VaLHALLA: Variable Latency History Aware Local-carry Lazy Adder

Ali Murat Gok
Electrical Engineering and Computer Science Dept.,
Northwestern University, Evanston, IL 60208, USA
amg@u.northwestern.edu

Nikos Hardavellas
Electrical Engineering and Computer Science Dept.,
Northwestern University, Evanston, IL 60208, USA
nikos@northwestern.edu

ABSTRACT

Current low-energy adder designs fail to address subtraction or signed operations, are imprecise, or have large overheads. These overheads include overlapping sub-adders, relatively complex carry speculation mechanisms and bulky error correction circuits, all of which consume considerable amount of energy. In this paper, we address all these problems and introduce Variable Latency History Aware Local-carry Lazy Adder (VaLHALLA). VaLHALLA employs sliced non-overlapping sub-adders and speculates the carry-ins for each slice by exploiting the temporal operand correlation of each instruction location (i.e., it considers both operand history and instruction locale). VaLHALLA achieves accurate results at lower energy-delay-product than competing designs.

Keywords

Power efficiency; Variable latency adders; Speculative adders

1. INTRODUCTION

The breakdown of Dennard scaling [6] and the advent of dark silicon [8] have elevated power consumption to a first-class design constraint [13]. Several techniques have been proposed to lower the power consumption, including dynamic voltage and frequency scaling [17], near-threshold computing, sleep states [5], approximate computing [7] and specialized cores (accelerators) [4]. In the quest to achieve higher power and energy efficiency, a significant amount of recent work focuses directly on decreasing the power consumption of functional units, and in particular, adders [10,16].

One of the most common methods is to slice the full bit range of an adder into smaller bit ranges ("slices") and run them in parallel. This architecture provides two potential benefits: (a) a reduction in the total number of gates which leads to lower power consumption, due to the superlinear relationship between bitwidth and hardware complexity, and (b) running smaller slices in parallel decreases the total delay of the circuit. This reduced delay can then be utilized to decrease power consumption by scaling down each slice's supply voltage to the lowest setting that allows the slice to still fit within the same cycle time [14]. However, running the slices in parallel generates complications in carry propagation, as the slices break the carry chain. The natural solution is to supply each slice with a speculative carry-in. Slice organization, carry propagation and the carry generation mechanisms for each slice constitute the main differences between current low-power adders.

Most low-power adders are categorized into two groups: approximate and variable latency speculative adders. The main difference between the two groups is on how mispredictions of carry-ins are handled. Approximate adders [10] finish the operation in a single cycle. However they lead to inaccuracies since they do not possess error correction mechanisms and wrong results are supplied whenever a carry-in is mispredicted. The main focus of approximate adders is to increase the accuracy in the output while constraining the execution to a single cycle. In contrast to approximate adders, variable latency speculative adders [16] always provide the correct output in one or two cycles. If all slices' carry-ins are speculated correctly, the operation completes in a single cycle. Otherwise, a carry misprediction is detected and a second cycle is employed to correct the output. Variable latency speculative adders focus on decreasing the number of two-cycle operations by increasing the accuracy of the carry-in speculation mechanism.

Unfortunately, the majority of previously proposed low-power adders only support unsigned addition and disregard signed operations or subtraction [9], and hence cover only a small portion of a real workload's instructions. In addition, even the most sophisticated prior attempts to increase the accuracy of carry speculation fall short of delivering on their promise. Some of the previous techniques exploit the correlation of arithmetic operations in real workloads (i.e., the temporal correlation of add instruction results) and use the prior operation to predict the carry-ins for the current one. However, consecutive add operations in the dynamic instruction stream interleave adds from different program locations, and are much less correlated than consecutive add operations from the same line of code. We contend that to improve the accuracy of carry-in speculation, low-power adders should exploit the correlation of consecutive invocations of the same static instruction.

In this paper we address both shortcomings of existing designs and propose VaLHALLA (Variable Latency History Aware Local-carry Lazy Adder), a variable latency adder design that:

- Exploits for the first time (to the best of our knowledge) spatial and temporal PC-based correlation in a low-power adder.
- Provides full support for real-world workloads (64-bit signed and unsigned addition and subtraction).
- Exhibits lower power and energy consumption and lower energy-delay product compared to the current state of the art.

GLSVLSI '17, May 10 - 12, 2017, Banff, AB, Canada

ACM 978-1-4503-4972-7/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3060403.3060437

This work was supported by NSF awards CCF-1218768 and CCF-1453853

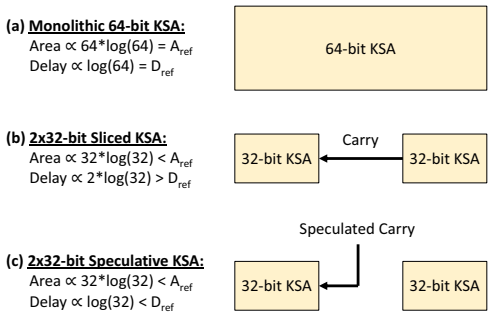

Figure 1. Hardware and delay scaling for KSA.

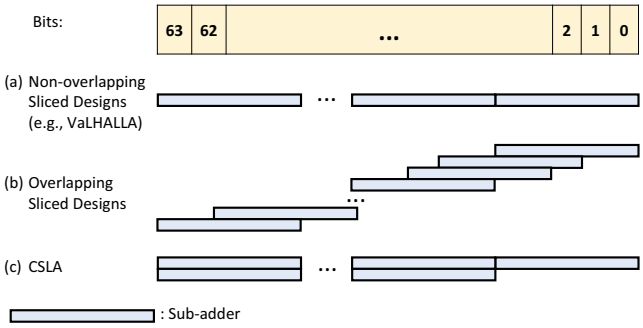

Figure 2. Sliced adder configurations.

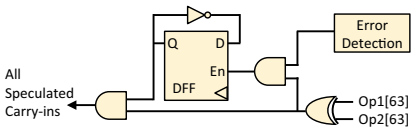

Figure 3. CASA [12] carry speculation technique.

2. BACKGROUND

2.1 Voltage Scaling

Voltage scaling (VS) exploits the trade-off between supply voltage and circuit delay [14] in circuit optimization. VS exchanges the existing slack in the timing budget for a reduction in the supply voltage to lower power consumption.

2.2 Hardware Scaling and Sliced Designs

Most traditional adder designs have a superlinear relationship between their bitwidth and hardware complexity. For example, the hardware complexity of a Kogge-Stone adder (KSA) scales in the order of $N \times logN$ and its delay scales in the order of $logN$, where N is the bitwidth of the adder. This illustrates a trade-off between hardware complexity and delay. As an example, it is possible to build a 64-bit adder from two 32-bit KSAs by concatenating one after another (Figure 1.b). This sliced approach decreases total area at the cost of increasing delay since the upper slice needs to wait for the lower one. On the other hand, if carry speculation is employed (Figure 1.c), both slices can run at the same time and attain both lower area and lower delay compared to the reference 64-bit KSA. However, speculated carries introduce inaccurate results when one or more of the carry-ins is speculated incorrectly.

Similar to KSA, most modern adders possess a similar trade-off between hardware complexity and delay, and face similar design choices in a sliced design. An example of a sliced adder configuration with non-overlapping slices is shown in Figure 2.a. Many of the current low-power adder designs employ slices with overlapping bit ranges (Figure 2.b), as "peeking" at the high-order bits of the lower slice may increase the accuracy of the carry speculation. However, this comes at the cost of employing more slices.

2.3 Carry Select Adder (CSLA)

The classical CSLA design [1] (Figure 2.c) resembles current low-power adders with its sliced design. However, CSLA slices do not overlap in terms of their bit ranges. Every slice in CSLA except the lowest one employs two sub-adders, one for each possible carry-in. Each one of the two slices in a bit range computes assuming a different carry-in. When the lower slice finishes its computation and produces a carry-out, that carry-out is matched against the two carry-ins used by the higher-order slice to determine which one of the two results is the correct result for the computation at hand.

2.4 Correlation-Aware Speculative Adders

Correlation-Aware Speculative Adder (CASA) [12] is a state-of-the-art speculative adder based on ACA [10]. CASA employs a new carry speculation technique in which the carry-ins of all slices are speculated to be 1 if the operand signs differ, and 0 otherwise. To improve accuracy, CASA uses a DFF as a dynamic prediction mechanism. The output of this DFF is used as an enable signal for the sign difference (Figure 3). This way CASA exploits the correlation between two temporally consecutive ADD/SUB operations.

3. THE ENERGY BENEFITS OF SLICING

To determine the potential benefit of employing a sliced design, we perform a limit study on the energy consumption of adders with non-overlapping slices (Figure 2.a) by exploring the trade-offs between bitwidth, number of sub-adders, hardware complexity, circuit delay and carry speculation accuracy. We assume a 64-bit adder design with 1 (i.e., conventional), 2, 4, or 8 non-overlapping slices, where each slice employs one sub-adder. We synthesize sub-adders of different bitwidths (8, 16, 32, or 64 bits) using the Synopsys Design Compiler. We feed these sub-adders with random vectors as inputs and with speculative carry-ins, and report the energy consumption of the overall adder for each configuration in Figure 4. To decouple the impact of slicing from the choice of supporting circuit design, only the sub-adders are taken into consideration for the energy and delay analysis. The rest of the circuit (e.g., error detection or carry generation circuitry) is assumed to have no delay or energy consumption.

The smallest possible energy consumption occurs when all the speculated carries are the correct ones and every slice computes only once. This case is indicated as "Best Case", where only one sub-adder is utilized for each slice. On the other hand, the largest energy consumption occurs when all the speculated carries are the

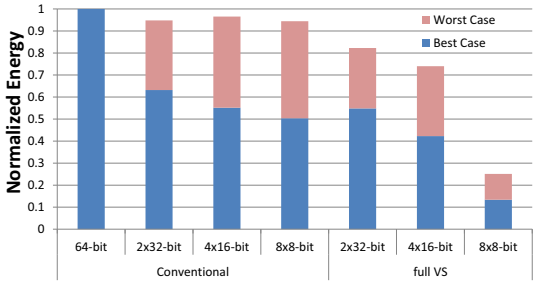

Figure 4. Limit study of the energy savings of sliced adders.

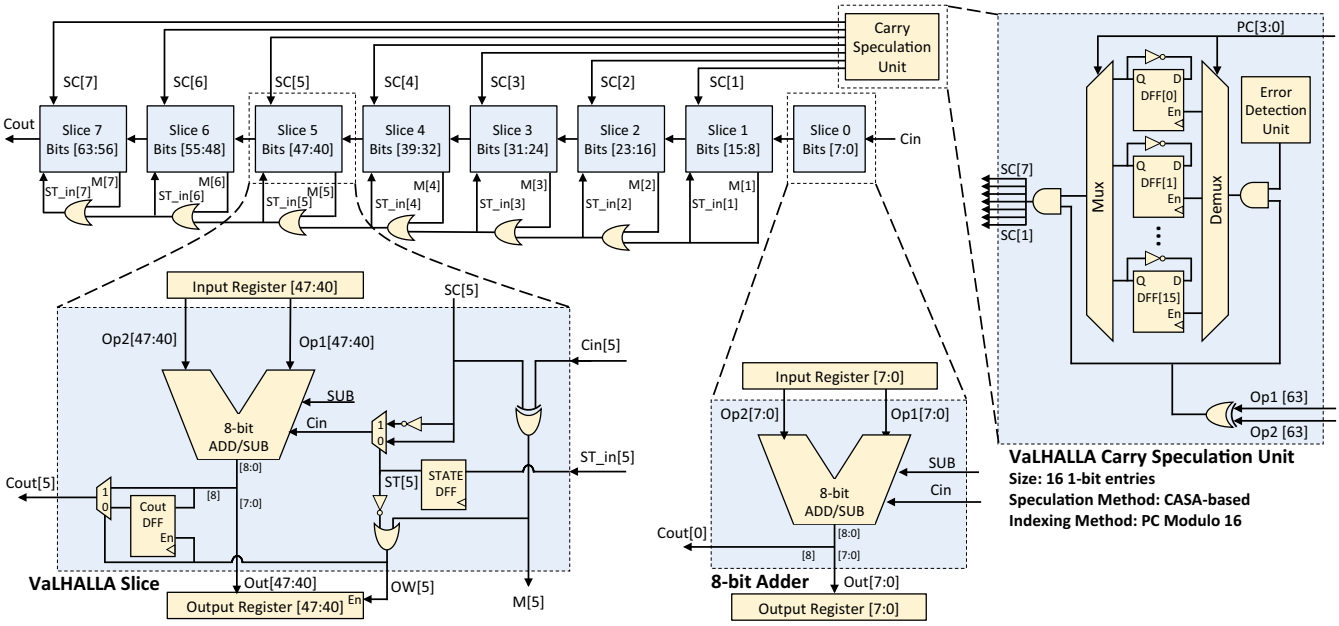

Figure 5. Example of a VaLHALLA adder design with 8x8-bit configuration. Slice 0 is a standard 8-bit adder, whereas all other slices are designed specifically for VaLHALLA. Slice 5 is shown in detail as an example of such slices.

wrong ones, and all sub-adders (except the lowest-order one) require an extra computation (this time with the correct carry-in). This case is indicated as "Worst Case". To investigate the potential for additional power savings, we apply voltage scaling to the sub-adders by decreasing their supply voltages until they have reached the same delay as the reference 64-bit single-slice adder. The resulting energy consumptions are given in Figure 4 as "Full VS".

It is possible to achieve significant power and energy savings by employing smaller sub-adders, as the actual power consumption will not necessarily be the worst case, but somewhere between the worst and best cases. The overall energy consumption directly depends on the carry-in speculation accuracy. Employing the minimum possible number of sub-adders to decrease the power consumption is one of the main ideas behind VaLHALLA, which is also a non-overlapping sliced design (Figure 2.a). As Figure 4 shows, employing voltage scaling with 2 or 4 slices shows a potential to save 18–58% of the adder energy per operation. Pushing to 8 slices, however, results in small enough sub-adders (just 8 bits) that allow the supply voltage to scale to 60% of the reference voltage, leading to 75–87% potential energy savings. While best-case savings are probably unachievable as only the sub-adders are considered in the analysis, they are however indicative of the high potential of sliced adder designs, and of VaLHALLA in particular.

4. VALHALLA

VaLHALLA is a CSLA-inspired design that employs multiple non-overlapping slices with only one sub-adder per slice. VaLHALLA performs an operation by computing for only one set of speculated carries in the first cycle of its operation and writing the results in its output register, optimistically expecting that no more work will be necessary (hence its designation as "lazy"). If additional work is needed to obtain the correct result, it is performed at an additional second cycle. Thus, VaLHALLA is a variable-latency "lazy" adder.

In most cases all carry-ins are speculated correctly and a second cycle of operation is not required. When a slice produces a carry-out at the end of the first cycle, VaLHALLA detects whether the next slice computed with the correct carry-in. If the carry-out of slice *k* does not match the carry-in used in slice *k+1* during the first cycle, VaLHALLA employs a second cycle during which slice *k+1* re-computes using the inverse of the previously speculated carry, i.e., the inverse of the carry-in that slice *k+1* used during the first cycle. At that time, however, it is unknown whether the carry-out produced by slice *k+1* at the end of the first cycle was the correct carry-in for the subsequent slice *k+2*. Thus VaLHALLA cannot determine whether slice *k+2* also needs to re-compute. In fact, a carry mismatch at slice *k+1* spreads towards the higher slices, causing all of them to conservatively require a second cycle of computation. At the end of the second cycle all correct carry-ins are known, which allows VaLHALLA to either keep the results already in the output register for a slice (i.e., the slice computation in the first cycle was the correct one) or overwrite them with the slice's result obtained at the second cycle of computation. Because only the necessary subset of slices re-computes (the slices higher than the first carry mismatch), VaLHALLA avoids unnecessary computations and exhibits significant power savings over CSLA. Figure 5 details the architecture of VaLHALLA.

Figure 6 illustrates a simple example where a 32-bit addition is performed using four 8-bit slices, where all of the carry-ins are speculated to be zeros. *SC[k]* denotes the speculated carry-in provided to slice *k*. *Cin[k]* denotes the carry-in used by slice *k* during that cycle. *Cout[k]* denotes the carry-out generated by slice *k* during that cycle. Because the carry-in of slice 0 is determined only by the type of the operation (addition or subtraction), it always gets the correct carry and computes correctly. At the end of the first cycle, there is no carry mismatch between *Cout[0]* and *Cin[1]*, also guaranteeing that slice 1 computed correctly. However, there is a mismatch between *Cout[1]* and *Cin[2]*, thus slice 2 in cycle 1 computed assuming the wrong carry in and should re-compute. Although there is no mismatch between *Cout[2]* and *Cin[3]*, slice 3 still needs the second cycle of computation because the correct-

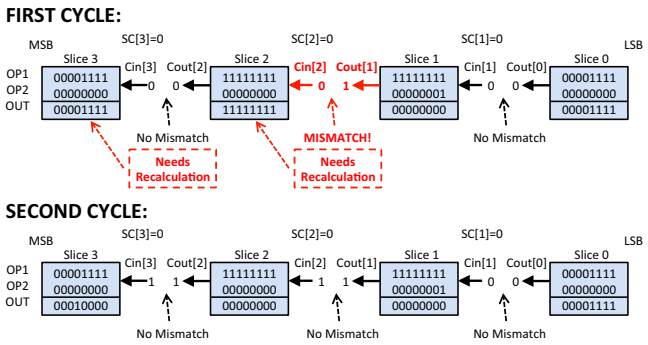

Figure 6. An example of VaLHALLA's 2-cycle operation

ness of *Cout[2]* cannot be determined. During the second cycle, slices 0 and 1 are idling because their first-cycle computation was already correct and the results are written in the adder's output register. Meanwhile, slices 2 and 3 re-compute using the inverse of their respective *SC*'s. At the end of the second cycle, slice 2 overwrites the output register with the newly-computed result and produces the **correct** *Cout[2]*. Similarly, *Cout[2]* and *Cin[3]* match in the second cycle, so slice 3's results from cycle 1 were erroneous. Thus, the output register is overwritten with the correct newly-computed value of slice 3, concluding the operation. If *Cout[2]* and *Cin[3]* had not match in the second cycle, slice 3's computation would have been deemed correct the first time and no further action (i.e., a second cycle for slice 3) would have been necessary.

4.1 VaLHALLA Slice Design

The design of VaLHALLA slices is illustrated in Figure 5. Each slice except the lowest-order one has a 1-bit DFF STATE register that keeps track of whether the slice is performing the first or the second cycle (STATE=0 or 1, respectively). When a new operation is assigned to the integer ALU, all STATE registers in it are reset to 0 to indicate that this is the first cycle of operation. After the first cycle, the STATE register of slice *i* is updated by OR-ing the mismatch signals of the current and all previous slices *i, i-1, ..., 1* and then stays at that value until a new operation is assigned:

$$\text{STATE}[i] \leftarrow M[i] \mid M[i-1] \mid \ldots \mid M[1] \qquad (1)$$

During the first cycle every slice is in state 0. Consequently, the speculated carry *SC* is used for computation and the output register is overwritten with the generated output. Every slice *i* generates its own mismatch signal *M[i]* by comparing the used carry-in *Cin[i]* with the carry-out of the lower slice *Cout[i-1]*. Then these mismatch signals are used to determine the states of the slices for the next cycle. If all mismatch signals are 0, then the operation completed correctly in a single cycle. If there is at least one mismatch signal set, then the second cycle is employed. During the second cycle, slices in state 0 wait idle, whereas slices in state 1 compute using the inverse of the carry-in used they used in the first cycle. At the end of the second cycle, correct carries gradually arrive from the lower to higher slices and VaLHALLA determines whether to overwrite the output register for each slice or not. To facilitate this operation, VaLHALLA retains the carry-out that a slice generates in the first cycle in a DFF, and also provides a direct path to propagate the carry-out from the sub-adder. This design ensures that the correct carries will arrive during the second cycle.

As Figure 4 indicates, employing a larger number of smaller slices increases the potential for energy savings. However, at the same time it increases energy and delay overheads for the rest of the circuit. More importantly, it increases the misprediction rate, as with more slices there is a higher chance that at least one carry-in speculation will be wrong. Thus, VaLHALLA strives to achieve high prediction accuracy for the speculated carry-ins.

4.2 Carry Speculation Unit

The Carry Speculation Unit provides speculated carries to VaLHALLA slices. To obtain the best VaLHALLA design we explored the design space for carry speculation in the following dimensions:

4.2.1 Carry Speculation Methods

We investigated four different carry speculation methods:

- *Perfect:* A limit study where correct carry-ins are generated at no energy or delay overheads.
- *Peek(k):* Each sub-adder is extended to *k* lower bits without changing its output range. Similar methods are frequently employed by other low-power adder designs. We implemented and evaluated *Peek(k)* for $k = 1, 2, 4, 8$. Increasing *k* decreases the misspeculation rate, but increases overheads.
- *Previous:* The carry-in of the current slice is the correct carry-out of the lower slice in the previous operation.
- *CASA-based*: Carry-ins are generated based on the difference between the operand signs, similar to CASA (Section 2.4).

4.2.2 Spatio-Temporal PC-Based Correlation

In real workloads, the dynamic instruction stream includes several instances of the same static instruction (i.e., an instruction at a specific PC). Consecutive operations stemming from the same line of code have a higher chance to have correlated operands and thus similar internal carries. For example, consider a static instruction which increments a loop iterator whose value ranges from 0 to 100, and we are using the Previous carry speculation method. All the dynamic instances executed from this static instruction will have their operands and outputs between 0 and 101, which means all the internal slice carry-ins in an 8-slice design and all higher operand bits will be zero. However, it is very unlikely that instances of this instruction will appear in the dynamic instruction stream consecutively without intervening add/subtract operations from other locations in the executing code. A single addition in the body of the loop would interleave with the increments of the loop iterator, and the operands of the loop-body instruction can be entirely outside the range of values used for the loop iterator. The Previous carry speculation method normally holds the carries of only the immediately preceding arithmetic operation observed by the functional unit, which may be irrelevant for the operation currently at hand.

VaLHALLA addresses this issue by using PC-based correlation. Instead of memory for a single carry bit per slice, VaLHALLA remembers an array of carries (Figure 5), each coming from the last invocation of a different static instruction. The register file that holds these carries for each slice is indexed using bits from the PC, allowing dynamic instructions that are instances of the same static instruction to interact with the same set of carry registers.

To the best of our knowledge, this is the first time that PC-based (spatial) correlation is used in a low-power adder design. VaLHALLA utilizes the history of carries generated by the same static instruction over time, hence it is a "history-aware" adder that predicts "local carry-ins" (i.e., carry-ins from the same instruction). Because the register file has finite size, it is possible that multiple static instructions map into the same register set (alias). Larger reg-

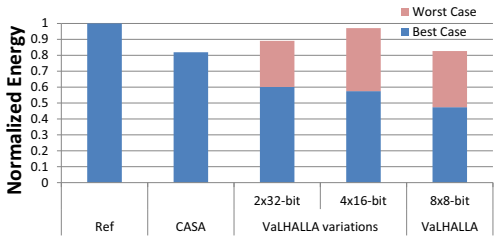

Figure 7. Potential for energy savings.

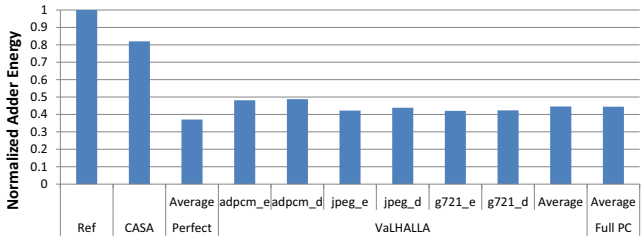

Figure 8. Normalized adder energy.

ister files would decrease aliasing. To arrive at an optimal design, we explored power-of-2 register file sizes from 2 to 128 entries per slice, as well as an infinitely-sized register file that removes all instruction aliasing with no latency or energy overheads.

4.2.3 PC-Based Register File Indexing Methods

To exploit PC-based correlation, the register file holding the carry-ins for a slice needs to be indexed by some combination of PC bits. Assuming a register file with 2^N entries, we evaluate the following two methods to index the register file:

- *PC Modulo:* index by *PC[N-1:0]*
- *PC Xor Hash:* index by *PC[N-1:0]* ⊕ *PC[2N-1:N]*

We explore the design space of VaLHALLA along all these dimensions to arrive at a near-optimal design.

5. METHODOLOGY

We model all adder designs (reference, CASA and VaLHALLA) in Verilog. We synthesize them using Synopsys Design Compiler with the same optimization parameters and the Synopsys SEAD 90nm library and obtain gate-level netlists. We convert these gate-level netlists into HSpice netlists and perform analog HSpice simulations of the resulting circuits to characterize their energy and delay. We characterize the performance of the adder designs by modeling them in gem5 [2] simulations that run a range of Mediabench [11] audio and video applications. Such media applications are among the most popular applications today [3]. CASA outperforms ACA and VLSA adders [12], thus we only compare VaLHALLA against the current state of the art, CASA.

Using this methodology, we perform a design space exploration across slice configurations, carry speculation methods, register file sizes and indexing methods, in search of the optimal VaLHALLA configuration. We simulate both in-order and out-of-order (OoO) x86 architectures with 6 integer ALUs. Space limitations prevent us from discussing the design space exploration in detail. The optimal configuration across our search space is a 64-bit VaLHALLA adder comprising 8x8-bit slices, CASA-based carry speculation, and a 4- or 16-entry register file (each register is 1 bit) for in-order or OoO x86 architectures, respectively. The register file in the optimal designs is indexed by the PC modulo method.

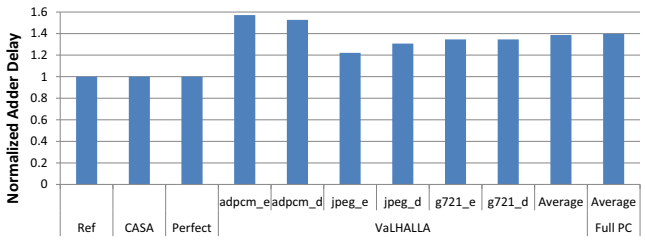

Figure 9. Normalized adder delay.

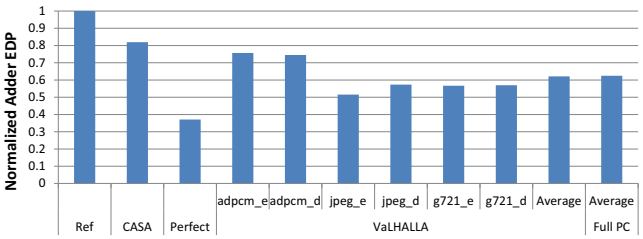

Figure 10. Normalized adder energy-delay product (EDP).

Because of its high relevance to modern processor designs, and due to space constraints, the rest of this paper discusses only the optimal VaLHALLA design for OoO processors. For readability, in the rest of the paper VaLHALLA refers to this optimal configuration.

6. EVALUATION

We demonstrate the potential energy savings of VaLHALLA in Figure 7. *Ref* represents the reference adder, which is a conventional 64-bit adder synthesized by Synopsys Design Compiler, whereas *CASA* represents a state-of-the-art carry-speculative approximate adder [12]. While the optimal VaLHALLA design employs eight 8-bit slices (Section 5), Figure 7 additionally shows two variations of VaLHALLA (2x32-bit and 4x16-bit slices) to demonstrate the exact benefits of the optimal VaLHALLA design over alternative slice bitwidths. The reference adder uses nominal supply voltage, whereas the other adders employ voltage scaling to decrease their supply voltages as much as possible as long as their delay does not exceed the reference adder's delay.

Best cases indicate the lowest possible energy consumption when the operation requires only one cycle to complete. Worst cases indicate the highest possible energy requirement when the second cycle is employed and all the slices (except the lowest one) need to re-compute for a second cycle. Energy consumptions of real workloads fall in between the best and the worst cases. High-accuracy carry speculation will bring the design closer to the best case.

In Figures 8, 9 and 10 we report normalized energy, delay and energy-delay product (EDP) for various adder types and applications. *Perfect* represents a limit study with an ideal carry speculation unit that always generates the correct carry-ins at zero energy cost. *Full PC* represents another limit study where the carry speculation unit is the same as in VaLHALLA, with the exception that it employs an ideal infinite-size register file indexed by the full PC that consumes zero power. Consequently it eliminates all aliasing among dynamic instructions, allowing them to enjoy the full effects of temporal correlation.

We find that temporal correlation alone (i.e., the previous carry speculation method) exhibits 53% average misprediction rate for an 8-sliced OoO design. A misprediction occurs when one or more

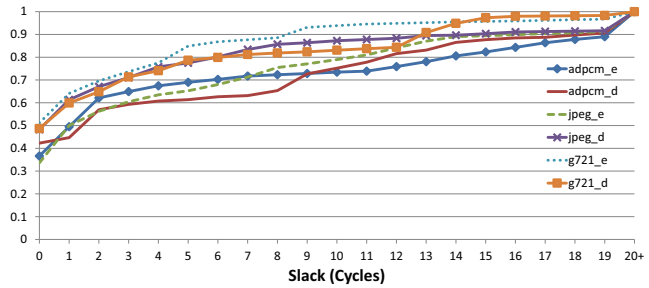

Figure 11. CDF of slack in consecutive ALU operations

carry-ins are mispredicted across all 8 slices (i.e., we measure not a per-slice misprediction rate but a per-ALU-operation one). In contrast, VaLHALLA's spatio-temporal PC-based correlation brings misprediction rate to 39% (1.4x reduction). For in-order designs, previous carry mispredicts 60% of the time, while VaLHALLA mispredicts only 35% (1.7x reduction). These results illustrate the accuracy potential of PC-based spatio-temporal correlation.

The area overhead of VaLHALLA consists of two 1-bit registers in each slice, 16 1-bit registers in the Carry Speculation Unit, a 16-to-1 mux/demux and a few combinational gates (Figure 5). The total area overhead over the reference design is 10.4%.

VaLHALLA consumes 55% less energy than Ref and 46% less energy than CASA (Figure 8), but mispredictions increase the average adder delay (i.e., the average number of cycles the adder is occupied per operation, Figure 9). However, the energy savings are significantly higher than the delay increase, which allows VaLHALLA to exhibit 24% and 38% lower average EDP than CASA and Ref respectively. When the full processor is considered and modeled using McPAT and gem5, including the execution overhead of VaLHALLA (1% for IO and 5% for OoO architectures), VaLHALLA saves 3x the energy saved by CASA. The total processor energy savings of VaLHALLA is 2% for the OoO processor we model, which employs only 6 ALUs. Applying VaLHALLA on ALU-intensive circuits (e.g., NVIDIA GTX-1080 Ti has 3584 ALUs) would yield significantly higher processor energy savings.

While the average adder delay can significantly increase due to misspeculated carries, the effect on the application runtime is drastically smaller because (a) processors typically employ multiple ALUs, and thus their execution delay overlaps, (b) only a fraction of the total executed instructions are additions or subtractions, and (c) modern OoO cores can overlap the additional delay with useful work if there is slack [15] between an ALU operation and the use of its result. We observe that 49–66% of ALU operations in our workload suite have at least 1 cycle of slack (Figure 11).

7. CONCLUSION

VaLHALLA is a variable-latency adder suitable for real-world applications that (unlike most other low-power adder designs) supports signed and unsigned addition and subtraction. VaLHALLA is a sliced adder with non-overlapping slices that employs only one sub-adder per slice, and exploits for the first time spatio-temporal PC-based correlation to increase the accuracy of carry-in speculations. We explored the design space of VaLHALLA and arrived at an optimized design that consumes 46% less energy per operation on average than the current state of the art (CASA), leading to 24% lower EDP. Compared to a conventional 64-bit adder, VaLHALLA saves 55% energy and exhibits 38% lower EDP.

8. REFERENCES

[1] O. J. Bedrij. Carry-select adder. *IRE Transactions on Electronic Computers*, pages 340–344, 1962.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, 2011.

[3] Cisco. The zettabyte era—trends and analysis, white paper, document id:1465272001812119. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html, June 2016.

[4] W. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R. Harting, V. Parikh, J. Park, and D. Sheffield. Efficient embedded computing. *Computer*, 41(7):27–32, 2008.

[5] S. Dawson-Haggerty, A. Krioukov, and D. E. Culler. Power optimization: a reality check. Technical Report UCB/EECS-2009-140, University of California, Berkeley, 2009.

[6] R. Dennard, V. Rideout, E. Bassous, and A. LeBlanc. Design of ion-implanted mosfet s with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5), 1974.

[7] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[8] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, 2011.

[9] J. Hu and W. Qian. A new approximate adder with low relative error and correct sign calculation. In *Proceedings of Design, Automation and Test in Europe*, 2015.

[10] A. B. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proceedings of the 49th ACM/EDAC/IEEE Design Automation Conference*, 2012.

[11] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, 1997.

[12] G. Liu, Y. Tao, M. Tan, and Z. Zhang. Casa: Correlation-aware speculative adders. In *Proceedings of the International Symposium on Low Power Electronics and Design,* 2014.

[13] T. Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[14] J. M. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[15] G. Tziantzioulis, A. M. Gok, S. M. Faisal, N. Hardavellas, S. Ogrenci-Memik, and S. Parthsarathy. Lazy pipelines: Enhancing quality in approximate computing. In *Proceedings of Design, Automation and Test in Europe,* 2016.

[16] A. K. Verma, P. Brisk, and P. Ienne. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Proceedings of Design, Automation and Test in Europe,* 2008.

[17] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner. Theoretical and practical limits of dynamic voltage scaling. In *Proceedings of the 41st Design Automation Conference,* 2004.