

Deep Learning-based Approximate Graph-Coloring Algorithm for Register Allocation

- Dibyendu Das, Shahid Asghar Ahmad, Venkataramanan Kumar

LLVM-HPC2020 Workshop

Nov 12, 2020 • Atlanta, Georgia



## Agenda

- Basic Concepts and Motivation
- Our DL-method using LSTM
- LLVM's GRA vs our method
- Results
- Conclusion and Future Work



## What is Graph-Coloring ?

- Graph-coloring is an important problem in CS with numerous applications
- A graph coloring is an assignment of labels, called colors, to the vertices of a graph such that no two adjacent vertices share the same color
- The chromatic number X(G) of a graph G is the minimal number of colors for which such an assignment is possible
- Solved using heuristics

A graph coloring for a graph with 6 vertices. It is impossible to color the graph with 2 colors, so the graph has chromatic number 3.





### **Register Allocation as a Graph-Coloring Problem**

- Register Allocation is an important problem in the area of compiler code generation
- Usually the number of registers available < number of variables used
- Create what is known as the \*interference graph\* which models registers which need to be \*live\* at the same time





#### **Motivation for a DL-based Register Allocation Algorithm**

- Register allocation (RA) is an important application of graph coloring in compilers
  - Heuristics + lots of tuning
- Idea is to see whether we can build a simple deep-learning model that can color graphs and can be used for RA
- We build an LSTM-based model and experiment with some popular graphs as well as interference graphs generated by LLVM's RA



#### Modeling Graph coloring using LSTMs

- Viewed as a sequence-2-sequence translation via LSTMs
- An input sequence where each item of the sequence corresponds to a node of the graph
  - Each item of the input sequence is an encoding of the adjacency vector of that vertex
- The output sequence is of the same length as the input sequence (number of nodes of the graph)
  - Each item of the output sequence is a \*color\* assignment for that node
- Adjacent nodes cannot carry the same color
  - But it is difficult to encode constraints in LSTM
- It is difficult to get perfect coloring for large graphs
  - Even perfect coloring for graphs with several hundred nodes is very time-consuming



### Our LSTM model

- Three stacked layers of LSTM followed by a FC(dense) layer and ReLU
- The figure shows an unrolled LSTM

 $0 \leq \text{Color}(v) \leq n-1$ 



1024 hidden units per LSTM



# **Training the model (1)**

- The aim is to apply such a model for moderate-sized graphs (few hundred nodes)
- Many interference graphs created by LLVM's RA for SPEC CPU 2017 are of similar size
- Use supervised training by generating random graphs and coloring them optimally
  - Used a package called \*nauty\* (<u>http://keithbriggs.info/very\_nauty.html</u>)
  - Generated random graphs up to 100 nodes of varying sparsity (very sparse to very dense)
  - Colored them optimally and generated the color assignments for the nodes
  - Used ~10000 samples for training
- Used 2 LONG INTS (128 bits) to encode the adjacency vertex of a node



# **Training the model (2)**

- One sample of the input+output sequence for training looks like this:
  - <<v0\_first\_64bits,v0\_second\_64bits>,...,<v99\_first\_64bits,v99\_second\_64bits>, < color(v0), color(v1), ..., color(v99)>>
- Sequences may be padded with zeros for graphs having a smaller number of vertices
- The model is trained for 100 epochs
- Up to a training error value of 5%



#### **Inference and Color-correction**

- For inference the input needed is just the adjacency vector encoding
  - < <v0\_first\_64bits,v0\_second\_64bits>,...,<v99\_first\_64bits,v99\_second\_64bits>, 0,0, ..., 0 >
- However there is a big catch with the inference output
  - The coloring assignment may be INVALID
  - Since constraints are not encoded in LSTM adjacent nodes having the same color can happen
- Our solution:
  - Use a post-pass called \*color correction\* to restore the validity of color assignment
    - May need additional colors



### **Color-correction**

- Forest-Fire graph
  - has a chromatic number of 5
- LSTM-based model colors with 4 colors resulting in 2 invalid edges
- These edges are  $\langle v_2, v_3 \rangle$  and  $\langle v_1, v_5 \rangle$
- v<sub>5</sub> can reuse the color c<sub>3</sub> as none of its neighboring nodes use c<sub>3</sub>. v<sub>2</sub> requires a new color c<sub>5</sub> as both v<sub>2</sub> and v<sub>3</sub>'s neighbors use all the colors
- So finally we also get 5 colors 🙄





Inference performance on some popular graphs

- Karate Graph X(G) 5
- LSTM uses 4 colors but 23 invalid edges out of 79 edges which is ~30% of the edges.
  On applying color correction, we use only one extra color resulting in coloring the graph optimally using 5 colors
- Baidu Graph X(G) 3
- LSTM uses 3 colors with 35 out of 90 edges being invalid which is ~38%. We are unable to reach the optimal number of 3 colors. But we can match the coloring number of the best heuristics available today by being able to color with 4 colors (after correction)





### **DL-model vs LLVM's GRA**

- GRA does not maintain an interference graph explicitly
- We create the interference graph at the end of the Live Interval Analysis phase
- The interference graph is then written out in the input format required for inferencing
- We collected the interference graphs for the functions of certain SPEC CPU<sup>®</sup> 2017 benchmarks. We ignored those functions which have more than 100 nodes in the interference graph
- We collect the register count of each function after codegen





# **DL-model vs LLVM's GRA**

# - Some results of mcf and leela

Functions	LLVM reg-alloc	DL before correction	DL after correction
switch_arcs	17	14	22
replace_weaker_arc	16	10	13
insert_new_arc	14	11	15
resize_prob	7	4	7
marc_arcs	12	10	12
refreshPositions	14	14	14
refreshArcPositions	8	4	7
master	23	13	24
worker	25	16	24
markBaskets	11	9	9
primal_bea_mpp	21	14	26
primal_feasible	9	10	10
flow_org_cost	14	7	10
flow_cost	13	8	10
refresh_neigbour_lists	10	6	9
update_tree	19	8	19
primal_start_artificial	11	7	9
primal_imnus	7	5	7
write_objective_value	5	5	8
main	6	3	4
TOTAL	262	174	257

Functions	LLVM reg-alloc	DL before correction	DL after correction
lzma_index_buffer_decode	11	7	9
index_decode	12	13	15
Izma_index_hash_append	10	10	10
lzma_index_hash_decode	14	16	18
lzma_stream_buffer_decode	14	6	11
stream_decode	14	19	19
lzma_stream_footer_decode	8	6	6
lzma_vli_decode	13	8	13
lz_encoder_prepare	13	6	13
lzma_lz_encoder_init	11	7	9
lz_encode	14	10	15
lzma_mf_hc3_find	21	9	19
TOTAL	732	500	715



### An architecture of a DL-based RA

- A plausible setup for LLVM





#### **Conclusions + Future Work**

- Preliminary and exploratory work to check whether an LSTM-based solution is feasible
  - Looks promising
- Baidu (2018) solution using Reinforcement Learning but costly and complicated
- One work using Graph NN (but not for color assignment)
- Can experiment with sequence input orders (ex: Topo-sort, DFS, BFS)
- Can experiment with newer LSTMs like Attention/Transformer
- Can use LLVM's graphs as training inputs rather than random graph

