ReCoNodes – Optimization Methods and Platform Design for Reconfigurable Hardware Systems

Prof. Dr. Sándor Fekete
Dr. Tom Kamphans
Dipl.-Math. Jan van der Veen
Dipl.-Math.oec. Nils Schweer

Prof. Dr.-Ing. Jürgen Teich
Dipl.-Inf. Josef Angermeier
Dipl.-Ing. Mateusz Majer
Outline

• ESM Platform:
  ▪ Motivation and Goal
  ▪ Design and Implementation
  ▪ Applications

• Optimization Methods
  ▪ Orthogonal Packing
  ▪ Scheduling with Reconfiguration Times
  ▪ Virtual Areas
  ▪ Defragmentation

• Publications
ESM Platform
ESM Platform: Motivation and Goal

- Why develop a new reconfigurable platform?
  - More flexibility applying dynamic reconfiguration: overcome limitations in placing partial modules freely
  - Close gap between theory and practice
- Difficulties in the design flow:
  - Multiple partial module implementation required for each different placement position
  - Idea: software and hardware support for relocatable partial modules
- Memory limitation:
  - Many applications, e.g. image processing, with high memory demand, memory access dependent on placement
  - Idea: architecture concept with flexible access to external memory banks
ESM Platform: Motivation and Goal

- I/O-pin limitation:
  - Static I/O pin assignment prevents relocation of partial modules.
  - Idea: architecture concept with I/O signals connecting dynamically periphery to the current partial module placement.

- Inter-module communication:
  - How to communicate at runtime without placement information at compile-time?
  - Idea: architecture concept allowing flexible communication independent of the placement.
ESM Platform: Design

- Slot-based, homogeneous architecture
- Reconfiguration manager device for relocating partial bitstreams
- SRAM-blocks for each partial module
- External control-CPU
- Flash memory for storing partial modules
- Virtex II 6000 FPGA
- Crossbar device for dynamic I/O-pin routing
ESM Platform: Design

- Inter-module communication:
  - Neighbour to neighbour
  - Shared memory
  - Reconfigurable bus
  - Over the crossbar
ESM Platform: Implementation

**MotherBoard:**

**BabyBoard:**
ESM Platform: Implementation

• Software:
  ▪ Linux with extended kernel modules
  ▪ Login and module transfer over network
  ▪ C libraries to manage reconfiguration and communicate with partial modules

• Users:
  ▪ Prof. Stechele – Fahrerassistenz, TU München
  ▪ Prof. Platzner – ReConOS, Uni Paderborn
  ▪ Prof. Rammig – Reconf.-Scheduling, Uni Paderborn
  ▪ Prof. Merker, TU Dresden
  ▪ Prof. Wong, NUS in Singapore
  ▪ Prof. Huss, TU Darmstadt
  ▪ Prof. Teich, Uni Erlangen and Prof. Fekete, TU Braunschweig
  ▪ Softgate GmbH
ESM Platform: Implementation

- **ESM people:**
  - Prof. Dr.-Ing. J. Teich
  - Dipl.-Ing. Mateusz Majer (coordination)
  - Dipl.-Inf. Josef Angermeier (Software+Firmware)
  - Felix Reimann (RMB Modules)
  - Andre Linarth (MotherBoard PCB design)
  - Thomas Haller (BabyBoard PCB design)
  - Ding Ji (Crossbar design)
  - Peter Asemann (Programmer´s Interface)
  - Christian Freiberger (Reconfiguration Manager)
  - Jan Grembler (Video + Audio Interfaces)
  - Dipl.-Ing. Diana Göhringer (communication)
  - Christof Lauer (Module Relocator)
  - Thomas Stark (Crossbar Software)
  - Bruno Kleinert (Reconfiguration Software)
  - Ulrich Batzer and Matthias Kovatsch (Taillight Demo)
ESM Platform: Applications

- Parallel sorting algorithms:
  - Multiple compare operations in parallel
  - E.g. bitonic sorter, running time $0.5 \log n (\log n + 1)$
  - Area usage of sorting network may be too large for device
  - Available space may be limited due to concurrent applications

=> Apply partial reconfiguration
ESM Platform: Applications

- Parallel sorting algorithms:
  - Multiple compare operations in parallel
  - E.g. bitonic sorter, running time $0.5 \log n (\log n + 1)$
  - Area usage of sorting network may be too large for device
  - Available space may be limited due to concurrent applications

=> Apply partial reconfiguration
ESM Platform: Applications

• Parallel sorting algorithms:
  - Multiple compare operations in parallel
  - E.g. bitonic sorter, running time $0.5 \log n (\log n +1)$
  - Area usage of sorting network may be too large for device
  - Available space may be limited due to concurrent applications

=> Apply partial reconfiguration
ESM Platform: Applications

- Parallel sorting algorithms:
  - Multiple compare operations in parallel
  - E.g. bitonic sorter, running time $0.5 \log n \cdot (\log n + 1)$
  - Area usage of sorting network may be too large for device
  - Available space may be limited due to concurrent applications

=> Apply partial reconfiguration
ESM Platform: Applications

- Driver assistance system: cooperation with Prof. W. Stechele, Dipl.-Ing. C. Claus, TU München
  - Adaption to driving situation: e.g. taillight recognition in tunnel
  - Integration of assistance module on ESM platform ✓
  - Latency constraint: $t = 19\,\text{ms} < 40\,\text{ms}$ (video @ 25FPS) ✓
Optimization: Scheduling

• Scheduling respecting reconfiguration peculiarities
  ▪ Reconfiguration times
  ▪ Exclusive reconfiguration port

• Optimization goals:
  ▪ Minimizing the complete makespan

• Model: Parallel Machine Scheduling with Single Server
Scheduling
Optimization: Scheduling

• New heuristic *ShuffleFit*
  - Determine schedule without reconfiguration peculiarities (Problem Pm||C_max) with known algorithms
  - Modify schedule, such that, no two jobs end at the same time instance

• Prevents worst case scenarios: cluster of small jobs

LPT
Optimization: Scheduling

- Experimental results:
  - With 8 slots; \( n = 1, \ldots, 50 \) tasks;
  - Max task duration \( p_{\text{max}} = 0, \ldots, 50 \)
  - Average deviation of 5-10% from corresponding optimal solution
  - Quality mainly depending on initial solution
Packing

Placing modules onto an FPGA
Orthogonal Placement

- Placement of orthogonal modules ➔ intervall graphs
- Find optimal placement by enumeration of valid interval graphs
- Works also with
  - heterogenities (RAM, DSPs, ...)
  - higher dimensional packings
Orthogonal Placement: Online

- Least Interference Fit:
  Place modules in those columns, that are used by as few other modules as possible
Orthogonal Placement: Online

• Least Interference Fit:
Place modules in those columns, that are used by as few other modules as possible.

Theorem
If all rectangles have the same width, then $LIF$ is 1-competitive.
Virtual Areas

- Divide an ESM into several virtual areas

- Advantages
  - Restrict resources
  - Protect applications
  - Different schedulers for each VESM

Task: Pack modules with time-varying resource requests
Virtual Areas: Example

Example: Schedule for 5 VESMs on 16 Slots
Virtual Areas: Delaying Requests

Without delaying

With delaying

DFG-Schwerpunktprogramm „Rekonfigurierbare Rechnersysteme“
ReCoNodes
FPGAtris: offline

- Solvable using an ILP
- Example:

  6120 Constraints
  2537 Variables
  6h with CPLEX (3.2 GHz Xeon)

Variables
- Slot assignment variables \( x_{si} \) module \( m_i \) is scheduled in slot \( s \)
- Time assignment variables \( y_{tij} \) request \( (i, j) \) is scheduled at time \( t \)
- Occupancy variables \( z_{stij} \) keep track of occupied slots
- Usage variables \( u_t \) indicates which times steps are used

Constraints
- Assignment Constraints
  \[
  \sum_{s=1}^{N} x_{si} = 1 \quad \forall i = 1, \ldots, M , \quad (1)
  \]
  \[
  \sum_{t=1}^{T} y_{tij} = 1 \quad \forall i = 1, \ldots, M, j = 1, \ldots, \ell_i . \quad (2)
  \]
- Boundary Constraints
  \[
  \forall i, j, s = s_{\text{low}}, \ldots, s_{\text{up}} : \quad x_{si} = 0 \quad (3)
  \]
- Order Constraints
  \[
  \sum_{t=1}^{T} t y_{tij} - \sum_{t=1}^{T} t y_{tij-1} > 0 \quad \forall i, j > 0 . \quad (4)
  \]
- Occupancy Constraints
  \[
  \forall i = 1, \ldots, M, j = 1, \ldots, \ell_i, s = 1, \ldots, N, t = 1, \ldots, T , \quad s' = s_{\text{low}}, \ldots, s_{\text{up}} \quad : \quad x_{si} + y_{tij} - z_{s'tij} \leq 1 \quad (5)
  \]
- Exclusive Constraints
  \[
  \forall t = 1, \ldots, T, s = 1, \ldots, N : \sum_{i=1}^{M} \sum_{j=1}^{\ell_i} z_{stij} \leq 1 \quad (6)
  \]
- Delay Constraints
  \[
  \forall i = 1, \ldots, M, j = 1, \ldots, \ell_i - 1, s = 1, \ldots, N , \quad t = 1, \ldots, T - 1 : \quad z_{stij} - z_{s(t+1)ij} - y_{(t+1)(j+1)} \leq 0 \quad (7)
  \]
- Usage Constraints
  \[
  \forall t = 1, \ldots, T , i = 1, \ldots, M, j = 1, \ldots, \ell_i : u_t - y_{tij} \geq 0 \quad (8)
  \]
  \[
  \forall t = 2, \ldots, T : u_{t-1} - u_t \geq 0 . \quad (9)
  \]

Objective Function
\[
\min_{i} \sum_{i=1}^{T} i u_t \quad \text{subject to Eq. (1)} - (9)
\]
\[
x_{si} \in \{0, 1\}, y_{tij} \in \{0, 1\}, z_{stij} \in \{0, 1\}, u_t \in \{0, 1\}
\]
FPGAtris: Simple Heuristics

- FirstFit
  - Choose the lowest, leftmost, overlapping-free position
- FirstFit with delays
  - Same as FF, but stretch modules
FPGAtris: Simple Heuristics

- **BestFit**
  - \( L \) (\( R \)) := unoccupied cells left (right) to placed module
  - rating := min\{L, R\}
  - take position with minimal rating
  - Quality of BestFit packing depends on insertion order

- **TabuSearch**
  - Try different insertion orders with BestFit
FPGAtris: Simple Heuristics

- Experiments with randomly generated input
- Lower bound: total area / width

$$LB = \frac{1}{N} \sum_{i=1}^{M} \sum_{j=1}^{\ell_i} r_{ij}$$

TabuSearch outperforms others
Defragmentation

Cleaning up an FPGA

Here: only columnwise reconfiguration
Defragmentation

FPGA

Scheduler

Module Library
Defragmentation
Defragmentation

DFG-Schwerpunktprogramm „Rekonfigurierbare Rechensysteme“
ReCoNodes
Defragmentation
Defragmentation

Scheduler

Module Library

FPGA
Defragmentation

FPGA

Scheduler

Module Library
Defragmentation
Defragmentation

FPGA

Scheduler

Module Library
Defragmentation:

- FPGA
- Scheduler
- Module Library
Defragmentation

Defragmentation: STOP

FPGA

Scheduler

Module Library
Defragmentation:

MOVE

FPGA

Scheduler

Module Library
Defragmentation:

RESUME

FPGA

Scheduler

Module Library
Defragmentation

FPGA

Scheduler

Module Library
Defragmentation

[Diagram showing a schematic of a system with modules, a scheduler, and an FPGA.]
Defragmentation, Approach I

- Use simple strategy for placement, defrag whole FPGA when necessary

Theorem

Rearranging an array of contiguous objects such that the free space is maximized is strongly NP-complete. Moreover, there is no PTAS within any polynomial approx. factor (unless $P = NP$).
Defragmentation, Approach I

- Easy to solve if density is low

\[ \delta := \frac{1}{\ell} \sum_{i=1}^{n} m_i \leq \frac{1}{2} - \frac{1}{2\ell} \cdot \max_{i=1,\ldots,n} \{m_i\} \]

\( \ell \) Width of FPGA

\( m_i \) Modules sizes

from right to left:
- shift every module to the right as far as possible

from left to right:
- shift every module to the left as far as possible
Defragmentation, Approach I

- Tabu-Search:
  - locally shift modules
  - accept shift that maximizes \( \frac{\text{Maximal free space}}{\text{Total free space}} \)

### Graphs

- Size of largest free space
- Number of free spaces

#### No heterogeneities

- **Virtex2 FPGA**
  - **Input**
    - Tabu Search
    - Greedy
  - **Density**
    - 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
    - **Size of Max. Free Space**
      - 0, 10, 20, 30, 40, 50, 60
    - **Density**
      - 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

#### Virtex2 FPGA

- **Input**
  - Tabu Search
  - Greedy
  - **Density**
    - 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
    - **Size of Max. Free Space**
      - 0, 10, 20, 30, 40, 50, 60
    - **Density**
      - 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

- **Graphs**
  - **Size of largest free space**
  - **Number of free spaces**
Defragmentation, Approach II

- Maintain FPGA to avoid complete defragmentation

- Strategies:
  - *AlwaysSorted*: Keep modules sorted by their size
  - *DelayedSort*: Delay sorting until necessary
  - *ClassSort*:
    - Organize modules in size classes of power of 2
    - Maintain free blocks in each size class
  - *LocalShift*:
    - Use BestFit if possible
    - Otherwise compact FPGA in small neighborhoods of free blocks
Defragmentation, Approach II

- Maintain FPGA to avoid complete defragmentation

**Theorem**

*AlwaysSorted* achieves the optimal makespan, if there is no time penalty for moves.

- *DelayedSort*: Delay sorting until necessary

**Theorem**

*ClassSort* performs $O(1)$ moves per operation. Amortized cost are $O(m_i \log M)$

$m_i$: module's size, $M$: largest module,
Defragmentation, Approach II

Array size: \( N = 2^{10}; k = 8 \) (LocalShift); 100000 modules per sequence; Expected duration: 300 units
Moves/Mass \( \times 10.000 \), Time: \( \times 300.000 \)
Defragmentation, Approach II

Number of moves

Number of moved columns

Makespan (moves are free!)

Array size: $N = 2^{10}$; $k = 8$ (LocalShift); 100000 modules per sequence; Expected duration: 300 units

Moves/Mass $\times 10,000$, Time: $\times 300,000$
Defragmentation, Approach II

... but a lot of moves

Sorting algorithms: optimal makespan

Array size: $N = 2^{10}$; $k = 8$ (LocalShift); 100000 modules per run

Moves/Mass $\times 10.000$, Time: $\times 300.000$
**Defragmentation, Approach II**

*ClassSort*: few moves, but a lot of overhead due to fragmentation

Array size: $N = 2^{10}$; $k = 8$ (LocalShift); 100000 modules per sequence; Expected duration: 300 units

Moves/Mass $\times 10,000$, Time: $\times 300,000$
Defragmentation, Approach II

LocalShift: simple, but effective

Array size: $N = 2^{10}$; $k = 8$ (LocalShift); 100000 modules per sequence; Expected duration: 300 units

Moves/Mass $\times 10.000$, Time: $\times 300.000$
Publications

M. A. Bender, S. P. Fekete, T. Kamphans, N. Schweer.  

J. Sim, W. Wong and J. Teich.  

J. Angermeier, A. Amouri, and J. Teich.  
Proceedings 19th International Conference on Field-Programmable Logic and Applications (FPL 2009), Prague, August, 2009.

[39] Online Square Packing.  
S. P. Fekete, T. Kamphans, N. Schweer.  
Proceedings 20th Algorithms and Data Structure Symposium (WADS).

[38] No-Break Dynamic Defragmentation of Reconfigurable Devices.  
Proceedings 18th International Conference on Field-Programmable Logic and Applications (FPL 2008), Heidelberg, September, 2008.

[37] A comparison of embedded reconfigurable video-processing architectures.  
Proceedings of 18th International Conference on Field Programmable Logic and Applications (FPL 2008), Heidelberg, Germany, September 8 - 10, 2008, pp. 587-590.
Publications

J. Angermeier and J. Teich.


[34] Offline and Online Aspects of Defragmenting the Module Layout of a Partially Reconfigurable Device.
S. Fekete, J. van der Veen, A. Ahmadinia, D. Göhringer, M. Majer, and J. Teich.


[32] Scheduling and communication-aware mapping of HW-SW modules for dynamically and partially reconfigurable SoC architectures.
Publications


Thank you