Comparison of two approaches for scheduling program graphs for dynamic SMP clusters with communication on the fly

Marek Tudruj, Łukasz Maśko

Institute of Computer Science Polish Academy of Sciences
Polish-Japanese Institute of Information Technology
Warsaw, Poland

Two compared algorithms

- Clustering-based algorithm using Moldable Tasks idea.
- List-scheduling based 2-phase algorithm

Moldable tasks

- **Parallel tasks** in a parallel program can be executed in parallel way on a variable number of processors.
- **Malleable tasks** are parallel tasks for which a number of assigned processors may change during execution.
- **Moldable tasks (MT)** are parallel tasks, for which the number of assigned processors is not fixed, but is determined before execution and then doesn't change.

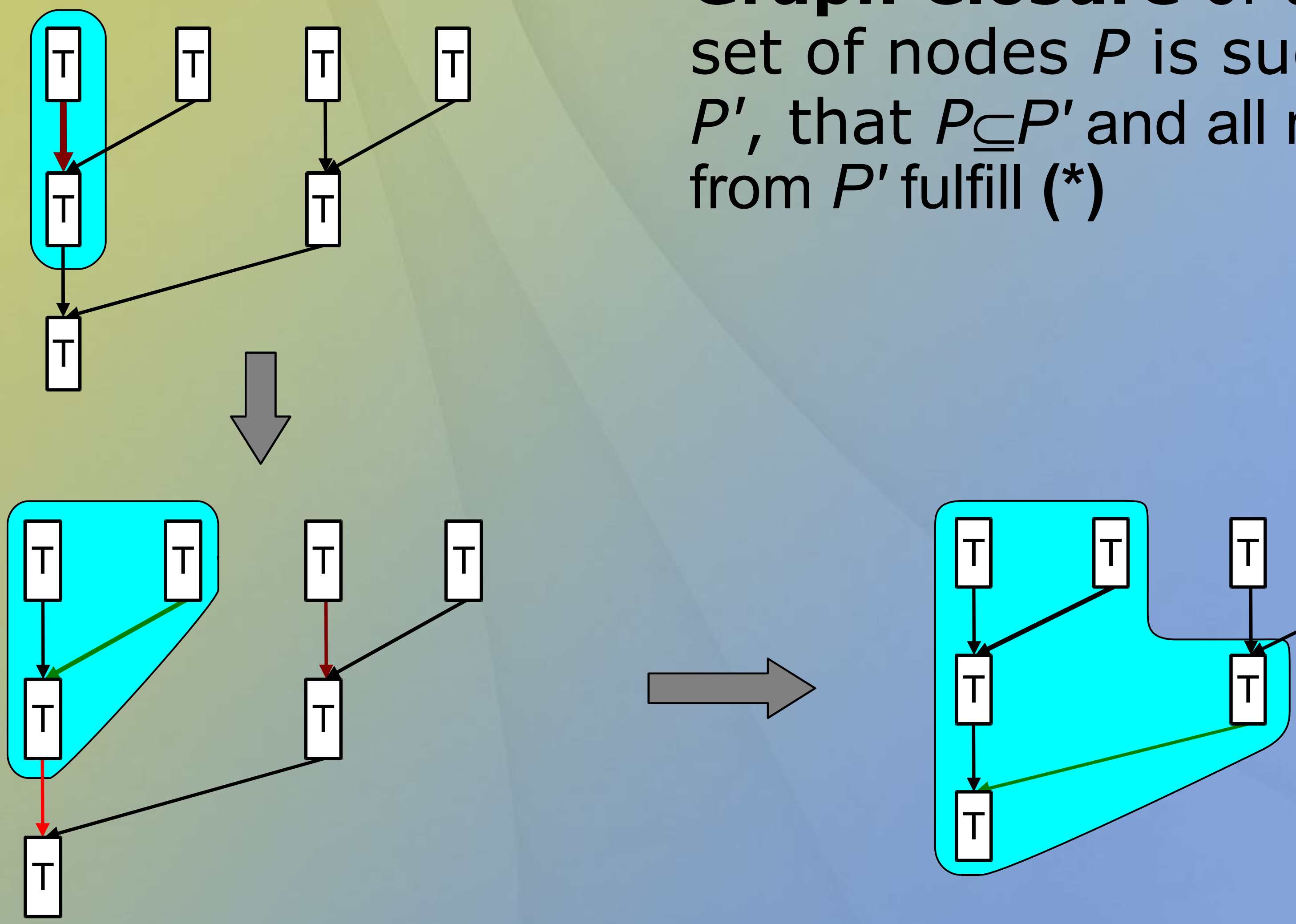
General MT-based algorithm

- This algorithm consists of three steps:
  - building a MT data flow graph, based on the program initial macro data flow graph,
  - defining the best internal structure of each MT node for each number of processors (schedule of component nodes to logical processors inside CMP modules of different sizes).
  - Defining an assignment of resources to MTs (allotment) and scheduling the MT graph in the architecture with simplified inter-CMP connections (fully connected network).

Step 1:
Definition of moldable tasks (*)

- A moldable task is built of nodes mapped to only one CMP module and its size is bounded.
- Moldable tasks are defined using a clustering algorithm, bounded by an assumed maximal cluster subgraph width.
- Clustering operation includes finding a graph closure of a task in respect to incoming and outgoing communication egdes.
- Clustering leads to reduction of global communication. Merging separate MTs to create a larger one transforms all the communication between them into locally executed communication inside a CMP module.

Graph closure of a MT graph



- **Graph Closure** of a given set of nodes *P* is such set *P'*, that $P \subseteq P'$ and all nodes from *P'* fulfill **(*)**

ETF-based clustering criterion

- A list scheduling algorithm with an ETF heuristics can be used to determine if the size of a clustered (merged) task is acceptable:
  - Schedule a task on unbounded number of resources using list scheduling with ETF heuristics.
  - Check the numbers of processors and busses used. These numbers constitute the size of this task.

Step 2:
Moldable tasks execution times

- Determining an execution time function for every MT obtained in the previous step. It is done by finding the best schedule for a given MT on 1 to N processors and determining its execution time (N is the number of processors in a SoC module).
- The extended graph representation includes reads on the fly, processor switchings and synchronization.
- An ETF list scheduling algorithm is used, based on basic control structures and their transformations – standard communication in subgraphs is converted into reads and communication on the fly. It determines the best heuristic MT execution times.

Step 3' : Processor assignment to MTs

- In this step an assignment of processors to moldable tasks is found.
- We have used a layer-based assignment algorithm:
  - A first layer of nodes are the nodes with no predecessors in the graph.
  - The layer n is a set of nodes with no predecessors in a graph with nodes from previous layers removed.
  - For each layer the best allotment is found using a greedy algorithm.

Step 3'' :
Final scheduling of MTs

- Step 3' assigns particular fixed numbers of processors to each MT. Therefore they can be scheduled as standard tasks with fixed resource requirements.
- This step consists in scheduling such tasks in the assumed architecture (assigning nodes to processors, communication to particular memory busses and modules) with a fully connected global network.
- For this step, a list scheduling algorithm with the ETF heuristics is used.

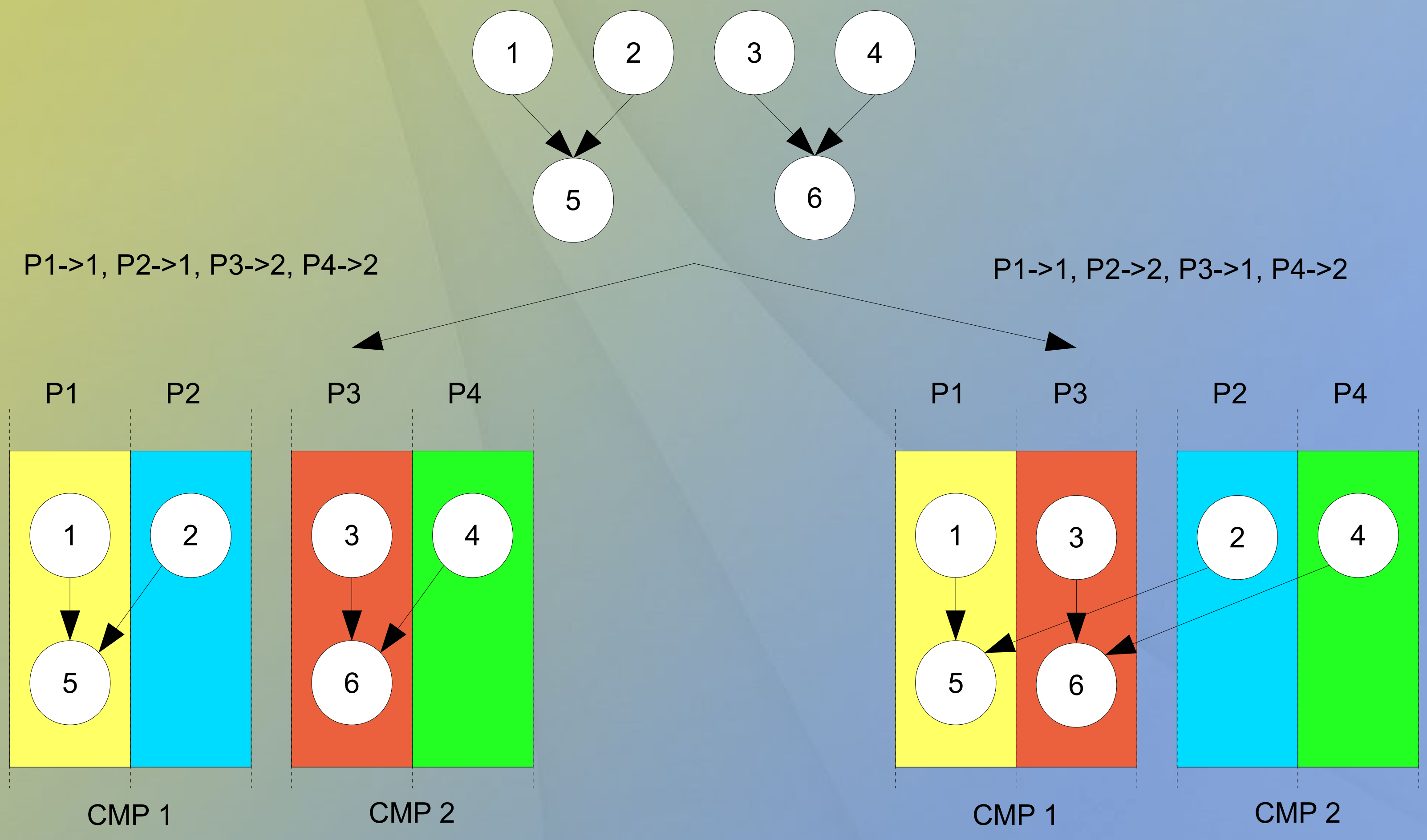
2-phase scheduling algorithm

- 2-phase scheduling algorithm derived from list scheduling.
- This algorithm works in 2 phases:
  - Distribution of nodes between processors and processors between SoC modules, using a genetic algorithm supported with list scheduling with ETF heuristics.
  - Structuring of communication between processors.

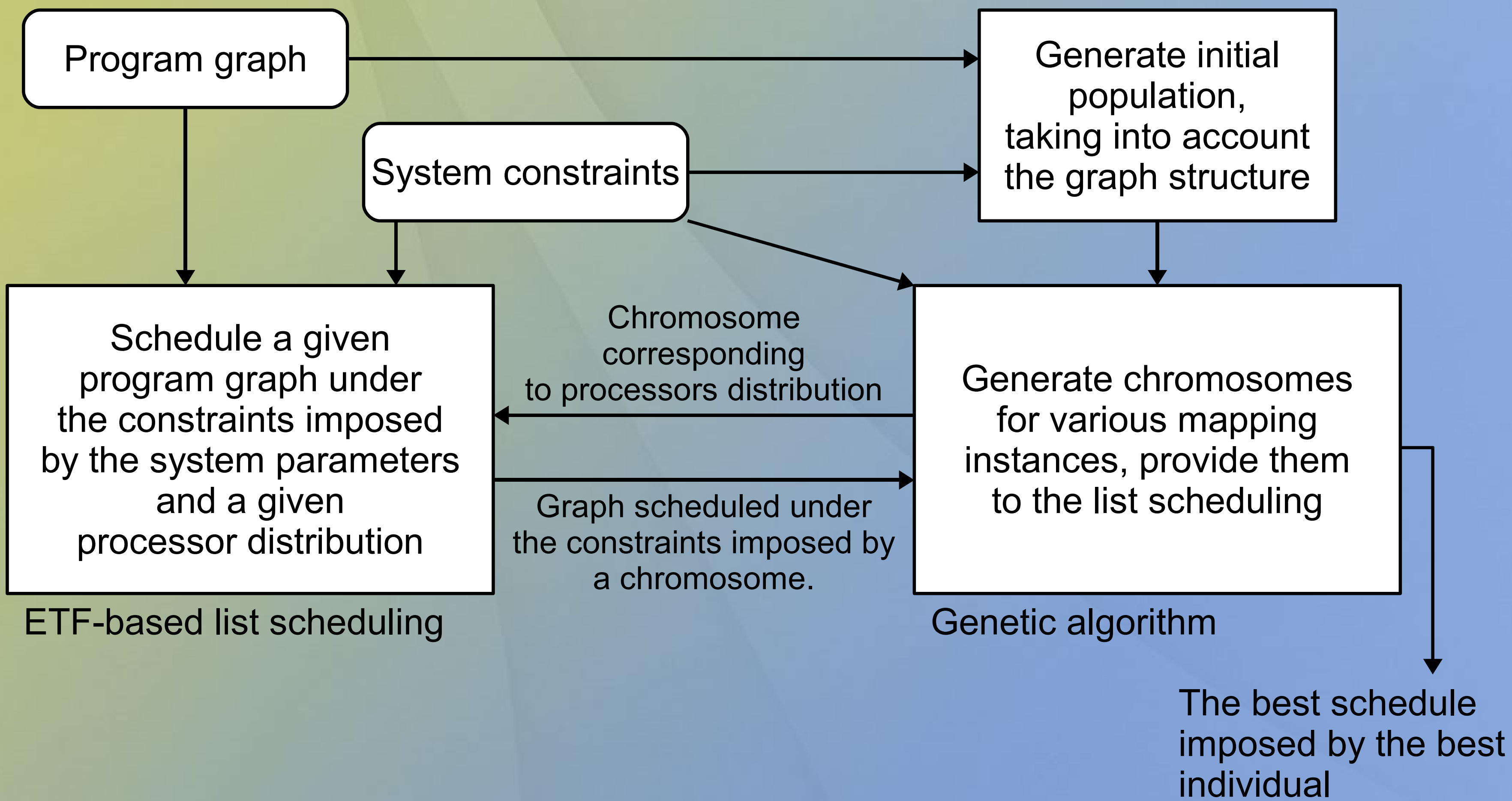
Phase 1

- In the first step, tasks are assigned to processors and processors are assigned to SoC modules in the system.
- This step reduces execution time of the graph by reduction of the number of global communication between processors.
- It uses a genetic algorithm supported by list scheduling with ETF heuristics.
- Each chromosome represents one distribution of processors between SoCs.

Why processor mapping is important



Phase 1


Program graph
System constraints
Generate initial population, taking into account the graph structure
Schedule a given program graph under the constraints imposed by the system parameters and a given processor distribution
Chromosome corresponding to processors distribution
Graph scheduled under the constraints imposed by a chromosome.
Generate chromosomes for various mapping instances, provide them to the list scheduling
ETF-based list scheduling
Genetic algorithm
The best schedule imposed by the best individual

Phase 1

- For each chromosome, the schedule is determined by a list scheduling with ETF heuristics, that takes into account both global and local communication.
  - Assignment of tasks without predecessors is determined by the order, in which these tasks are scanned by the list scheduler – each consecutive task is assigned to the first available processor (ordered with their indexes).
  - Further assignments are influenced by distribution of processors between CMP modules – global communication is slower.

Phase 1

- A value of a fitness function *Fit* for a chromosome *C* is determined by the following formula:

$$Fit(C) = \frac{\sum_{v \in E} end(v)}{|E|}$$

were *E* is a set of nodes without successors in graph *G* scheduled according to a distribution defined by chromosome *C*.

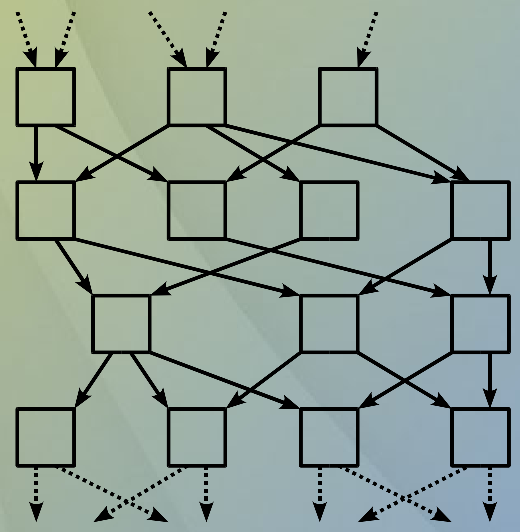
Phase 2

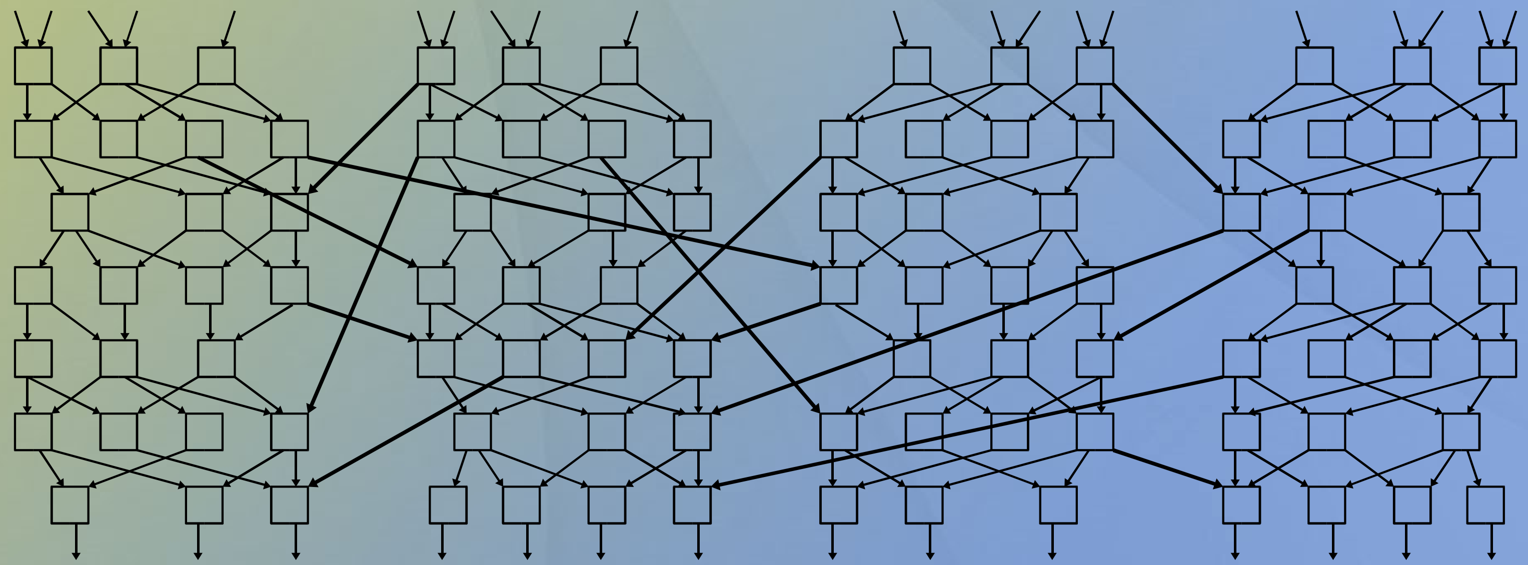
- After the first phase, all the nodes in the graph are assigned to processors distributed between SoCs. Therefore for each communication it is determined, if it is a global or local one.
- In the second phase, for each local communication it is determined, if it will be performed on the fly or in the standard way.
  - Data transfers on the fly are faster, but they may introduce latencies and require additional graph structuring.

Phase 2

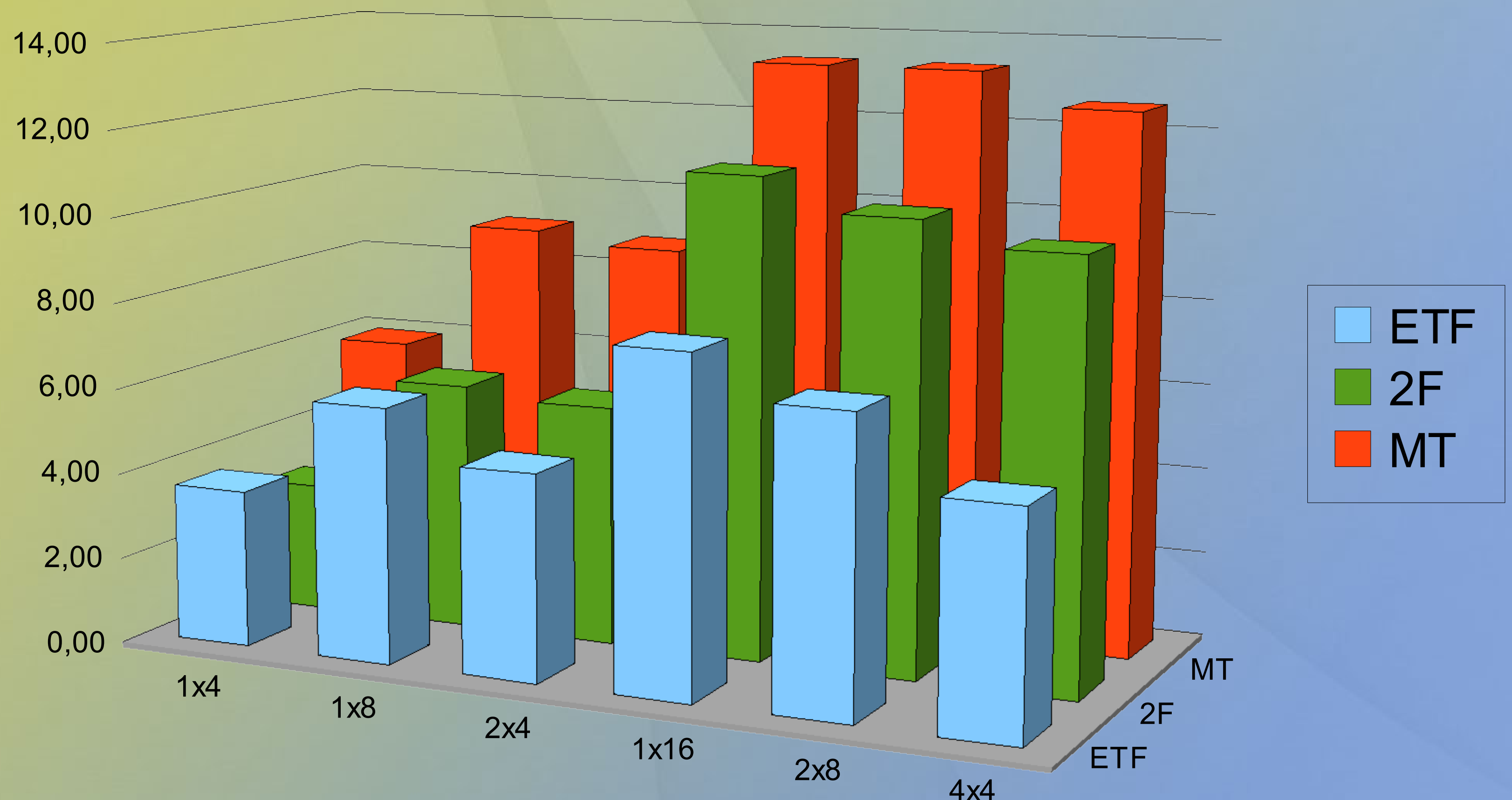
- For transformation of communication, basic structures and their transformations are used.
- The basic structures are determined and ordered using graph traversal based on Critical Path.

Experimental graphs

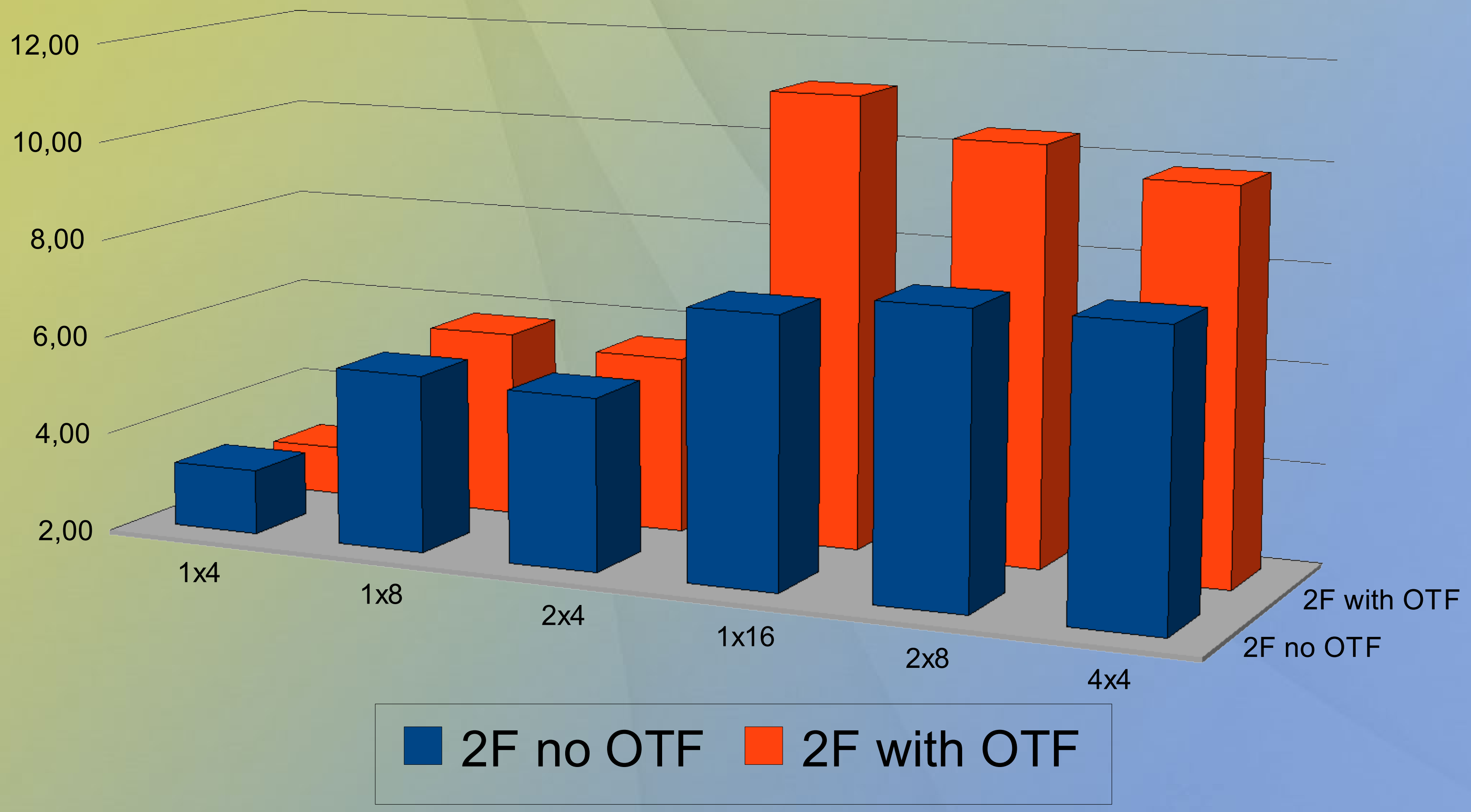




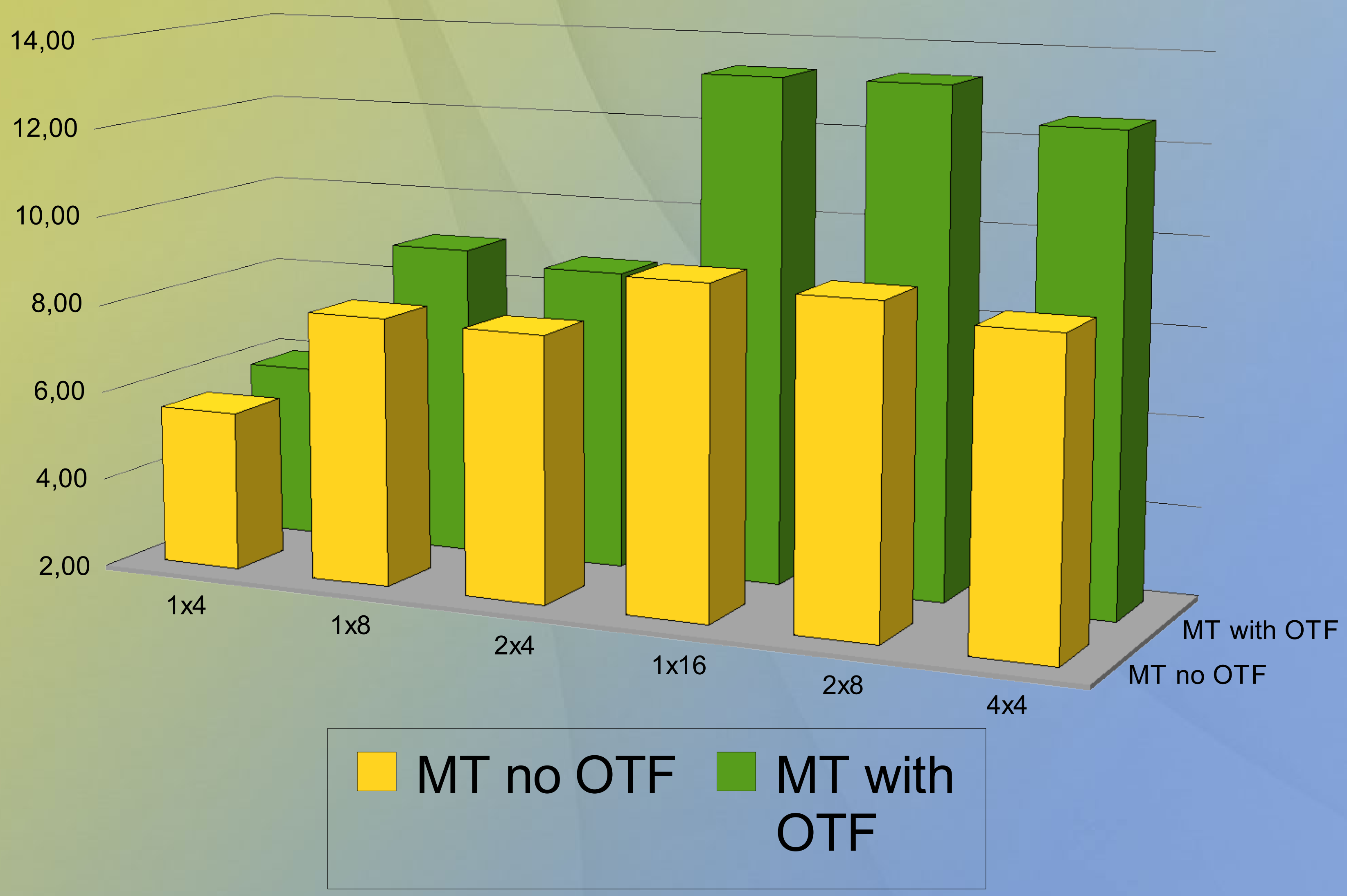
Comparison of results



Influence of reads on the fly (2F)



Influence of reads on the fly (MT)



The end