THEME ARTICLE: TOP PICKS FROM THE 2023 COMPUTER ARCHITECTURE CONFERENCES

Decoupled Vector Runahead for Prefetching Nested Memory-Access Chains

Ajeya Naithani, *Ghent University, B-9000, Ghent, Belgium*
Jaime Roelandts, *Ghent University, B-9000, Ghent, Belgium*
Sam Ainsworth, *University of Edinburgh, EH8 9AB, Edinburgh, U.K.*
Timothy M. Jones, *University of Cambridge, CB3 0FD, Cambridge, U.K.*
Lieven Eeckhout, *Ghent University, B-9000, Ghent, Belgium*

Abstract—Decoupled Vector Runahead (DVR) exploits massive amounts of memory-level parallelism to improve the performance of applications featuring indirect memory accesses by dynamically inferring loop bounds at run-time, recognizing striding loads, and speculatively vectorizing subsequent instructions that are part of an indirect chain. DVR runs as an on-demand, speculative, in-order, lightweight hardware subthread alongside the main thread within the core. DVR incurs a minimal hardware overhead while delivering a substantial performance boost.

Index Terms: *CPU, microarchitecture, vector runahead, prefetching, speculative vectorization, indirect memory accesses*

Out-of-order cores are bigger than ever, with the latest processors featuring reorder buffers of many hundreds of entries. And yet, although modern-day out-of-order (OoO) processors are given more than ample resources, and thus their out-of-order queueing resources are rarely filled to capacity, they are still memory-bound especially for workloads that feature chains of dependent memory accesses, or indirect memory accesses. One recent proposal, Vector Runahead [1], presents a potential method for doing better. Rather than work-skipping as earlier runahead proposals do [2,3,4,5] to keep uncovering memory-level parallelism, Vector Runahead reformulates the transient execution performed within runahead mode to be primarily based on loop-level parallelism, following independent groups of many different dependent chains of memory accesses from future loop iterations in the program, and speculatively executing them in a vectorized manner to reduce front-end and back-end pipeline resource requirements. Vector Runahead (VR) can successfully follow and prefetch the complex memory-access patterns. However, like the underlying out-of-order core, even with a large reorder buffer (ROB), Vector Runahead is still memory-bound. Because the large reorder buffer rarely fills up, the resource starvation that triggers Vector Runahead rarely occurs, and so its benefits over even resource-bountiful out-of-order execution are not allowed to shine.

Decoupled Vector Runahead (DVR) contributes three fundamental innovations to data prefetching and runahead execution in general, and VR in particular, enabling effective and accurate prefetching for challenging graph analytics workloads with chains of dependent memory accesses on today's most aggressive out-of-order cores. (1) DVR does not wait for the reorder buffer in an out-of-order core to fill up before triggering runahead execution, as performed in traditional runahead. Instead, *DVR decouples runahead execution* by continuously monitoring the main instruction stream and, when deemed beneficial, DVR initiates a separate lightweight runahead subthread to execute in-order alongside the main out-of-order core to prefetch along multiple chains of dependent

Digital Object Identifier 10.1109/XXX.0000.0000000

loads in parallel and in a vectorized manner. This enables the main thread to continue making forward progress while at the same time using few hardware resources for in-order vector-runahead execution (as opposed to traditional runahead techniques, which reuse the main out-of-order core for prefetching). (2) Prefetching in general is a balancing act in hiding latency versus polluting on-chip caches and generating excessive memory bandwidth. *DVR detects loop bounds* at run time in hardware to precisely determine how many parallel chains of dependent loads to prefetch to avoid over-prefetching. (3) Prefetching needs to be timely, which implies that one needs to prefetch far into the future, i.e., several hundreds, if not thousands, of instructions ahead of time. This is challenging, particularly for graph workloads with irregular memory-access patterns where the number of edges per vertex determines the number of chains of dependent loads one can prefetch in parallel. *DVR includes nesting*, which enables prefetching chains of dependent loads for multiple vertices in parallel, thereby exposing prefetching opportunities far into the future.

Vector Runahead

Before explaining DVR's key innovations in more detail, we first need to introduce Vector Runahead (VR) and elaborate on its limitations. VR is a recently proposed microarchitectural technique that generates high degrees of memory-level parallelism (MLP) by transiently vectorizing a chain of memory accesses for the purpose of prefetching. VR is a major departure from prior runahead techniques (such as Precise Runahead[6]) that must fetch the future instruction stream to generate speculative prefetches. More importantly, prior runahead techniques cannot prefetch chains of dependent memory accesses. Consequently, they deliver limited performance gains for graph workloads on out-of-order processors. Instead, VR predicts the initial addresses for multiple chains of dependent memory accesses, and simultaneously generates prefetches for many chains in a vectorized manner.

Figure 1 shows how VR works for a chain with two load instructions. The first load accesses memory in a sequential manner while the second load is an indirect load, as it depends on the value accessed by the first load, i.e., its memory accesses are irregular. The otherwise underutilized processor back-end when executing a chain of memory accesses would now be inundated with a large number of (scalar) instructions generated by VR. Therefore, VR combines multiple scalar instructions into vectors and executes them in parallel on the underlying vector execution units.

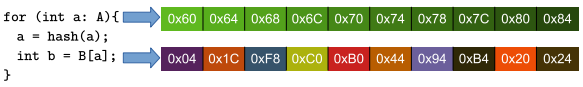


FIGURE 1. Vector Runahead (VR) detects striding access patterns and generates vector prefetches for the chain of instructions originating from the striding load, via an architecturally transparent transient runahead execution. This allows it to bring in complex data-dependent chains of memory accesses that are otherwise unpredictable.

While VR invented a new way of generating prefetches even for chains of dependent memory accesses with complex address calculation patterns, it is hampered by five key limitations. (1) VR is initiated only after the ROB is filled with instructions. Therefore, its performance gain decreases with increasing ROB size. (2) VR does not adapt the vector length, i.e., the number of scalar-equivalent (load) instructions generated for each instruction in the chain is fixed. Having a single, fixed vector length leads to cache pollution and wasted DRAM bandwidth for many applications. (3) VR cannot deliver high MLP for applications with short inner loops. (4) VR cannot generate prefetches along diverging vector lanes after a branch instruction, which leads to a missed opportunity for improving performance. (5) VR delays the main thread from making progress until the prefetches have been generated for the complete indirect chain.

Decoupled Vector Runahead

Decoupled Vector Runahead (DVR) eliminates these limitations by proposing three major contributions.

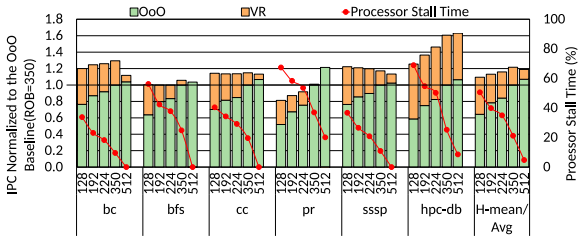


FIGURE 2. Vector Runahead's performance improvement diminishes with large ROBs, because it is only triggered when the processor stalls on a full ROB.

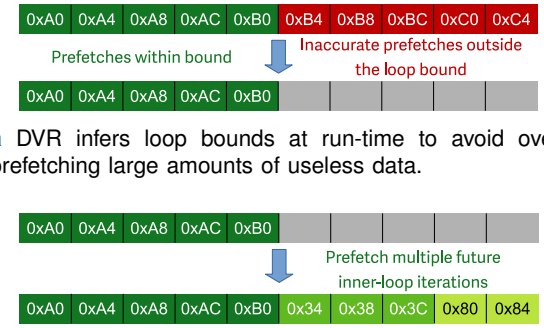
Contribution #1: Decoupling. Runahead has traditionally been initiated after filling the reorder buffer of an out-of-order processor[6]. However, this condition

has become a limiting factor as microarchitectural structures have continued to increase in size over the past two decades, with modern-day processors featuring ROBs with more than 500 entries, see for example Intel's Alder Lake processor[7]. While being an attractive approach to trigger accurate future memory accesses by pre-executing the application's own code, the usefulness of runahead techniques hinges upon this trigger condition to fill the ROB. In fact, we find that VR's performance benefit over a baseline out-of-order core diminishes from 70% for a 128-entry ROB to merely 5% for a 512-entry ROB, see Figure 2.

Decoupling vector runahead, or liberating runahead from its pre-condition to fill the ROB, is DVR's first key contribution. A stride detector in the core continuously checks for indirect access patterns, and when it finds one, the core initiates DVR by offloading the chain to a separate, in-order, speculative subthread context. The subthread speculatively vectorizes the indirect chain and executes the generated vector instructions on the core's vector execution units. This subthread context does not need out-of-order execution resources to extract high performance. In fact, the runahead subthread executes in-order and hence it incurs limited hardware overhead. The decoupled vector runahead subthread executes transparently alongside the main execution thread, which executes on the out-of-order execution engine, i.e., the core continues to execute instructions out-of-order while the subthread prefetches future memory accesses in-order. Speculative vectorization by the decoupled vector runahead subthread continues until the last indirect load in the chain has been vectorized.

Contribution #2: Loop-bound detection. A key trade-off for prefetching, in general, is to determine how far into the future to prefetch. More specifically for a vector-runahead technique this translates into determining the right number of future iterations of the inner loop to speculatively vectorize and prefetch, as this determines the number of dependent chains one can prefetch in parallel. Under-prefetching leaves performance on the table, while over-prefetching pollutes the caches and wastes precious DRAM bandwidth. Because the prior Vector Runahead technique considers a fixed degree of vectorization, its effectiveness diminishes severely for real-world graph workloads with a varying number of edges per vertex.

Accurately determining the number of future iterations by inferring loop bounds at run-time, see Figure 3a, is the second novel contribution of DVR. We develop a set of novel microarchitectural analyses to track loops via the existence of backwards edges in the control flow graph at run-time, using the inputs to innermost loops (also detected at run-time) to track remaining iterations, and mask the excess. We find that loop-bound detection is accurate and substantially reduces the number of excess prefetches.



a DVR infers loop bounds at run-time to avoid over-prefetching large amounts of useless data.

b DVR achieves extreme memory-level parallelism even on short inner loops, by analyzing the macro-level structure of inner- and outer-loops simultaneously to overlap future work from many variable-length inner loops at once.

FIGURE 3. DVR employs (a) loop bound detection and (b) nested runahead to uncover as many correct prefetches as possible within the available vector length.

Contribution #3: Nesting. For many workloads, the number of future iterations of a loop can still be small because many vertices in a graph may have a small number of edges, as we observed on real-world graphs. Initiating DVR for a small number of future iterations only is suboptimal as the processor will execute them in the near future regardless. Therefore, it is critical to generate prefetches far into the future, by prefetching several future invocations of the inner loops. This can only be accomplished if we can target future invocations of the loop, determine loop bounds for each invocation, and then generate prefetches for iterations belonging to each invocation. Nested runahead, the third novel contribution of DVR, achieves this by vectorizing the outer loop to discover loop bounds for multiple invocations of the inner loop. It then generates prefetches for the multiple invocations of the inner loop as illustrated in Figure 3b. This allows DVR to jump ahead into program execution much further than even the largest reorder buffer to find accurate prefetches.

DVR Microarchitecture

DVR operates as follows. When the core discovers that it is executing a loop with dependent loads, based on a striding load that can be used to predict future loop

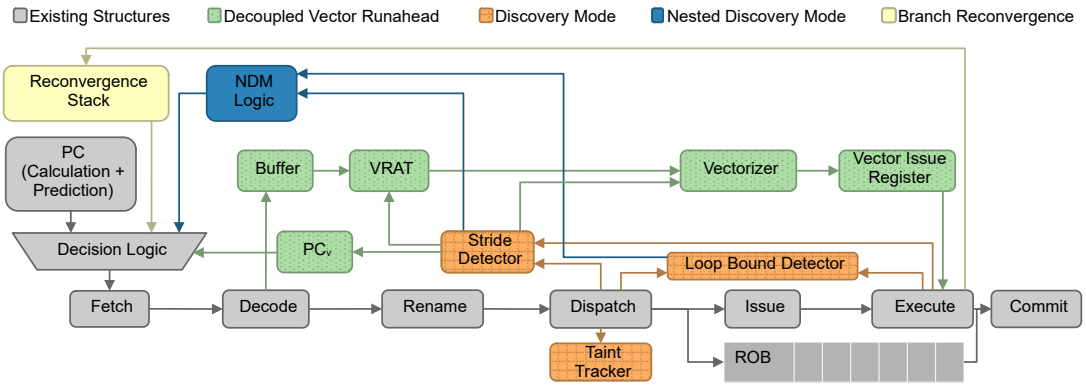


FIGURE 4. DVR processor pipeline.

iterations, a specialized vector-runahead subthread is activated on the same core as the currently executing main thread. This subthread is dynamically generated to prefetch many memory accesses into the future, but without affecting the semantics of the main thread. The vector-runahead subthread runs alongside the main thread on the same core, much like how threads co-execute in simultaneous multithreading (SMT)[8], except that the subthread is microarchitecturally generated, transparent to software, transient (to prefetch into the cache rather than achieve real computation), speculative, reordered to achieve extremely high memory-level parallelism, and significantly simpler, i.e., the subthread executes in-order. The vector-runahead subthread is also closely related to simultaneous subordinate microthreading[9], which also aims at improving performance of the main thread. Whereas a subordinate microthread is written in microcode featuring specialized machine-specific instructions, the vector-runahead subthread is dynamically generated and derived from the main application thread.

To achieve high memory-level parallelism from this in-order vector-runahead subthread, even while following chains of dependent loads that stall the subthread, we use single-instruction multiple-thread (SIMT) data-level parallelism, to execute large numbers of each instruction from the front-end, each representing a different loop iteration, simultaneously, thereby prefetching far into the future. Since this happens continuously, and overlaps with the execution of the main thread, most of the main out-of-order thread's memory accesses hit in the L1 cache by the time it reaches them — thus even for very large processors with massive windows, significant speedups can be achieved.

Figure 4 provides a schematic of a processor's microarchitecture enhanced to support DVR. The stride detector obtains information about loads from the dispatch and execute stages of the pipeline. Once a stride is detected, DVR enters Discovery Mode, which uses the Taint Tracker and Loop-Bound Detector to discover information for the subsequent vector-runahead execution, i.e., it determines how many loop iterations to speculatively vectorize and thus generate prefetches for. The Nested Discovery Mode logic will be used if Discovery Mode finds too few elements of the loop to vectorize, and if so, multiple future invocations of the inner loop will be speculatively vectorized by analyzing the outer loop. Once Discovery Mode is complete, the vector program counter (PC_V) will be populated with the PC of the striding load, the VRAT will be populated with the striding load addresses and a copy of the main thread's scalar registers, and the decoupled vector-runahead subthread will initiate. The Reconvergence Stack will engage upon divergence in control-flow between the vector lanes.

We refer to the conference paper[10] for details about the implementation. Overall, DVR requires minimal changes to the processor pipeline and incurs only 1,139 bytes of overhead.

Key Results

Figure 5 shows the key results for DVR assuming a baseline 350-entry ROB out-of-order core. Across a wide set of complex graph analytics, database and high-performance computing (HPC) applications on modern-sized cores, DVR offers a considerably higher average speedup of 2.4× versus 1.2× for VR. While VR improves performance by 1.2×, decoupling increases the performance speedup by 1.5×. Eliminating incorrect prefetches through loop bound detection further increases the performance to 1.8×. Nested runa-

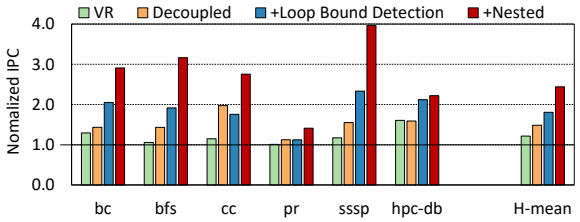


FIGURE 5. Breaking down DVR's performance normalized to a baseline out-of-order (OoO) core with a 350-entry ROB: (1) Vector Runahead[1], (2) 'Decoupled' triggers a vector-runahead subthread whenever a stride is detected, (3) '+Loop Bound Detection' further improves prefetch accuracy, and (4) '+Nested' completes DVR by prefetching over multiple short inner loops.

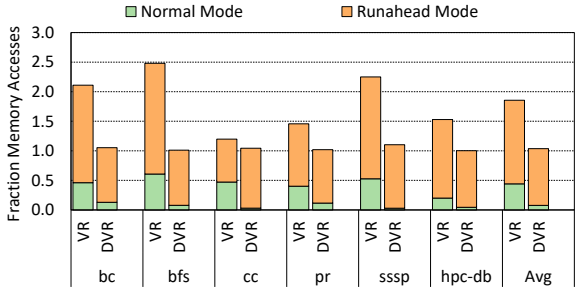


FIGURE 6. Number of off-chip memory accesses for VR and DVR normalized to OoO, and fraction of memory accesses in normal versus runahead execution mode. VR substantially over-prefetches in contrast to DVR which accurately prefetches most memory accesses.

head successfully prefetches across many invocations of the same loop for vertices with a small number of edges, and propels the performance to an overall 2.4× speedup compared to the baseline out-of-order core.

Figure 6 shows both the total number of main memory accesses performed, and the fraction within the main thread and runahead mode or subthread. Both DVR and VR are given for comparison, relative to the same out-of-order baseline. DVR is extremely accurate because of its Discovery Mode, over-fetching only 3.7% compared to the baseline. By contrast, Vector Runahead over-fetches by on average 85.6% because it lacks loop-length analysis. As well as being more accurate, DVR also covers far more of each application during runahead execution, due to triggering more eagerly, and because Nested Mode prefetches multiple future loop invocations.

Figure 7 shows that the performance of DVR continues to increase with increasing ROB sizes. This

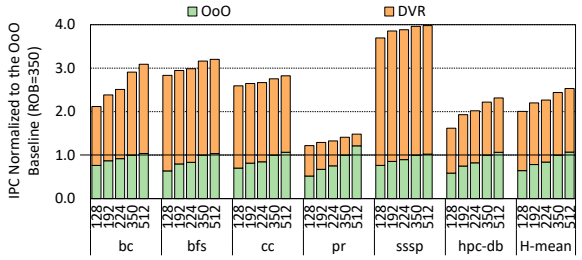


FIGURE 7. Performance of DVR with increasing ROB size, relative to a baseline out-of-order core with a 350-entry ROB.

demonstrates that DVR is indeed future-proof, unlike any of the prior runahead techniques, including VR.

Looking Forward

Decoupled Vector Runahead (DVR) completes the job started by VR. While VR offered a whole new form of memory-level parallelism, invisible at the programmer's interface, we believe that DVR is the missing link that will allow the concept of vector runahead to attain widespread commercial viability and adoption. As aforementioned, we believe that DVR contributes three fundamental innovations that substantially advance the state-of-the-art in data prefetching in general and (vector) runahead execution in particular, for some of the most challenging workloads (graph analytics with irregular and dependent chains of memory accesses). Decoupling, loop-bound detection, and nesting enable vector runahead execution to run continuously, using a lightweight in-order subthread context, prefetching far into the future without polluting the on-chip caches and without generating excess prefetches, while at the same time continuing to be effective for out-of-order cores with increasingly large ROB sizes.

The workloads targeted by DVR are not only challenging, they also greatly matter: many emerging workloads from graph applications to machine learning demand a tremendous performance boost from today's underlying hardware. Application-specific hardware, i.e., accelerators, have undoubtedly been a dominant focus in the past decade. However, they are still a 'part' of the chip as the simplicity and programmability of general-purpose processors is difficult to outshine. DVR offers a tempting and tremendous performance improvement for graph workloads on today's general-purpose CPUs.

At the same time, DVR is simple to implement. Its extreme memory-level parallelism ability through vectorization obviates the need for out-of-order execution

capabilities in the runahead subthread. In fact, the decoupled vector-runahead subthread running alongside the main execution thread context executes in program order while sharing the vector units with the main core. We hence believe that DVR has the potential to be deployed beyond the aggressive out-of-order cores with large ROBs as considered in this work, everywhere in the device stack: from smartphones to high-end servers. Indeed, as we have demonstrated in this work, DVR is equally effective for small-ROB architectures as it is for large-ROB architectures. We want to go even further than that: the form of parallelism we have discovered (i.e., parallel chains of dependent loads exposed through speculative vectorization) is so simple, that there is no reason to believe that it cannot be repurposed in pure in-order cores, where resources are even tighter and parallelism is even harder to find.

The extremely high performance of DVR is only possible because of a combination of novel analysis techniques that are likely to bear fruit in many other scenarios. In particular, DVR ends up discovering complex properties of loop control-flow in order to look far into the future. Questions about the fusion of techniques such as branch runahead[11] and vector runahead become inevitable. This potentially opens up new optimization goals other than memory-level parallelism, bringing it into the standard set of tools architects use to extract new forms of performance. Indeed, what further classes of applications might it be able to accelerate, with a wider definition of induction variables than the one currently inferred by looking at sequential stride loads? It is likely that the broad approach can pay dividends by marching ahead into many kinds of diverse loop structures.

Note further that DVR is also a form of speculative parallelism, albeit one that receives significant gains from not having to obey any correctness guarantees with respect to data dependencies or ordering. Indeed, its primary targets are workloads that modern vectorizers in optimizing compilers will not touch. However, many insights from DVR bring about a new generation of mechanisms that are able to speculate and roll back on data dependencies, able to parallelize compute as well as memory accesses, all while achieving orders of magnitude higher MLP than today's basic-block-scale vectorizers.

Conclusion

Decoupled Vector Runahead offloads the runahead execution to a simple, in-order, SIMT, vector subthread that is initiated whenever the core detects an indirect memory access pattern. Unlike prior runahead techniques, DVR does not wait for the reorder buffer to stall, and by discovering the loop bound at run-time, it can adjust the degree of vectorization to better suit application characteristics. DVR generates prefetches from multiple invocations of a loop when the discovered degree of vectorization for one invocation is not sufficient to achieve high memory-level parallelism. DVR incurs minimal hardware overhead while delivering a substantial performance boost for some of the most challenging graph workloads.

The benefits of reordering-based runahead over invalidation runaheads will usher in a new era of processors with the latency insensitivity of GPUs while maintaining the programmability and single-threaded performance of CPUs. In an era where single-threaded performance is so difficult to enhance, we suspect that a radical yet easy-to-implement and application-transparent solution like DVR holds significant potential.

Acknowledgments

This work is supported in part by the UGent-BOF-GOA grant No. 01G01421, the Research Foundation Flanders (FWO) grant No. G018722N, the European Research Council (ERC) Advanced Grant agreement No. 741097, and the Engineering and Physical Sciences Research Council (EPSRC) grant reference EP/W00576X/1. Additional data related to this publication is available on request from the lead author.

REFERENCES

1. A. Naithani, S. Ainsworth, T. M. Jones, and L. Eeckhout, "Vector runahead," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA '21. Los Alamitos, CA, USA: IEEE Computer Society, 2021, p. 195–208. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00024
2. J. Dundas and T. Mudge, "Improving data cache performance by pre-executing instructions under a cache miss," in *Proceedings of the 11th International Conference on Supercomputing*, ser. ICS '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 68–75. [Online]. Available: https://doi.org/10.1145/263580.263597
3. O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead execution: An alternative to very large instruction windows for out-of-order processors," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.* Los Alamitos, CA, USA:

IEEE Computer Society, Feb 2003, pp. 129–140. [Online]. Available: https://doi.org/10.1109/HPCA.2003.1183532
4. O. Mutlu, H. Kim, and Y. N. Patt, "Techniques for efficient processing in runahead execution engines," in *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, ser. ISCA '05. Los Alamitos, CA, USA: IEEE Computer Society, 2005, p. 370–381. [Online]. Available: https://doi.org/10.1109/ISCA.2005.49
5. M. Hashemi and Y. N. Patt, "Filtered runahead execution with a runahead buffer," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: Association for Computing Machinery, 2015, p. 358–369. [Online]. Available: https://doi.org/10.1145/2830772.2830812
6. A. Naithani, J. Feliu, A. Adileh, and L. Eeckhout, "Precise runahead execution," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, Feb 2020, pp. 397–410. [Online]. Available: https://doi.org/10.1109/HPCA47549.2020.00040
7. E. Rotem, A. Yoaz, L. Rappoport, S. J. Robinson, J. Y. Mandelblat, A. Gihon, E. Weissmann, R. Chabukswar, V. Basin, R. Fenger, M. Gupta, and A. Yasin, "Intel Alder Lake CPU architectures," *IEEE Micro*, vol. 42, no. 3, pp. 13–19, 2022. [Online]. Available: https://doi.org/10.1109/MM.2022.3164338
8. D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, ser. ISCA '95. New York, NY, USA: Association for Computing Machinery, 1995, p. 392–403. [Online]. Available: https://doi.org/10.1145/223982.224449
9. R. S. Chappell, J. Stark, S. P. Kim, S. K. Reinhardt, and Y. N. Patt, "Simultaneous subordinate microthreading (SSMT)," in *Proceedings of the 26th Annual International Symposium on Computer Architecture*, ser. ISCA '99. Los Alamitos, CA, USA: IEEE Computer Society, 1999, p. 186–195. [Online]. Available: https://doi.org/10.1145/300979.300995
10. A. Naithani, J. Roelandts, S. Ainsworth, T. M. Jones, and L. Eeckhout, "Decoupled vector runahead," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 17–31. [Online]. Available: https://doi.org/10.1145/3613424.3614255
11. S. Pruett and Y. Patt, "Branch runahead: An alternative to branch prediction for impossible to predict branches," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 804–815. [Online]. Available: https://doi.org/10.1145/3466752.3480053

Ajeya Naithani is a postdoctoral researcher at Ghent University, Belgium. His research interests are in the area of computer architecture with an emphasis on designing novel techniques to improve performance, energy-efficiency, and reliability of modern processors. He received the PhD degree in computer science engineering from Ghent University in 2019. Contact him at ajeya.naithani@ugent.be.

Jaime Roelandts is a PhD student at Ghent University, Belgium. His research interests include computer architecture with an emphasis on simulation and graph processing. He received the MSc degree in computer science engineering from Ghent University in 2020. Contact him at jaime.roelandts@ugent.be.

Sam Ainsworth is a research consultant working in industry, and a Visitor at the University of Edinburgh, where he supervises two PhD students. His research looks at runtime, systems and hardware security, along with architectural and compiler techniques for data prefetching in software and hardware, and efficient techniques for hardware error detection and correction. He received a PhD in Computer Science from the University of Cambridge in 2018. Contact him at sam.ainsworth@ed.ac.uk.

Timothy M. Jones is a Full Professor in Computer Architecture and Compilation at the University of Cambridge. His research interests span compiler and microarchitectural schemes for performance, reliability and security, especially focused on tackling challenges using different forms of parallelism. He received a PhD in Informatics from the University of Edinburgh in 2006. Contact him at timothy.jones@cl.cam.ac.uk.

Lieven Eeckhout is a Full Professor at Ghent University, Belgium. His research interests include computer architecture performance analysis and modeling, CPU/GPU microarchitecture and resource management, and sustainability. He received a PhD degree in computer science engineering from Ghent University in

2002. He is an IEEE and ACM Fellow. Contact him at lieven.eeckhout@ugent.be.