EES2: Software and Algorithms

July 20, 2023

Brian Hirano

Outline

- Computer Language Efficiency and Software Optimization
- Interaction between Software and Architecture
- Importance of Architecture in “Novel Compute”
- Workloads/Benchmarks to Measure Power Use

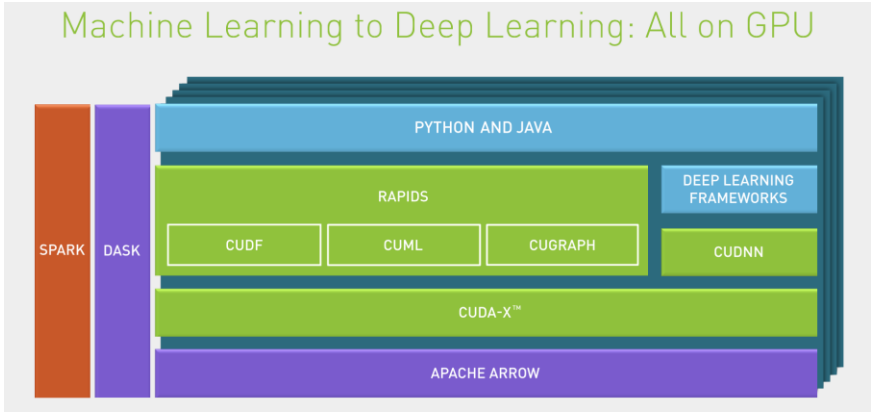
Table 4: Normalized global results for Energy, Time, and Memory

Total					
	Energy (J)		Time (ms)		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Pereira, Rui, et al., *Ranking Programming Languages by Energy Efficiency*, Science of Computer Programming

Micron

Not all Python code runs only Python



Alternatives to Interpreted Python

Mojo

cinder

julia

codon

Software Performance Example

TABLE I. Summary of Appel's speedups

Design Level	Speedup Factor	Modification
Algorithms and Data Structures	12	A binary tree reduces $O(N^2)$ time to $O(N \log N)$
Algorithm Tuning	2	Use larger time steps
Data Structure Reorganization	2	Produce clusters well-suited to the tree algorithm
System-Independent Code Tuning	2	Replace double-precision floating point with single precision
System-Dependent Code Tuning	2.5	Recode the critical procedure in assembly language
Hardware	2	Use a floating-point accelerator
Total	400	

From: Jon Bentley, *Programming Pearls: Perspective on Performance*, Communications of the ACM, Volume 27, Issue 11, November,1984

Software Performance Example

total speedup factor of 400; Appel's final program runs a 10,000-body simulation in about one day. The speedups were not free, though. The trivial $O(N^2)$ algorithm may be expressed in a few dozen lines of code, while the fast program required 1,200 lines of Pascal. The design and implementation of the fast program required several months of Appel's time. The speedups are summarized in Table I.

From: Jon Bentley, *Programming Pearls: Perspective on Performance*, Communications of the ACM, Volume 27, Issue 11, November,1984

Implications of Inefficient Software

- We reexamine a number of claims [9, 19, 21, 32, 42, 45, 47, 53] that GPUs perform 10X to 1000X better than CPUs on a number of throughput kernels/applications. After tuning the code for **BOTH** CPU and GPU, we find the GPU only performs 2.5X better than CPU. This puts CPU and GPU roughly in the same performance ballpark for throughput computing.

Lee, Victor, et al., *Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU*, ISCA '10: Proceedings of the 37th annual International Symposium on Computer Architecture, June, 2010.

Interaction of Software and CPU Architecture

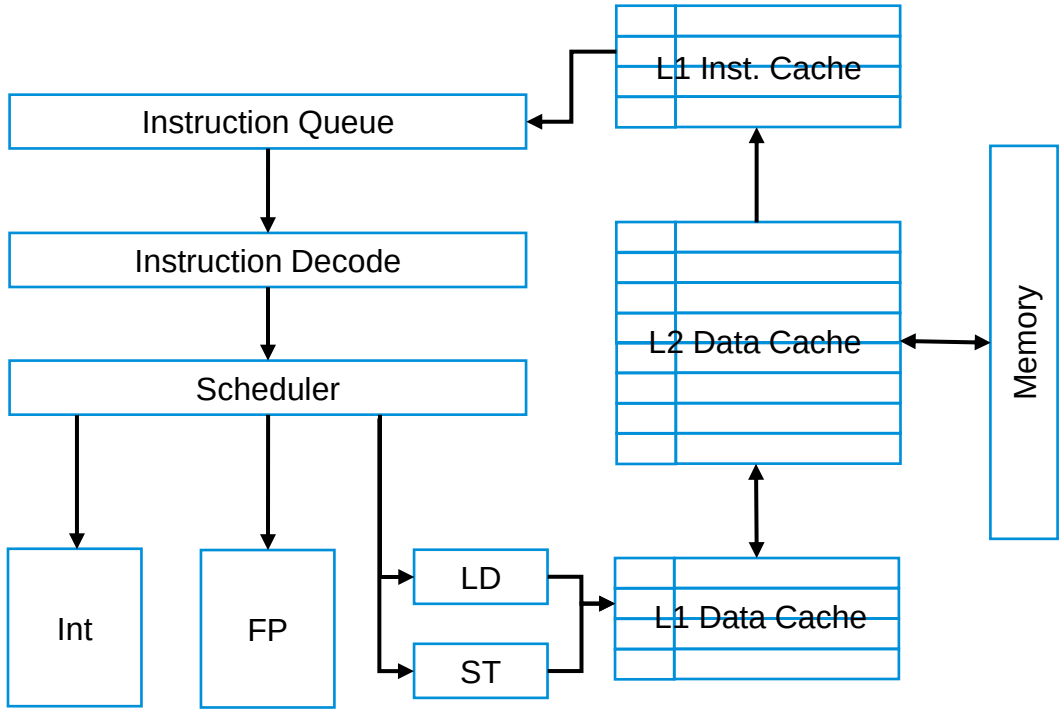
C Example

```
int foo(unsigned int *a, unsigned int *b)
{
    return *a + *b;
}
```

Compiled code to execute foo()

```
; %r1 – r4 input registers, %r0 return reg
__foo:
ld [%r1], r5        ; *a -> register 5
ld [%r2], r6        ; *b -> register 6
add %r5, %r6, %r0   ; add,sum in return reg
ret
```

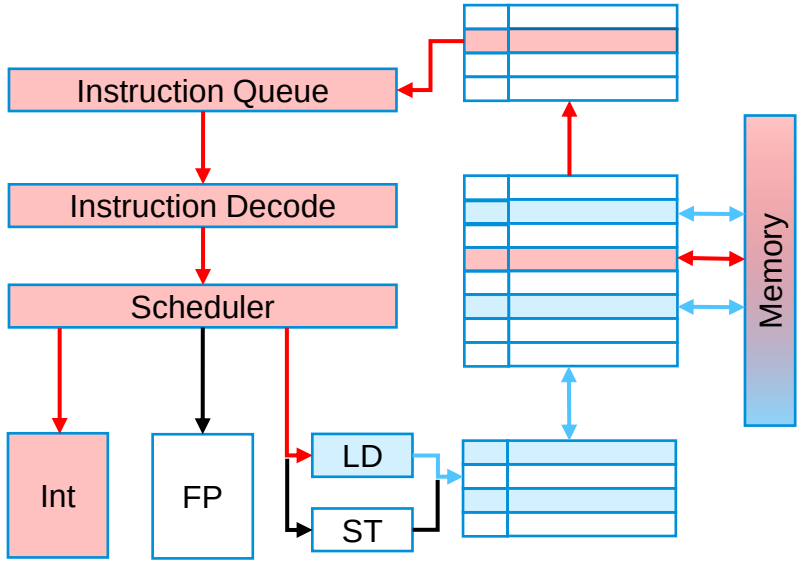
Toy CPU Architecture



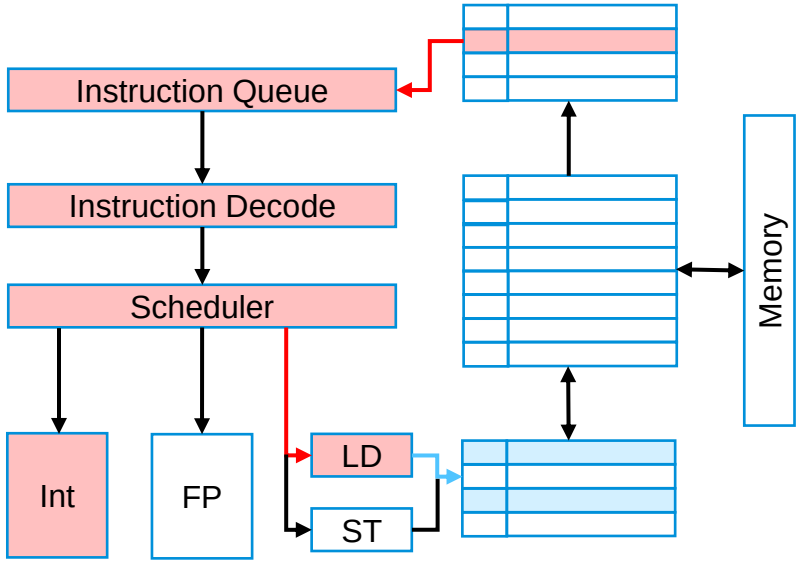
Things I am not including: structures for handing out-of-order instruction dispatch, branch predictors, data and instruction TLBs, last-level caches, memory controller… and many other things.

Parts of a CPU to execute `foo()`

```
__foo:
ld [%r1],%r5
ld [%r2],%r6
add %r5, %r6, %r0
ret
```



First call to `foo()`



Second call to `foo()`
(same arguments)

Instruction Use

Data Use

Architecture for SFQ-based Computing

ISCA Tutorial 2023

Design and Integration of Superconductive Computation for Ventures beyond Exascale Realization

The success of CMOS has overshadowed nearly all other solid-state device innovations over recent decades. With fundamental CMOS scaling limits close in sight, the time is now ripe to explore disruptive computing technologies. As a viable post-CMOS computing technology, superconductor electronics can deliver ultra-high performance and energy efficiency at scale, thereby paving the way for seminal innovations in integrated electronics, sustainable exascale computing, and acceleration of machine learning. This half-day tutorial will cover the challenges and opportunities associated with the first-time design of a superconductive system of cryogenic computing cores (SuperSoCC), which is the focus of a recently awarded NSF Expeditions in Computing award. To demonstrate SuperSoCC, a slew of challenges related to physical scaling, chip-level integration, compact modeling and design tool support, on-chip memory design, architecture design, and full-system design and integration including interfacing to room temperature electronics must be addressed. These issues will be addressed through a series of talks given by experts in the field.

ISCA 2023
Orlando, Florida

Location: Marriott World Center Orlando
Date: Sunday June 18, 2023
Time: 1:30 – 4:30 PM
Agenda:

- 3:10 – 3:45 PM | Murali Annavaram (USC): Architectural Implications of Superconductive Electronics
- 3:45 – 4:20pm | Dilip Vasudevan (LBNL): Design of the Building Blocks for the Superconducting Imaging Processing System Using Temporal Logic

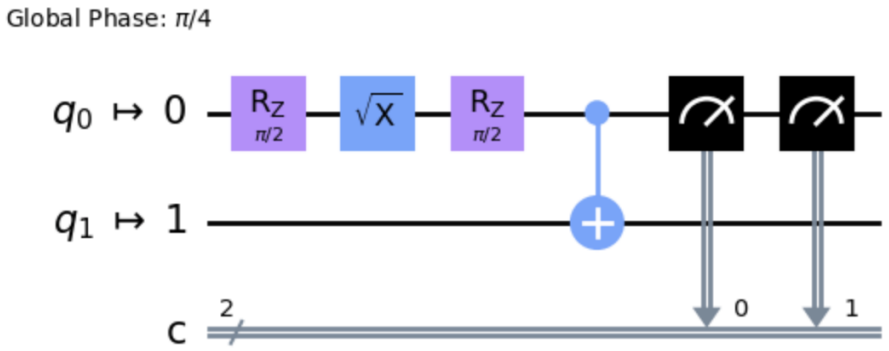
https://discoverexpedition.usc.edu/index.php/isca-tutorial-2023/

Software Enablement Example: Quantum (IBM)

```
from qiskit import qasm3, QuantumCircuit, transpile

# Creating a bell circuit
qc_bell = QuantumCircuit(2, 2)
qc_bell.h(0)
qc_bell.cx(0, 1)
qc_bell.measure(0, 0)
qc_bell.measure(0, 1)

qc_bell = transpile(qc_bell, backend)
qc_bell.draw(output="mpl", idle_wires=False)
```

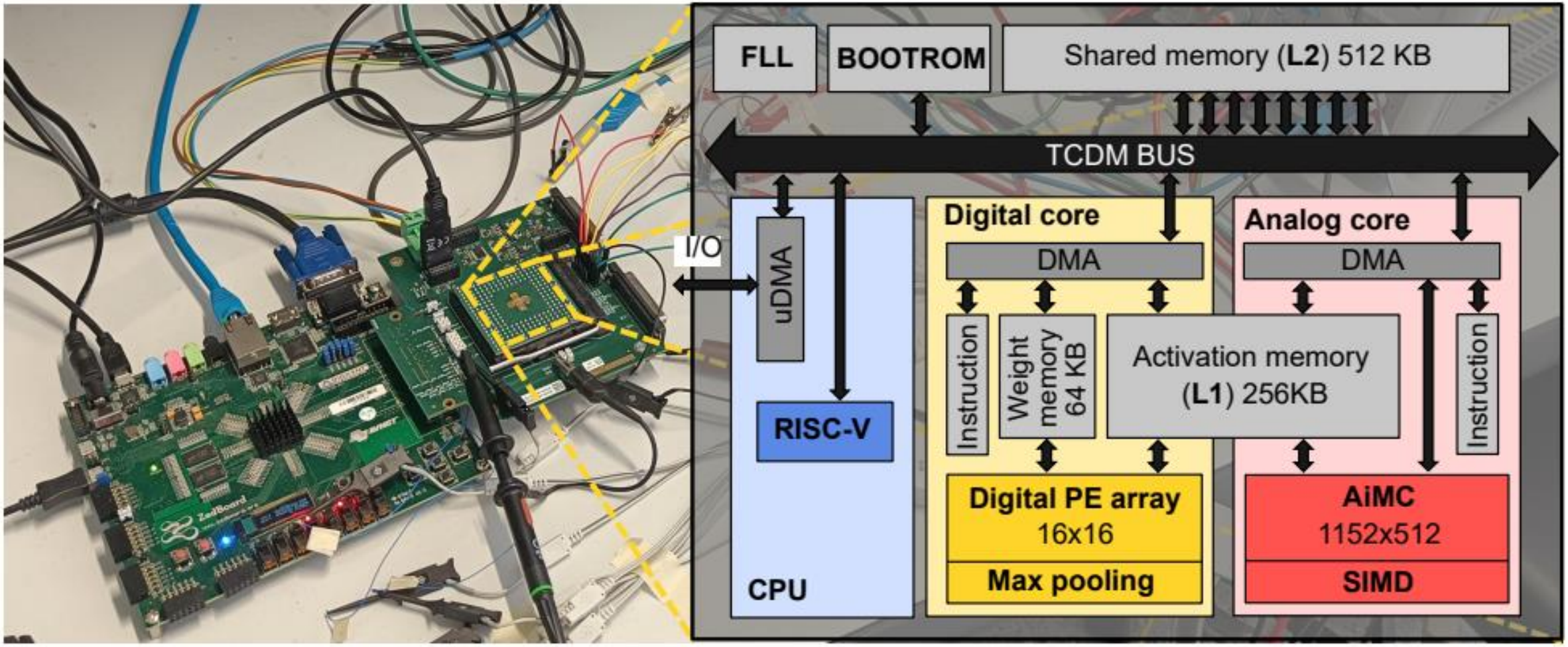


Benchmark	Description	Algorithms
adder	Quantum ripple-carry adder	Quantum Arithmetic
basis_change	Transform the single-particle baseis of an linearly connected electronic structure	Quantum Simulation
basis_trotter	Implement Trotter steps for molecule LiH at equilibrium geometry	Quantum Simulation
bell_state	Bell State	Logical Operation
cat_state	Cat State	Logical Operation
deutsch	Deutsch algorithm with 2 qubits for $f(x) = x$	Hidden Subgroup
dnn	Quantum Deep Neural Network	Quantum Machine Learning
fredkin_n3	Fredkin gate benchmark	Logical Operation
qec_dist3	Error correction with distance 3 and 5 qubits	Error Correction
grover	Grover's algorithm	Search and Optimization
hs4	Hidden subgroup problem	Hidden Subgroup
inverseqft	Performs an exact inversion of quantum Fourier tranform	Hidden Subgroup
iSWAP	An entangling swapping gate	Logical Operation
linearsolver	Solver for a linear equation of one qubit	Linear Equation
lpn	Learning parity with noise	Machine Learning
pea	Phase estimation algorithm	Hidden Subgroup
qaoa	Quantum approximate optimization algorithm	Search and Optimization
qec_sm	Repetition code syndrome measurement	Error Correction
qec_en	Quantum repetition code encoder	Error Correction
qft	Quantum Fourier transform	Hidden Subgroupe
qrng	Quantum Random Number Generator	Quantum Arithmetic
quantumwalks	Quantum walks on graphs with up to 4 nodes	Quantum Walk
shor	Shor's algorithm	Hidden Subgroup
toffoli	Toffoli gate	Logical Operation
teleportation	Quantum Teleportation	Quantum Communication
jellium	Variational ansatz for a Jellium Hamiltonian with a linear-swap network	Quantum Simulation
vqe_uccsd	Variational Quantum Eigensolver with UCCSD ansatz	Search and Optimization
wstate	W-state preparation and assessment	Logical Operation

Cross, Andrew, et al., *OpenQASM 3: A Broader and Deeper Quantum Assembly Language*, ACM Transactions on Quantum Computing, Volume 3 Issue 3, September 2022.

Li, Ang, et. al, *QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation*, https://arxiv.org/abs/2005.13018

KU Leuven: Prototyping Neuromorphic Compute



Multi-core ML Platforms and Custom Compilation Infrastructure -- https://micas.esat.kuleuven.be/research/domains/hardware-efficient-ai-and-ml

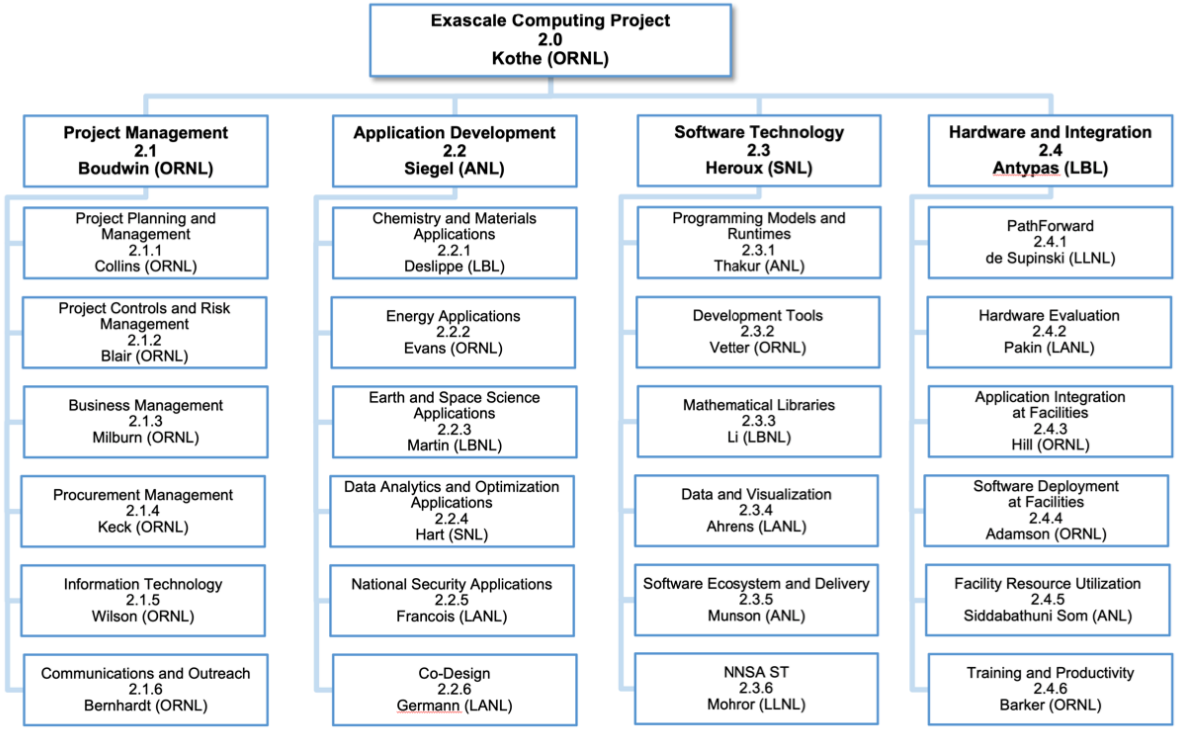
Inspiration: Exascale Reading List

ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems

Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

September 28, 2008

https://people.eecs.berkeley.edu/~yelick/papers/Exascale_final_report.pdf



ECP Software Technology Capability Assessment Report
https://www.exascaleproject.org/wp-content/uploads/2021/01/ECP-ST-CAR-v2.5.pdf

Driving Use Cases across Multiple Domains

Domain	Software	Benchmarks
AI/ML	• Frameworks • ML Compilers • Integrating new AI/ML accelerators • Data prep techniques	• MLCommons benchmarks • NeuroBench • DataPerf • Domain-specific • Training/Inference perf tests Models for Science?
Cloud (“Datacenter Tax”)	• Open Source • ”Cloud” versions of enterprise apps • REST API services	• Fleet Bench (Google) • Others should be coming out soon
HPC	[Pick some target kernels]	[based on kernels picked]
Enterprise	• Enterprise-class Database • In-Memory Databases • Back-Office Applications • Supply Chain • CRM	• TPC Benchmarks (C, E, H, DS) • SpecJBB • SpecVIRT/Vm • VMmark Virtualization

What domains are we missing?

Micron

Software Improving Power Use

Optimize Software/Model Development

- Better performance tools: reduce time to optimization
- Tools to optimize for power or identify high-lower code
- AI-assisted software development wizards
- Curate better datasets to train more efficiently
- Optimize model development
- AI-assisted algorithmic exploration

Enabling New or Old Architectures

- Neuromorphic
- Differential Analyzer
- Reservoir Computing
- Ising Rings in CMOS
- Silicon Photonics Analog (AI Inference)
- Content Addressable Processors
- Logic circuits in yeast
- *Architectures yet to be proposed*

Education for Software Optimization

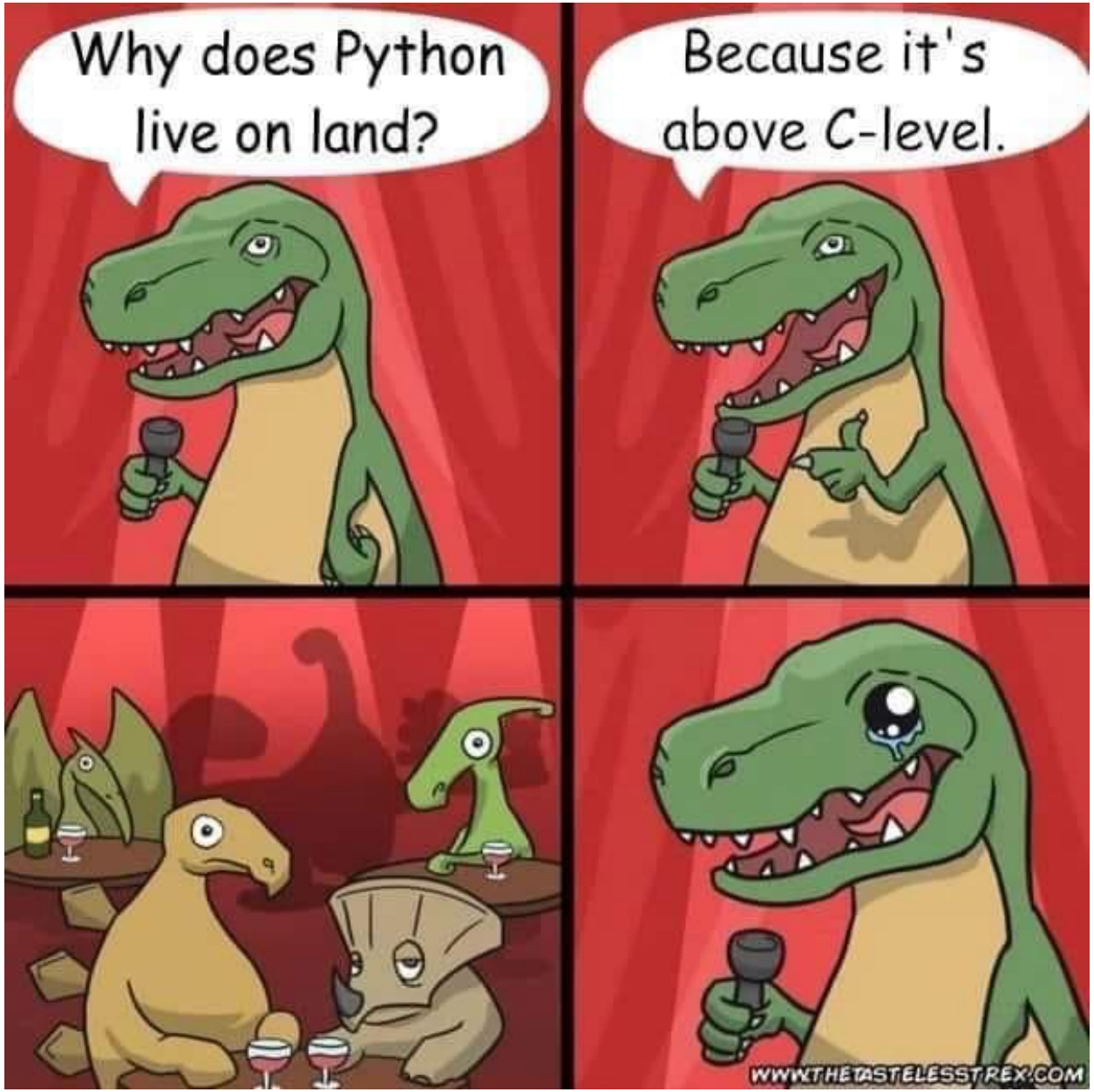
- Software Performance Engineering (e.g., MIT 6.016)
- Classes or contests to design applications (and systems) to minimize power use

Managing Infrastructure Power

- Infrastructure software for managing resources for power-efficiency
- Runtime management of jobs

Summary

- Software optimization and algorithm happens continually and has been happening for a long time
- Specifying the interaction between software and novel hardware architectures is vitally important to enable new devices/new materials to find use cases with material benefits and understand impact on overall system design
- Any hardware solution has a tension between specificity and generality
- Software and Algorithms will help to enable new hardware, but need help to speed up on existing hardware



https://twitter.com/BeingHorizontal/status/1152797848850710531

References

Pereira, Rui, et al., *Ranking Programming Languages by Energy Efficiency*, Science of Computer Programming, volume 205. Elsevier, 2021.

Benchmarking and Workload Characterization, https://www.nersc.gov/research-and-development/benchmarking-and-workload-characterization/ (broken link)

Lee, Victor, et al., *Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU*, ISCA '10: Proceedings of the 37th annual International Symposium on Computer Architecture, June 2010, https://doi.org/10.1145/1815961.1816021

Anonymous, *Mojo – a new programming lanuage for all AI developers*, https://www.modular.com/mojo.

Cinder Github page, https://github.com/facebookincubator/cinder.

Codon Github page, https://github.com/exaloop/codon.

Bentley, Jon, *Programming Pearls: Perspective on Performance*, Communications of the ACM, November 1984, Volume 21 Number 1, https://dl.acm.org/doi/pdf/10.1145/1968.381154.

Bentley, Jon, *Programming Pearls: The Envelope is Back*, Communications of the ACM, March, 1986, Volume 29 Number 3. https://dl.acm.org/doi/pdf/10.1145/5666.315593.

Jon Bentley, MIT, 6.172 (now 6.106) *Tuning a Traveling Salesman Problem Algorithm*, https://www.youtube.com/watch?v=SS5KfIFzfEE.

Cross, Andrew, et al., *OpenQASM 3: A Broader and Deeper Quantum Assembly Language*, ACM Transactions on Quantum Computing, Volume 3 Issue 3, September 2022.

Li, Ang, et. al, *QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation*, https://arxiv.org/abs/2005.13018.

McSherry, Frank, et al., *Scalability! But at what COST?*, 15th Workshop on Hot Topics in Operating Systems, 2015, https://www.usenix.org/system/files/conference/hotos15/hotos15-paper-mcsherry.pdf.

Dean, Jeffrey, Barroso Luiz André, *The Tail at Scale*, Communications of the ACM, Volume 56, No. 2, February, 2013, https://dl.acm.org/doi/pdf/10.1145/2408776.2408794.

Kanev, S, et al., *Profiling a warehouse-scale computer*, 2014, ISCA '15 Proceedings of the 42nd Annual International Symposium on Computer Architecture, https://research.google/pubs/pub44271/.

Lucas, *Ising formulations of many NP problems*, Frontiers in Physics, vol. 2, 2014. [Online]. Available: https://doi.org/10.3389%2Ffphy.2014.00005.

Foster, Caxton C (1976), *Content Addressable Parallel Processors*, Van Nostrand Reinhold, ISBN 0-442-22433-8.

Gander, M., Vrana, J., Voje, W. et al. Digital logic circuits in yeast with CRISPR-dCas9 NOR gates. Nat Commun 8, 15459 (2017). https://doi.org/10.1038/ncomms15459.


Micron®