Better Performance Models for MLGO Training

EuroLLVM '24

Presented by Viraj Shah; working with Ondřej Sýkora, Mircea Trofin, and Aiden Grossman

Background

Why do we need these cost models?

- Benchmarking is noisy, many runs needed to compensate.
- Benchmarking is also expensive.
- Care needs to be taken to obtain consistent results.

- Better, more accurate cost modeling improved reward signal quality more capable MLGO models.

What are we using now?

- A weighted sum of six code features (e.g. loads and stores).
- Works fairly well for RegAlloc.
- Generally inaccurate.

- State-of-the-art static analysis based and learnt models are also available.
- Generally decently accurate.

What is missing from what we have now?

- All of these assume ideal execution environments.
- Non-ideal runtime events like cache misses and branch mispredictions affect results by an order of magnitude.

- Non-ideal behavior is very hard to model statically.

The Goal

What do we want?

- We need a more dynamic cost model.
- Can use profiling information to give the cost model hints.

- We can achieve this by:
  - Build a data collection pipeline that covers additional runtime information.
  - Modifying learnt cost models so that they can consume this data.
  - Modifying the training and inference processes accordingly.

Metrics

- Standard ML accuracy metrics like MAPE.
- Ordering of blocks by performance.

Methodology

Collecting Runtime Information

- Modern CPUs have Performance Monitoring Units.
- PMU events cover all kinds of runtime phenomena.

- For example, Intel Skylake has[1]:
    - MEM_LOAD_RETIRED.L3_MISS
    - MEM_TRANS_RETIRED.LOAD_LATENCY_GT_128
    - BR_MISP_EXEC.ALL_BRANCHES

[1] https://perfmon-events.intel.com/

A Simple Approach

- Collect cache miss counts.
- Use a simple linear model to find the overhead resulting from misses.
- Essentially multiplying by cost per cache miss.

A Simple Approach

Benchmark memory access patterns like:

```
...
FlushLinkedListFromCache(head);    // "Cold" accesses

Node *current = head;
int sum = 0;
while (current) {
  sum += current->value;
  current = current->next;         // Pointer chasing
}
```

A Simple Approach

- This is not good enough.

- The “cost per cache miss” varies.
- Reasonably accurate when the exact type of access is known.
- Good for the individual “categories”, does not generalize.
- Some categories are not particularly well defined.

A Better Approach

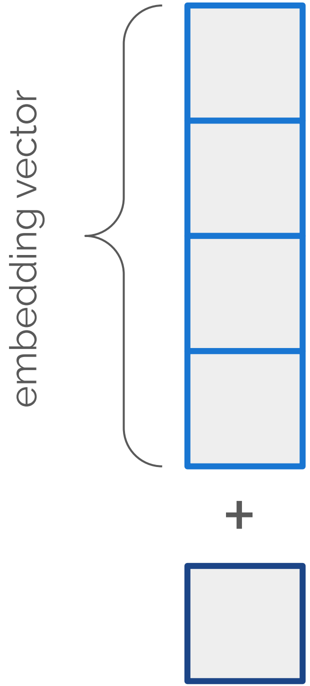
- Models need both static context and runtime information.

- "Base" learnt basic block cost models:
  - Recurrent, like the LSTM-based Ithemal[1],
  - GNN-based, like GRANITE[2].

[1] Mendis et al, "Ithemal", [2] Sýkora et al, "GRANITE"

A Better Approach

- Use this extra information to calculate node embeddings.
- Simply concatenate instruction-representing nodes embeddings with runtime information vector.



Challenges

- Building a large enough dataset with representative cache miss information is a huge task.
- The data collection pipeline isn't suited for building datasets of this scale.

- Possible solution: fine-tuning with runtime information.

Future Directions

- Expand to other runtime behaviors.

- Use basic block predecessor frequencies/execution traces and supply them to the models as well.

Questions?

Also, feel free to contact me:

- shahvirajbiren@gmail.com
- https://www.linkedin.com/in/viraj-b-shah/
- https://github.com/virajbshah/