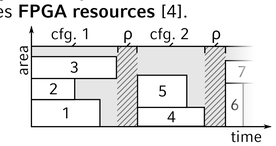
Task Scheduling on FPGA-Based Accelerators without Partial Reconfiguration

Pascal Jungblut, LMU Munich, Germany
Advisor: Dieter Kranzlmüller

METHODOLOGY

Motivation

Task scheduling on Field Programmable Gate Arrays (FPGAs) is primarily implemented using Partial Reconfiguration (PR). The scheduling approaches work on PR has a high **development overhead**, is **hardly portable** and occupies **FPGA resources** [4].



We want to understand task scheduling on FPGAs **without** the need for **PR**, **compare** appropriate approaches and **analyze** them **systematically**.

Inputs

Algorithms
- over 25 years of reasearch for task scheduling on FPGAs
- different levels of abstraction
- focus on static algorithms

Hardware
- large influence on possible schedules
- hardly comparable between models and vendors
- possibly large bitstreams
- heterogeneous FPGAs

Traces
- easily obtainable from a host runtime
- require one event per task
- agnostic to programming models

Machine models

Machine models are build around **Processing Elements (PEs)**. Each PE can execute at least one task and has a **set** of **properties**.

- Properties describe the **capabilities of a PE**.
- They are based on
  - **implications** of the scheduling **algorithm** and
  - the target **hardware** and **software**.
- The models are an **abstraction over** very different task scheduling **approaches**.

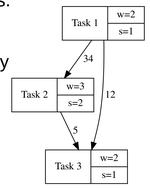
A **set of PEs** and their properties is a **machine model**.

PE				
Class	**Property**	**Domain**	**Default**	**Example**
Configuration	Bandwidth	bit/s	∞	400 MB/s
	Concurrency	N	∞	2
	Placement	Location	{}	{slot0, slot1}
	Size	N	0	12 MB
Dependency	Exclusivity	PEs	{}	{p_3, p_8}
	Inclusivity	PEs	{}	{p_1}
Function	Quantity	N	1	2
	Type	Task	{}	{v2, v_4}
	Vectorization	N	1	4
Overhead	Start	s	0	1 ms
	Reconfiguration	s	0	12 ms
Performance	Bandwidth	bit/s	∞	12 GB/s
	Computation	s/Task	0	30ms/v_3

Task graphs

Task graphs are **generated from trace data** and annotated with **FPGA-specific** information.

- **Automatic generation** from for example OpenCL traces.
- Dependencies contain **data** movement **cost**.
- **Task cost** for each property can be integrated.
- **Multiple** traces can be **combined** for better accuracy.



Predicates

First-order predicates restrict the set of **valid schedules**. Predicates can be trivially translated to **constraint programming** environments.

Example
Property reconfiguration overhead $\mathcal{P}_{o,r}$ describes the time overhead for configuring the FPGA to execute a task. The start time $t_s(v,p)$ on PE *p* must be adjusted accordingly:

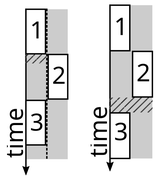
$$\forall v \in V : t_s(v,p) \geq t_r(v,p) + \mathcal{P}_{o,r}$$

Each PE-**property maps to** at least one **predicate**. A machine model can be **automatically converted** to a set of predicates.

Schedules

Schedules specify **where and when** a task is executed.
- Cost functions provide comparability, e.g. makespan, energy usage.
- Best-case scenarios are obtained using constraint programming environments.

Result highlights
- **High-level programming without PR** has a low percentage overhead, while easing the development and improving the portability.
- Reconfiguration-aware dynamic scheduling algorithms can generate **near-optimal schedules in polynomial time**.
- **Transparent integration** into high-level programming environments like OpenCL is possible.



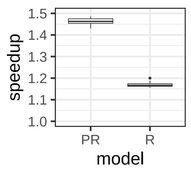
Ongoing
We are still investigating the **optimization** of high level **code** based on schedules.

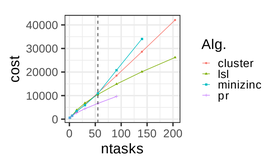
Key Contributions

1. Flexible and arbitrarily accurate **machine models** for FPGA-based accelerators.
2. Automated derivation of **CP programs** from machine model.
3. Three **heuristic-based polynomial-time scheduling** algorithms.
4. Automated **recommendations** for HLS **code**.
5. Traces of OpenDwarf executions on FPGAs.

Outputs and Results

Two types of results from our approach:
- **statistical analysis** of valid schedules
- **inductive proofs** for lower bounds, achievable parallelism, …





The hardware-agnostic model allows straightforward comparison of distinguished scheduling algorithms and FPGAs [2].
The **left figure** shows the speedup for an identical model except for PR support.
The **right figure** depicts lower bounds for some models/alg. depending on the # of tasks.

TECHNOLOGY

Tracing

We demonstrate automatic generation of task graphs with traces from the **OpenDwarf** benchmark suite with OpenCL [1].

```
__kernel void lud_internal(__global float *m, int matrix_dim, int offset) {
    int global_row_id = offset + (get_group_id(1)+1)*BLOCK_SIZE;
    int global_col_id = offset + (get_group_id(0)+1)*BLOCK_SIZE;
    float sum = 0;
    for (int i=0; i < BLOCK_SIZE; i++)
        sum += m[(global_row_id+get_local_id(1))*matrix_dim+offset+i] *
            m[(offset+i)*matrix_dim+global_col_id+get_local_id(0)];
    m[(global_row_id+get_local_id(1))*matrix_dim+global_col_id+get_local_id(0)] -
sum;
}
```

```
<<<< clSetKernelArg -> CL_SUCCESS
 >>>> clEnqueueNDRangeKernel( lud_internal ): queue = 0x7fda74a6c4b8,
kernel = 0x162c210,
    global_work_size = < 16 >, local_work_size = < 16 >
 <<<< clEnqueueNDRangeKernel created event = 0x27e9c20 -> CL_SUCCESS
 >>>> clFinish: queue = 0x7fda74a6c4b8
 <<<< clFinish -> CL_SUCCESS
Device Timeline for lud_internal (enqueue 379)
    2993788267098459 ns (queued),
    2993788267168510 ns (submit),
    2993788267181221 ns (start),
    2993788267408694 ns (end)
```

- Generating traces is done with an adapted version of the **Intercept Layer for OpenCL Applications** [6].
- Traces give insights into **PE properties**.
- Supports **automatic dependency detection**
- Supports **data movement detection** from `clEnqueue(Read|Write)Buffer` calls.
- No manual action required.

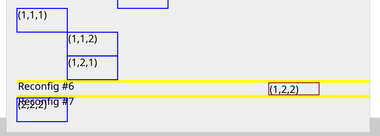
Simulation

We introduce two **reconfiguration-aware polynomial time** scheduling algorithms **without PR**:

- A **clustering algorithm** that decomposes the task graph into clusters, each representing a single bitstream.
- A **list scheduling** algorithm with a **lookahead** that counters „flip-flopping“ of active bitstreams.

A **framework** build with C++17 and the Boost Graph Library **generates schedules** in milliseconds.

The **figure** to the right depicts an example schedule for a 3x3 blocked LU-decomposition. Time flows from top to bottom:
Each **column** represents a **PE**,
each **rectangle** a schedule **task** an
its **color** the **bitstream**.
Reconfigurations are represented as yellow lines.



Constraint programming

The predicates generated from the machine model can be trivially **converted** to **constraint programming (CP) inputs**. A software generates valid schedules for a given set of predicates.

- We provide case studies for two machine models: **one with and one without using PR**.
- CP can be used to find **optimal solutions**:
  - Valuable as a **lower bound** for polynomial time algorithms.
  - Easy **comparability** of **PE properties** and *what-if* scenarios.
- Current implementation using ORTools [3].
- More flexible version in OR-Tools is in-progress.
- Only viable for **small task graphs**.

```
% Start times of tasks
array[V] of var int: t_s;
% Finish times of tasks
array[V] of var int: t_f;
% PE a task runs on
array[V] of var int: process;
array[V] of string: tasklabels;
predicate no_overlap(var int: v1, var int: v2) =
        t_f[v1] < t_s[v2] \/ t_s[v1] > t_f[v2];
```

Contact

Pascal Jungblut

pascal.jungblut@nm.ifi.lmu.de
http://mnm-team.org/~jungblut
http://github.com/pascalj

LMU Munich
Oettingenstr. 67
81369 Munich, Germany

References

[1] Krommydas, Konstantinos, Wu-chun Feng, Christos D. Antonopoulos, and Nikolaos Bellas. “Opendwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures.” Journal of Signal Processing Systems 85, no. 3 (2016): 373–92.
[2] Jungblut, Pascal, and Dieter Kranzlmüller. “Optimal Schedules for High-Level Programming Environments on FPGAs with Constraint Programming,” 96–99. IEEE Computer Society, 2022.
[3] Nethercote, Nicholas, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. “MiniZinc: Towards a Standard CP Modelling Language.” In International Conference on Principles and Practice of Constraint Programming, 529–43. Springer, 2007.
[4] Vipin, Kizheppatt, and Suhaib A. Fahmy. “FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications.” ACM Computing Surveys (CSUR) 51, no. 4 (2018): 1–39.
[5] Jungblut, Pascal, and Dieter Kranzlmüller. “Dynamic Spatial Multiplexing on FPGAs with OpenCL.” In International Symposium on Applied Reconfigurable Computing, 265–74. Springer, 2021.
[6] https://github.com/intel/opencl-intercept-layer