A New Coarse-Grained Reconfigurable Architecture with Fast Data Relay and Its Compilation Flow

Lu Wan Chen Dong Deming Chen
ECE, University of Illinois at Urbana-Champaign
{luwan2, cdong3, dchen}@illinois.edu

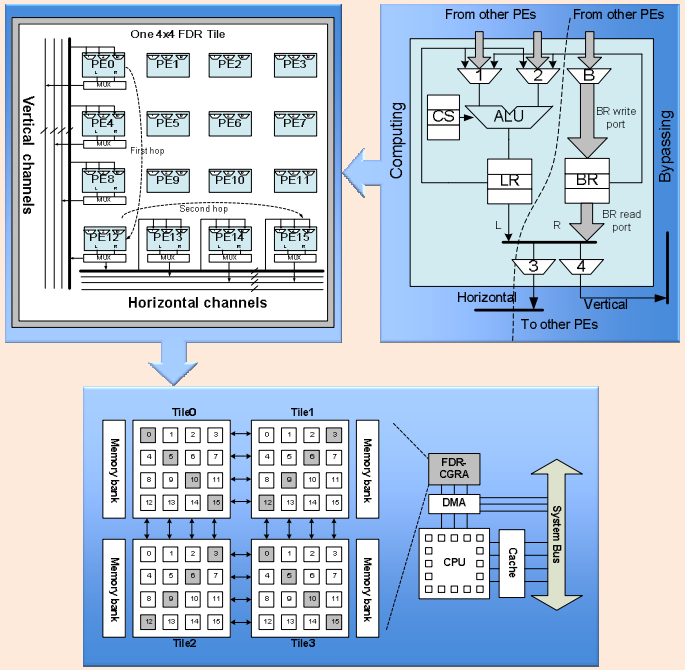
Enabling Hardware Support for Fast Data Relay

❑ **Architectural overview:**
- ➢ Host CPU initializes the kernel computation on our FDR-CGRA.
- ➢ DMA is in charge of system-level dataflow.
- ➢ FDR-CGRA can focus on accelerating time-consuming kernels.

❑ **Processing elements:**
- ➢ Support concurrent computation and communication with dedicated computing path and bypassing path for Fast Data Relay.
- ➢ Use bypassing register file to temporary deposition of intermediate data from other PEs.

❑ **Wire-based companion channels:**
- ➢ Inter-tile communication: direct connections between PEs on the tile boundaries.
- ➢ Intra-tile communication: using companion channels.
  - Vertical : send scalar data from the sender PE to any PE in the same column.
  - Horizontal: send scalar data from the sender PE to any PE in the same row.

(a) Companion channels glue PEs together (b) Structure of Processing Element

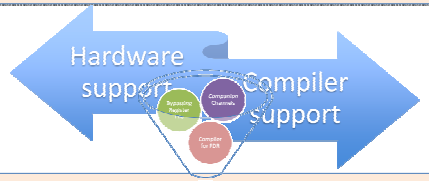


(c) FDR-CGRA system overview

Figure 1: Hardware Support for Fast Data Relay

❑ **Fast Data Relay (FDR)**
- ➢ FDR is a combination of hardware and compiler techniques to enable efficient multi-cycle data communication among *Processing Elements* (PEs) in a *Coarse-Grained Reconfigurable Architecture* (CGRA). We name the proposed architecture FDR-CGRA.

❑ **Advantages of FDR-CGRA:**
- ➢ A corner-to-corner transmission in a tile can be finished in two cycles;
- ➢ Data communication can be done as a background operation without disturbing computation;
- ➢ Source operands can have multiple copies in different PEs so that a dependent PE can find a local copy in its vicinity.



Higher Performance

❑ Benchmark programs are extracted from xvid 1.1.3 and JM7.5b

❑ Preprocess with *Low Level Virtual Machine* (*LLVM*)

❑ Performance is evaluated as *Instructions-Per-Cycle* (IPC)

❑ **FDR-CGRA vs. ADRES [1]:**
- ➢IPC for IDCT_ROW: 35.7 (gain 29%) vs. 27.7
- ➢IPC for IDCT_COL: 35.9 (gain 9%) vs. 33.0

❑ **FDR-CGRA vs. RCP [2]:**
- ➢ For IDCT, 16-issue RCP achieved IPC of 9.2
- ➢ For IDCT, 1-tile (16-PE) FDR-CGRA can achieve IPC of 11.1 (gain 21%)

❑ **Kernel functions of H.264**
- ➢ On 1-tile FDR-CGRA IPC for "get_block" is about 9 and on 4-tiles IPC is more than 36, which is better than IPC of 29.9 reported in [8].

Table 1: Experimental results for video application kernels

Arch.	App.	Large Cfg.: 4 tiles, each tile has 4x4PEs ops	cycles	Avg. IPC	Perf. Gain
ADRES	idct_row(8x8)	-	-	27.7	-
FDR-CGRA	idct_row(8x8)	857	24	35.7	29%
ADRES	idct_col(8x8)	-	-	33.0	-
FDR-CGRA	idct_col(8x8)	1185	33	35.9	9%
FDR-CGRA	interpolate8x8_avg4_c	1193	40	29.8	-
FDR-CGRA	interpolate8x8_halfpel_hv_c	1295	38	34.1	-
FDR-CGRA	sad16_c(16x16)	3441	106	32.5	-

Arch.	App.	Small Cfg.: 1 tile, 4x4PEs each tile ops	cycles	Avg. IPC	Perf. Gain
RCP	idct(row+col)	-	-	9.2	-
FDR-CGRA	idct(row+col)	2042	184	11.1	21%
FDR-CGRA	interpolate8x8_avg4_c	1193	136	8.8	-
FDR-CGRA	interpolate8x8_halfpel_hv_c	1295	135	9.6	-
FDR-CGRA	sad16_c(16x16)	3441	339	10.2	-
ADRES	get_blocks (64 PEs)	-	-	29.9(64 PEs)	
FDR-CGRA	get_block(H)	340	38	8.9	-
FDR-CGRA	get_block(V)	296	37	8.0	-
FDR-CGRA	get_block(V+H)	899	93	9.7	-
FDR-CGRA	get_block(H+V)	900	95	9.5	-
FDR-CGRA	Adjusted Avg. (4 tiles)	-	-	36.1(4 tiles)	21%

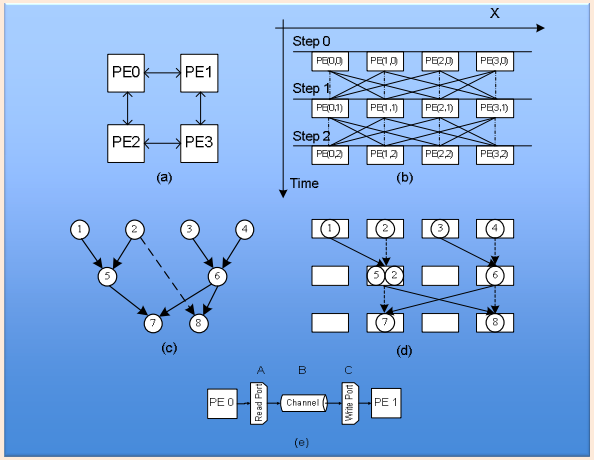


Figure 2: Compilation for FDR-CGRA. (a) A 2×2 PE array, (b) the routing region, (c) a DAG, the dashed edge represents multi-cycle dependency, which may require FDR. , (d) the mapping and routing solution, (e) constraint model.

Compilation for Fast Data Relay

❑ **Problem statement:** Given the Directed Acyclic Graph (DAG) representing the data dependency among operations in the kernel code, find a valid scheduling and mapping of all operations on the routing region so that all data dependency is satisfied and no resource conflict exists.

❑ **Routing region:** The routing region is constructed as shown in Fig. 2 by the following steps: (1) Line up all PEs in the two-dimensional computation grid along the *x* axis. (2) Duplicate the PE row as many times as the minimum feasible (regardless of the detailed routing congestion) scheduling steps along the time axis. (3) Add edges between any nearby PE rows to reflect the connectivity among PEs according to the connectivity specification.

❑ **Resource constraints:** Each segment of a communication link is modeled as a cost function of three parts (Fig. 2(e)): read port, channel, and write port.

Placement and routing for operations

❑ **Placement frontend:**
- ➢It only suggests a trial operation mapping and leaves the detailed resource conflict problems to be resolved later in the backend routing stage, virtually any scheduling algorithm can be used in the frontend stage. We find from experiment that even the simple and fast *list scheduling* can give a fairly good initial placement.

❑ **Routing backend:**
- ➢*Rip-up & reroute*: Iteratively perform operations as follows: rip-up and reroute will perform the following steps: (1) Pick a routed path, usually a path that can afford extra latency without degrading overall performance; (2) rip-up (or break) the path by de-allocating all the resources (i.e., channels and ports) taken by this path; (3) based on the existing congestion information of the routing region, find an uncongested detour and reroute the path.
- ➢*Non-volatile copy*: distribute reusable operands across a tile as non-volatile copies so that a dependent PE can find a local copy in its vicinity.
- ➢*Schedule step relaxation:* If rip-up and reroute and non-volatile copy cannot resolve all the congestions, schedule step relaxation will be invoked to insert extra scheduling steps into the most congested regions.

References

1.B. Mei, et al., "Design methodology for a tightly coupled VLIW/reconfigurable matrix architecture: a case study", in *Proceedings of DATE*, p. 21224. 2004
2.O. Colavin, et al., "A scalable wide-issue clustered VLIW with a reconfigurable interconnect," in *Proceedings of CASES*, pp. 148–158. 2003.
3.M. B. Taylor, et al., "Evaluation of the RAW microprocessor: an exposed-wire-delay architecture for ILP and streams," in *Proceedings of ISCA*, pp. 2–13 2004.
4.G. Dimitroulakos, et al., "Design space exploration of an optimized compiler approach for a generic reconfigurable array architecture," *J. Supercomputing*, vol. 40, issue 2, pp. 127–157, 2007.
5.S. Sait, et al, *VLSI Physical Design Automation: Theory and Practice*. Hackensack, NJ: World Scientific Publishing, 1999.
6.C. Lattner, "Introduction to the LLVM Compiler Infrastructure," presented at the 2006 Itanium Conference and Expo, San Jose, California, April 2006.
7.T. J. Todman, et al., "Reconfigurable computing: architectures and design methods," *IEE Proc. Comp. Digit. Tech.*, Mar. 2005.
8.B. Mei, et al., "Mapping an H.264/AVC decoder onto the ADRES reconfigurable architecture," in *Proceedings of FPL* , pp. 622–625, 2005