Toward Practical 128-bit General Purpose Microarchitectures

Chandana S. Deshpande, Arthur Perais, Frédéric Pétrot

Toward Practical 128-bit General Purpose Microarchitectures

Chandana S. Deshpande, Arthur Perais and Frédéric Pétrot

***Abstract*—Intel introduced 5-level paging mode to support 57-bit virtual address space in 2017. This, coupled to paradigms where backup storage can be accessed through load and store instructions (e.g., non volatile memories), lets us envision a future in which a 64-bit address space has become insufficient. In that event, the straightforward solution would be to adopt a flat 128-bit address space. In this early stage letter, we conduct high-level experiments that lead us to suggest a possible general-purpose processor micro-architecture providing 128-bit support with limited hardware cost.**

***Index Terms*—128-bit microprocessors, microarchitecture, clustered microarchitectures, region based compression**

I. Introduction

THE virtual address (VA) space has steadily grown over the past decades. For instance, x86 extended VA width from 48 to 57 bits in 2017 [1]. Moreover, there is already a signal from industry that we may run out of 64-bit addresses within two decades in specific use cases (e.g., 8 EiB –2^{63}– files by 2040 [2]). In that event, the most straightforward strategy would be to double the VA width again, to 128 bits [3]. Such a change would impact the whole stack, from compilers and operating systems to processor cores. In this context, it is necessary to start stirring the pot as early as possible, as the band-aids (e.g., Physical Address Extensions) that were added during the 32- to 64-bit transition phase turned out to be burdens that disappeared as soon as it became possible to remove them. In this letter, we focus on microarchitecture and propose a 128-bit processor microarchitecture, in a context where Dennard scaling and Moore's Law cannot be entirely trusted to absorb hardware cost [4]. In particular, naively scaling datapaths and structures is bound to increase area, latency, and power consumption. A "business as usual" approach to 128-bit microarchitecture is therefore likely to incur a performance penalty. Under the high-level assumption that integer width used in programs need not increase with the width of VAs, we propose to divide and conquer hardware complexity using clustering. Specifically, we architect the processor back-end around a 128-bit cluster mostly dedicated to address computations, while general purpose integer computations remain performed on a distinct 64-bit cluster.

Manuscript received 16 May 2023; revised 7 June 2023; accepted 15 June 2023. This work was supported by ANR project Maplurinum under Grant ANR21-CE25-0016. Chandana S. Deshpande, Arthur Perais and Frédéric Pétrot are with the TIMA, CNRS, Grenoble INP, University Grenoble Alpes, 38000 Grenoble, France (e-mail: chandana.deshpande@univ-grenoble-alpes.fr; arthur. perais@univ-grenoble-alpes.fr; frederic.petrot@univ-grenoble-alpes.fr). Digital Object Identifier 10.1109/LCA.2023.3287762

II. Naive 128-bit Microarchitecture

Despite opening access to more memory, naively supporting 128-bit VAs in a modern microarchitecture [5], [6] by doubling datapath width and the size of specific structures will incur significant area, power and latency costs.

In *Fetch*, the iTLB tags (virtual page number, VPN) and data (physical page number, PPN) as well as I-Cache tags (PPN or VPN for VI* and PI* caches respectively) almost double in size. The BTB and indirect branch predictor data (VAs) suffer the same increase. BTB, direction and indirect branch predictor tags may not grow as full tags are not required. Nonetheless, iTLB, I-Cache, BTB and predictors area, power, and latency are impacted. In *Decode*, the predicted targets of direct branches are checked against the targets computed from the instruction bytes. Both comparison and computation now require 128-bit operators, whose latency scales logarithmically with operand width. *Register Rename* is not affected as the stage only manipulates register names, whose width depends on the number of registers, not their width. *Dispatch* is similarly not affected, as no operand value or address is manipulated. The exception is instruction PCs that are inserted in the Reorder Buffer (ROB) or a microarchitecture-specific FIFO to handle pipeline flushes. From the *Scheduler*, instructions are scheduled for execution if operands and a functional unit (FU) are available. Since operand values are not stored in the scheduler, the stage is not affected. However, once scheduled, the instructions read operands from either the *Physical Register File* (PRF) or *bypass network*. Their area, power and latency increase as 128-bit values are now stored and routed. The logic determining whether a value should be taken from the bypass network or the PRF is unchanged (comparison on the register name), and the muxes depth does not change, although the muxes are now wider. Broadly speaking, the latency of common *functional units* (ALU, shift, multiplication, division) grows linearly or better with operand width, while the area grows linearly or worse. Finally, on the data access side, dTLB and D-Cache suffer the same area, power and latency increase as the iTLB and I-Cache. Moreover, the Load/Store Queue area, power and latency increase as the structure is responsible for determining memory *read-after-write* hazards by comparing physical addresses (PAs) of incoming load/stores with PAs stored in LSQ entries. Since modern LSQ support 128-bit accesses for SIMD instructions, the data fields of LSQ entries remain unchanged.

To summarize, in the front-end, the transition to 128-bit mostly impacts caching structures, as the front-end does not deal with actual operand values. The exception is any control

flow speculation or checking logic (e.g., next PC selection, direct target check in Decode). In the back-end, both storage (e.g., PRF, D-Cache) and logic (e.g., functional units, bypass network) are affected. Depending on the specific design, this could impact frequency, e.g., if the PRF read is the critical path. Combined with the increased footprint of pointers in memory, this transition will probably decrease overall performance.

The virtual memory paging structures may also be impacted. However, this would not impact first level TLBs who would still map a VPN to a PPN regardless of the page table structure. As a result, this letter does not discuss changes to the virtual memory structures themselves. Rather, our objective is to address part of the hardware overhead incurred by a 128-bit transition.

III. A Practical 128-bit Microarchitecture

Implemented VA width will not initially be 128-bit, just as currently implemented VAs are not yet 64-bit [1]. This significantly limits the increase in tag size in, e.g. TLBs. Moreover, techniques such as region-based compression [7] can be leveraged to mitigate the increase in tag and/or data arrays of relevant structures. The idea is to map high-order bits to region identifiers (RIDs) such that structures store the lower bits and an RID, while a small table stores the higher bits mapped to each identifier. Since the tags (caches, TLBs, BTB, predictors) and –part of the– data (TLBs, BTB, LSQ, ROB) of the structures are addresses, they exhibit high locality. Therefore, given a sufficiently large region size, only a handful of RIDs (i.e., mapping table entries) are necessary to minimize RID reclamation, which is a costly operation. In the back-end, the same scheme can be applied to the PRF, with lower efficiency. Indeed, the PRF contains both addresses and plain integers, which increases value entropy and implies more RIDs to minimize RID reclamations. Nevertheless, we note that despite the transition from 32- to 64-bit, some computations are still performed on 32-bit in current programs. In fact, the typical width of a C *int* is 32-bit. Moreover, although a 32-bit unsigned loop counter might not be enough to walk through a large structure, a 64-bit counter can cover any structure up to 16 Ei-entries. As a result, we hypothesize that future 128-bit programs will only use 128-bit instructions for address manipulation, as would be the case if an existing code source were taken and compiled for a 128-bit machine today. Under that assumption, we propose to divide and conquer the microarchitecture complexity using *clustering*.

In a nutshell, clustering partitions critical parts of the microarchitecture (scheduler, PRF, FUs, bypass network) into *clusters* and steers instructions to respective clusters using specific criteria. A steering mechanism is responsible for distributing instructions [8]. Modern processors already implement integer and floating point clusters [5], [6] and naturally steer instructions to clusters based on their type. However, address calculation is performed in the integer cluster: Addresses are treated as integers.

In this letter, we propose to implement a 128-bit cluster dedicated to executing instructions that manipulate addresses (loads, stores, indirect branches and all producers), and a 64-bit cluster for other integer instructions. As a result, most values in the 128-bit PRF will be addresses, enabling highly efficient region-based compression of the structure.

A. A Clustered 128-bit Address/Value Microarchitecture

This microarchitecture implements three clusters with dynamic steering: Address (A, 128 bits), integer (I, 64 bits), and FP/SIMD (FP, implementation dependent). The A and I clusters each maintain their own PRFs (128- and 64-bit respectively), and communication is transparent to the programmer. Each integer rename map table entry is augmented with a bit to determine if a value resides in the A or I cluster. If the steering policy finds that an instruction it is sending to the A cluster requires a value currently living in the I cluster, it will inject a *copy* μ-op in the pipeline to perform the copy, using an execution unit in the I cluster and dedicated cluster-to-cluster wires, incurring latency [9]. Note that steering only applies to instructions that are not explicitly 128-bit (e.g., 64-bit or 32-bit arithmetic operations). Instructions that are explicitly 128-bit (e.g., 128-bit arithmetic operations) are trivially steered towards the 128-bit cluster. Depending on the percentage of instructions to steer to the 128-bit cluster as well as their nature (e.g., addition vs. multiplication), the hardware footprint of the A cluster may vary.

In the next section, we quantify the number of instructions that would be steered to the A cluster. We further highlight that region-based compression is indeed an adapted tool to mitigate the area increase of microarchitectural structures.

B. Experimental Framework

As RISC-V already features material for a 128-bit extension [3], we use the functional simulator Spike [10] to study metrics in Polybench [11] (*standard* inputs) and SPEC CPU 2017 speed [12] (*ref* inputs). *omnetpp*, *xalancbmk*, *pop2* and *fotonik3d* are excluded due to lack of system support by the proxy kernel provided with Spike.

For SPEC CPU 2017, we skip the first 100 B instructions to avoid the initialization phase of the workload, then run 10 B instructions. For Polybench, we run the first 10 B instructions. We analyze only 10 B because our first experiment maintains an arbitrarily large dependency graph of the execution in memory to gather statistics of interest. We assume that the current benchmarks are representative enough to draw meaningful conclusions for our experiments, as the nature of the future 128-bit workloads is still hypothetical.

C. Experiments and Results

1) Quantifying Address Calculating Instructions: The backward address slice (BAS) of address generating (AGEN) instructions (loads, stores, and indirect branches) is a directed graph with instruction nodes and register data dependency edges. The last node is the AGEN instruction, and all previous nodes are direct or indirect producers. To identify instructions on BASes, which would be steered to the address cluster, we build the dynamic register data dependency graph at runtime

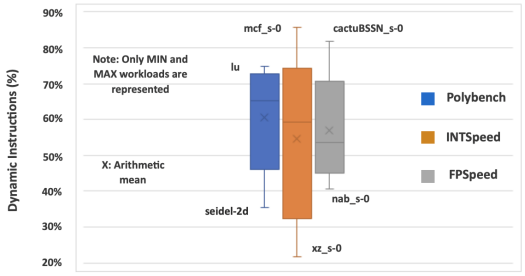

Fig. 1: Percentage of dynamic instructions on (at least) a BAS.

and walk it backwards for each AGEN instruction. The walk stops when an instruction does not have producers (e.g., *load immediate*) or when another load instruction is reached, as we separately build its own BAS. A comparable algorithm can be implemented in hardware to iteratively build BASes and drive the steering decision for instructions that are not explicitly 128-bit but are indirect or direct producers of AGEN instructions [13]. Figure 1 shows that around 55% (INPSpeed), 58%(FPSpeed) and 63% (Polybench) of the dynamic instructions are on a BAS for a global average of 59%. Figure 2 further depicts that most non AGEN instructions on BASes are ALU or shift instructions. The number of *div* instructions on BASes is negligible, suggesting that a microcoded or purely software implementation of 128-bit division may be acceptable. The number of *mul* instructions is significant in a handful of cases: *deriche, deepsjeng, exchange2* and *nab*. For instance, 79 M dynamic *mul* instructions are on a BAS in *deepsjeng*, with 43% of them in a single function, *FindFirstRemove* (deepsjeng/bits.cpp:49-53). As a result, a hardware 128-bit multiplier may be required to guarantee good performance in the address cluster. We also found that on average, 45%, 17% and 1% of the dynamic instructions are on load, store and indirect branches BASes. However, the ratio of the sum of dynamic instructions on the different BAS types to the number of dynamic instructions on BASes is on average 1.11, indicating that a moderate amount of dynamic instructions are on two or more BAS types. Despite implementing vastly different algorithms (especially SPEC), most workloads have a reasonably balanced utilization of the address and integer clusters. As a consequence, the address cluster will in fact require enough resources and especially functional units to deliver high throughput, although complex operations can be pushed to software to save area.

2) Region-based Compression: In addition to functional units, the address cluster will require a large enough instruction window to allow many instructions in flight in the cluster. This includes physical registers, which have now doubled in area, increasing access time as a side effect.

Fortunately, most values in the address cluster registers will be addresses. As a result, and as discussed in Section III, we can leverage region-based compression to not only compress tags and or data in caches, predictors and TLBs, but also greatly reduce the size of the physical registers in the address cluster.

For the PRF, a 128-bit datum is compressed before being written back to the PRF by matching the upper bits to an RID in the mapping table (associative search) and concatenating it with the lower bits. A compressed value is uncompressed after being read from the PRF by retrieving the upper bits corresponding to the RID in the mapping table (indexed read) and concatenating them to the lower bits of the compressed value.

For caches, BTB and predictors, similar operations are needed. For tag matches, the tag array is read in parallel to the mapping table being associatively searched with the VPN or PPN of the address of interest, and only the lower bits of the tag and RIDs are actually compared. Writing a new tag to the array also requires an associative search. For (VI/PI)PT instruction and data caches, a single mapping table associative search is needed to perform the TLB tag match followed by the cache tag match since the PPN is already compressed in the TLB data array. However, it is likely that two mapping tables will be required, one for PAs and one for VAs. Each compressed structure may implement its own table.

For the PRF, however, enough RIDs must be available to represent a valid architectural state. This means that we need at least 32 RIDs in case i) All architectural registers are currently in the address cluster PRF and ii) All architectural registers have different upper bits. This requirement will impact the timing of the mapping table, which is on the critical path of register reads (indexed read) and writes (associative search). An RID is reclaimed from the mapping table when a value that cannot be compressed using existing mappings is written to the PRF (or another structure that uses that mapping table). Reclaiming an RID requires ensuring that all the physical registers using that RID are unreachable. This can be done non-speculatively by waiting for the problematic instruction to reach Commit, flushing the pipeline, then reclaiming any RID not currently used by architectural registers living in the address cluster. If all RIDs are used, then each architectural register uses its own RID. Thus, the RID used by the architectural register being written can be reclaimed safely. While slow, this process is straightforward and the region size can be chosen such that reclamation is rare, at the cost of slightly less efficient compression. RID reclamation caused by ROB (PCs) and LSQ (PAs) would be handled similarly by waiting for the pipeline to drain and reclaiming an RID. For other structures (caches, TLBs, BTB), reclaiming an RID implies invalidating all the tags or data using that RID, which can be done non speculatively in an iterative fashion.

To determine what a good region size would be, we sample the number of unique regions being accessed every 40 M instructions (1 IPC @ 4GHz during 10 ms) for region sizes from 116 bits (4 KB region offset) to 92 bits (64 GB region offset), using only SPEC CPU 2017. We excluded Polybench from this experiment because the control flow is highly regular and prevents access to many unique memory regions within a sample. Figure 3 depicts the average number of accessed regions per sample, across all 40 M samples, for each region size, for each workload. The Figure confirms the intuition that as the region size grows, the number of regions accessed within an epoch increases. In this experiment, using a region size of 102 upper bits (64 MB region offset) yields fewer than 32 regions tracked within an epoch, on average (19 and 13

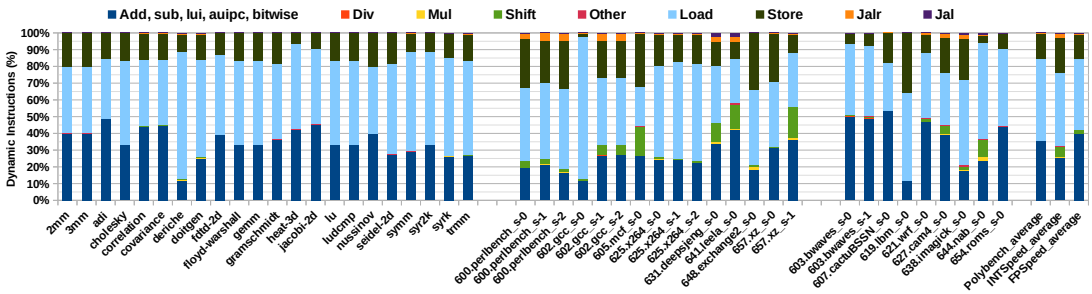

Fig. 2: Dynamic distribution of instruction type on BAS (Groups from L to R: Polybench, INTSpeed, FPSpeed)

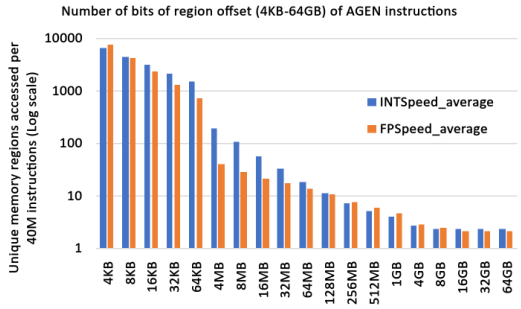

Fig. 3: Average unique memory regions accessed over 40M instructions, per region size, per workload suite. Logscale.

regions for INTSpeed and FPSpeed). Using this region size would save 66% of the PRF storage if a 256-entry 128-bit PRF and a 32-entry RID table are considered. Nevertheless, future machines running workloads with large datasets could want to consider smaller regions to ensure region identifier reclamation remains rare. For instance, using a region size of 94 upper bits (16 GB region offset) would still save 60% of the PRF storage.

IV. Related Work

State of the art clustered microarchitectures can be categorised as homogeneous or heterogeneous. In the former, an instruction may be executed by multiple identical clusters [14], and each cluster maintains a copy of the physical register file. Conversely, in heterogeneous clustering, instructions have to be executed in a specific cluster [5], [6], each featuring their own disjoint register files. Our proposal is a form of heterogeneous clustering, although some instructions can technically be executed on both A and I clusters (e.g. 64-bit *add* initially executing on I cluster before it is determined to belong to a BAS). However, in existing heterogeneous designs, transferring values between clusters is done through memory using loads and stores or directly using dedicated instructions [3], as the clusters usually own different portion of the architectural registers (e.g., integer vs. FP). Since both A and I clusters own the integer architectural registers, value transfer between clusters is done implicitly by the hardware. Steering can be performed dynamically in hardware or statically by the compiler. The former does not require ISA change and generally performs better as the compiler does not have all the dynamic latency information that hardware does [8], [9], [14]. Since out-of-order machines are notorious for their latency variability, dynamic steering appears the natural choice. Regarding 128-bit machines, only RISC-V defines a flat 128-bit general purpose extension [3]. However, no actual 128-bit microarchitectures implementing this extension ISA have been proposed.

V. Conclusion and Future Work

This letter proposes an efficient 128-bit microarchitecture. Instructions are steered to the dedicated 128-bit address cluster by iteratively learning [13] the backward address slices (BAS) of address generating instructions. Other 64- and 32-bit instructions are steered to the cheaper 64-bit cluster. This letter showed that this would allow around 41% of the dynamic instructions to not consume 128-bit resources, on average, under the hypothesis that non address related integer computations would remain performed on 32 or 64 bits when compiling for a 128-bit machine. We further suggest to leverage an existing region-based compression scheme to significantly reduce the footprint of many common structures, including the 128-bit physical register file in the address cluster.

References

[1] Intel Corporation, "5-Level Paging and 5-Level EPT," https://www.intel.com/content/www/us/en/content-details/671442/5-level-paging-and-5-level-ept-white-paper.html, pp. 1–31, 2018.

[2] M. Wilcox, "Zettalinux: It's not too late to start," in *Linux Plumbers Conference*, Sep. 2022.

[3] RISC-V Foundation, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2," https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf, pp. 1–145, 2017.

[4] J. L. Hennessy and D. A. Patterson, "A New Golden Age for Computer Architecture," *Comm. of the ACM*, vol. 62, no. 2, pp. 48–60, Jan. 2019.

[5] D. Suggs, M. Subramony, and D. Bouvier, "The amd "zen 2" processor," *IEEE Micro*, pp. 45–52, 2020.

[6] 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*, pp. 13–19, 2022.

[7] A. Seznec, "Don't use the page number, but a pointer to it," in *Intl. Symp. on Computer Architecture*, 1996, p. 104–113.

[8] P. Michaud, A. Mondelli, and A. Seznec, "Revisiting clustered microarchitecture for future superscalar cores: A case for wide issue clusters," *ACM Trans. on Arch. and Code Optim.*, pp. 1–22, 2015.

[9] R. Canal, J. Parcerisa, and A. Gonzalez, "Dynamic cluster assignment mechanisms," in *Intl. Symp. on High-Performance Computer Architecture*, 2000, pp. 133–142.

[10] RISC-V Foundation, "Spike RISC-V ISA Simulator," https://github.com/riscv-software-src/riscv-isa-sim/tree/a0298a33e7b2091ba8d9f3a20838d96dc1164cac, 2021.

[11] L.-N. Pouchet *et al.*, "Polybench: The polyhedral benchmark suite," *URL: http://www. cs. ucla. edu/pouchet/software/polybench*, 2012.

[12] Standard Performance Evaluation Corporation, "SPEC CPU," https://www.spec.org/cpu2017/, 2017.
[13] T. E. Carlson, W. Heirman, O. Allam, S. Kaxiras, and L. Eeckhout, "The load slice core microarchitecture," in *Intl. Symp. on Computer Architecture*, 2015, pp. 272–284.
[14] R. E. Kessler, "The alpha 21264 microprocessor," *IEEE micro*, pp. 24–36, 1999.