⬅️ Connect with CASS
https://tinyurl.com/2024-CASS-BOFS

The Consortium for the Advancement of Scientific Software

June 11 – 13, 2024

https://cass.community/bofs

Announcing CASS

The Consortium for the Advancement of Scientific Software

CASS Basics

- A newly-formed organization
- Sponsored by DOE Office of Advanced Scientific Computing Research (ASCR)
- Established by DOE Software Stewardship Organizations (SSOs)

CASS Goals

- Forum for SSO collaboration and coordination
- Bigger than the sum of its parts
- Vehicle for advancing the scientific software ecosystem

CASS Status

- Defining governance structure
- Establishing community awareness
- Building a team of teams
- Collaborating on outreach

Software Stewardship Organization (SSO) Basics

- Each SSO represents a specific software ecosystem concern
- **Product SSOs:** Programming systems, performance tools, math packages, data/viz packages
- **Portfolio SSO:** Curating & delivering software stack to the community
- **Community SSOs:** Workforce, partnerships

Engage with CASS

- Participate in June 11-13 CASS Community BOF Days: https://cass.community/bofs
- Visit https://cass.community

8 Software Stewardship Organizations (SSOs)

DOE Office of Advanced Scientific Computing Research (ASCR) Post-ECP Projects

COLABS
Training, workforce development, and building the RSE community

CORSA
Partnering with foundations to provide sustainable pathways for scientific software

FASTMATH
Stewardship, advancement, and integration for math and ML/AI packages

PESO
Stewarding, evolving and integrating a cohesive ecosystem for DOE software

RAPIDS
Stewardship, advancement, and integration for data and viz packages

S4PST
Stewardship, advancement and engagement for programming systems

STEP
Stewardship, advancement of software tools for understanding performance and behavior

SWAS
Stewardship and project support for scientific workflow software and its community

Exploring the Landscape of AI and ML in Compiler Development: Pros and Cons

Speakers
Mircea Trofin, Google
William Moses, UIUC
EJ Park, Qualcomm
Aiden Grossman, UC Davis
Sunita Chandrasekaran, U Delaware
Gokcen Kestor, PNNL

Moderator:
Johannes Doerfert, LLNL

June 11, 2024

Mircea Trofin, Google

Value statements / trade-off analysis are in a context

My context: LLVM, production, data center binaries

How much can we rely on models?

How much can we rely on an advice from a stranger?

(depends… e.g. on consequences; maybe also track record?)

Compiler construction & ML:

+ Cleaner separation of correctness vs policy
+ Stronger feedback signal for optimizations
+ Found unexpected “holes” / blind spots (in LLVM)

AI: Reviewing vs authoring

- different skills

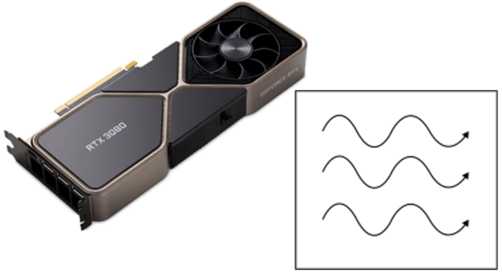
- can one deeply learn something (“grok”) without authoring?

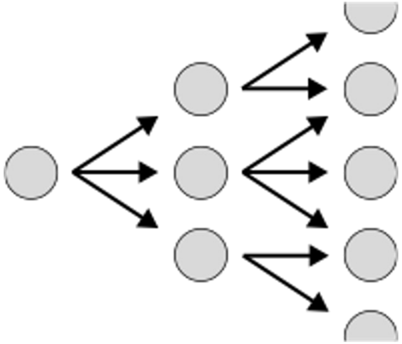
William Moses, Optimization Science Lab @ UIUC

How do we represent and transform programs to enable ***anyone*** to leverage the latest in HPC/ML/etc?

>100x speedup!

Prior:
5 days (cluster)
Enzyme-Based:
1 hour (laptop)





Efficient Differentiation (Training) of Arbitrary Programs [1] [2] [3]

Synthesize GPU & parallel programs with Polygeist/MLIR [4] [5] [6]

Use ML to discover the fastest programs [7] [8] [9]

[1] Instead of Rewriting Foreign Code for Machine Learning, Automatically Synthesize Fast Gradients. NeurIPS ’20.
[2] Reverse-mode automatic differentiation and optimization of GPU kernels via Enzyme. SC'21
[3] Scalable Automatic Differentiation of Multiple Parallel Paradigms through Compiler Augmentation. SC'22
[4] High-Performance GPU-to-CPU Transpilation and Optimization via High-Level Parallel Constructs. PPoPP'23
[5] Polygeist: Raising C to Polyhedral MLIR. PACT'21
[6] Retargeting and Respecializing GPU Workloads for Performance Portability. CGO'24
[7] AutoPhase: Compiler Phase-Ordering for HLS with Deep Reinforcement Learning. MLSys '20.
[8] ComPile: A Large IR Dataset from Production Sources. arxiv'24
[9] Enabling Transformers to Understand Low-Level Programs. HPEC'22

Currently taking students!

Why is AI so successful now (and not 20 years ago)?

How do we emulate that success in program optimization?

…and push even further?

“Good Old Fashioned AI” aka Symbolic AI

Can we build AI by writing a sufficiently expressive set of rules?

Analyzing language by modelling stages of language (tokenizing, features, etc)

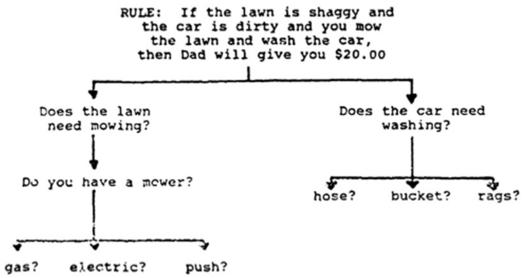
Parse Tree

Sophisticated image filters

Canny Edge Detection (1986)

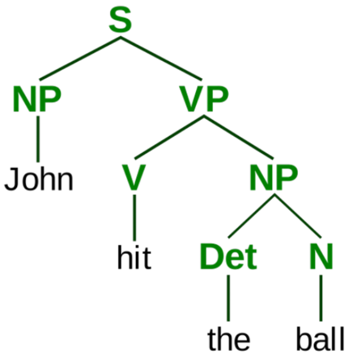
“Good Old Fashioned AI” aka Symbolic AI

Can we build AI by writing a sufficiently expressive set of rules?



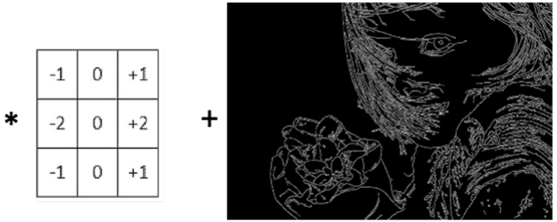
Analyzing language by modelling stages of language (tokenizing, features, etc)

Parse Tree



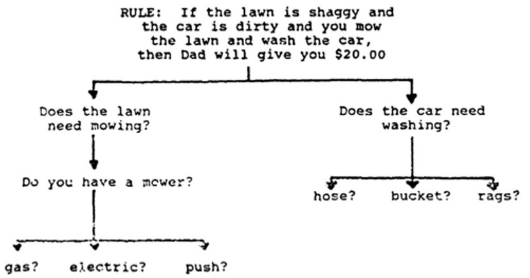
Sophisticated image filters

Canny Edge Detection (1986)



“Good Old Fashioned AI” aka Symbolic AI

Can we build AI by writing a sufficiently expressive set of rules?

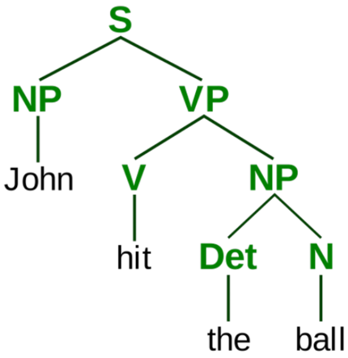


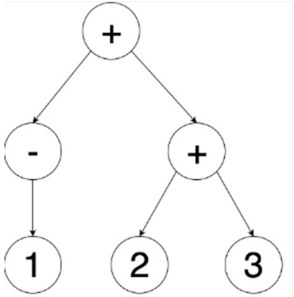

```
y(i) = A(i,j) * x(j)
```

```
for (int32_t i0 = 0; i0 < ((A1_dimension + 31) / 32); i0++) {
    for (int32_t i1 = 0; i1 < 32; i1++) {
      int32_t i = i0 * 32 + i1;
      double tjy_val = 0.0;
      for (int32_t jA = A2_pos[i]; jA < A2_pos[(i + 1)]; jA++) {
        int32_t j = A2_crd[jA];
        tjy_val += A_vals[jA] * x_vals[j];
      }
      y_vals[i] = tjy_val;
    }
  }
}
```

Analyzing language by modelling stages of language (tokenizing, features, etc)

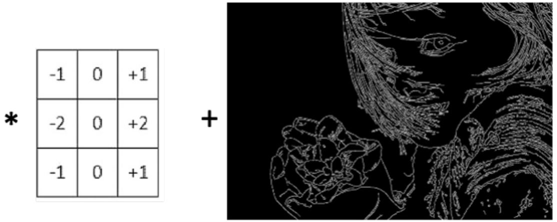
Parse Tree





Sophisticated image filters

Canny Edge Detection (1986)




```
f1 has 100 instructions
f2 has 10 instructions
f3 has 1000 instructions
```

GOFAI lost to current “gen-AI” wave because running more *unstructured* training cycles is cheaper than writing more rules.

Compiler researchers (myself included) are correctly embracing these techniques but….

limited by structured data & correctness guarantees (not 97% accuracy)

How do we combine the best of neural ***and*** symbolic reasoning:

- transformations
- program representation
- data

EJ Park, Qualcomm

Let ML improve (ML) Compilers:

- **Multiple Objectives:** Need of faster and smaller code on small devices is becoming increasingly important. (e.g., Inferences on devices)
- **Adaptive Learning/Transfer Learning:** ML models that can adapt to new SW/HW changes instead of collecting new training data and retraining.

Challenges:

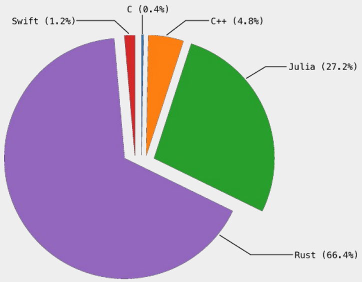
- **Human readability** becomes more challenging as ML models become more complex and elaborated.
- **Integrating ML models** into diverse environments remains challenging.
    - e.g., using PyTorch ML model within compilers written in C++

Aiden Grossman, UC Davis

Datasets:

Programming Language	Bitcode (GB)	Deduplicated Bitcode (GB)	Licensed Bitcode (GB)	Licensed Text (GB)
C	16	8	2	10
C++	109	74	29	103
Julia	200	184	164	1088
Rust	656	580	400	1524
Swift	8	7	7	36
Total	**990**	**853**	**602**	**2761**

https://github.com/llvm-ml/llvm-ir-dataset-utils



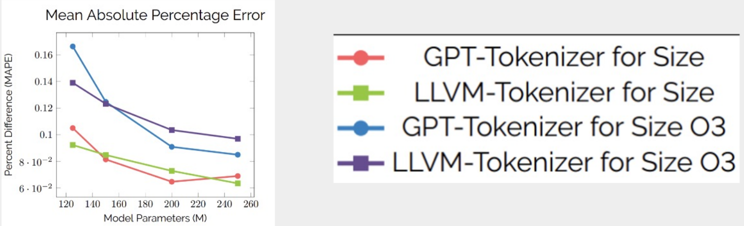
https://huggingface.co/datasets/llvm-ml/ComPile

LLMs for IR:

Model Performance

Parameters	Tokenization	Task	Percent Error
450M	GPT	Size	5.0%
250M	LLVM	Size	6.3%
450M	GPT	Size O3	6.3%
250M	LLVM	Size O3	9.7%

Table 2. Performance of best performing models (that have been trained so far) in each category.



Cost Modeling:

BB Count	MAPE
1M	14.3%
2.5M	5.5%
10M	5.8%
10M[1]	4.7%

- 1B+ BBs from ComPile
- SOA results on znver2

https://github.com/google/gematria

Sunita Chandrasekaran, U Delaware

Building validation and verification testsuites using LLMs

- Automate the process of manual tests generation as the specifications evolve
- Currently focusing on directive-based programming model
- Used several prompt-engineering techniques, parameter-efficient fine-tuning (peft) with low rank adaptation (lora), i.e. freezing model weights and training small additional layers
- Generated 35 testsuites, over 5000 tests
  - Deepseek’s Deepseek-coder-33b-instruct, Meta’s Codellama-34b-Instruct, Phind’s Codellama-34b-v2, GPT-3.5-Turbo and GPT-4-Turbo and fine-tuned all except the last one

Open Questions

- Metric for accuracy of test beyond human analysis?
- Building a larger and more relevant dataset?
- Pre-training an open-source LLMs with corpus of relevant data?
- Train with reinforcement learning using a reward function?

Pre-print: https://arxiv.org/abs/2310.04963 (Accepted to FGCS journal, 2024)
GitHub: https://github.com/chrismun/LLM4VV

Gokcen Kestor, PNNL

The next Big Thing:

- Most ML training/inference is based on dense model/data structures/computing.
- Cost of dense attention grows quadratically with the query length, it is essential to embrace sparse methods, including graph-neural networks and recommender systems
- Current ML systems lack of expressing sparse ML models, only a few handwritten sparse operations.
- We need compiler infrastructures to support the increasing adaptation of sparse methods in ML framework

Compilers for sparse AI models

- Compilers take advantage of sparsity and support some key functionalities such as tile and fuse sparse kernels,
- Some recent efforts to develop compilers and libraries for sparse AI computation (TACO, PyTorch.sparse, cuSparse, etc.)

The COMET compiler support efficient generation of spare computation kernels: https://github.com/pnnl/COMET

- Multiple sparse storage formats (COO, BCSR, …) and code generation for combination of those - users do not need to specify the data structures of computed tensors
- Graph oriented operators (semiring, masking, etc.)
- Sparse optimizations (mixed mode kernel fusion, masking, etc.)

Questions:

- What are the challenges to support for distributed computation as well as support for targeting domain-specific hardware?
- How do we ensure that code can adapt to different architectures? GenAI?
- How do we integrate domain-knowledge in the compiler code generation (pragmas? Intermediate artifacts? directives?)

Panel — Please unmute & ask questions!