FPGA-based Adaptive Computing for Correlated Multi-stream Processing

Ming Liu[‡†], *Zhonghai Lu*[†], *Wolfgang Kuehn*[‡], *Axel Jantsch*[†]

‡ II. Physics Institute
Justus-Liebig-University Giessen (JLU), Germany
{ming.liu, wolfgang.kuehn}@physik.uni-giessen.de

† Dept. of Electronic Systems
Royal Institute of Technology (KTH), Sweden
{mingliu, zhonghai, axel}@kth.se

***Abstract*—In conventional static implementations for correlated streaming applications, computing resources may be inefficiently utilized since multiple stream processors may supply their sub-results at asynchronous rates for result correlation or synchronization. To enhance the resource utilization efficiency, we analyze multi-streaming models and implement an adaptive architecture based on FPGA Partial Reconfiguration (PR) technology. The adaptive system can intelligently schedule and manage various processing modules during run-time. Experimental results demonstrate up to 78.2% improvement in throughput-per-unit-area on unbalanced processing of correlated streams, as well as only 0.3% context switching overhead in the overall processing time in the worst-case.**

I. Introduction

Adaptive computing enables various computation algorithms to be tailored to ambient conditions during system run-time. Typically an adaptive system changes the design architecture or adjusts parameters on the fly according to its workloads, computation interests or other environmental factors. One major technical precondition of adaptive computing is the reconfigurability of the computing platform. In contrast to conventional static reconfigurability, Partial Reconfiguration (PR) is able to dynamically stop and change specific algorithm modules, while basic functions of an FPGA design are to be maintained.

Stream processing is a computation paradigm in which data streams are continuously generated and flow through processing steps until final results come out. Compared to random data access architectures such as databases, stream processing results in much faster real-time response and less storage requirements for data archiving, but raises more challenges with regard to intermediate data buffering and computing capabilities to the system design [1]. For single-streaming applications, parallel computation architectures may be easily organized to meet high performance requirements. However in multi-stream processing with correlated data, complex issues on data dependency, synchronization and communication among processing units must be taken into account. An example application is the correlated pattern recognition processing for data acquisition and triggering in nuclear and particle physics experimental facilities [2].

II. Related Work

Relevant contributions exist concerning adaptive computing and hardware resource management on reconfigurable devices. For instance in [3], the single processor scheduling algorithm is investigated and applied to task hardware module reconfiguration; In [4], [5] and [6], the authors focus on designing online placement and scheduling of tasks on reconfigurable devices. However, the above cited investigations are limited to the modeling level and do not take into account practical constraints of reconfigurable designs. In [7], the authors implement a practical hardware/software environment to manage reconfigurable blocks on FPGAs. A Linux kernel is enhanced to support hardware processes and schedule them altogether with normal software processes. However the modification work in the Operating system (OS) kernel space is error prone and makes the schedulable system difficult to be ported in different computing scenarios.

In [8], the authors propose a design optimization framework for adaptive realtime streaming applications. Their investigations are restricted to single-stream processing. In [9], a resource allocation model is presented for load-balancing processing of multi-tasks. But the complicated hierarchical architecture makes it difficult and impractical for hardware implementation. Moreover, experimental results in both [8] and [9] rely only on simulation rather than showing practical measurements on reconfigurable circuits. In this paper, we will present our adaptive design for multi-streaming applications, with on-FPGA measurement results showing performance improvement and ease of implementation.

III. Correlated Multi-streaming Models

A. Static Model

Synchronous Data Flow (SDF) [10][11] has been widely used to model and analyze streaming applications. We establish the multi-streaming model with static stream processors to appropriately describe multi-tasking applications. As shown in Figure 1, multiple correlated producers (P_i, $i\in[1,\ n]$) continuously generate data streams, and respective algorithm processors (consumer C_i, $i\in[1,\ n]$) digest their relevant data to obtain sub-results. Afterwards all sub-results are synchronized, correlated or assembled by the barrier synchronizer (Sync) for final results. We quantify the peak data producing or consuming capabilities of different nodes by Greek letters with Latin letters as coefficients. Specifically P_i generates data at a maximum rate of $k_i\rho$. It is proportional to an overall event rate of ρ, since all producers may react on the same events and strictly or statistically generate data at an approximately fixed ratio. For instance multimedia processing features statistically proportional Audio/Video data streams in the long run. Each consumer C_i consumes its respective data stream at a maximum rate of θ_i and produces sub-results at $j_i\theta_i$. Then the synchronizer collects all sub-results belonging

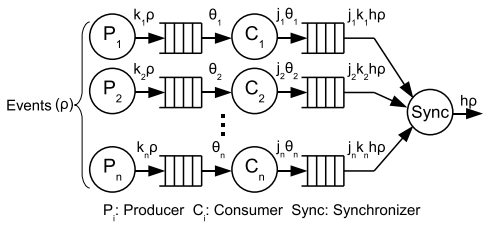


Fig. 1. Static SDF model for multi-streaming applications

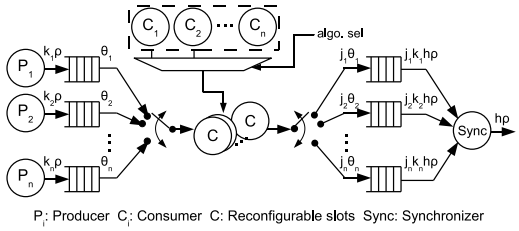


Fig. 2. Adaptive SDF model for multi-streaming applications

to the same event at $j_i k_i \rho$ and recovers events which occur at the rate ρ. We may assume a higher processing capability of the synchronizer than algorithm processors. This is either true in many applications containing very complex algorithms, or can be realized by a hierarchical architecture in which the overall synchronization work is partitioned into multiple sub-synchronizers for parallel processing. This assumption aims to exclude the synchronizer from the system bottleneck, focusing on algorithm designs. To indicate the assumption, we scale a coefficient h (h $\rightarrow \infty$) on both input and output parameters of the synchronizer.

Along the flow paths, intermediate data are buffered in finite-depth FIFOs. When any consumer is not able to cope with the high-speed data generation, data will be backlogged in the FIFO and then either pause the producer from injecting, or be lost. In hardware implementations, the processing capabilities on correlated streams are hardly guaranteed to be equivalent and synchronized, due to practical factors such as algorithm complexity and implementation differences, system integration constraints, etc. Therefore, if excluding the synchronizer, the final result throughput is restricted by the comparatively weakest consumer which generates the bottleneck and leads to unbalanced processing. The system performance can be derived from the following mathematical operations: We describe the performance parameters in a matrix format with rows for streams and columns for parameters of different nodes as listed in Figure 1:

$$A = \begin{pmatrix} k_1\rho & \theta_1 & j_1\theta_1 & j_1k_1h\rho \\ k_2\rho & \theta_2 & j_2\theta_2 & j_2k_2h\rho \\ ... & & & \\ k_n\rho & \theta_n & j_n\theta_n & j_nk_nh\rho \end{pmatrix} \quad (1)$$

All data streams are normalized by being divided by the coefficient k_i respectively:

$$\begin{pmatrix} \frac{1}{k_1} & 0 & ... & 0 \\ 0 & \frac{1}{k_2} & ... & 0 \\ ... & & & \\ 0 & 0 & ... & \frac{1}{k_n} \end{pmatrix} * A = \begin{pmatrix} \rho & \frac{\theta_1}{k_1} & \frac{j_1\theta_1}{k_1} & j_1h\rho \\ \rho & \frac{\theta_2}{k_2} & \frac{j_2\theta_2}{k_2} & j_2h\rho \\ ... & & & \\ \rho & \frac{\theta_n}{k_n} & \frac{j_n\theta_n}{k_n} & j_nh\rho \end{pmatrix} \quad (2)$$

In the ideal situation when all consumers are capable of digesting their respective data streams instantly ($\{\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n}\} \geq \rho$), the final result throughput for the static design (P_{stat}) is equal to the event rate ρ and it presents a balanced and realtime processing capability. Elsewise ($\frac{\theta_i}{k_i} < \rho$), P_{stat} is proportionally restricted by the weakest processor on path *i* whose normalized data consumption rate is $\frac{\theta_i}{k_i}$ = Min($\frac{\theta_1}{k_1}$, $\frac{\theta_2}{k_2}$, ..., $\frac{\theta_n}{k_n}$), as formulated in Equation (3). In unbalanced processing of correlated multi-streams, all faster cores have to wait for the slowest stream sub-results to reach the barrier, and hence computing resources are wasted. We define the key parameter **Degree of Unbalance (DU)** as the relative stream consuming capability difference between the fastest consumer and the slowest one, as formulated in Equation (4).

$$\frac{\frac{\theta_i}{k_i}}{P_{stat}} = 1 \Rightarrow P_{stat} = \frac{\theta_i}{k_i} = Min(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n}) \quad (3)$$

$$DU = \frac{Max(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n}) - Min(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n})}{Min(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n})} \quad (4)$$

B. Adaptive Model

The adaptive architecture for multi-streaming is modeled as shown in Figure 2. Reconfigurable slots of computing resources are reserved and they can be sequentially loaded with different algorithm processors. Flexible criteria may be adopted to select algorithms. To achieve balanced processing, algorithms can be loaded for different time spans on each slot C. We equalize the normalized data consuming capabilities of all algorithms on one slot as shown in Equation (5), where t_i (i$\in$[1, n]) stands for the effective work time of processor *i* in a single time unit.

$$\begin{cases} \frac{\theta_1}{k_1}t_1 = \frac{\theta_2}{k_2}t_2 = ... = \frac{\theta_n}{k_n}t_n = \psi \\ t_1 + t_2 + ... + t_n = 1 \end{cases} \quad (5)$$

Since processing capabilities of all algorithms are well balanced as ψ, we may derive its value as:

$$\psi = \{\frac{\theta_i}{k_i}t_i \mid i \in [1, n]\} = \frac{1}{(\frac{k_1}{\theta_1} + \frac{k_2}{\theta_2} + ... + \frac{k_n}{\theta_n})} \quad (6)$$

With the same number of *n* consumer modules (same resource utilization) as in the static model, the normalized data consuming throughput for each algorithm is $\frac{n}{(\frac{k_1}{\theta_1} + \frac{k_2}{\theta_2} + ... + \frac{k_n}{\theta_n})}$, also equal to the final result throughput P_{adpt}. Because we know that the arithmetic mean of a set of numbers is no greater than their maximum, as shown in Equation (7),

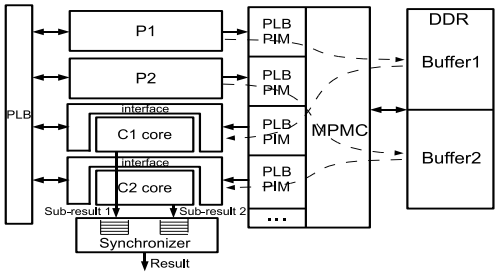
$$\frac{(\frac{k_1}{\theta_1} + \frac{k_2}{\theta_2} + ... + \frac{k_n}{\theta_n})}{n} \leq Max(\frac{k_1}{\theta_1}, \frac{k_2}{\theta_2}, ..., \frac{k_n}{\theta_n}) \quad (7)$$

we conclude that the overall performance of balanced adaptive computing is no less than static computing (Equation 8), and the improvement degree depends on the parameter DU:

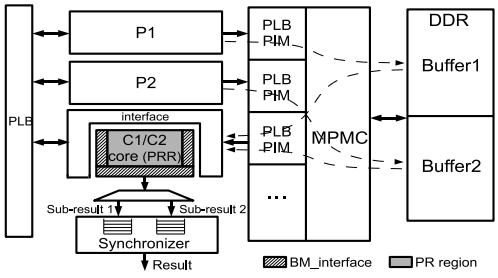
$$(P_{adpt} = \frac{n}{(\frac{k_1}{\theta_1} + \frac{k_2}{\theta_2} + ... + \frac{k_n}{\theta_n})}) \geq (P_{stat} = Min(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n})) \quad (8)$$

IV. PR-based Adaptive Design Framework

In FPGA-based adaptive computing, algorithm modules are respectively designed according to different computation requirements. Analogous to software processes running on OSes, algorithm instances are treated as hardware processes and managed by a central scheduler to be equipped/unloaded in/from the PR Region (PRR). Runtime reconfiguration is technically realized by our customized MST_HWICAP core [12],



(a) Static computing



(b) Adaptive computing

Fig. 3. Implementation of multi-streaming models on the FPGA

which achieves a configuration throughput of about 235 MB/s at a system clock frequency of 100 MHz. Context switching happens when the currently activated hardware process is being overwritten by another. It is required to properly save the context, and restore it when the current algorithm module resumes to work next time. A detailed description on the adaptive design framework can be found in [13].

V. Experiments

The multi-streaming models are implemented on a Xilinx Virtex-4 FX60 FPGA. Shown in Figure 3(a) for static and Figure 3(b) for adaptive computing, producers generate data streams and buffer them in dedicated DDR memory blocks (512 MB each). Algorithm consumers are statically or adaptively mapped on FPGA resources. The synchronizer has moderate buffering capability and correlates sub-results into final results. For analysis simplicity, we assume the following experimental setup:

1) There are only two correlated data streams.
2) Each consumer design utilizes one unit resource or on-chip area.
3) The synchronizer does only simple correlation work and is excluded from the performance analysis.
4) In the adaptive design, only one reconfigurable slot is reserved for implementation simplicity. Accordingly we measure and study the performance per unit area.

Producers and consumers are modeled with counters and they periodically produce or consume fixed-size data packets. The consumer design consists of the consumer core and a system interface, via which buffered data are fetched from DDR by a bus master and the CPU releases commands to change parameters. To minimize the reconfigurable area, only the consumer core is reserved as the PR region. The interface part is incorporated in the static design. In addition, the

```
//A simplified scheduler which automatically balances the processing on multi-streams
char scheduling(void) {
  static char turn_to_which_bit = 1; //By default bitstream 1 is active to process stream 1.

  if((consumed_data1 - consumed_data2) >= BIT_SWITCH_THRESHOLD){
    turn_to_which_bit = 2; //Context switching to hw process 2 to consume stream 2.
  }
  else if((consumed_data2 - consumed_data1) >= BIT_SWITCH_THRESHOLD){
    turn_to_which_bit = 1; //Context switching to hw process 1 to consume stream 1.
  }
  //Elsewise the current hw process keeps working.
  return turn_to_which_bit; //Return the decision to the main function.
}
```

Fig. 4. The simplified scheduler in C for balanced processing

resource utilization overhead of BM_interfaces is small and negligible in the PR design.

The scheduler program for hardware management is implemented in C as a standalone application on the PowerPC processor [13]. It keeps track of the processed data amount of each stream. Hardware processes are switched when their processed data amount difference exceeds an adjustable threshold (e.g. 8 MB in experiments). Thus this scheduling mechanism guarantees alternate processing on multi-streams, and features an intrinsic capability to automatically balance it. The simplified scheduler is demonstrated in Figure 4.

The consumer context includes buffered raw data in FIFOs and address information of DDR buffers in registers. Context switching will be postponed until all buffered raw data have been digested. In this way, computing resources are kept busy and there is no need to save the FIFO context. The register context can be read out via PLB and saved in DDR variables. It will be restored when the corresponding consumer resumes to work.

Referring to the normalized parameter matrix in Equation (2), we did normalized-setup experiments at various degrees of unbalance, as shown in Table I. We choose ρ equal to 100 MB/s, which is a reasonable input data throughput for a high-speed serial communication channel such as the optical link. The coefficient j_i of 10^{-3} implies a data selection rate of 1/1000. Therefore the final result throughput is equal to 100 KB/s in ideal static processing when both data streams can be instantly and evenly digested by algorithms. In the PR design, partial bitstreams are 96 KB for both consumers. Based on Equation (3) and (8), theoretical result throughput is separately listed for static and adaptive computing. Measurement results on experimental circuits take into account bus conflicts and dynamic reconfiguration overhead, and are accordingly listed in the last table column. Considering the double-unit area utilization in the static model, both theoretical and measured result throughputs per unit area are illustrated in Figure 5 to compare the static and the adaptive multi-streaming models. We observe that schedulable adaptive computing intrinsically features the capability to balance the processing on correlated streams. It can more efficiently utilize the computing resources and achieve a much higher throughput-per-unit-area (increased from 11.9/2 to 10.6/1 by 78.2% in Test 3) in unbalanced processing conditions. Practical measurement results are only slightly lower than theoretical values.

To further investigate the context switching overhead in adaptive computing, we changed the bitstream sizes and

Exp. setup	Peak data producing rate of P1 (ρ)	Peak data producing rate of P2 (ρ)	Processing capability of C1 ($\frac{\theta_1}{k_1}$)	Processing capability of C2 ($\frac{\theta_2}{k_2}$)	Degree of unbalance (DU)	Data selection rate (j_1, j_2)	Results	Theoretical result throughput	Measured result throughput
Test 0 (ideal)	100 MB/s	100 MB/s	100 MB/s	100 MB/s	0	1/1000	static (area=2)	100 KB/s	95.2 KB/s
							adaptive (area=1)	50 KB/s	47.4 KB/s
Test 1	100 MB/s	100 MB/s	100 MB/s	50 MB/s	1	1/1000	static (area=2)	50 KB/s	47.6 KB/s
							adaptive (area=1)	33.3 KB/s	31.6 KB/s
Test 2	100 MB/s	100 MB/s	100 MB/s	25 MB/s	3	1/1000	static (area=2)	25 KB/s	23.8 KB/s
							adaptive (area=1)	20 KB/s	19.0 KB/s
Test 3	100 MB/s	100 MB/s	100 MB/s	12.5 MB/s	7	1/1000	static (area=2)	12.5 KB/s	11.9 KB/s
							adaptive (area=1)	11.1 KB/s	10.6 KB/s

TABLE I
EXPERIMENTAL RESULTS OF THE STATIC/ADAPTIVE COMPUTING PERFORMANCE

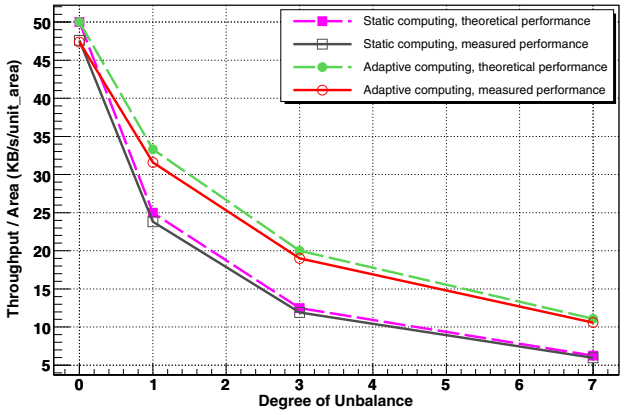


Fig. 5. Result throughput-per-unit-area of static/adaptive computing

measure on the sample setup of Test 2 (DU = 3). Experimental results are shown in Table II. We record the data processing time for 300 times reconfiguration. The total reconfiguration time and the software overhead (including the context saving and restoring time, etc.) are profiled and respectively listed. From the calculated percentages in the fifth column, we observe that the overall PR and software overhead takes only a tiny proportion of the overall measurement time (worst-case 0.30%). The result throughput is not exacerbated much by context switching, due to the dominant time proportion taken by consumers to effectively process data.

Partial bitstream size (KB)	PR counts	Measured processing time (s)	PR time (ms)	SW overhead (ms)	PR+SW overhead (%)	Measured through-put (KB/s)
9.6	300	125.83	12.77	187.2	0.16%	19.01
21.3	300	125.85	28.30	191.7	0.18%	19.01
61.2	300	125.90	81.24	188.8	0.21%	19.00
96.0	300	125.95	127.40	192.6	0.25%	19.00
145.3	300	126.01	192.87	187.1	0.30%	18.98

TABLE II
MEASUREMENT RESULTS OF THE CONTEXT SWITCHING OVERHEAD.

Although the above discussed experiments are based on two streams, the adaptive architecture has the same performance improvement effect for more streams. This is due to the mechanism with which computing resources are efficiently kept busy on data processing of different streams. Performance comparisons with static implementations can be estimated with Equation (8) listed in Section III.

VI. CONCLUSION AND FUTURE WORK

In this paper, we adopt FPGA-based adaptive computing in correlated multi-streaming applications. Multi-streaming models are analyzed and formulated by comparing adaptive computing with the conventional static approach. We implement and verify both models on the FPGA. Experimental results reveal large performance improvement and tiny context switching overhead of the adaptive design on unbalanced processing of multi-streams.

In the future work, real algorithm processors for pattern recognition will replace the consumer model for further verification. Inter-process communication mechanisms are also to be investigated.

ACKNOWLEDGMENT

This work was supported in part by BMBF under contract Nos. 06GI179 and 06GI180, FZ-Juelich under contract No. COSY-099 41821475, and HIC for FAIR.

REFERENCES

[1] M. Stonebraker, U. Cetintemel, and S. Zdonik, "The 8 Requirements of Real-time Stream Processing", *ACM SIGMOD Record*, Volume 34, Issue 4, Dec. 2005.
[2] M. Liu, J. Lang, et al., "ATCA-based Computation Platform for Data Acquisition and Triggering in Particle Physics Experiments", *In Proc. of the International Conference on Field Programmable Logic and Applications*, Sep. 2008.
[3] F. Dittmann and M. Goetz, "Applying Single Processor Algorithms to Schedule Tasks on Reconfigurable Devices Respecting Reconfiguration Times", *In Proc. of the 20th International Parallel and Distributed Processing Symposium*, Apr. 2006.
[4] X. Zhou, Y. Wang, X. Huang, and C. Peng, "Fast On-line Task Placement and Scheduling on Reconfigurable Devices", *In Proc. of the International Conference on Field Programmable Logic and Applications 2007*, Aug. 2007.
[5] H. Walder and M. Platzner, "Online Scheduling for Block-partitioned Reconfigurable Devices", *In Proc. of the Design Automation and Test in Europe Conference and Exhibition 2003*, Dec. 2003.
[6] C. Steiger, H. Walder, and M. Platzner, "Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks", *IEEE Transactions on Computers*, Nov. 2004.
[7] H. K. So, A. Tkachenko, and R. Brodersen, "A Unified Hardware/Software Runtime Environment for FPGA-based Reconfigurable Computers using BORPH", *In Proc. of the 4th International Conference on Hardware/Software Codesign and System Synthesis*, Oct. 2006.
[8] J. Zhu, Ingo Sander and Axel Jantsch, "Performance Analysis of Reconfiguration in Adaptive Real-time Streaming Applications", *In Proc. of the 6th Workshop on Embedded Systems for Real-time Multimedia*, Oct. 2008.
[9] T. Ito, K. Mishou, Y. Okuyama, and K. Kuroda, "A Hardware Resource Management System for Adaptive Computing on Dynamically Reconfigurable Devices", *In Proc. of the Japan-China Joint Workshop on Frontier of Computer Science and Technology*, Nov. 2006.
[10] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow", *In Proc. of the IEEE*, Vol. 75, No. 9, Sep. 1987.
[11] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", *IEEE Transactions on Computers*, Vol. C-36, No. 1, Jan. 1987.
[12] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration", *In Proc. of the International Conference on Field Programmable Logic and Applications*, Aug. 2009.
[13] M. Liu, Z. Lu, W. Kuehn, S. Yang, and A. Jantsch, "A Reconfigurable Design Framework for FPGA Adaptive Computing", *In Proc. of the International Conference on ReConFigurable Computing and FPGAs*, Dec. 2009.