Sustainable High-Performance Instruction Selection for Superscalar Processors

Saeideh Sheikhpour[†], David Metz[‡], Erling Jellum[‡], Magnus Själander[‡], and Lieven Eeckhout[†]
[†]Ghent University, Belgium
[‡]Norwegian University of Science and Technology (NTNU), Norway

ABSTRACT

Sustainability is a grand societal challenge, which requires our urgent attention given the significant and growing contribution of electronic devices to global warming. The environmental footprint of an electronic device comprises of two major contributors: (1) the embodied footprint due to raw material extraction, manufacturing, assembly, end-of-life-processing, and (2) the operational footprint due to device use during its lifetime. Sustainable hardware design hence requires a holistic approach that encompasses the entire lifetime of an electronic device.

In this paper, we demonstrate how to leverage conventional performance-power-area (PPA) analysis towards sustainable hardware design by investigating the sustainability-performance trade-off of a non-trivial hardware circuitry, namely the dynamic instruction selection logic in superscalar processors. We assess five previously proposed complexity-effective and power-efficient instruction selection approaches compared to conventional out-of-order (OoO) selection, namely Casino, Load Slice Core (LSC), Forward Slice Core (FSC), Delay-and-Bypass (DnB) and Freeway. We find that Casino, FSC and OoO are Pareto-optimal, optimally balancing the environmental footprint against performance; in contrast, LSC, DnB and Freeway are suboptimal. In addition, based on these insights, we further improve FSC's environmental footprint and propose FSC++ as a compelling sustainable design point: hardware synthesis to a 7 nm technology node and cycle-accurate FPGA simulation of complete SPEC CPU2017 benchmarks show that FSC++ reduces the environmental footprint by around 40% while degrading performance by only 1.7% compared to an OoO baseline.

1 INTRODUCTION

Sustainability in general and global warming in particular are grand societal challenges. Computer systems demand substantial materials and energy resources during production and use. A recent study by Freitag et al. [11] reports that information and communication technology (ICT) contributes 2.1 to 3.9% of the world's global greenhouse gas (GHG) emissions — on a par with the aviation industry — and this contribution is likely to increase further into the future.

A key question is how hardware designers can reduce the environmental footprint of electronic devices. The total footprint of a computing device consists of two major contributors: the embodied footprint (i.e., due to product manufacturing) and the operational footprint (i.e., due to product use during its lifetime). While the ratio of embodied versus operational footprint varies across devices [14], the embodied footprint is increasing under current scaling trends and is expected to become the predominant contributor in the near future [9]. Teehan and Kandlikar [26] report that silicon dies account for around 20% of a product's total embodied emissions (reaching up to more than 30%) across a wide spectrum of computing devices from tablets to laptops, desktops, and servers. Given current scaling trends, it is reasonable to assume that the environmental footprint of silicon dies has increased over the last decade [12], and furthermore, it is to be expected that it will continue to increase in the near future.

The most important silicon die in a computer system arguably is the processor, which in today's high-end systems (be it for personal computing or in the datacenter) contains multiple CPU cores, each featuring a superscalar out-of-order (OoO) microarchitecture. Unfortunately, while OoO cores deliver high performance, they are complex, resulting in large chip area, and high energy and power consumption. To tackle the complexity and power concern, a significant body of recent work [2, 6, 15, 17, 18] has proposed a variety of complexity-effective and power-efficient dynamic instruction selection mechanisms with the goal of approaching out-of-order performance at the complexity of a low-power in-order core. It is unclear though what their environmental footprint is.

In this paper, we evaluate five dynamic instruction selection mechanisms: Load Slice Core (LSC) [6], Freeway [17], Casino [15], Delay-and-Bypass (DnB) [2], and Forward Slice Core (FSC) [18]; and we compare these against the Berkeley Out-of-Order Machine core [27]. Through cycle-accurate FPGA-accelerated simulation of complete SPEC CPU2017 benchmarks and hardware synthesis to a 7 nm standard-cell technology node, we find that the different mechanisms lead to different trade-offs in environmental footprint versus performance. The overall conclusion is that *the Casino, FSC and OoO cores are Pareto-optimal*. In contrast, *LSC, DnB and Freeway are suboptimal*: they deliver too low performance for the environmental footprint they incur, or vice versa, they incur too high an environmental footprint for the delivered performance. Casino and OoO are the extremes along the Pareto frontier, i.e., Casino minimizes the environmental footprint (with the lowest performance) while OoO incurs the highest footprint (but also achieves the highest performance). FSC is a Pareto-optimal compromise design balancing footprint against performance.

Unfortunately, the sustainability-performance trade-off is somewhat underwhelming: while FSC reduces the environmental footprint by 25 to 33% (depending on the scenario), it also leads to a (non-negligible) performance degradation of 5% compared to an

ICCAD '24, October 27–31, 2024, New York, NY, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1077-3/24/10.
https://doi.org/10.1145/3676536.3676757

OoO core. We therefore propose a novel mechanism, FSC++, which offers a much more compelling design point: *FSC++ reduces the environmental footprint by 37 to 41% for a performance degradation of only 1.7% compared to the OoO baseline.* FSC++ achieves this (1) by running at a higher clock frequency than the OoO core which in part compensates for the reduction in instructions executed per cycle (IPC), and (2) by featuring an instruction queue in which half the entries are out-of-order while the other half are in-order, requiring less chip area and achieving higher performance than two in-order instruction queues in FSC.

By doing so, this paper demonstrates how to evaluate hardware sustainability using conventional performance-power-area (PPA) analysis — widely used in academia and industry — coupled with cycle-accurate FPGA simulations. We further demonstrate that optimizing for energy efficiency does not necessarily lead to the most sustainable design. Finally, we argue that combined sustainability metrics can be misleading, arguing for a comprehensive sustainability-performance trade-off analysis instead.

2 EVALUATING SUSTAINABILITY

Reducing the environmental footprint of an electronic device requires a holistic approach in which chip area, energy, power, clock frequency, and performance are optimized as a whole. It further requires that the entire lifetime is taken into account from raw material extraction, to manufacturing, assembly, transportation, usage, and end-of-life processing. This is a non-trivial endeavor because it is a multi-objective optimization problem while suffering from inherent data uncertainty.

We use the first-order FOCAL model [10] to assess the environmental footprint of a hardware design. Motivated by the inherent data uncertainty regarding the environmental impact of computing, the model is based on first principles using proxies for the embodied and operational footprint. The overall footprint of a computing device consists of two major contributors: (1) the *embodied footprint* due to manufacturing and fabrication, and (2) the *operational footprint* due to device usage over its entire lifetime. A parameter weighs the relative importance of the embodied versus operational footprint to account for variation in product use and lifetime.

Manufacturing a computer chip incurs a substantial environmental footprint, which can be decomposed into three scopes according to the GHG Protocol [21]. *Scope-1* refers to the chemicals and gases emitted during the manufacturing process, i.e., fluorinated gases with a carbon dioxide equivalent potential that is more than 20,000 times higher than CO_2. *Scope-2* refers to the carbon dioxide output due to the energy consumed during chip production; the more brown energy is used during manufacturing, the higher the scope-2 contribution. *Scope-3* refers to the footprint upstream to chip manufacturing due to material extraction and transportation. While empowering the fabs with green energy sources may drastically reduce the environmental footprint of chip production, it only affects scope-2, but not scope-1 nor scope-3 [9], hence it is not a silver bullet for reducing the footprint of chip manufacturing. Because the unit of fabrication is a wafer, a useful proxy for the embodied footprint of a chip is its die size. Indeed, the larger the chip, the larger its embodied footprint. Conversely, the smaller the chip, the smaller its embodied footprint.

The operational footprint of a processor is proportional to its energy usage during its entire lifetime. Assuming that the amount of work done is constant when comparing two designs, i.e., a *fixed-work* scenario, a useful proxy for the operational footprint of a chip is hence its energy usage. Unfortunately, this fixed-work scenario does not (always) reflect reality. It is expected that a system that is more efficient, e.g., higher performance or lower energy, will be used more intensively. This is the infamous rebound effect, also called Jevons' paradox, which states that a more efficient system incentivizes its usage, ultimately leading to a larger overall footprint [1]. Assuming that a more efficient design is used for the same time as a less efficient design, i.e., a *fixed-time* scenario, or for an even larger fraction of time, the operational footprint of a chip is proportional to its power consumption.

The first-order model features a parameter α ($0 \leq \alpha \leq 1$) to weigh the relative importance of the embodied versus operational footprint. The motivation for doing so is that the embodied versus operational footprint varies across computing devices [14]: while the embodied footprint tends to dominate for mobile personal devices (e.g., smartphones, tablets) as well as datacenter infrastructure, the operational footprint dominates for always-connected personal devices (e.g., desktop computers, gaming consoles). The relative importance of embodied versus operational footprint is further affected by the lifetime of the device, its use case, rebound effects, geographic location (i.e., green versus brown energy mix at the user's location), etc. Using a range of the α value allows for assessing sustainability despite inherent uncertainty about the actual manufacturing, deployment, and usage.

As such, FOCAL computes the *normalized carbon footprint (NCF)* of design X relative to design Y by weighing the normalized embodied and operational proxies, for the fixed-work scenario:

$$NCF_{\textit{fixed-work}} = \alpha \frac{A_X}{A_Y} + (1-\alpha)\frac{E_X}{E_Y},$$

with A_X and A_Y the chip area for designs X and Y, respectively, and E_X and E_Y their respective energy usage. For the fixed-time scenario, the NCF is computed as follows:

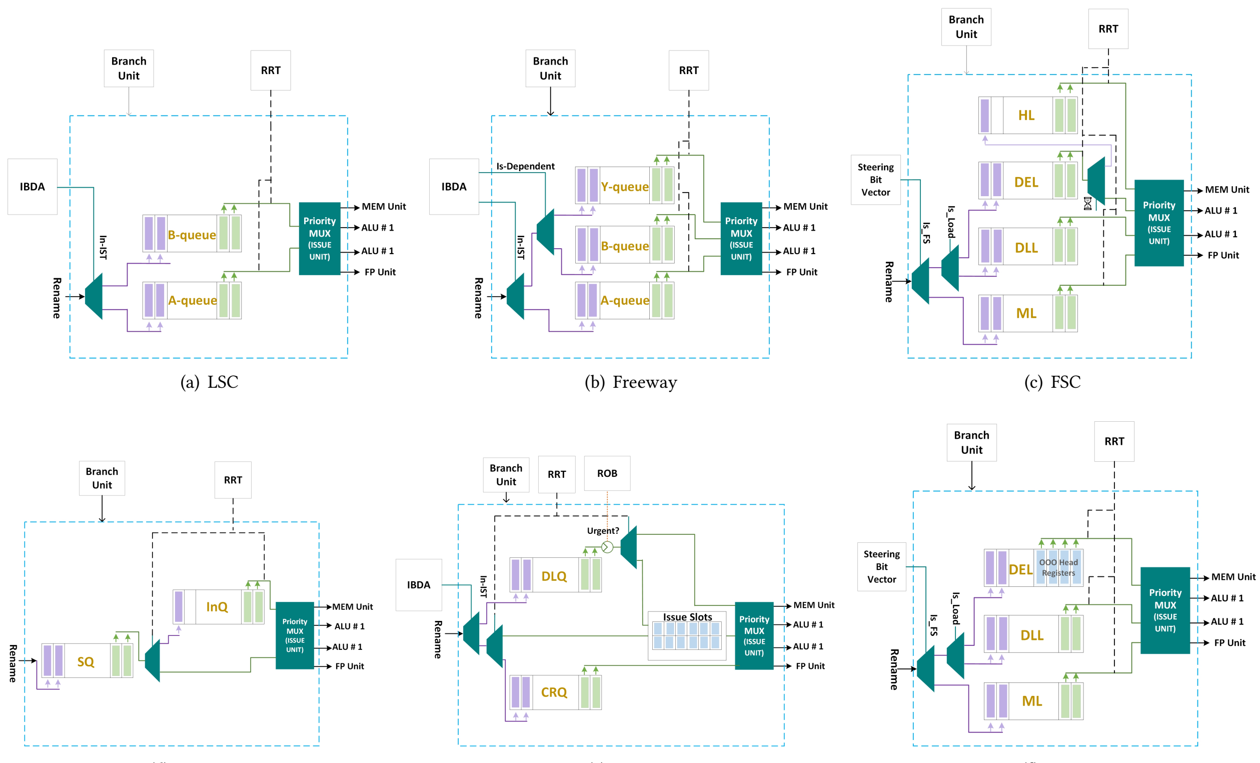
$$NCF_{\textit{fixed-time}} = \alpha \frac{A_X}{A_Y} + (1-\alpha)\frac{P_X}{P_Y},$$

with P_X and P_Y the respective power consumption. By considering both the fixed-work and fixed-time scenarios and by varying the α parameter, one can understand the impact of a design choice on their environmental footprint despite the inherent data uncertainty.

3 INSTRUCTION SELECTION LOGIC

We consider a total of five recently proposed complexity-effective and power-efficient dynamic instruction selection mechanisms for general-purpose superscalar processors, see also Figure 1. It is unclear though what their environmental footprint is.

(a) Load Slice Core (LSC) [6] features two in-order FIFO queues: the A and B-queue. The B-queue enables load and store instructions and their *backward slices* (i.e., the address-generating instructions leading directly or indirectly up to these memory operations) to bypass arithmetic instructions in the dynamic instruction stream, which are sent to the A-queue. Instructions are issued to the functional units in program order from the A and B-queues, while the



(a) LSC (b) Freeway (c) FSC (d) Casino (e) DnB (f) FSC++

Figure 1: Microarchitecture implementations: (a) LSC, (b) Freeway, (c) FSC, (d) Casino, (e) DnB, and (f) FSC++. FSC+ is similar to FSC except that the HL contains only 4 entries instead of 8; likewise, FSC+++ is similar to FSC++ except that the DEL contains only 4 entries instead of 8, i.e., DEL is a 4-entry OoO queue. Green boxes indicate the number of read ports to a queue; purple boxes indicate write ports; blue boxes indicate out-of-order entries.

queues may issue instructions out of program order with respect to each other. LSC requires register renaming and a reorder buffer (ROB) to keep track of program order and to enable precise exceptions. The motivation behind the LSC microarchitecture is to increase the amount of parallelism in the memory subsystem by steering load instructions and their dependents to different queues (i.e., loads are steered to the B-queue while their dependents are steered to the A-queue unless the dependents also lead to a memory operation). This enables independent loads to be issued in parallel to the memory hierarchy from the B-queue, despite load-dependent instructions in-between the independent loads in the dynamic instruction stream.

Identifying backward slices is done through a hardware mechanism called *iterative backward dependence analysis (IBDA)* which requires two hardware tables, the *register dependence table (RDT)* and the *instruction slice table (IST)*. The RDT contains as many entries as there are physical registers, with each entry holding the instruction pointer of the instruction that last wrote to the physical register. The IST is a cache that holds the instruction pointers of backward-slice instructions. When a load or store instruction is steered to the B-queue, IBDA adds its instruction pointer to the IST. For any instruction with its instruction pointer in the IST, IBDA adds the instruction pointers stored in the RTD of the instruction's source operands to the IST as well. This denotes that the instruction(s) is (are) now part of the memory instruction's backward slice. This procedure iteratively builds up the backward slice in the IST, starting with the memory operation upon its first execution (e.g., in the first iteration of a loop), the instruction(s) it depends on, directly or indirectly, upon subsequent executions (e.g., subsequent iterations of the loop). All instructions that hit in the IST are backward-slice instructions, and are steered to the B-queue; other instructions are steered to the A-queue.

LSC's hardware complexity is significantly reduced compared to an OoO core because it does not require expensive out-of-order instruction scheduling involving content-addressable memory (CAM) and long bypass wires. Instead, the instruction dispatch and issue unit consists of two in-order queues. In our setup, the RDT and IST incur a hardware overhead of 400 and 640 bytes, respectively.

(b) Freeway [17] makes the observation that an independent load may get stuck behind a load that depends on another, possibly long-latency, load in the B-queue of the LSC microarchitecture. The independent load cannot execute because it needs to wait for the dependent load to execute. This unnecessarily limits the amount of parallelism that can be exploited in the memory hierarchy. The Freeway microarchitecture addresses this performance issue by adding a third in-order queue, called the yielding queue or Y-queue for short, to which dependent loads are sent such that younger, independent loads can execute from the B-queue. More specifically, Freeway splits backward slices that contain multiple loads into two: a *producer slice* (i.e., the backward slice leading up to the initial load) and a *dependent slice* (i.e., the backward slice that starts after the initial load and ends with the dependent load). The dependent slice is steered to the Y-queue such that a younger independent backward slice can go ahead and execute from the B-queue, thereby improving memory-level parallelism and overall performance. Freeway is more complex than LSC by adding a third in-order queue.

(c) Forward Slice Core (FSC) [18] steers *forward slices*, i.e., the chain of instructions that depend on a load instruction, rather than backward slices to significantly reduce hardware overhead while at the same time improve performance compared to both LSC and Freeway. FSC steers instructions into three in-order queues: non-forward-slice instructions are steered from the pipeline's front-end into the so-called Main Lane (ML), while forward-slice instructions are steered into one of the Dependent Lanes, i.e., the loads are steered to the Dependent Load Lane (DLL) while the arithmetic (non-load) instructions are steered to the Dependent Execute Lane (DEL). Instructions that reside at the head of the DEL for more than a preset number of cycles (i.e., the L1 D-cache access latency) are redirected to a fourth queue, the Holding Lane (HL), such that independent instructions can execute from the DEL. To force loads and stores to execute in program order, FSC features a technique called *store-address replication (SAR)*: a store instruction is broken up in a store-address (STA) and store-data (STD) micro-op, of which the STA micro-op is replicated across the ML and DLL lanes; the STA micro-op executes only when all replicates have reached their respective lane heads.

The intuition of the FSC proposal is to steer instructions that depend on a load instruction that is yet to be executed to separate in-order queues such that independent instructions can execute from the main lane. Furthermore, instructions in the DEL that depend on long-latency loads are sent to a separate holding lane, such that younger independent instructions can execute as soon as possible.

FSC requires a so-called *steering bit vector (SBV)* to identify forward slices: the SBV is a bit vector with as many entries as there are physical registers to track which registers transitively depend on a load. When steering a load, the SBV bit corresponding to its destination register is set. An instruction for which at least one of its source registers' SBV bits is set, is a forward-slice instruction, and its destination SBV bit is set. The SBV bit of the physical destination register is cleared upon instruction execution. This mechanism identifies the forward slices of loads that are yet to be executed.

FSC incurs less hardware overhead than LSC as it does not rely on IBDA and does not need the RDT and IST hardware structures; instead, the SBV is a small hardware structure of 18 bytes.

(d) Casino [15] features two cascading FIFO queues: the Speculative Instruction Queue (SQ) and the In-Order Queue (InQ). These queues serve distinct purposes to enhance performance over a single in-order core. All instructions are initially placed in the SQ. Any instruction at the head of the SQ that is ready for execution is promptly issued to a functional unit. A non-ready instruction at the head of the SQ is redirected to the InQ from which it executes in program order. Instructions issued from the SQ execute OoO with respect to redirected instructions to the InQ.

(e) Delay-and-Bypass (DnB) [2] builds upon the notions of instruction criticality and readiness. An instruction is *critical* if it is a memory operation or an instruction that belongs to a backward slice. (DnB uses the IBDA mechanism from LSC to detect backward slices.) An instruction is *ready* if it has all of its operands available in the front-end of the pipeline. The key observation that underpins DnB is that ready instructions do not need complex OoO scheduling, hence they can be sent to in-order queues and execute immediately. Only critical and non-ready instructions can potentially benefit from dynamic scheduling from an OoO queue, such that they can execute as soon as their dependences resolve. Non-critical and non-ready instructions can be delayed in an in-order queue until they are about to be ready, at which time they are steered to the out-of-order queue — this further reduces the pressure on the out-of-order queue. Implementing this design principle requires DnB to complement an out-of-order instruction queue (IQ) with two in-order queues: the delay queue (DLQ) and the critical-ready queue (CRQ). The DLQ contains non-critical, non-ready instructions, while the CRQ contains the critical, ready instructions; the IQ is the OoO queue to which critical, non-ready instructions are steered as well as non-critical instructions when they are about to be ready. The IQ is (much) smaller than the instruction queue in a conventional OoO core because it contains only a small fraction of the instructions.

4 REDUCING FSC'S FOOTPRINT

As we will report in Section 7, FSC reduces the environmental footprint by, depending on the scenario, 25 to 33% for a 5% performance degradation compared to OoO. While FSC meaningfully balances environmental footprint versus performance, it would be even better to further reduce its footprint *and* at the same time improve its performance. This triggered us to evaluate three optimizations.

FSC+. The baseline FSC architecture features four equal-length queues with eight entries each. Because we find the occupancy in the HL to be relatively low compared to the other queues, the first optimization is hence to explore unequal queue lengths. In particular, we consider a 4-entry HL rather than an 8-entry FL, while keeping the other queues unchanged (eight entries each).

FSC++. The second optimization is to merge the 4-entry HL into the DEL, which also features relatively low occupancy, in such a way that the top-four entries in the DEL effectively belong to the HL and are eligible for selection in case an instruction is ready, see also Figure 1(f). In other words, the four oldest instructions in the DEL can be selected out-of-order; the other four entries in the DEL are in-order. If an instruction is selected for execution, the DEL shifts the younger instructions through the DEL such that the four oldest instructions in the DEL can be selected for execution. The intuition behind this optimization is to enable younger instructions to bypass older instructions that depend on a long-latency load.

FSC+++. The third optimization further reduces the DEL to only four entries, with all four entries being eligible for selection. In other words, the DEL effectively is an out-of-order queue, albeit of a small size, namely four entries.

5 HARDWARE IMPLEMENTATION DETAILS

We now dive deeper into some of the hardware implementation details for the various microarchitectures. We use the SonicBOOM core [27] as our starting point, leveraging its existing structures, and directing our focus on the dynamic instruction execution engine, which includes the steering logic, the various (in-order and out-of-order) queues, and the selection logic. The core can fetch, decode, rename, dispatch, and commit two instructions per cycle, and can select up to four instructions per cycle for execution on one of the two ALUs, the memory unit, or the floating-point unit.

All queues are implemented using a circular buffer with head and tail pointers and consist of eight entries, unless mentioned otherwise. Each queue has a specific number of read and write ports, indicated in green and purple, respectively, see Figure 1. To handle branch mispredictions and to update the branch masks, the dispatch/issue unit requires a dedicated port connected to the branch unit. The instruction selection logic requires access to the *register ready table (RRT)*, which contains as many bits as there are physical registers. A ready bit in the RRT is set when the corresponding physical register value is computed. The ready bit is reset upon physical register allocation in the rename stage.

The issue queue selection logic is implemented using a priority multiplexer with a ready-valid interface connected to the heads of the various FIFO queues. The selection logic acts as a control mechanism, determining which instructions from the queues are selected for execution in a given cycle depending on their type and readiness information. A static priority scheme is followed across the various queues. We now discuss the implementation details that are specific to each of the microarchitectures.

LSC. Up to two instructions are steered to either the A or B queue per cycle. An instruction for which its instruction pointer is stored in the IST, i.e., this is a backward-slice instruction, is steered to the B queue. Up to four instructions can be selected for execution. Due to in-order processing inside each queue, the instruction in the second head position of a queue can only be selected for execution if the instruction at the head position is selected. Priority is given to the B-queue for instruction selection, followed by the A-queue.

Freeway. An instruction that belongs to a dependent backward slice is steered to the Y queue, while an instruction that is part of a producer backward slice is steered to the B queue. Up to four instructions can be selected from six queue head positions (again, in-order selection per queue); priority is given to the Y-queue, then the B-queue, and then the A-queue.

FSC. A forward-slice instruction is steered to the DLL or DEL depending on whether it is a load or not. A non-forward-slice instruction is steered to the ML. An instruction that sits at the head of the DEL for more than four cycles — implemented using a simple 2-bit down-counter — is re-directed to the HL. Up to four instructions can be selected (in-order per queue) out of eight instruction slots across the four queues, while giving priority to HL, then DEL, then DLL, and then ML. Note that FSC+ is similar to FSC except that the HL contains only four entries instead of eight.

Casino. Up to two ready instructions are selected from the head of the SQ for execution. If the head of the SQ is non-ready, it is re-directed to the InQ. Up to two instructions can be selected from the head of the InQ. Priority is given to the InQ.

DnB. Backward-slice instructions are steered to the OoO IQ if not ready; otherwise, they are steered to the CRQ. Non-backward-slice instructions are steered to the DLQ. DnB selects up to four instructions from the 16 instruction slots in the three queues: two at the head of the DLQ (in-order selection), two at the head of the CRQ (in-order selection), and any of the 12 entries in the IQ (out-of-order selection). Priority is given to the DLQ, IQ and CRQ, respectively.

FSC++. Instruction steering is the same as for FSC. The HL is integrated in the DEL as a small OoO queue at the head of the DEL from which up to four instructions can be selected. The DEL is implemented as a shift register rather than a circular buffer, i.e., when instructions are selected for execution, the queue is compacted by shifting up to two instructions towards the head. FSC+++ is the same as FSC++ except that the DEL contains four entries instead of eight.

6 EXPERIMENTAL SETUP

Cycle-Accurate FPGA Simulation. We use Chipyard's FireSim [20] to evaluate cycle count for the various instruction selection mechanisms which we implemented in Chisel by modifying the 2-wide fetch, 4-wide issue SonicBOOM Berkeley Out-of-Order Machine core [27]. We obtain cycle-accurate performance results using two on-premise Xilinx U250 Alveo Data Center accelerator cards. FireSim uses FASED [5] to accurately model DRAM timing during FPGA-accelerated simulation. The core configurations that we simulate are listed in Table 1: they all run at 3.2 GHz and feature the same total number of queue entries, namely 32, except for Casino (16 entries), FSC+ (28 entries), FSC++ (24 entries), and FSC+++ (20 entries). The other core components (number of functional units, branch predictor, caches, etc.) are kept constant across the different instruction selection mechanisms.

PPA Analysis. We synthesize to a standard-cell technology to obtain cycle time, chip area and power consumption for the various designs using the open-source ASAP7 Predictive Process Design Kit (PDK) 7 nm FinFET standard-cell library [7]. The back-end flow (placement, routing, clocking, and optimization) is done using Cadence Innovus 2021.

To estimate power consumption, we use the following methodology. Because it is impossible to simulate complete SPEC CPU2017 benchmarks of the final ASIC implementation to compute activity factors, we instead simulate a set of microbenchmarks (provided by Chipyard [3]) using Verilator to generate activity factors for all signals in a design. These activity factors then serve as input to Cadence Voltus to quantify power consumption, assuming a supply voltage of 0.7 V and ambient temperature of 25°. This methodology yields a power estimate per instruction selection mechanism.

We limit the evaluation of chip area and power consumption to the core's execution engine, as described in Section 5. The other stages in the pipeline are the same across all designs, and hence we do not synthesize them.

Benchmarks. We simulate nine SPEC CPU2017 benchmarks, which we run to completion with their reference inputs. (We considered the ten 'intspeed' SPEC CPU2017 benchmarks supported by the FireSim framework, but had to exclude the perlbench benchmark because of inconsistent performance results due to non-determinism.) The benchmarks were compiled with the gcc compiler v7.5.0 at optimization level -O3 using the FireMarshal framework [20]. The number of dynamically executed instructions varies between 1.12 trillion for 620.omnetpp_s and 9.51 trillion for 657.xz_s. We run the Linux kernel version 5.7.0 on top of FireSim.

Table 1: Simulated instruction selection mechanisms, incl. queue depth (number of entries) and selection logic (InO vs OoO).

	Casino	*LSC*	*Freeway*	*FSC*	*FSC+*	*FSC++*	*FSC+++*	*DnB*	*OoO*
Queues	SQ (#4, InO) InQ (#12, InO)	AQ (#16, InO) BQ (#16, InO)	AQ (#16, InO) BQ (#8, InO) YQ (#8, InO)	ML (#8, InO) DLL (#8, InO) DEL (#8, InO) HL (#8, InO)	ML (#8, InO) DLL (#8, InO) DEL (#8, InO) HL (#4, InO)	ML (#8, InO) DLL (#8, InO) DEL (#8, InO/**OoO**)	ML (#8, InO) DLL (#8, InO) DEL (#4, **OoO**)	IQ (#12, **OoO**) CRQ (#10, InO) DLQ (#10, InO)	INT (#20, **OoO**) MEM (#12, **OoO**) FP (#16, **OoO**)
IBDA	–	RDT (400 B) IST (2-way, 640 B)	RDT (400 B) IST (2-way, 640 B)	–	–	–	–	RDT (400 B) IST (2-way, 640 B)	–
In-flight instructions	64-entry reorder buffer; 16-entry load queue; 16-entry store queue								
Register file	80-entry integer RF (6 read ports, 3 write ports); 64-entry floating-point RF (3 read ports, 2 write ports)								
Pipeline	2-way superscalar execution; 2 ALUs, 1 LD/ST unit, 1 FP unit; 10 pipeline stages								
Front-end	TAGE branch predictor w/ 64-bit global history and 6 tables, 1K-entry BTB, 32-entry μBTB, 8-byte fetch per cycle, 16-entry fetch buffer								
Caches	16 KB 2-way L1 I-cache, 16 KB 2-way L1 D-cache (4 cycles load-to-use latency), 512 KB 8-way L2 cache (14 cycles)								
TLBs	32-entry L1 I-TLB, 8-entry L1 D-TLB, 1024-entry L2 TLB								
Memory	16 GB DDR3, 16 GB/s maximum bandwidth, 14-14-14 (CAS-RCD-RP) latencies @ 1 GHz								

7 RESULTS

We first discuss the environmental impact of the various microarchitectures, after which we analyze their performance, chip area, and power consumption in more detail.

7.1 Sustainability-Performance Trade-Off

Figure 2 reports normalized carbon footprint as a function of performance for the scenario (a) when the embodied footprint dominates (i.e., $\alpha = 0.8$ with the error bars for $\alpha \in [0.7, 0.9]$) and (b) when the operational footprint dominates (i.e., $\alpha = 0.2$ with the error bars for $\alpha \in [0.1, 0.3]$).[1] We report results for the fixed-work scenario only due to space constraints; the fixed-time scenario results are fairly similar though, leading to similar conclusions. The ideal instruction selection mechanism should deliver high performance and incur a low environmental footprint, i.e., be situated in the bottom right of Figure 2. Environmental footprint is computed as described in Section 2 using the chip area and energy numbers obtained through hardware synthesis. We reach several conclusions.

Finding #1: Casino, FSC, and OoO are Pareto-optimal while LSC, Freeway and, DnB are not. It is interesting to observe that different selection mechanisms lead to different trade-offs in performance versus environmental impact. The Casino, FSC, and OoO cores are Pareto-optimal, i.e., no other mechanism yields higher performance at a lower environmental footprint. LSC, Freeway and DnB on the other hand are suboptimal, i.e., they deliver too low performance for the environmental footprint they incur, or vice versa, the environmental footprint is too high for the performance they deliver. This conclusion holds true for both the embodied-dominated and the operational-dominated scenarios. The Pareto-optimal designs offer different trade-offs in performance versus environmental footprint. While Casino incurs the lowest footprint, it also delivers the lowest performance; in contrast, the OoO core delivers the highest performance but also incurs the highest footprint. FSC is a compromise design yielding high performance at moderate footprint.

Finding #2: FSC++ delivers a compelling sustainable design point substantially reducing the environmental footprint compared to an OoO core with a negligible performance drop. The FSC optimizations further reduce the footprint of the FSC mechanism while in addition improving performance (FSC+ and FSC++) or at the cost of a limited performance degradation (FSC+++). In particular, FSC++ is a compelling design point: it reduces the environmental footprint by 37% (when embodied footprint dominates) and 41% (when operational footprint dominates) compared to the OoO core while reducing average performance by only 1.7%.

Finding #3: Which design point is most sustainable depends on the scenario. It is interesting to note that DnB (as well as Freeway to some extent) is more sustainable than the baseline OoO core when the operational footprint dominates, while being less sustainable when the embodied footprint dominates. This indicates that DnB's higher embodied footprint does not get amortized by its reduced operational footprint compared to the OoO core.

7.2 Performance

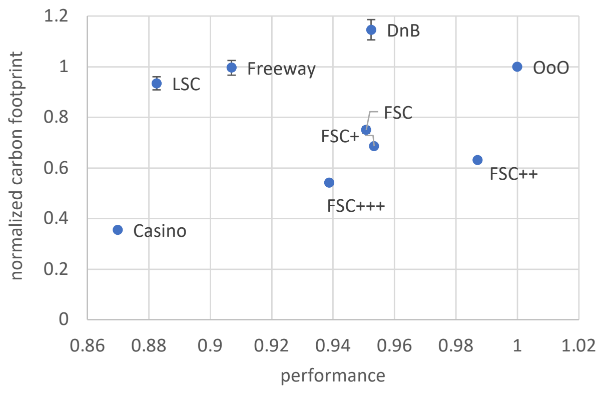
We now analyze performance in more detail. Single-core performance is the product of the number of instructions executed per cycle (IPC) times clock frequency. The methodology used in this work allows for evaluating both factors: hardware synthesis yields cycle time and thus clock frequency, and FPGA-accelerated simulation yields IPC. We set the same target frequency for all designs during synthesis, and we find that Casino, LSC and Freeway achieve a similar (less than 0.1% difference) clock frequency as the OoO core. DnB, FSC, and FSC+ slightly improve clock frequency by 0.7%, 0.8% and 1.4%, respectively. FSC++ and FSC+++ both achieve a 3.3% higher clock frequency than OoO.

Finding #4: FSC++ achieves an average performance degradation of only 1.7% compared to OoO, significantly outperforming the other designs. Figure 3 reports normalized performance (IPS or instructions per second) for the various designs. (The vertical axis starts at 0.7 which corresponds to the performance achieved by the in-order Rocket core [4] assuming that its frequency is identical to the OoO core.) The key conclusion is that FSC++ degrades performance by only 1.7% compared to OoO. For some benchmarks (omnetpp, gcc and mcf), FSC++ achieves even higher performance than OoO because of its higher clock frequency while achieving the same (or similar) IPC. The other complexity-effective designs deliver substantially lower performance, degrading performance by on average 5.2% (FSC+), 5.2% (DnB), 5.3% (FSC), 6.4% (FSC+++), 9.8% (Freeway), 12.2% (LSC) and 13.2% (Casino).

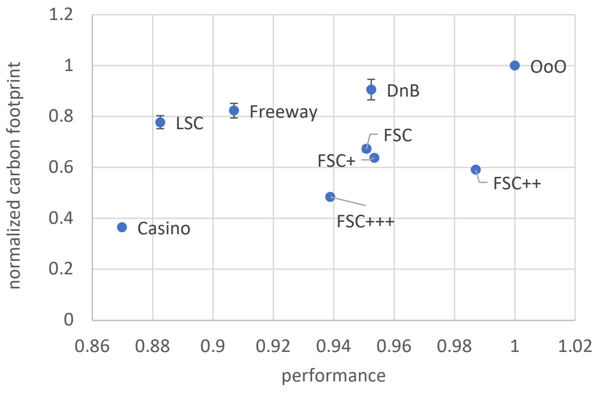
7.3 Chip Area

Figure 4 (on the left) reports chip area for the various microarchitectures normalized to the OoO core.

[1]When the error bars are not visible in Figure 2, this means that the impact is small when changing α. This happens when the relative embodied and operational footprints are similar.



(a) Embodied footprint dominates



(b) Operational footprint dominates

Figure 2: Normalized carbon footprint as a function of performance normalized to the OoO core when (a) the embodied footprint dominates ($\alpha = 0.8$) and (b) the operational footprint dominates ($\alpha = 0.2$), assuming a fixed-work scenario. *Casino, FSC and OoO are Pareto-optimal designs, in contrast to LSC, Freeway and DnB which are suboptimal.*

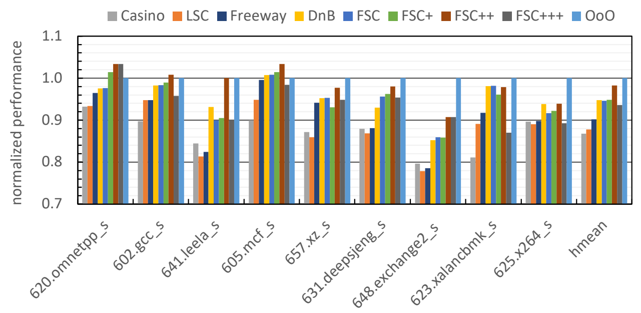


Figure 3: Performance (IPS) normalized to OoO. *FSC++ achieves an average performance degradation of only 1.7% compared to OoO, significantly outperforming the other complexity-effective designs.*

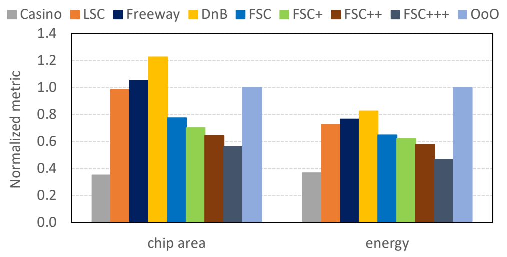


Figure 4: Chip area (left) and energy usage (right) normalized to the OoO core. *Casino results in the smallest design, followed by FSC and its variants; Freeway and DnB incur more chip area than the OoO core. Casino uses the least amount of energy followed by FSC and its variants, and then LSC, Freeway and DnB.*

Finding #5: FSC++ and FSC+++ are smaller than FSC. Casino incurs the smallest chip area: it consists of only two queues (4-entry SQ and 12-entry InQ). Interestingly and surprisingly perhaps, FSC++ and FSC+++ are smaller sized than FSC. The 8-entry DEL with a 4-entry OoO sub-queue in FSC++ and the 4-entry OoO DEL in FSC+++ incur less chip area than the 8-entry in-order DEL plus HL in FSC. This is something that is hard, if not impossible, to assess without doing actual hardware synthesis, which further illustrates the need for a hardware-synthesis-driven methodology as advocated in this work.

Finding #6: LSC, Freeway, and DnB are larger than FSC (and variants) even though the aggregate queue size is the same (total of 32 entries); the reason for the larger chip area is the RDT and IST hardware structures in LSC and Freeway versus the much smaller SBV for FSC. DnB is the largest, due to the out-of-order IQ, in addition to the DLQ and CRQ.

7.4 Energy Usage

Figure 4 (on the right) reports normalized energy usage. Casino is the most energy-efficient design, followed by FSC and its variants. LSC, Freeway, and DnB use more energy than FSC. All complexity-effective designs use less energy than the OoO core.

8 DISCUSSION

Energy efficiency. It is probably common to believe that optimizing the energy efficiency of a design also leads to the most sustainable design. This is not true.

Finding #7: A more energy-efficient design is not necessarily more sustainable. Take the example of DnB in comparison to OoO. As illustrated in Figure 4, DnB uses less energy than OoO, and hence its operational footprint is less. However, DnB also incurs a larger chip area, hence its embodied footprint is higher. If the embodied footprint dominates, which is the case for battery-operated devices as well as for servers in hyperscale datacenters [14], this leads to a higher overall carbon footprint, as illustrated in Figure 2(a). Although DnB emits less carbon than OoO during usage, looking at the entire lifecycle including manufacturing leads to the opposite conclusion that OoO incurs more carbon, i.e., DnB's lower operational footprint does not amortize its higher embodied footprint.

Combined sustainability metrics. Prior work proposed a variety of combined sustainability metrics, i.e., these metrics combine carbon footprint with another metric such as performance or energy usage into a single metric to assess a design's sustainability. In particular, Gupta et al. [13] propose carbon-delay product

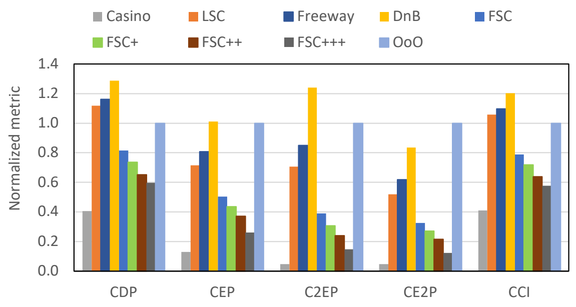


Figure 5: Previously proposed combined sustainability metrics for the various instruction selection mechanisms normalized to the OoO baseline. *Combined sustainability metrics do not expose trade-offs in sustainability versus performance.*

(CDP), carbon-energy product (CEP), carbon-square-energy product (C^2EP) and carbon-energy-square product (CE^2P), in which C stands for the embodied footprint; all metrics are lower-is-better. Switzer et al. [25] propose the computational carbon intensity (CCI) metric, which is defined as the ratio of the total (embodied plus operational) carbon footprint and performance. Figure 5 reports these five previously proposed sustainability metrics for the various designs normalized to the OoO baseline assuming that the embodied footprint dominates ($\alpha = 0.8$) under a fixed-work scenario.

Finding #8: Combined sustainability metrics cannot expose sustainability-performance trade-offs. None of the combined sustainability metrics provide the insight that Casino, FSC, and OoO are Pareto-optimal, and that LSC, Freeway, and DnB are suboptimal. Interestingly, both the CDP and CCI metrics point out that LSC, Freeway, and DnB are suboptimal compared to OoO, unlike the other metrics. This suggests that the reduction in embodied footprint (for CDP) and total footprint (for CCI) is smaller than the performance degradation. However, a value below one for CDP or CCI does not imply that a design is Pareto-optimal. Pareto-optimality can only be assessed by explicitly reporting the environmental footprint versus performance trade-off as previously shown in Figure 2.

9 RELATED WORK

Sustainability. Sustainability just recently received attention by the systems community. In particular, Gupta et al. [14] conducted a comprehensive survey to assess the environmental footprint of computing. One key conclusion from this study is that the carbon footprint is mostly dominated by the embodied footprint for mobile personal devices (smartwatches, smartphones, tablets) as well as for servers in hyperscale datacenters. Personal always-connected devices (game consoles, desktop computers) on the other hand are mostly dominated by the operational footprint. Eeckhout [9] reaches a similar conclusion by analyzing how current trends in microprocessor chip demand and semiconductor manufacturing energy and carbon footprint: the embodied footprint of computing devices is likely to continue to grow in the foreseeable future up to the point that it will be the dominant contributor to the total footprint if that is not already the case today.

A number of carbon footprint models have been proposed, such as GreenChip [16] and ACT [13] which are bottom-up data-driven approaches to evaluate a computing device's environmental footprint. Motivated by the inherent data uncertainty pertaining to sustainability, we opted in this work to use the first-order top-down FOCAL model [8, 10]. The key asset of FOCAL in the context of this work is that it uses proxies that hardware designers have control over and can reason about to make a holistic sustainability assessment when designing hardware circuitry.

Complexity- and power-efficient instruction selection. Ample research effort has been devoted to improve the power and energy efficiency of modern-day microarchitectures [24]. A variety of techniques have been developed to reduce energy and power consumption, including dynamic voltage and frequency scaling, clock gating, power gating, pipeline gating, etc. These techniques have collectively led to a significant reduction in operational emissions for a broad range of computing devices, from wearables to smartphones, tablets, laptops, desktops, and rack servers [14].

The embodied footprint of a processor can only be reduced by reducing its complexity and thus the chip area that it incurs [8, 10]. OoO cores are known to be complex and incur substantial chip area. A variety of proposals have been made in the literature to reduce the design complexity of superscalar processors. Besides the dynamic instruction selection mechanisms evaluated in this work, complexity-effective superscalar processors (CESP) [19] steer chains of dependent instructions to in-order queues that operate out-of-order with respect to each other. Because instruction steering stalls when an independent instruction cannot be steered to an empty queue, CESP yields lower performance compared to FSC, especially when few queues are deployed for CESP [18].

A couple other microarchitecture proposals leverage readiness and criticality of instructions to simplify the out-of-order back-end, similar to the Delay-and-Bypass sOoO proposal [2]. In particular, the front-end execution architecture [23] promptly executes ready instructions in the front-end, while steering non-ready instructions to the out-of-order back-end. Because fewer instructions are steered to the back-end, it can be made less complex. Instead of executing from the front-end, long-term parking [22] eagerly allocates back-end resources for critical instructions while postponing back-end resource allocation for non-critical instructions for as long as possible, thereby saving power.

10 CONCLUSION

Assessing the environmental footprint of a hardware design requires a holistic and detailed analysis of its chip area, power, energy, and performance. This paper contributes: (1) the insight that conventional PPA analysis coupled with cycle-accurate FPGA simulation enables comprehensively assessing the sustainability impact of a design; (2) some previously proposed complexity-effective and power-efficient dynamic instruction selection mechanisms (Casino and FSC) offer Pareto-optimal sustainability-performance trade-offs, while others (LSC, Freeway, and DnB) are suboptimal; and (3) the novelly proposed FSC++ mechanism provides a compelling sustainable design point reducing the environmental footprint by around 40% while degrading performance by only 1.7% compared to an OoO baseline. We hope that this paper inspires and provides the insights and methodologies for hardware designers to explore sustainability trade-offs when designing new hardware features.

REFERENCES

[1] B. Alcott, "Jevons' paradox," *Ecological Economics*, vol. 54, no. 1, 2005.

[2] M. Alipour, S. Kaxiras, D. Black-Schaffer, and R. Kumar, "Delay and bypass: Ready and criticality aware instruction scheduling in out-of-order processors," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2020, pp. 424–434.

[3] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[4] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[5] D. Biancolin, S. Karandikar, D. Kim, J. Koenig, A. Waterman, J. Bachrach, and K. Asanovic, "FASED: FPGA-accelerated simulation and evaluation of DRAM," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019, pp. 330–339.

[6] T. E. Carlson, W. Heirman, O. Allam, S. Kaxiras, and L. Eeckhout, "The load slice core microarchitecture," in *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 272–284.

[7] L. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthya, and G. Yeric, "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, Jul. 2016.

[8] L. Eeckhout, "A first-order model to assess computer architecture sustainability," *IEEE Computer Architecture Letters*, vol. 21, pp. 137–140, July–August 2022.

[9] L. Eeckhout, "Kaya for computer architects: Toward sustainable computer systems," *IEEE Micro*, vol. 43, pp. 9–18, Jan/Feb 2023.

[10] L. Eeckhout, "FOCAL: A first-order model to assess processor sustainability," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, vol. 2, 2024, pp. 401–415.

[11] C. Freitag, M. Berbers-Lee, K. Widdicks, B. Knowles, G. S. Blair, and A. Friday, "The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations," *Patterns*, vol. 2, no. 9, 2021.

[12] M. Garcia Bardon, P. Wuytens, L.-A. Ragnarsson, G. Mirabelli, D. Jang, G. Willems, A. Mallik, A. Spessot, J. Ryckaert, and B. Parvais, "DTCO including sustainability: Power-performance-area-cost-environmental score (PPACE) analysis for logic technologies," in *IEEE International Electron Devices Meeting (IEDM)*, 2020.

[13] U. Gupta, M. Elgamal, G.-Y. W. G. Hills, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "ACT: Designing sustainable computer systems with an architectural carbon modeling tool," in *Proceedings of the ACM/IEEE Inernational Symposium on Computer Architecture (ISCA)*, 2022, pp. 784–799.

[14] U. Gupta, Y. G. Kim, S. Lee, J. Tse, H.-H. S. Lee, G.-Y. Wei, D. Brooks, and C.-J. Wu, "Chasing carbon: The elusive environmental footprint of computing," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 854–867.

[15] I. Jeong, S. Park, C. Lee, and W. W. Ro, "CASINO core microarchitecture: Generating out-of-order schedules using cascaded in-order scheduling windows," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2020, pp. 383–396.

[16] D. Kline, N. Parshook, X. Ge, E. Brunvand, R. G. Melhem, P. K. Chrysanthis, and A. K. Jones, "GreenChip: A tool for evaluating holistic sustainability of modern computing systems," *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 322–332, Jun. 2019.

[17] R. Kumar, M. Alipour, and D. Black-Schaffer, "Freeway: Maximizing MLP for slice-out-of-order execution," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 558–569.

[18] K. Lakshminarasimhan, A. Naithani, J. Feliu, and L. Eeckhout, "The forward slice core microarchitecture," in *Proceedings of the IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2020, pp. 361–372.

[19] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, Jun. 1997, pp. 206–218.

[20] N. Pemberton and A. Amid, "Firemarshal: Making HW/SW co-design reproducible and reliable," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 299–309.

[21] J. Ranganathan and et al., "The greenhouse gas protocol: A corporate accounting and reporting standard – revised edition," 2004. [Online]. Available: https://ghgprotocol.org/sites/default/files/standards/ghg-protocol-revised.pdf

[22] A. Sembrant, T. E. Carlson, E. Hagersten, D. Black-Schaffer, A. Perais, A. Seznec, and P. Michaud, "Long term parking (LTP): criticality-aware resource allocation in OOO processors," in *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, Dec. 2015, pp. 334–346.

[23] R. Shioya, M. Goshima, and H. Ando, "A front-end execution architecture for high energy efficiency," in *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, Dec. 2014, pp. 419–431.

[24] M. Själander, M. Martonosi, and S. Kaxiras, *Power-Efficient Computer Architectures: Recent Advances.* Morgan & Claypool Publishers, 2014.

[25] J. Switzer, G. Marcano, R. Kastner, and P. Pannuto, "Junkyard computing: Repurposing discarded smartphones to minimize carbon," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, vol. 2, Mar. 2023, pp. 400–412.

[26] P. Teehan and M. Kandlikar, "Comparing embodied greenhouse gas emissions of modern computing and electronics products," *Environmental Science and Technology*, vol. 43, no. 9, pp. 3997–4003, May 2013.

[27] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "SonicBOOM: The 3rd generation Berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, 2020.