A Reconfigurable Design Framework for FPGA Adaptive Computing
-- Application for Online Trigger Algorithms

II. Physikalisches Institut,
Justus-Liebig-Universität Giessen, Germany

Ming Liu for the PANDA collaboration group
Outline

• Introduction & Motivation
• Reconfigurable Framework for Adaptive Computing
  – HW infrastructure
  – OS, device drivers & scheduler SW
  – Context saving and restoring
  – Inter-Process Communication (IPC)
• Technical Perspectives in applications
• Conclusion & Future Work
Introduction & Motivation

• Adaptive computing: algorithms adapted to ambient conditions during system run-time.
• Benefits:
  – Higher performance
  – Lower power consumption
  – Multitasking on limited resources
• Conventional adaptive multitasking on general-purpose CPUs + OSes: well-fledged as the development of OSes & scheduler
  – Computing resources (CPU) intelligently and efficiently utilized
• In the FPGA world???
  – Static designs? Not adaptive
  – Partial Reconfiguration (PR)? Technical support
• Motivation: a complete design framework for more efficient hardware resource management, based on FPGA PR technology.
Design Framework for Adaptive Computing

- A comprehensive framework in different HW/SW layers
  - HW PR design (static computer systems + reconfigurable algorithm modules as HW processes)
  - OS kernel for the base computer systems
  - Device drivers for algo. modules
  - Software scheduler for HW processes

- Reconfiguration speed is critical for performance
  - Module reconf. time in the order of magnitude of Micro-seconds (us), with our customized MST_HWICAP design
  - Reconf. time depending on design complexity
Partial Reconfiguration Technology

- PR Region (PRR) dynamically loaded with different design modules (partial bitstreams)
- Designs can be switched in the system run-time for different algorithms
- HW resources are multiplexed by different PR Modules (PRM)
HW Infrastructure

- Xilinx PR design flow
- Modular algorithm designs (A1, A2, …)
- PR Region (PRR)
- PR communication interface (BMs)
- System manager (GP CPU)
- Peripherals, memories, …
- ICAP design
  - mst_hwicap
  - Conf. speed: ~235 MB/s
  - Conf. overhead: xx μs to xxx μs
OS, Drivers & Scheduler

- Embedded OS or Standalone
- Device drivers for algorithm modules
  - Software control registers
  - Interrupts
- Algorithm scheduler
  - Application programs (flexible & portable)
  - Monitors ambient conditions and triggers algorithm switching
  - HW processes are preemptable and comply with the scheduler
  - Flexible disciplines

```c
int scheduling(void) {
    if((data_in_fifo0 - data_in_fifo0) > THRESHOLD) {
        switching_to_hw_process = 0; // Context switching to hw process 0,
        // due to much buffered data in fifo0,
    } else if((data_in_fifo1 - data_in_fifo0) > THRESHOLD) {
        switching_to_hw_process = 1; // Context switching to hw process 1,
        // due to much buffered data in fifo1,
    } else {
        switching_to_hw_process = switching_to_hw_process; // Keep unchanging,
        // to reduce reconfiguration overhead.
    }
    return switching_to_hw_process;
}
```
Context Switching

• Context
  – Control registers
  – Buffered incoming data
  – Intermediate calculation results
  – To be saved and restored for algorithm modules in many cases

• Concrete approaches [1][2]:
  – Register read and write
  – Bitstream readout and analysis

Inter-Process Communication (IPC)

- Static process communications: pipes (FIFOs) between nodes
- Time-multiplexing process communications: writing to pipes by $A_n$ and reading by $A_{n-1}$ ($A_n \rightarrow A_{n-1}$)
- Larger pipe sizes: smaller reconfiguration overhead, higher throughput, but longer latency
Adaptive Computing for online Triggering

Motivation:

• Multiple pattern recognition algorithms in DAQ & trigger systems (RICH ring recog., MDC tracking, TOF, Shower, …)
• Multiple cores for each algorithm for massive parallel processing
• Computation steps distributed on FPGAs
• Difficult to manage and modify the large system (many FPGAs, many algorithms, many cores, different FPGA bitstreams, long design synthesis & implementation time, …)
• Different computation features for algorithms (computation-bounded, memory-bounded, …)
• Traditionally all partitions are considered by designers during system development process -- NEITHER flexible, NOR efficient!!!
Adaptive Computing for online Triggering

One promising solution: Adaptive computing

- Algorithm cores designed as PR modules
- Modules can be adaptively loaded during experiments, according to external factors (workload, sub-event types, …)
- Uniform DAQ & trigger design to interface with optical hubs, which delivers all kinds of sub-events
- Performance improvement due to the balance of computation and memory accesses, as well as more efficient utilization of FPGA resources?
- Other merits?... (to be explored)
Comparison

Traditional non-PR design:

- Sub-event allocation needed
- Static algorithm module placement, Many design versions
- Inter-FPGA channel bandwidth requirements
- No data distribution requirements for optical hubs (all kinds of sub-events fed into all FPGAs)
- Uniform design in adaptive computing – easy to maintain system designs
- Balanced computing and more efficient FPGA resource utilization

PR design for adaptive computing:

- No SE allocation needs
- Uniform base system design, dynamically & automatically loaded modules, balanced processing capabilities, intelligent HW managements
- Correlations inside FPGAs, low on-board or inter-board bandwidth needs.
Summary

- FPGA PR-based adaptive computing is introduced in particle physics applications for online triggering.
- A comprehensive design framework is under research.
- Large benefits are foreseen with adaptive computing, in terms of design version control, system performance, HW utilization efficiency, costs, etc.

In the future:
- Trigger algorithms are to be implemented on FPGAs.
- The adaptive design framework is to be elaborated in different aspects.
- Real algorithm cores are to be applied for adaptive triggering
Thanks for your attention!