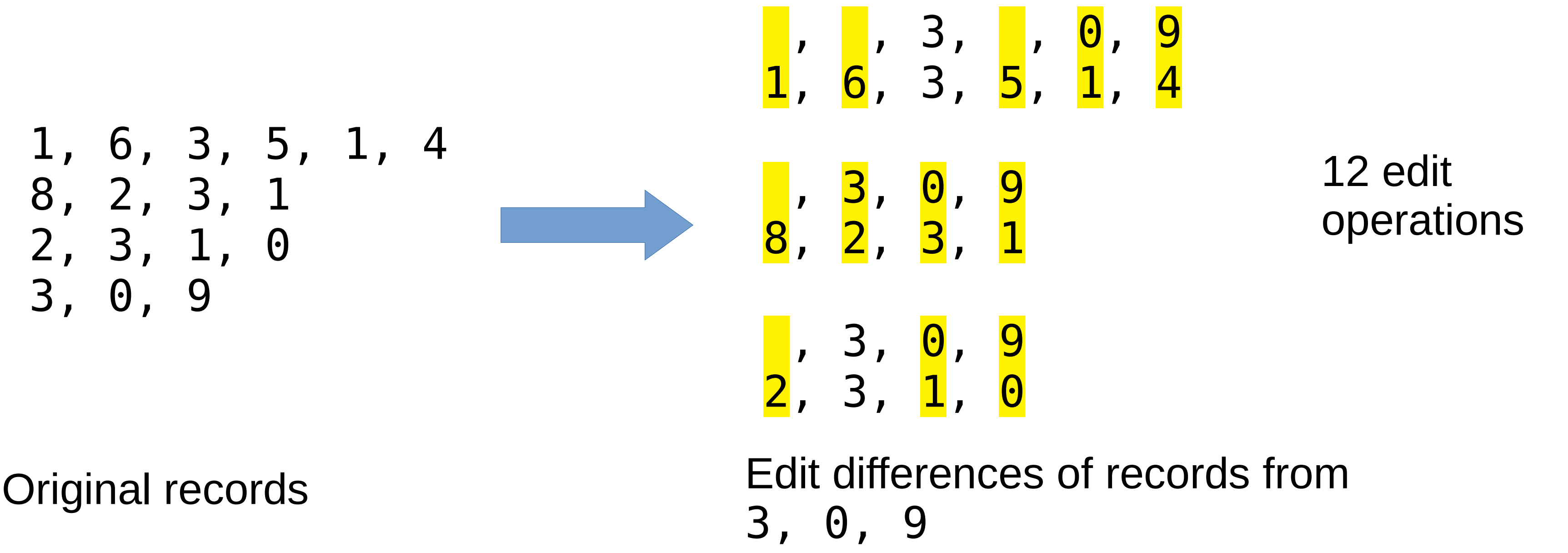
```
1, 6, 3, 5, 1, 4
8, 2, 3,  , 1,  

1, 6, 3, 5, 1, 4
 , 2, 3,  , 1, 0

1, 6, 3, 5, 1, 4
 ,  , 3,  , 0, 9
```

13 edit operations

Edit differences of records from
`1, 6, 3, 5, 1, 4`

Which differences??!

- Clearly, the number of stored edit differences depends on the record from which the difference is computed



```
1, 6, 3, 5, 1, 4
8, 2, 3, 1
2, 3, 1, 0
3, 0, 9
```

Original records

```
 ,  , 3,  , 0, 9
1, 6, 3, 5, 1, 4

 , 3, 0, 9
8, 2, 3, 1

 , 3, 0, 9
2, 3, 1, 0
```

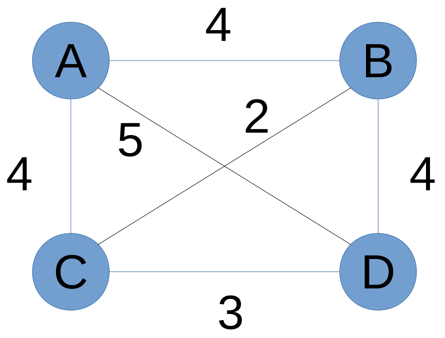
12 edit operations

Edit differences of records from `3, 0, 9`

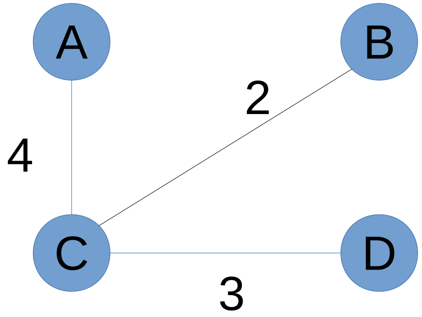
Tree of distances

- Better approach
  - Compute the edit difference between every pair of records

```
A: 1, 6, 3, 5, 1, 4
B: 8, 2, 3, 1
C: 2, 3, 1, 0
D: 3, 0, 9
```



  - Find the *minimum spanning tree* on the complete graph (e.g., using Prim’s algorithm)



9 edit operations

Semester project assignment

- Implement the **sequential** and **parallel** version of the data storage optimization
  - You have to decide how to parallelize the code (algorithms or some other parts?)
    - Use Intel Parallel Studio to find the **bottlenecks**!
  - Choose either C++11 threads, OpenMP or MPI
    - If you choose MPI see UploadSystem requirements
  - Use the provided skeleton code that already implements the reading/writing of input/output
- Prepare the **presentation** to show the achieved results during Lab 14 (**mandatory**, otherwise no points given for semester project!)

Evaluation of semester project

- UploadSystem, independent evaluation (7 points)
  - Upload your parallel algorithm to UploadSystem
  - Automatic evaluation on a set of private instances (the size of instances will be known)
  - The number of given points depends on how fast your algorithm finds the correct solution
- UploadSystem, contest (3 points)
  - Comparison of students
  - Points given based on how fast your algorithm is in comparison with the others
- Report (optional)
  - Describe how the parallelization was done (1 point)
  - Profile the bottlenecks for sequential code (1 point)
    - Profiler outputs (Intel Parallel Studio), detection of bottlenecks, analysis.
  - Parallel code executed on Metacentrum (2 points)
    - Scalability and performance graphs and other performance metrics (measured on Metacentrum).
    - Metacentrum: PBS scripts, hardware info, how to carry out the experiments.

Requirements for your presentation

- **Base structure**:
  - Introduction
  - Scalability graph
  - Performance graph
  - Discussion and conclusion
- The presentation should have at max **10 slides**.

Presentation - Introduction

- Mention the used technology (C++11 threads, OpenMP, MPI)
- What was parallized, bottlenecks analysis

Presentation – graphs

- Graphs:

  1) Speedup of *parallel CPU version* vs *sequential version* (**scalability graph**)

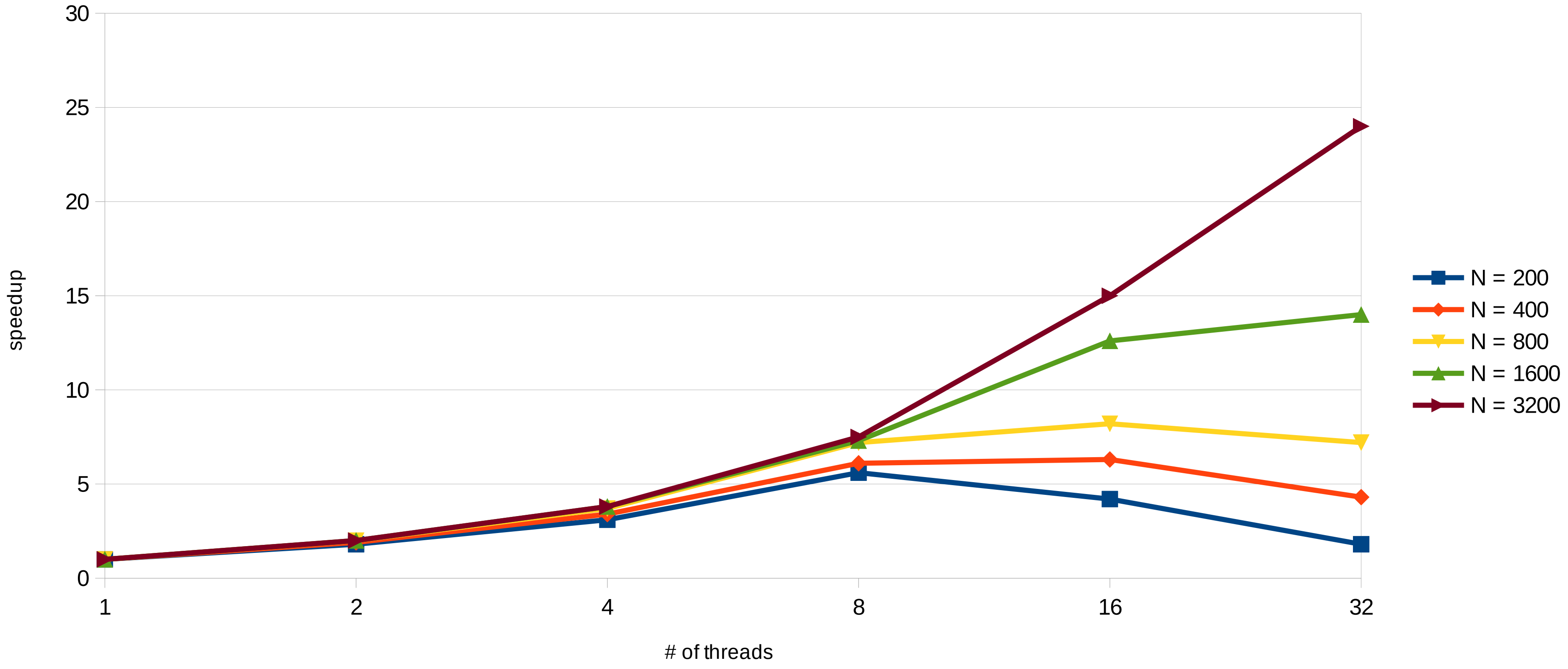
  2) Graph showing the algorithm runtime based on the size of an input instance (**performance graph**).

- Each graph should have a title, legend, and an appropriate format of axes (+units)
- Description of the hardware and software

Presentation – Scalability graphs

- Shows the speedup with respect to the number of used threads
  - Scalability graph for up to 256 threads

Presentation – Scalability graph

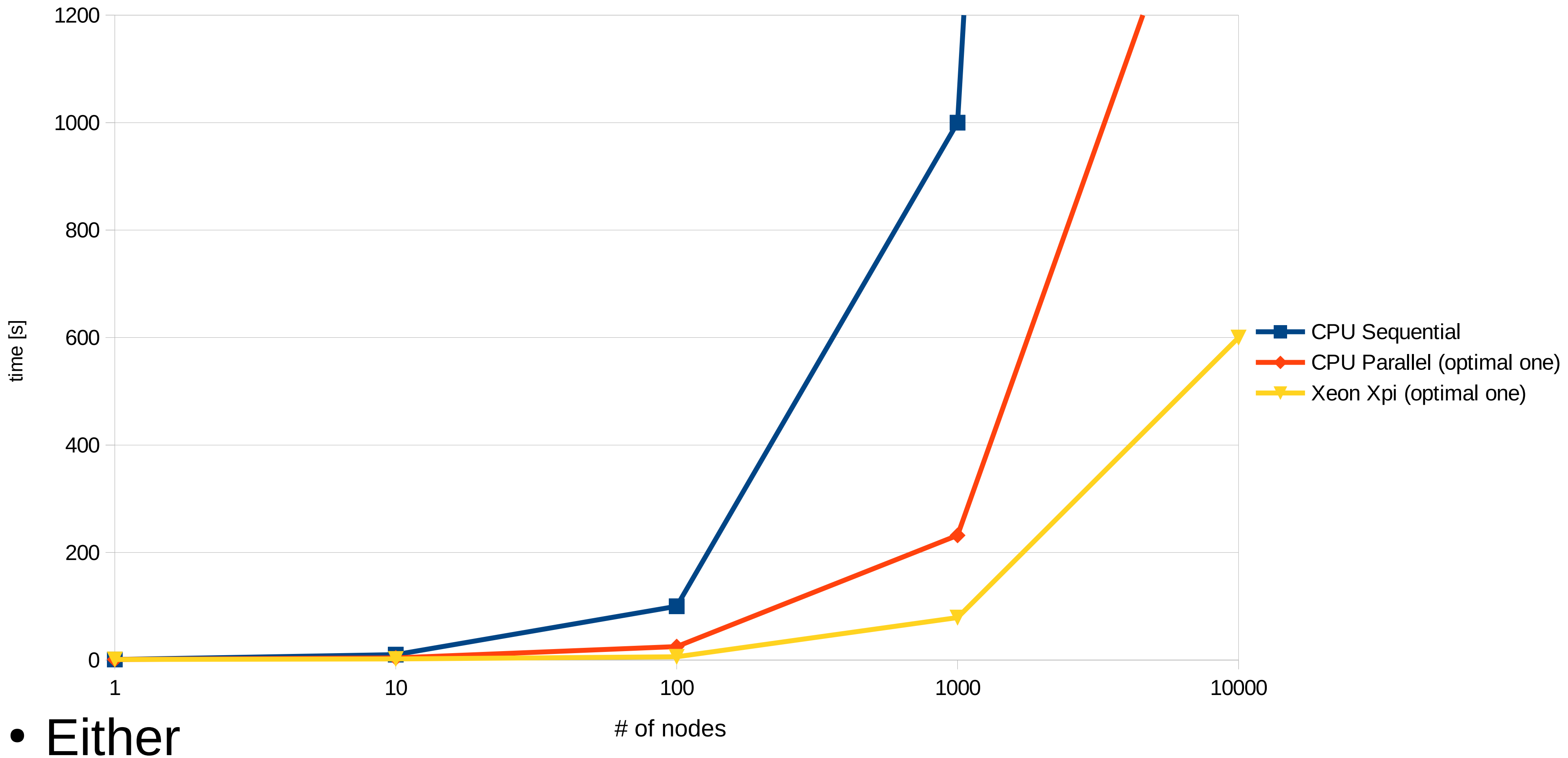


- N – number of records

Presentation – Performance graph

- Shows how much time the algorithm takes to finish the computation depending on the number of records and their maximum length

Presentation – Performance graph



- Either
  - 2D graph (# of nodes on one axis, max record length on other axis)
  - Fix one parameter (e.g., number of nodes) and vary the other parameter (e.g., max record length) + vice versa

Discussion and conclusion

- Explain what was the most complicated part and why the results are as provided.
- What is the limiting factor of the parallelisation in your algorithm.