Gematria/GRANITE Learned performance models

Ondrej Sykora
Compiler Research+Operations Research, Google

Scoring/Rewards

How do we tell code A is better than code B?

- Should be easy: it runs faster, uses less memory, ...

But how do we actually tell? Benchmarks?

- Useful and important for deployment.
- Must be written and maintained along with the code.
- Sadly, impractical for compilation.

A more practical alternative?

- Static *performance models*!

Code performance models from 30k feet

Performance model = an approximation of a benchmark

- Instead of running the code, analyze it statically

They have some disadvantages:

- Less precise than a benchmark

But also a lot of advantages:

- Deterministic! Always available! Safe!
- And also, faster than an actual benchmark.

A closer look

Typically, oriented at basic blocks

- Basic block = sequence of instructions with no control flow
- Simulate/analyze the flow of instructions through the CPU pipeline, measure simulated (inverse) throughput.

Simplifying assumptions:

- Code running in a tight loop
- Ideal memory access patterns: no cache misses
- Ideal execution patterns: no branch mispredictions

Practical challenges

CPUs are very complex beasts

- Pipelined execution with many stages
- Out of order execution with many instructions in flight
- Many parts moving at the same time

...that could be documented much better

- Sparsely documented, many details are missing from docs
- Documentation is prose, very few machine-readable docs

Typology of performance models

Analytical models

- A simulator of the target machine.
- Very crude to very precise. May provide explanations.
- Very, very laborious to develop.

Learned performance models

- Machine learning to the rescue.
- Good precision/effort trade-off, limited explanations.
- "Easy" to scale model development, open for automation.

A few existing performance models

- Intel IACA: analytical, Intel-only, discontinued in 2019
- LLVM-MCA: analytical, open-source, part of the LLVM
- uops.info + uiCA: analytical, open-source, very precise
- Ithemal: learned, based on LSTM networks (2018)
- GRANITE: learned, based on Graph networks (2022)

GRANITE model

Graph representation of code

- Natural representation of code
  - A lot of semantic information is embedded in the representation
  - Fewer symmetries than text
- Long tradition in compilers and general computer science
  - Abstract syntax trees
  - Control flow graphs
  - Data flow graphs
  - ...

Graph neural networks

Machine learning models for graphs

- Input: Graph structure + node, edge, and graph feature vectors
- Output: Updated node, edge, and graph feature vectors
- Computation: Fixed number of message passing iterations along the edges of the graph.
- Excellent model for relational data!

Graph neural networks

Machine learning models for graphs

- Input: Graph structure + node, edge, and graph feature vectors
- Output: Updated node, edge, and graph feature vectors
- Computation: Fixed number of message passing iterations along the edges of the graph.
- Excellent model for relational data!

Message passing algorithm

- Initial feature vectors are supplied by the user.
- In each step, node, edge,and graph feature vectors are updated based on values from their "neighbors".
- The update function is a (learned) neural network.

Graph neural networks

Machine learning models for graphs

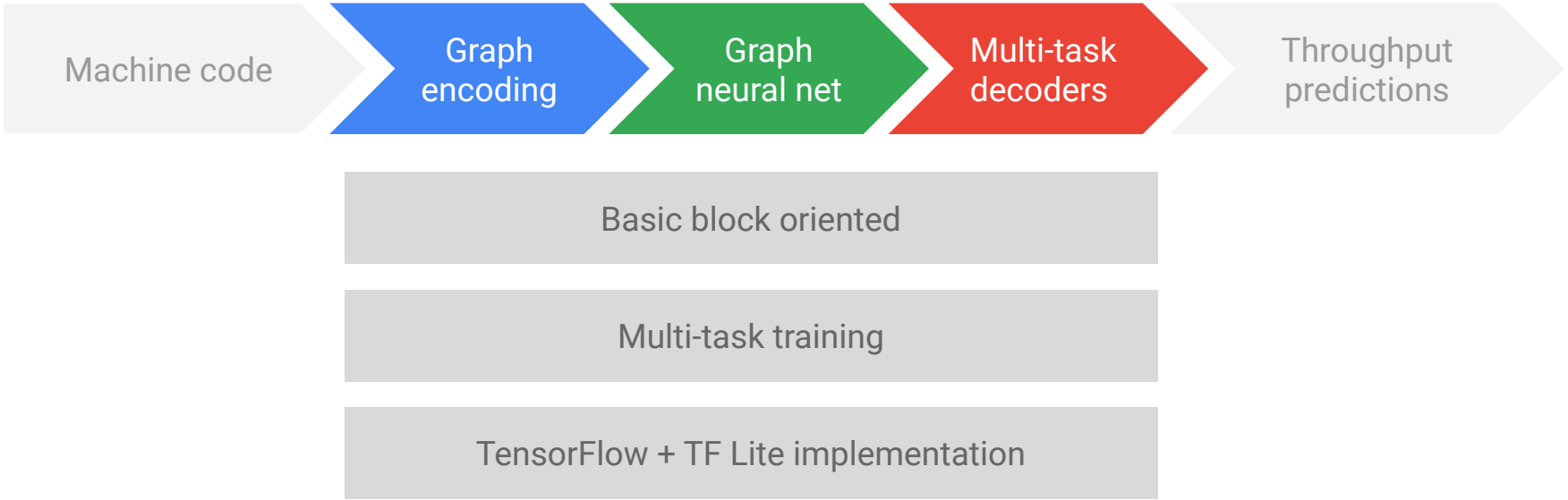
- Input: Graph structure + node, edge, and graph feature vectors
- Output: Updated node, edge, and graph feature vectors
- Computation: Fixed number of message passing iterations along the edges of the graph.
- Excellent model for relational data!

Message passing algorithm

- Initial feature vectors are supplied by the user.
- In each step, node, edge,and graph feature vectors are updated based on values from their "neighbors".
- The update function is a (learned) neural network.

Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks.", 2018.

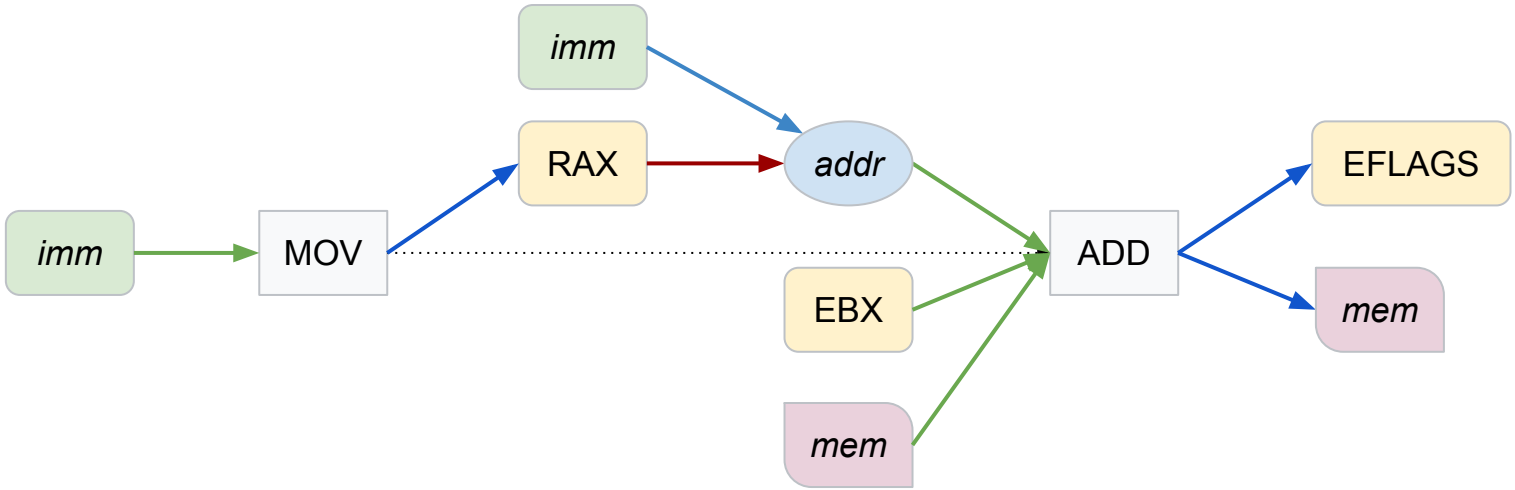
High level GRANITE design



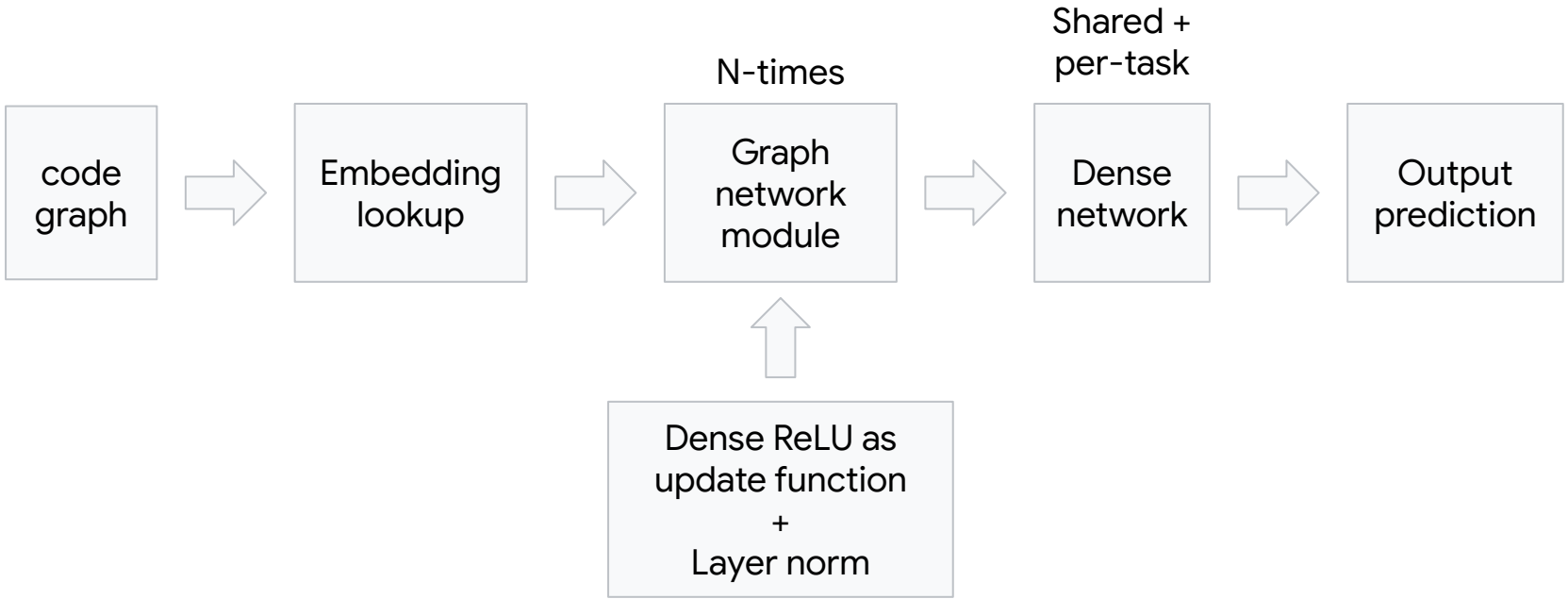
Graph representation of code in GRANITE

```
MOV RAX, 12345
```

```
ADD DWORD PTR [RAX+16], EBX
```



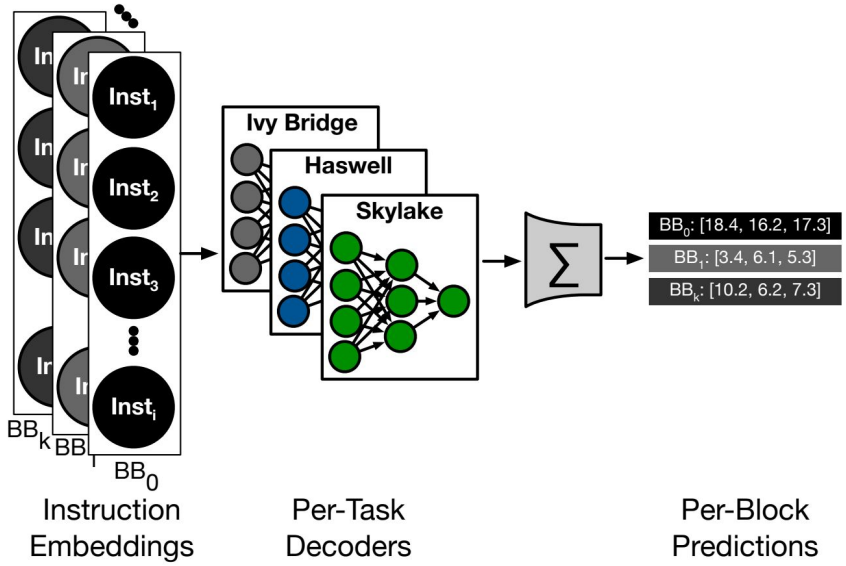
GRANITE model structure



Multi-task decoders

Get more results with less effort (with multi-task learning)

- Task = target microarchitecture
- Multiple microarchitectures per model and data sample
- Graph network computes instruction embedding
- Decoder(s) transform embeddings into per-instruction throughput
- Block prediction = sum of instruction throughputs



Evaluation methodology

Baseline: Ithemal (Mendis et al, 2019):

- A machine-learning based model for throughput prediction
- Two-level Long-short term memory (LSTM) neural networks
- Single task, simple decoder network

Metrics:

- Mean absolute percentage error:
- Training time (per batch)
- Inference time (per batch)

Ithemal+ - extended baseline model

Complex decoder network and multi-task training are independent on graph representation!

We introduce Ithemal+ to evaluate their contributions to model precision

- A two-level LSTM model, similar to original Ithemal, but...
- ... has a complex decoder network
- ... is trained using multi-task training

Prediction precision

Dataset	Microarchitecture	Model	MAPE	Spearman / Pearson
Ithemal	Ivy Bridge	Ithemal	8.34%	0.9640 / 0.2768
		Ithemal+	7.89%	0.9744 / **0.9631**
		GRANITE	**6.67%**	0.9721 / 0.8936
	Haswell	Ithemal*	9.90%	0.9720 / 0.3615
		Ithemal+	8.82%	0.9777 / **0.9231**
		GRANITE	**7.61%**	0.9752 / 0.8255
	Skylake	Ithemal	8.30%	0.9643 / 0.2871
		Ithemal+	7.51%	0.9754 / **0.9035**
		GRANITE	**6.47%**	0.9717 / 0.7888

* We obtained the results by training the models with mean squared percentage error. The accuracy results when training the models with mean *absolute* percentage error was significantly worse.

Effects of multi-task training

Microarchitecture	Model	MAPE (Single-Task)	MAPE (Multi-Task)
Ivy Bridge	Ithemal	8.34%	8.82%
	Ithemal+	8.37%	7.89%
	GRANITE	7.02%	**6.67**%
Haswell	Ithemal	9.90%	9.62%
	Ithemal+	8.87%	8.82%
	GRANITE	**7.76** %	7.82%
Skylake	Ithemal	8.30%	8.77%
	Ithemal+	7.65%	7.51%
	GRANITE	7.34 %	**6.75** %

Computational efficiency

Model	Microarchitecture	GPU training	GPU inference	CPU inference
Ithemal single task	Ivy Bridge	0.0996s	0.0491s	0.0551s
	Haswell	0.1236s	0.0501s	0.0558s
	Skylake	0.0775s	0.0502s	0.0556s
GRANITE single task	Ivy Bridge	0.0354s	0.0147s	0.0749s
	Haswell	0.0367s	0.0147s	0.0750s
	Skylake	0.0349s	0.0146s	0.0751s
Ithemal^{+} multi-task	Ivy Bridge & Haswell & Skylake	0.1086s	0.0515s	0.0602s
GRANITE multi-task	Ivy Bridge & Haswell & Skylake	0.0361s	0.0157s	0.0768s

Closing words

What we learned and achieved

Graph representation and graph neural networks

- An expressive representation that helps the model precision
- Efficient training and inference on GPUs (~3x speedup over LSTM)

A more complex decoder network

- Significant effect on prediction precision; model agnostic

Multi-task training

- Further improve prediction precision/help avoid overfitting
- Additional significant speed-up: A multi-task model is almost as fast to train as one single-task model

What are we doing now?

Open-source development

- Our code is on GitHub, contributions welcome!
- https://github.com/google/gematria
- LLVM ML community on slack (ask Mircea to add you)

Ongoing projects (open-source)

- Public training data set, Function-level performance models: llvm-cm,
- Performance models with additional context

More project ideas (waiting for you! :))

- Transfer learning for performance models
- Support for more architectures & microarchitectures

Thank you for listening!

- Questions? Comments?