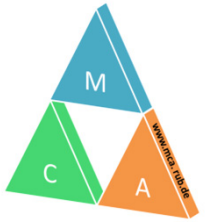
RUHR-UNIVERSITÄT BOCHUM

RUB

Clustering and Mapping Algorithm for Application Distribution on a Scalable FPGA Cluster

Lester Kalms | Diana Göhringer
Ruhr-University Bochum, Germany
lester.kalms@rub.de | diana.goehringer@rub.de



Research Group for
Application-Specific Multi-Core Architectures

RUB

Content

- Motivation
- Application Distribution
  - General Description
- Implementation
  - Overview
  - Description
- Evaluation
- Summary
- Outlook

Motivation

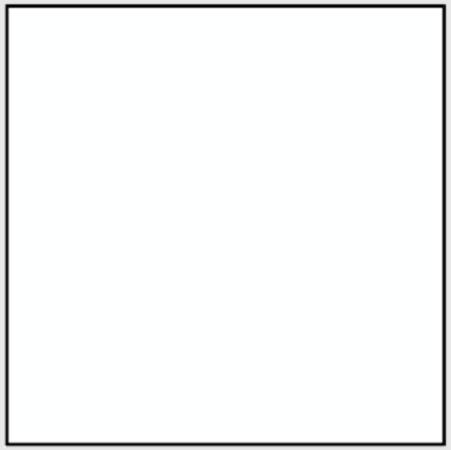
Why do we need an FPGA cluster?

Advantage	Disadvantage
Reduced costs	Latency
Expandability	Bandwidth
Bigger designs	Fragmentation

Major tasks

- Which topology?
- How does the communication between the FPGAs take place?
- How will the application(s) be distributed among the FPGAs?

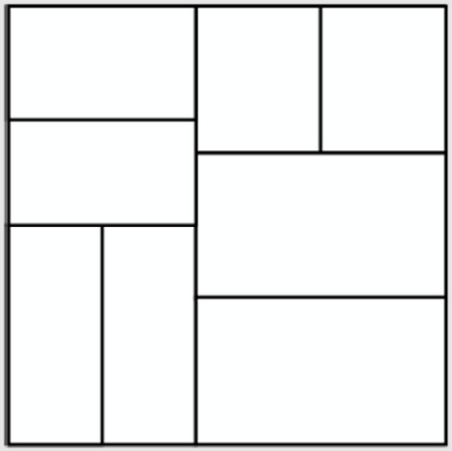
Application Distribution



Partitioning

Start with an application

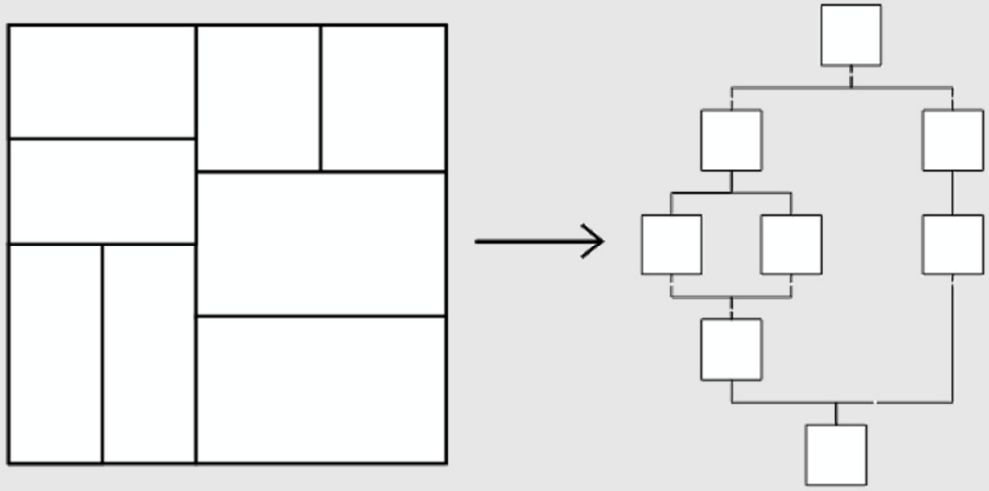
Application Distribution



Partitioning

Partition the application into tasks

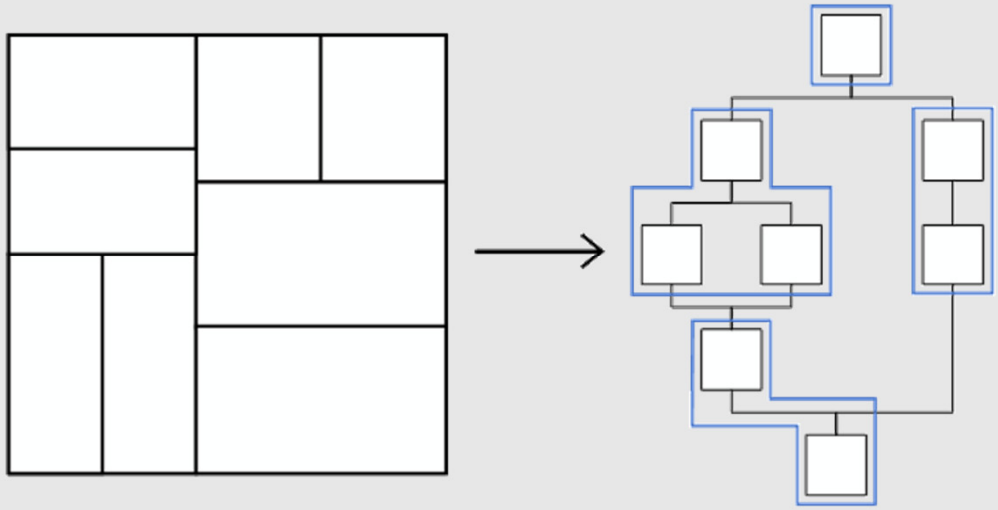
Application Distribution



Partitioning Clustering

Create a task interaction graph (TIG)

Application Distribution

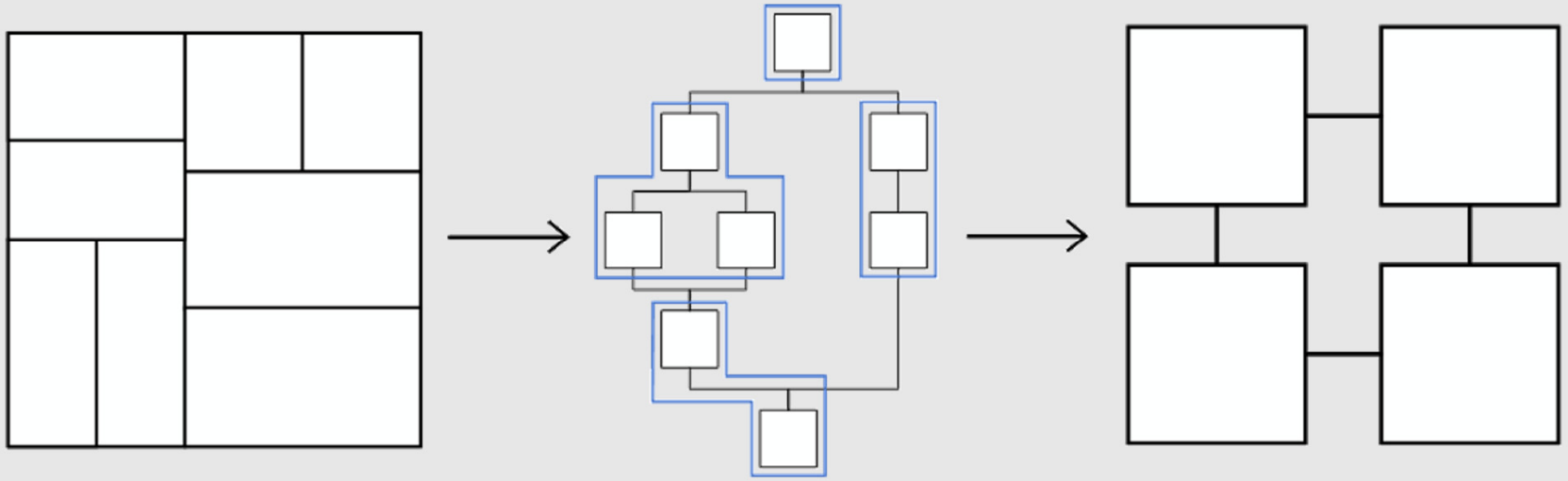


Partitioning

Clustering

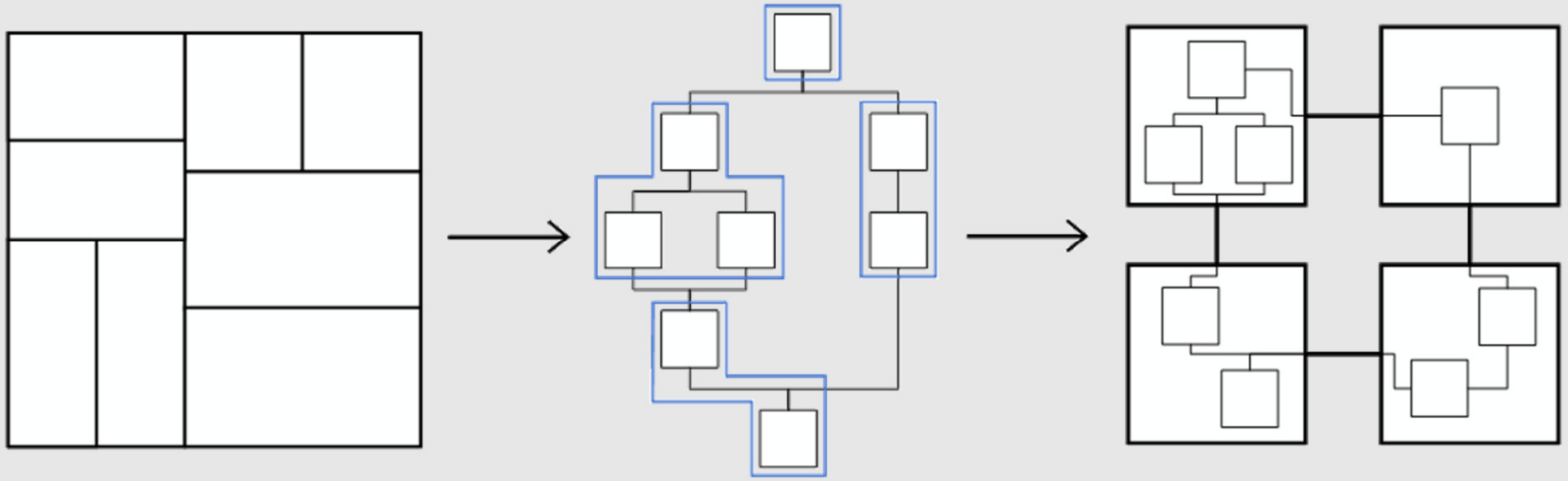
Cluster these tasks

Application Distribution



Map the cluster to the board connection graph (BCG)

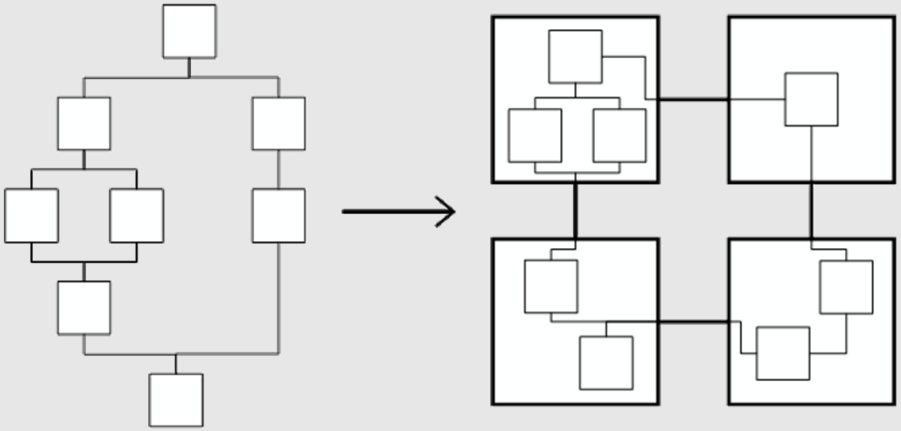
Application Distribution



Clustering and mapping in 2 separate steps holds the danger of some loss of optimality

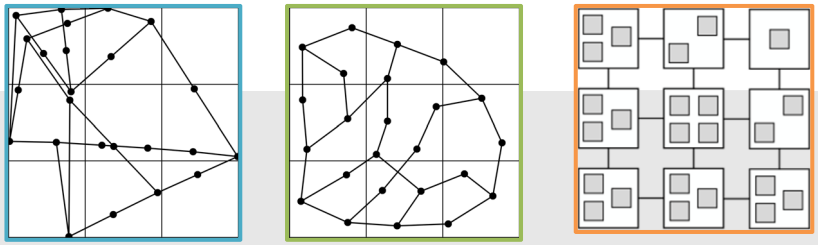
Application Distribution

- This work focuses on clustering and mapping (in 1 step)
- Partitioning is done by the developer



- Clustering and mapping are known as NP-hard for irregular graphs
- E.g. 9 FPGAs and 24 tasks would result in 9^{24} different mapping solutions
- Therefore we make use of heuristics and load balancing

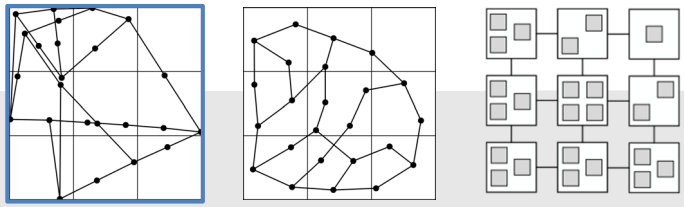
Algorithm Overview



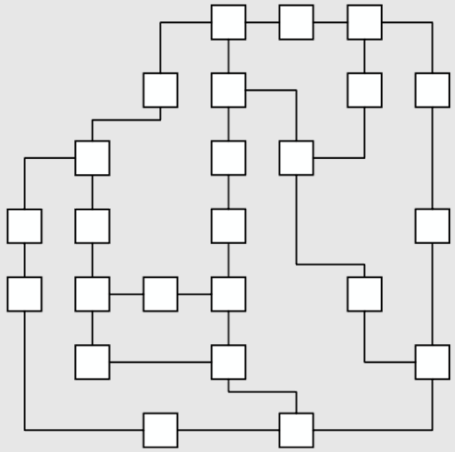
1. Create initial graph
   - Based on the Task Interaction Graph
2. Load Balancing
   - Connected and unweighted edges are forced together
   - All tasks are forced apart from each other based on their capacity utilization
3. Optimization Steps
   - Reduce the maximum dilation between the tasks
   - Reduce the maximum capacity utilization of the FPGA cluster

Several solutions are computed in parallel and the best solution is chosen

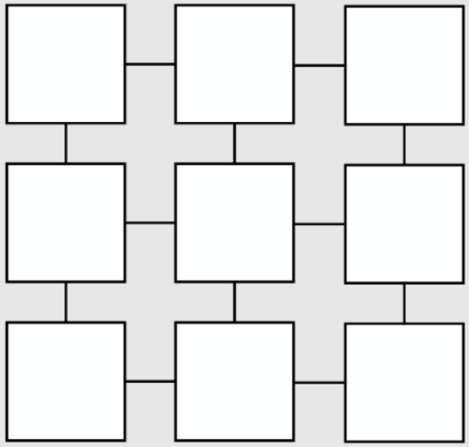
Create Initial Graph

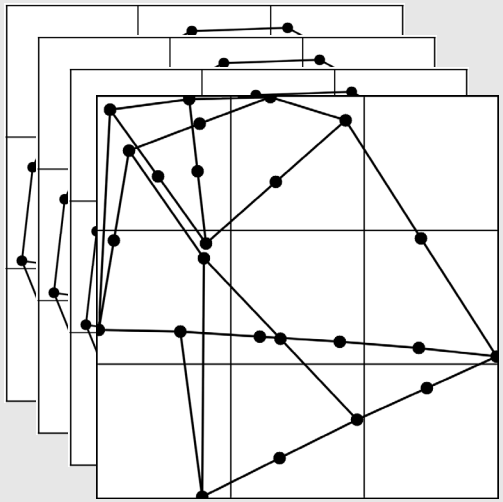


- An initial guest graph is created based on the TIG
- It is created in a 2 dimensional Euclidean vector space
- For each solution 1 guest graph is created
- The guest graph will be embed into a host graph (BCG)



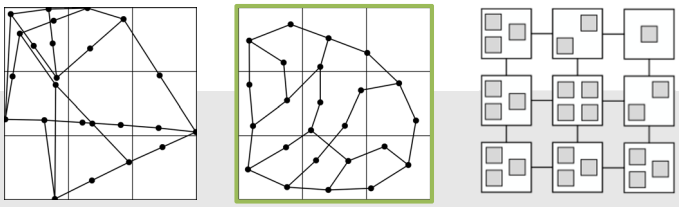
&



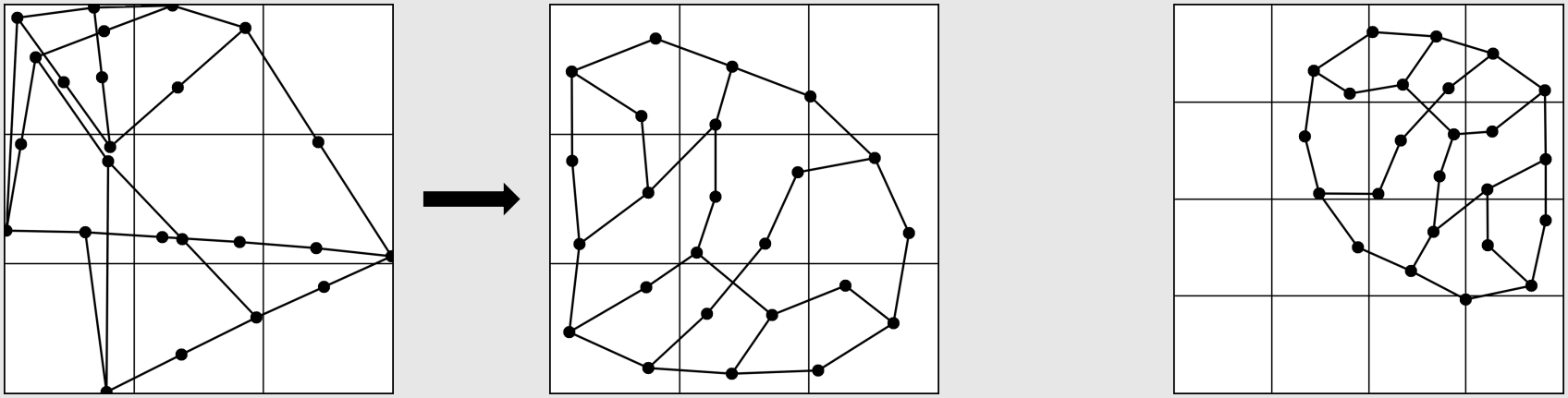


Task Interaction Graph (TIG) Board Connection Graph (BCG) Host Graph / Guest Graphs

Load Balancing



- The guest graph is unfolded within the boundaries of the host graph
- Connected and unweighted edges are forced together, like a rubber
- All tasks are forced apart from each other based on their capacity utilization
- Guest graph does not utilize much more area than needed within the host graph

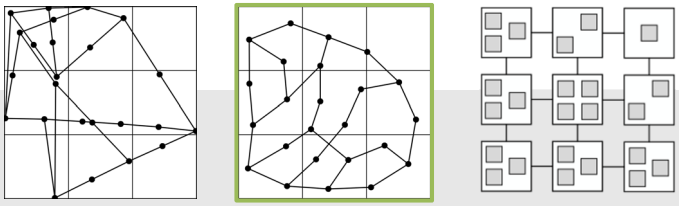


Initial graph

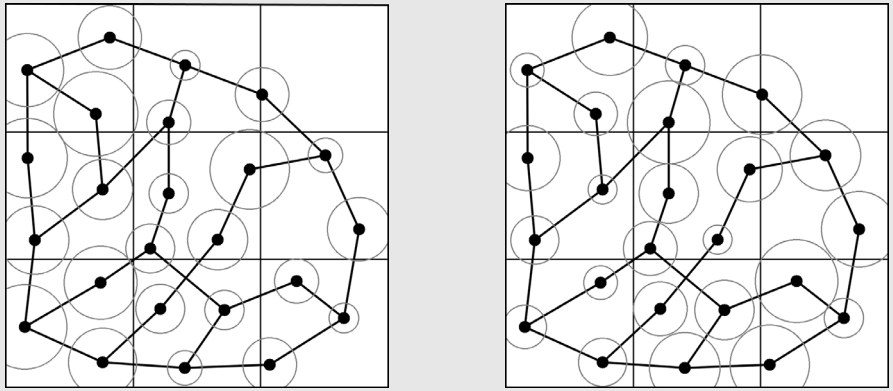
Graph after load balancing

Graph within a bigger cluster

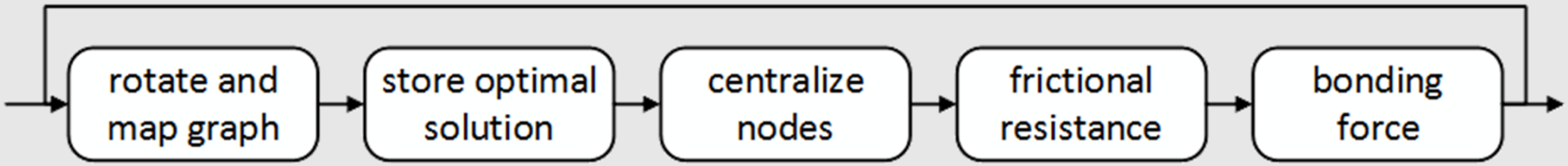
Load Balancing



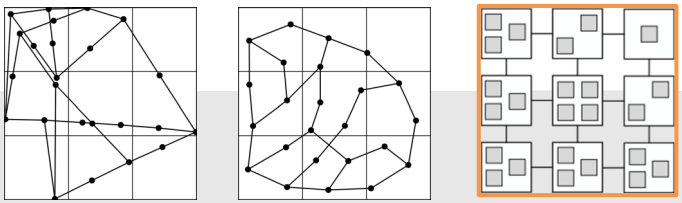
- Graph after load balancing, showing the different capacity utilizations



- Load balancing contains several steps, which are repeated (e.g. 256 times)
- Not every step is computed all the time

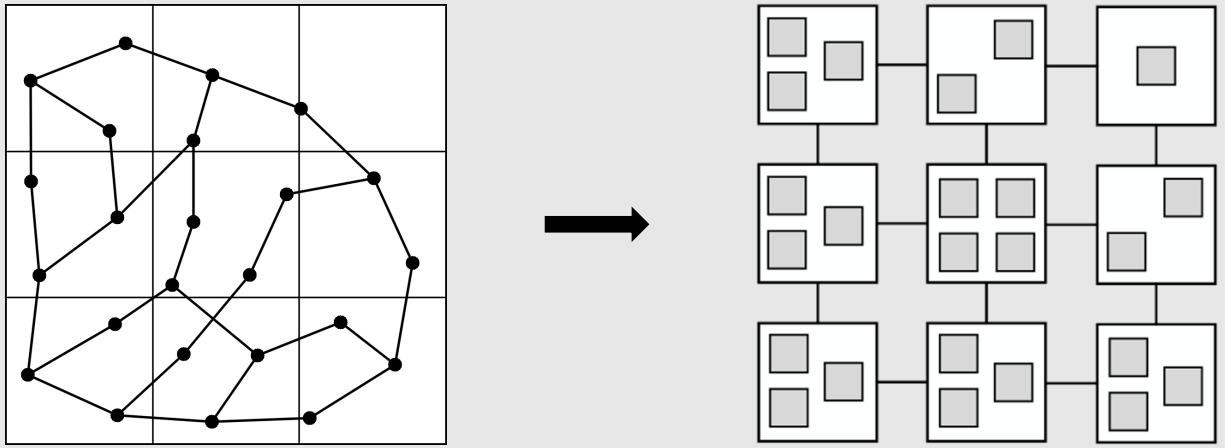


Optimization



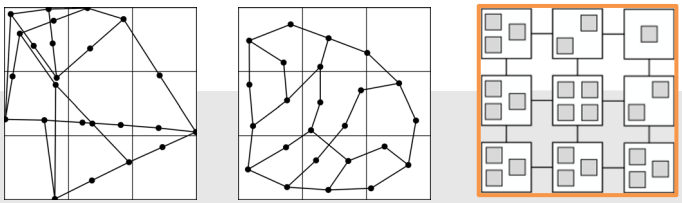
After load balancing

- Each node is mapped to the corresponding FPGA in the host graph
- Nodes, which are mapped to the same FPGA, are clustered together



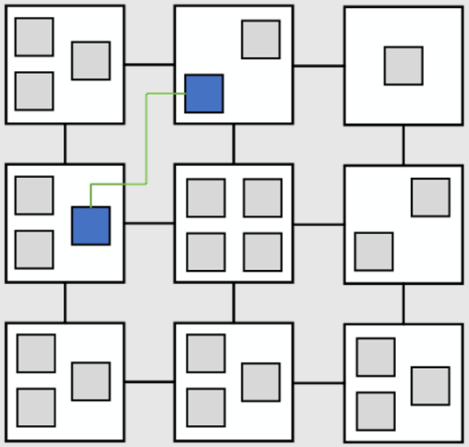
Initial mapping

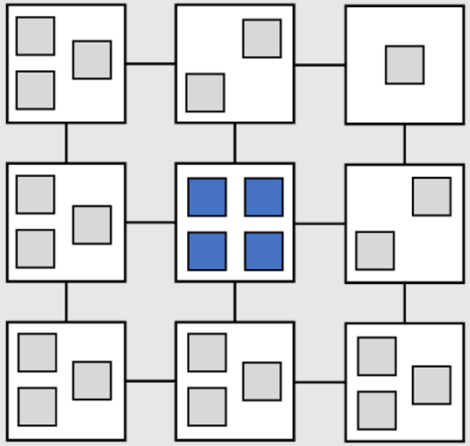
Optimization



Optimization steps

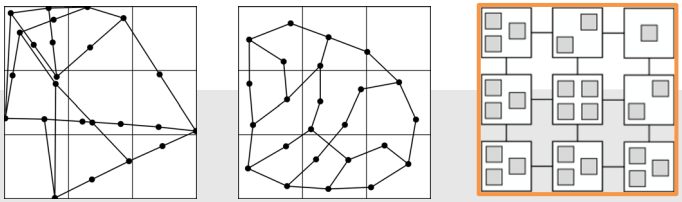
Reduce maximum dilation between tasks





Reduce maximum FPGA capacity utilization of cluster

Optimization



Using heuristics

- Gradient Descent

  This is an improved variant of local search that performs the best elementary step of all possible elementary steps

- Taboo Search

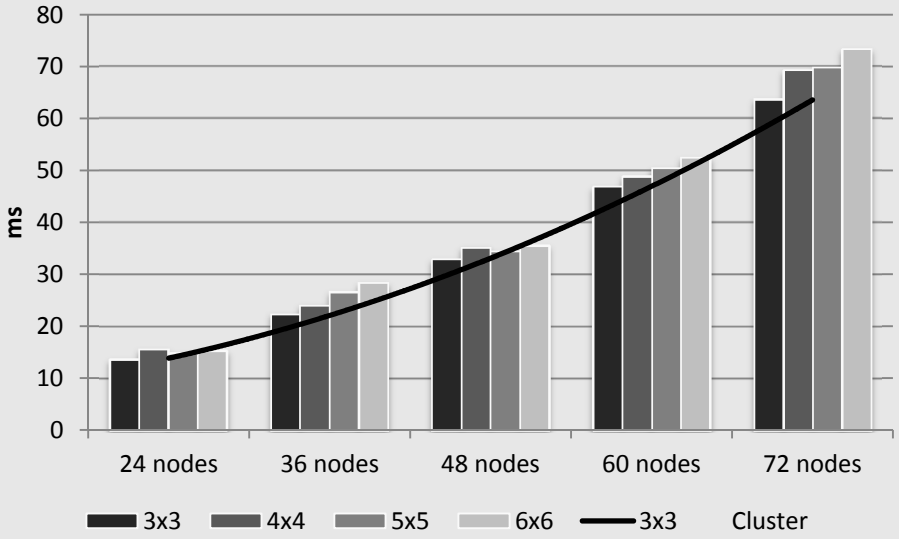
  A memory is added to roll back to the best solution, if no further solution can be found or a cycle is detected.

- Simulated Annealing

  To leave solutions that are stuck in local minima, the next solution is allowed to be worse by a parameterizable percentage.

Evaluation

- Core i7 4790 CPU
- Five random TIGs created with same complexity
- Utilize 60% resources of a reference FPGA 3x3 cluster

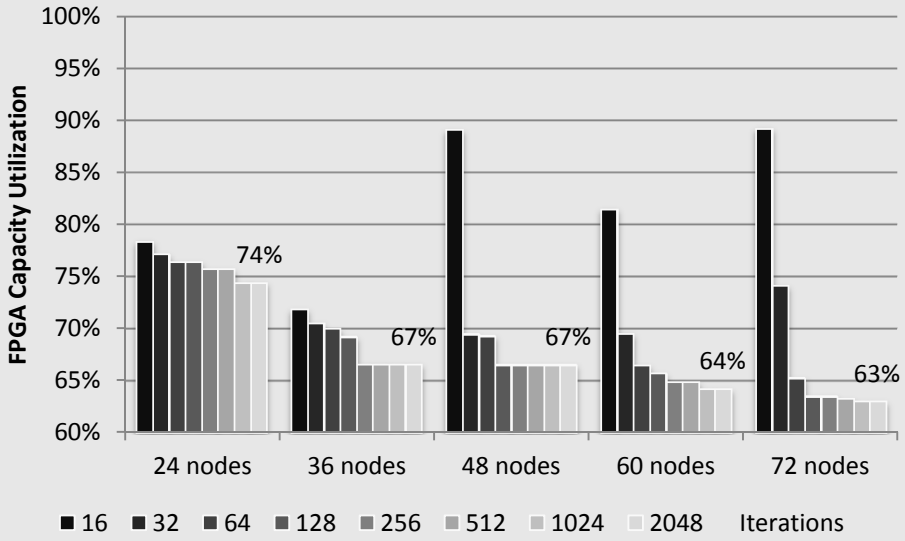


Computation time of the algorithm

- Different amount of nodes
- Different amount of FPGAs
- Parallelization degree: 4
- Maximum capacity utilization: 85 %

Evaluation

- Core i7 4790 CPU
- Five random TIGs created with same complexity
- Utilize 60% resources of a reference FPGA 3x3 cluster



Maximum FPGA capacity utilization

- Different amount of iterations in the optimization process
- Different amount of nodes
- Parallelization degree: 4
- FPGA cluster: 3 x 3

Summary

- Clustering and mapping algorithm for application distribution
- On a scalable FPGA cluster (Mesh like topology)
- Clustering and mapping are done in 1 step

Algorithm

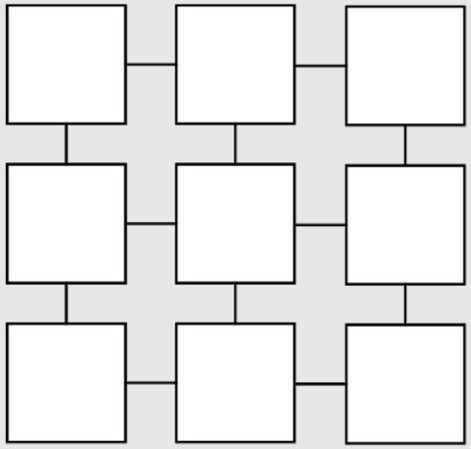
1. Create initial graph
2. Load Balancing
3. Optimization Steps

Results

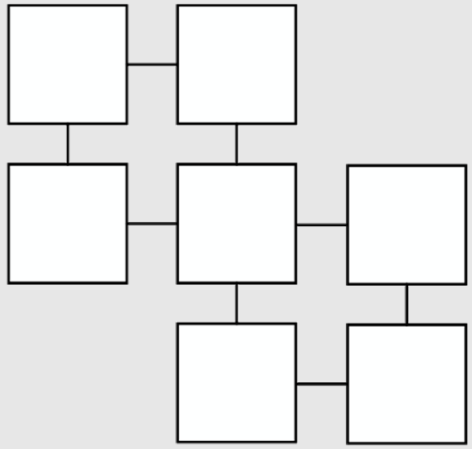
1. Achieved scalability for the amount of FPGAs that are contained in a cluster
2. Scales with the amount of tasks in a quadratic complexity class ≈ $O(n^2)$
3. An efficient solution can be found in a reasonable time

Outlook

- Support different FPGA types (heterogeneous cluster)
- Support irregular host / board connection graphs
- Integrate the optimization for latency and bandwidth



Regular Graph



Irregular Graph

RUB

Questions

Thank You!

Questions?