DiAG: A Dataflow-inspired Architecture for General-purpose Processors

Dong Kai Wang Nam Sung Kim
University of Illinois at Urbana-Champaign

1. Motivation

Modern high performance microprocessors suffer poor energy efficiency due to complex front-end instruction overheads [3]. Out-of-order architectures can exploit instruction level parallelism (ILP) but require expensive hardware structures that rename registers and reorder instructions to aggressively issue multiple instructions per cycle. As hardware accelerators take the spotlight in industry and research, general-purpose processor architecture has largely remained stagnant for decades.

This paper proposes DiAG, a novel, energy efficient CPU microarchitecture that accomplishes three main goals: **1.** Dynamically extracting ILP with little control overhead. We do so by transparently constructing dataflow graphs (DFGs) of the running program in hardware. **2.** Exploiting instruction reuse [8], such as loop iterations or commonly used functions. The datapaths we constructed before can be reused at no cost, eliminating the need for fetching and decoding the same instructions. **3.** Using thread-level pipelining to exploit data level parallelism (DLP). Identical threads running in parallel can share the same datapath by pipelining its functional units. We achieve these goals while maintaining generality and transparency to software. DiAG processors are plug-and-play: they can support existing ISAs (we use RISC-V) without requiring special extensions, compilers, or libraries. Though DiAG works standalone, it can also be enhanced with ISA extensions and further compiler support.

2. Limitations of the State of the Art

We focus on dataflow architectures in this section, comparisons to similar superscalar techniques are in Section 3 of the paper. In general, a dataflow processor spatially maps a program's DFG onto a planar grid of processing elements (PEs). DiAG addresses two important limitations of past works:

1. Granularity of control. Most dataflow architectures cannot fully handle control flow changes at the instruction level. They use compilers to break down a program into control-free sequences of code, e.g. 'blocks' in TRIPS [6] or 'waves' in Wavescalar [9]. These sequences are then mapped and executed block-wise in hardware. As a result, supporting an arbitrary branch instruction or precise interrupts is difficult to realize. DiAG does not decompose the program into subgraphs and handles all control flow changes at the instruction level. It supports **precise interrupts** and **speculative execution** fully (e.g. even if all instructions in a program are branches).

2. General compatibility. Most dataflow architectures require special instruction sets and/or compilers and/or software libraries to work with the hardware [6, 7, 4, 9, 2, 5]. Thus, there is a high barrier to adoption as existing binaries for commonly used ISAs must all be recompiled to work on the platform. Furthermore, control limitations in **1.** only make it more difficult to run all types of applications. DiAG differs from past dataflow works in that instructions are mapped **in program order** but **execute out-of-order** as shown in Figure 1. Register lanes described in the next section construct the DFG dynamically and completed instructions retire in-order.

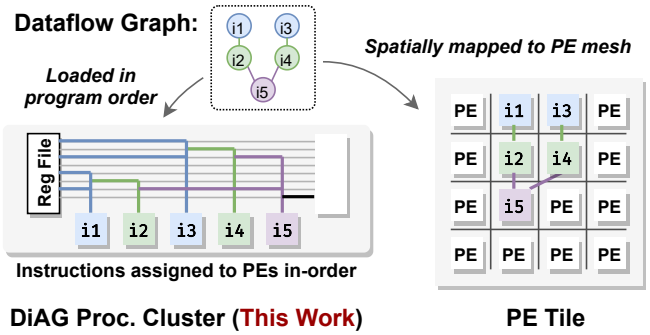

Figure 1: (*Left*) In DiAG, instructions are assigned in program order to PEs, register lanes form the DFG dynamically. (*Right*) In dataflow archs, the DFG is mapped to a mesh of PEs.

3. Key Insights

The DiAG architecture is designed based on three key insights:

1. Transparent dataflow execution. We can easily build the program's DFG in hardware as shown in Figure 2. We extend the register file into lanes, which are 32-bit wires (for 32-bit ISAs) along with a valid bit for each register. Rather than constructing the DFG explicitly, we assign program instructions in-order to a row of processing elements (PEs). Each PE reads its assigned instruction's source operands from the register lanes and writes its output to the destination lane. Observe that a flattened DFG naturally arises in Figure 2(c) since register lanes essentially forward all values from producer to consumer instructions. PEs begin executing as soon as their operands are valid (in out-of-order fashion) and the example program completes in the same 3 cycle latency as the original DFG.

2. Branches and instruction reuse. The DFGs we built from register lanes in **1.** can be reused (e.g. loop iterations) to avoid re-fetching and decoding the same instructions. This is automatically done by backward branches in the program. To support branches and jumps, we dedicate a PC lane that carries the program counter through each PE much like register lanes. Since instructions are assigned in-order, each PE checks its instruction address against the input PC, then increments the

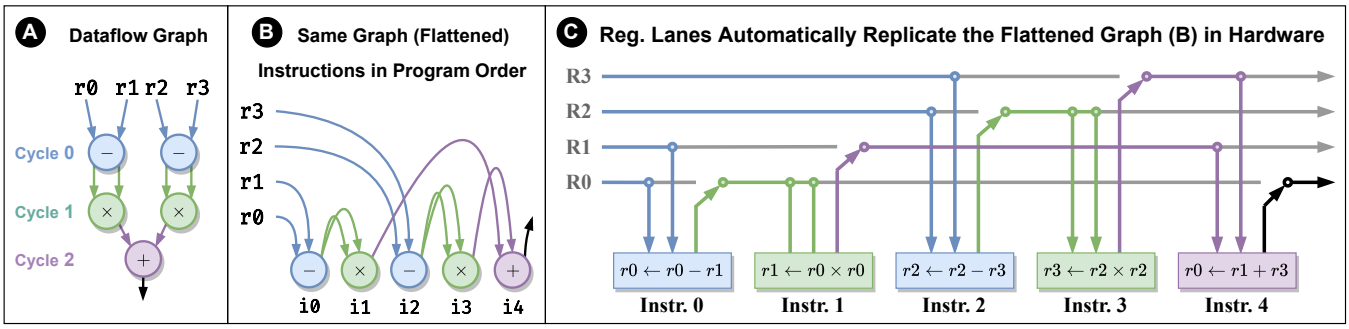

Figure 2: Dataflow execution of a program with five instructions. Instruction assigned to functional units in program order implicitly constructs the flattened dataflow graph of the program, i.e. the DFG in (A) and (B) are identical.

output PC by a word if matched, otherwise the PE is disabled and does not modify the PC lane. When a branch or interrupt occurs at instruction i, the PC is changed to the target address and subsequent instructions at $i+1, i+2, \ldots$ will no longer match and these PEs are effectively flushed, allowing only prior instructions to write and store values. Since PEs are in-order, they serve the same role as the reorder buffer to commit changes. Figure 3 is a simplified high-level diagram of a DiAG processor, PEs are grouped into clusters that buffer register lanes and share load-store units.

3. Temporal parallelism. If multiple identical threads can execute concurrently, we pipeline the threads at the granularity of clusters to exploit DLP.

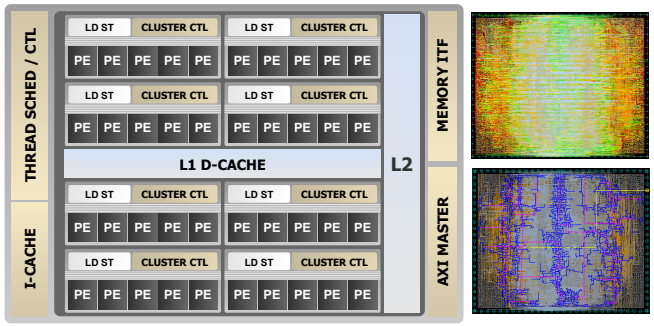

Figure 3: DiAG High-level Architecture.

4. Main Artifacts

We implement a DiAG processor prototype that supports the RISC-V 32-bit ISA (RV32IMF) in SystemVerilog. Multiple configurations of this processor with 32 to 512 PEs are synthesized with commercial EDA tools to evaluate hardware area, timing, and power. We also demonstrate the processor running bare metal RISC-V programs on a Xilinx VC709 FPGA board. Additionally, we apply custom RISC-V instruction extensions to support thread pipelining.

5. Key Results and Contributions

We use RTL simulation to evaluate DiAG configurations with up to 512 PEs in the Rodinia benchmark suite [1].

- Against a simulated 12 core 8-issue OoO ARM processor with normalized frequency, a 512 PE DiAG processor achieves a mean 1.15× single-thread perf., 1.31× multi-thread perf. (with TLP and DLP), and 1.77× improvement in energy efficiency.
- Synthesis results on a 45nm library show that DiAG uses significant hardware area (88mm^2 for 512 PEs), but a small fraction is dynamically active.

DiAG presents a dataflow-based alternative to the conventional out-of-order μarch that heavily improves energy efficiency in compute-centric applications at the cost of higher chip area.

6. Why ASPLOS

Although we present DiAG as a work of specialized CPU microarchitecture, our results show that compiler assistance to enable thread-level pipelining is valuable to unlock its full potential. Our philosophy is that a novel architecture like DiAG must first meet baseline support for a standard ISA and existing code, which is important to commercial adoption. Once satisfied, it delivers additional optimization methods through compiler support and software libraries. We believe that DiAG is a suitable work for the ASPLOS audience.

7. Citation for Most Influential Paper Award

DiAG is a general-purpose processor architecture that dynamically and transparently builds a reuseable dataflow graph of the executing program in hardware. It emerges as a novel alternative to the longstanding out-of-order model, reducing control overheads and bridging the energy efficiency gap between CPU and accelerator for compute-centric applications.

8. Revisions

Architecture: Added improved support for speculative execution and precise interrupts. Register lanes are now buffered once in a cluster rather than at each PE, sharply reducing area with little performance impact. **Evaluation:** Added sections for hardware area and timing analysis to address previously raised concerns on area cost and feasibility. We also improve the accuracy of simulations to model performance. **Paper:** A proper related works section is added that distinguishes DiAG from past dataflow works. Architecture and evaluation sections are expanded with new results and details.

References

[1] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IEEE International Symposium on Workload Characterization*, 2009.

[2] V. Govindaraju, C. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim. DySER: Unifying functionality and parallelism specialization for energy-efficient computing. *IEEE Micro*, 32(5):38–51, 2012.

[3] Stephen W. Keckler, William J. Dally, Brucek Khailany, Michael Garland, and David Glasco. GPUs and the Future of Parallel Computing. *IEEE Micro*, 31(5):7–17, September 2011.

[4] Walter Lee, Rajeev Barua, Matthew Frank, Devabhaktuni Srikrishna, Jonathan Babb, Vivek Sarkar, and Saman Amarasinghe. Space-time scheduling of instruction-level parallelism on a Raw machine. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VIII, page 46–57, New York, NY, USA, 1998. Association for Computing Machinery.

[5] T. Nowatzki, V. Gangadhar, N. Ardalani, and K. Sankaralingam. Stream-dataflow acceleration. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 416–429, 2017.

[6] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Nitya Ranganathan, Doug Burger, Stephen W. Keckler, Robert G. McDonald, and Charles R. Moore. TRIPS: A polymorphous architecture for exploiting ILP, TLP, and DLP. *ACM Trans. Archit. Code Optim.*, 1(1):62–93, March 2004.

[7] A. Smith, J. Burrill, J. Gibson, B. Maher, N. Nethercote, B. Yoder, D. Burger, and K. S. McKinley. Compiling for edge architectures. In *International Symposium on Code Generation and Optimization (CGO'06)*, pages 11 pp.184–195, 2006.

[8] Avinash Sodani and Gurindar S. Sohi. Dynamic instruction reuse. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, ISCA '97, pages 194–205, New York, NY, USA, 1997. Association for Computing Machinery.

[9] Steven Swanson, Andrew Schwerin, Martha Mercaldi, Andrew Petersen, Andrew Putnam, Ken Michelson, Mark Oskin, and Susan J. Eggers. The WaveScalar architecture. *ACM Trans. Comput. Syst.*, 25(2), May 2007.