RETROSPECTIVE: Evaluation of the RAW Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams

Michael B Taylor, University of Washington
David Wentzlaff, Princeton University
Saman Amarasinghe, MIT
Anant Agarwal, MIT

I. Introduction

This paper, published in 2004, was an evaluation of our prototype of a scalable general-purpose tiled multicore processor called Raw.

Original Proposal. An early vision of Raw was proposed in the famous Sept 1997 IEEE Computer "Billion Transistor Microprocessor" issue [O48], where a number of leading architecture groups proposed how to productively put to use a billion transistors in a single chip processor. At the time, computer architecture researchers were very wedded to the promise of single-threaded performance, which was advancing rapidly due to Dennard Scaling, circuit design techniques refined in the GHz frequency wars, and the rapid development of out-of-order superscalar technology. The idea of parallelizing programs by hand had become unpalatable to the architectural community (and indeed, program committees). As a result, most of the "Billion Transistor" proposals focused on hiding resources behind the microarchitecture.

Our 1997 paper proposed that limits due to wire delay, rapidly declining efficiency due to microarchitectural overheads, and complexity would ultimately require a shift to tiled multicore architectures that exposed parallel VLSI resources to software in a physically scalable way (i.e. cores). The paper further proposed the use of what we later termed scalar operand networks [O45], a class of networks optimized for minimal latency of ordered communication of scalars between cores, and compiler technology for automatically parallelizing programs across cores using this network. Finally, it proposed an idea which we internally called "all-software hardware", which was to try to implement many traditional hardware mechanisms in software, to reduce hardware area and allow more parallelism to be exposed.

II. The Paper

Seven years later, many Raw internal research efforts had been pursued among these different thrusts, focusing on parallelizing compilers [O24, O5, O25], all-software hardware, scalar operand networks, and exposing parallel resources through tiled multicore. We implemented a full-scale end-to-end system, including the architecture and microarchitecture, a state-of-the-art 180nm 425-MHz 16-core 331 mm^2 chip [1], several compilers, a motherboard, and a suite of software applications. The 2004 paper reported on the fully-fleshed out, optimized system. A 4-chip, 64-core system was also constructed which demonstrated the glueless interconnect built into the Raw chip. We connect below the ideas from the proposal paper to the evaluation paper and also discuss some new ideas that arrived after the proposal:

Scalable Tiled Multicores. Perhaps the most successful of the thrusts was the one that focused on designing a scalable tiled multicore architecture and its VLSI implementation. The prototype worked through many of the details of a realistic on-chip network and scalable I/O system to service these cores, looked at some of the important questions of provisioning and optimizing the cores, reducing latency, maximizing bandwidth, and preventing deadlock. In addition to providing architectural scalability, the tiled structure was especially for managing the exhaustive amount of computation required for physical design when taping out a chip. Many of these design decisions are detailed in [2], which cleverly includes a full-scale chip layout where you can zoom in and view individual gates of the design.

All-Software Hardware. On the all-software hardware front, we had explored many cases where a combination of compiler and runtime technology could replace hardware. The most successful of these efforts was implementing software instruction caching using binary rewriting [3], which today remains salient for embedded processors with fixed-size instruction memory. We had also tried ideas like software data caching, software virtual memory, software branch prediction and software floating point but realized less success. Paradoxically, many of these efforts ended up taking up more hardware area when implemented in software, because of their software footprint in on-chip memory. Overall, the conclusion could be distilled relatively simply: if you have to do it all the time, do it in hardware to minimize energy, latency and area. If you only have to do it some of the time, consider software more strongly.

Automatically Parallelizing Compilation. On the compiler front, we developed an impressive ILP automatically-parallelizing compiler, RawCC, based on the Stanford SUIF infrastructure, and a key part of that was the hardware scalar operand network that reduced latencies enough to make parallel speedup attainable. However, due to the ever-present alias-analysis problem, and the resulting lack of large amounts of ILP in legacy SPEC binaries, this limited the scope of single-threaded conventional applications where speedup was attainable. We also found that it was hard to mitigate the high software costs of handling dynamic effects in the compiler.

NEW: Domain Specific Languages. We spent several years mapping applications to the Raw in order to generate the results and make the case for the architecture. Frustrated with the difficulty of scaling performance with ILP-based models, we started developing a language, StreamIt, that could be used to express streaming computations at a higher level, and paradoxically, make the task of automatic parallelization even easier. StreamIt was one of the earliest examples of a domain specific languages (DSLs) whose purpose was not only to improve programmer productivity, but to constrain the expression of programs in a way that simultaneously improves productivity and makes the parallelization problem tractable [O11].

NEW: Multiprogramming. Perhaps relatively unsurprising in retrospect, the results from the paper showed that tiled multicores were very good at running unrelated unparallelized applications with relatively little performance interference. Although this was not considered a conference grade insight at the time, it is very relevant to how multicores are used today in datacenters and cloud computing.

III. Resonance with the Future

The progression of technology is often less a progression of individual works and more of a collective contribution to an evolving technological zeitgeist. Without diminishing the contributions of many others in the architecture community, we highlight a few resonances between this paper and subsequent developments in the field.

Scalable multicore (and manycore) has resonated across industry, and tiled variants have appeared, for example, in the Tilera 64-core TILE64 [4][5], Tile64Pro, and 72-core TileGX architectures, Intel's 80-core Teraflop chip [6], and perhaps most significantly, the Intel Skylake SP processor that is widely used in today's datacenters. A few members of the original team have also continued refinements to manycore processor chip design, in the form of the open source Celerity [7] and Piton [8][9] processors.

Raw was one of the earliest examples of a scalable network-on-chip (NOC) and the analysis of its on-chip networks [O19] laid a foundation for a wide variety of network-on-chip literature. The work on scalar operand networks has proven useful to other parallel architects who have needed mechanisms for tight synchronization of data between cores; and Raw has served as an early inspiration for CGRA architectures, which share Raw's desire to precisely orchestrate the flow of operands between plentiful parallel resources.

Raw's StreamIt DSL served as an early example of the efficacy of domain-specific languages for attaining parallel performance, which has been reflected in the success of other DSLs like PyTorch, CUDA, OpenCL and Halide.

References

[1] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffmann, P. Johnson, W. Lee, A. Saraf, N. Shnidman, V. Strumpen, S. Amarasinghe, and A. Agarwal, "A 16-issue Multiple-Program-Counter Microprocessor with Point-to-Point Scalar Operand Network," in *ISSCC*, 2003.

[2] M. Taylor, "Tiled Microprocessors," Ph.D. dissertation, Massachusetts Institute of Technology, 2007.

[3] J. E. Miller and A. Agarwal, "Software-based instruction caching for embedded processors," in *ASPLOS*, 2006.

[4] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64 processor: A 64-core SoC with mesh interconnect," in *ISSCC: Digest of Technical Papers of the IEEE International Solid-State Circuits Conference*, 3-7 2008, pp. 88–89,598.

[5] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, pp. 15–31, 2007.

[6] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28tflops network-on-chip in 65nm cmos," in *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, 2007, pp. 98–589.

[7] S. Davidson, S. Xie, C. Torng, K. Al-Hawaj, A. Rovinski, T. Ajayi, L. Vega, C. Zhao, R. Zhao, S. Dai, A. Amarnath, B. Veluri, P. Gao, A. Rao, G. Liu, R. K. Gupta, Z. Zhang, R. Dreslinski, C. Batten, and M. B. Taylor, "The Celerity Open-Source 511-core RISC-V Tiered Accelerator Fabric," *Micro, IEEE*, Mar/Apr. 2018.

[8] M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, J. Balkind, A. Lavrov, M. Shahrad, S. Payne, and D. Wentzlaff, "Piton: A manycore processor for multi-tenant clouds," *IEEE Micro*, pp. 70–80, March/April 2017.

[9] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "OpenPiton: An open source manycore research framework," *ASPLOS*, 2016.