Learning to Combine Instructions in LLVM Compiler

LLVM Performance Workshop at CGO 2022

Presenters

Sandya Mannarswamy
sandya.mannarswamy@intel.com

Dibyendu Das
Dibyendu.das@intel.com

Motivation

- Instruction Combiner a critical pass in all modern compilers
- Thousands of instruction-combining patterns
- Patterns need to be frequently updated over time as software coding patterns/idioms/applications evolve
- IC is the most frequently updated component in the LLVM compiler [Zhou et al. 2020].
- Considerable human effort, high software maintenance costs

Is it possible to replace traditional IC with a machine learnt model?

Learning to Combine Instructions

- Can we replace the hand-coded rule driven pattern matching IC pass with a machine learnable IC pass?
- Modelled as monolingual machine translation task
- Neural Machine Translation (NMT) translates from source to target language
- Both source and target languages are LLVM Instruction IR
- We leverage neural Seq2Seq models for this task
  - State of art models using LSTMs and Transformers

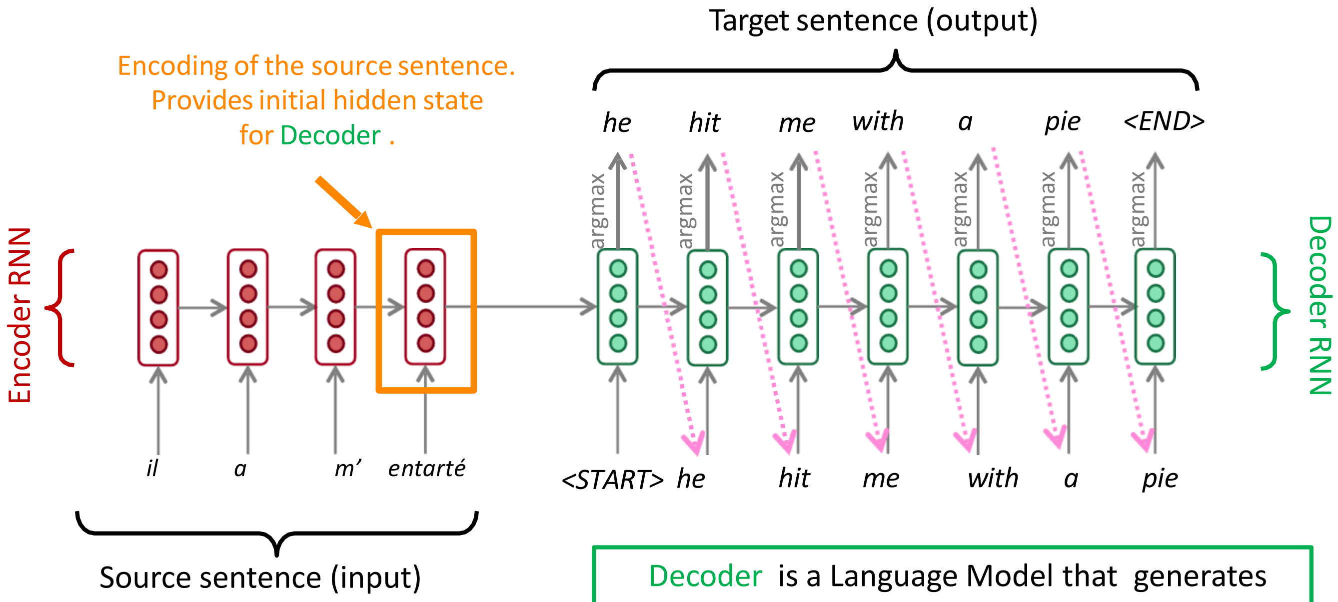
Design choices for NIC

- What should be the input sentence encoding for Seq2Seq model?
- How can we find/build a dataset for this task?
- How do we integrate a machine learnt IC module into the overall pipeline?
- How do we validate the IR generated from NIC?

Neural Instruction Combiner (NIC)

- NIC has three major components
- **NIC inputter**: (non-ML) creates an encoded representation from LLVM IR instruction corresponding to a basic block
- **NIC Converter**: (Seq2Seq Neural network model) takes the output from NIC Inputter and generates an equivalent optimized encoded instruction sequence
- **NIC Outputter**: (non-ML) converts the NIC Converter output back to full-fledged LLVM IR instruction sequence of a basic block. It also performs a set of IR verification checks and translation validity checking

Overview of Seq2Seq Models



Encoder produces an encoding of the source sentence.

Decoder is a Language Model that generates target sentence, *conditioned on* *encoding*.

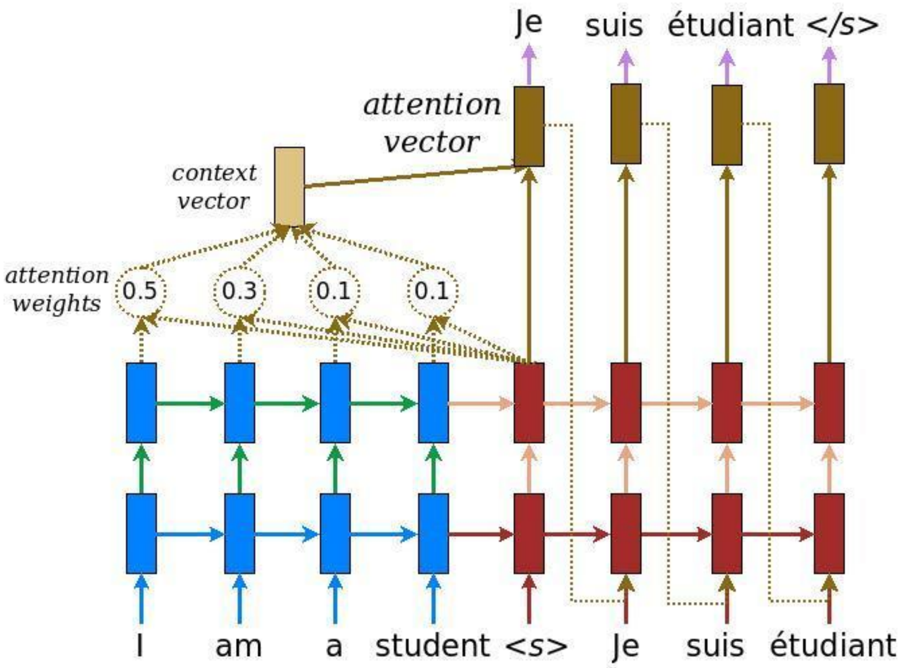
Note: This diagram shows test time behavior: decoder output is fed in ·····> as next step's input

Attention Mechanism

- Vanilla Seq2Seq models have the information bottleneck problem due to single encoder output vector
- Attention provides a solution to the bottleneck problem
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source

Seq2Seq Model with Attention

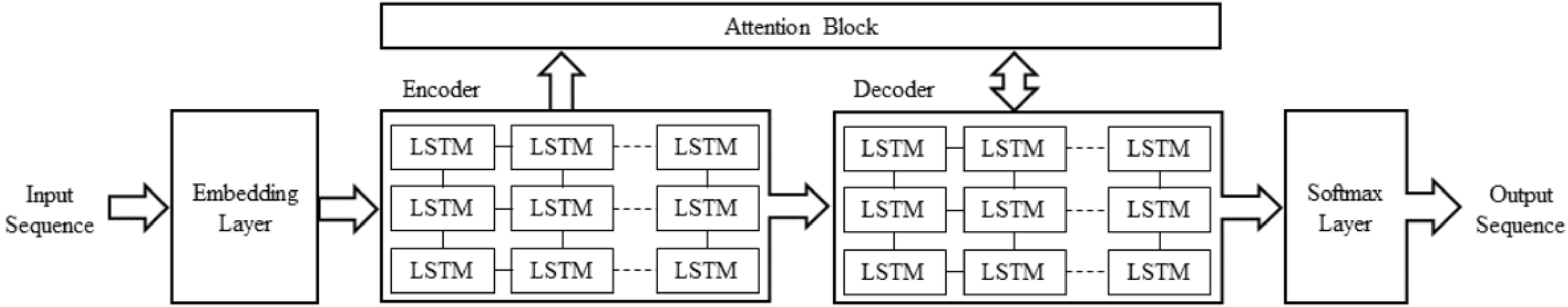


- Seq2Seq models typically contain an encoder, decoder and attention mechanism
- Encoder creates a distilled representation of input.
- Decoder generates the output based on the encoder outputs and each previously generated output symbol
- Attention weights selectively weigh the encoder outputs
- Each encoder/decoder block can be a RNN (LSTM) or a transformer block (Multihead Attention)

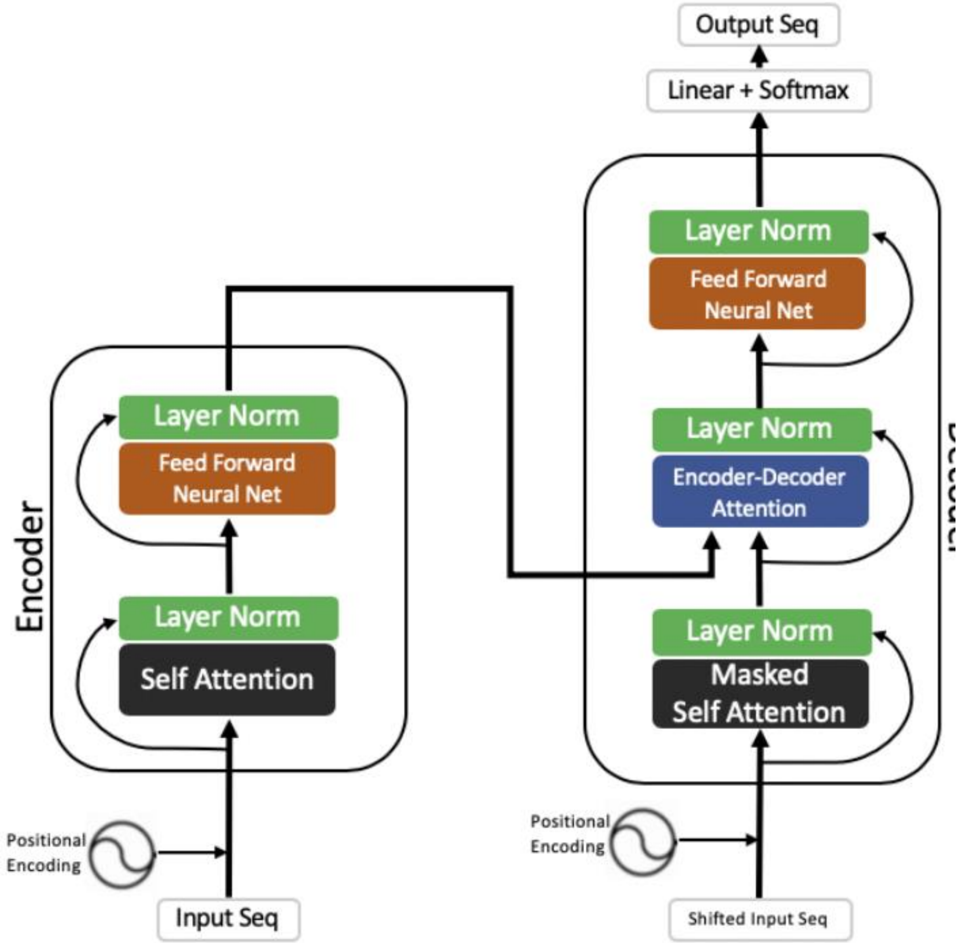
NIC Building Blocks

- NIC inputter is the input (non-ML) module for NIC
  - Creates a distilled representation of the IR instruction sequence for each BB
- NIC Converter (ML Module)
  - Model trained offline and employed in inference mode in optimizer pipeline
  - Two variants: RNN based and Transformer based
  - NIC Converter uses two attention mechanisms
    - Standard attention mechanism of Seq2Seq models
    - A novel Compiler guided attention mechanism
- NIC outputter is the output module (non-ML) for NIC
  - Takes the NIC converter output along with source BB instruction list
  - Validates the instruction stream and emits the optimized IR instruction list

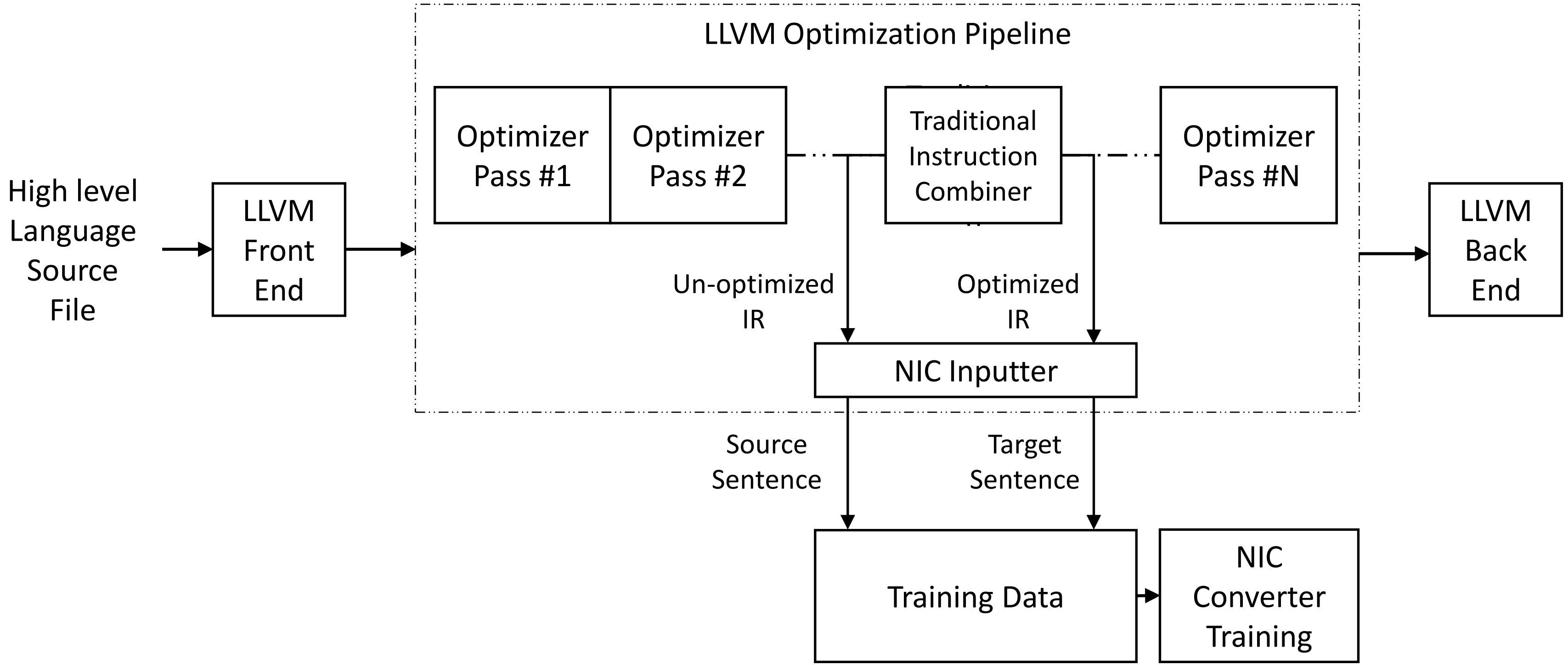
NIC Converter – RNN Based



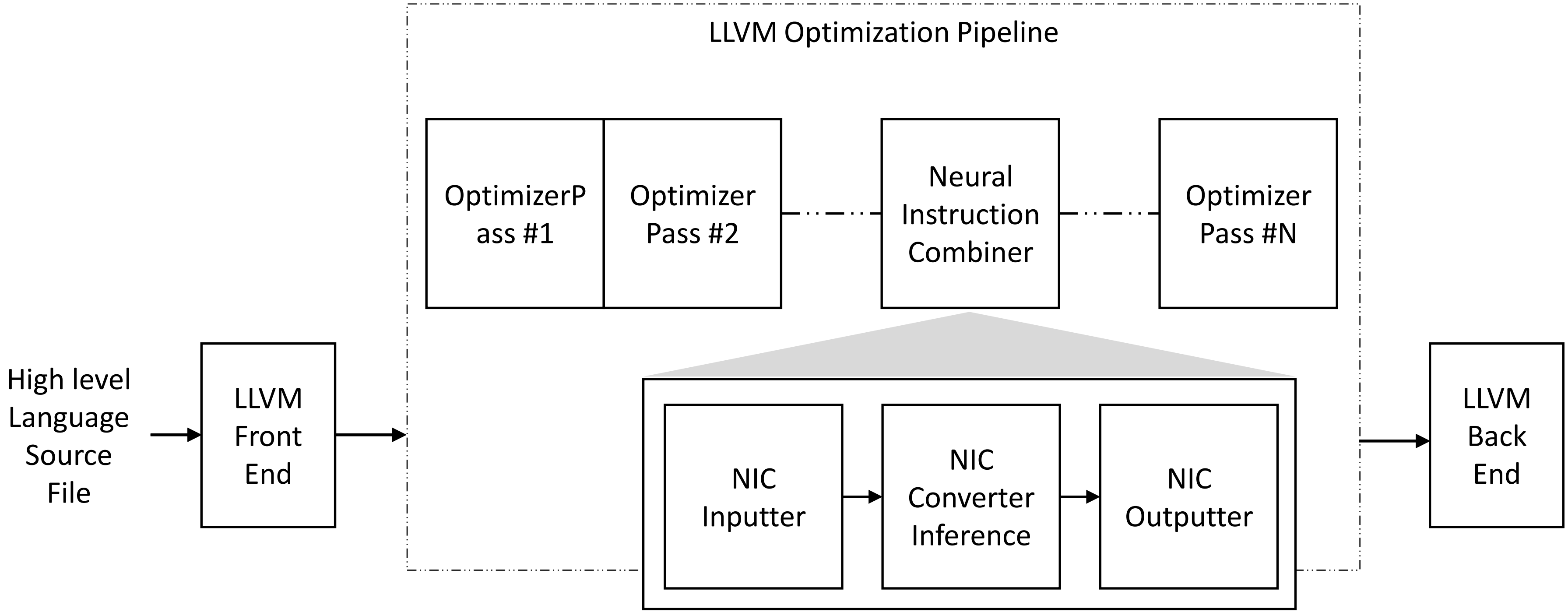
NIC Converter – Transformer Based



NIC Converter Training



NIC Converter Inference



Compiler Guided Attention

- Leverage the compiler knowledge in improving the soft attention alignments
- During training data generation, a compiler guided attention matrix CA is created
- CA matrix terms are fixed attention scores provided by the compiler and are not learnt during training.
- Each element CA[i, j] corresponds to the probability of whether the ih token in target sentence maps to jth token in source sentence.
- Force the learnt attention weights to be closer to CA during the training process
  - by adding an additional loss term to the training objective

Experimental Evaluation

- Created 300K samples dataset from LLVM application test suite & AnghaBench
- Trained the NIC seq2seq models using mini-batch gradient descent
- standard cross-entropy loss and Adam optimizer
- The trained NIC converter was then deployed in inference mode in the optimizer pipeline
- Evaluated with test data set

Model	Description
A	LSTM 3-layer bidirectional stacked encoder with 3-layer unidirectional greedy decoder.
B	Transformer: num_layers = 4, d_model = 128, dff = 512, num_heads = 8, dropout_rate = 0.1
C	Transformer: num_layers = 6, d_model = 512, dff = 2048, num_heads = 8, dropout_rate = 0.1
D	Same as B, with num layers = 2
E	Same as B, with No POS Embedding
F	Same as B, with 16 heads
G	Model A with compiler guided attention
H	Model B with compiler guided attention

Model Performance Metrics

- Standard Machine Translation metrics are Bleu s& Rouge Scores
  - BLEU evaluates the quality of translation, a number between 0 to 1
    - 1 -> machine translation and human translation were identical.
    - Bleu precision evaluated at multiple n-gram level with average across all n-gram levels being reported as a single final score.
  - Rouge-n score represents the n-gram overlap between the machine generated and ground truth reference translations
- Task specific metric is Exact Match (EM) comparison results
  - for each BB between the predicted sequence and the ground truth
  - Reported separately for optimized and unoptimized sequences

Experimental Results

Metric	A	B	C	D	E	F	G	H
Bleu precision	0.93	0.94	0.91	0.93	0.94	0.93	0.93	0.94
Rouge-1 r score	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Rouge-1 p score	0.90	0.90	0.90	0.90	0.90	0.90	0.89	0.90
Rouge-2 r score	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96
Rouge-2 p score	0.91	0.91	0.91	0.91	0.92	0.91	0.91	0.92
Rouge-l r score	0.97	0.97	0.96	0.97	0.97	0.97	0.96	0.97
Rouge-l p score	0.93	0.94	0.93	0.94	0.94	0.93	0.94	0.93
Exact Match (un-opt)	0.93	0.94	0.93	0.93	0.94	0.94	0.93	0.94
Exact Match (opt)	**0.68**	**0.72**	0.71	0.70	0.70	0.71	**0.70**	0.72

Exact Match Error Analysis

- NIC correctly fixes up the uses of the replaced opcode with the newly generated opcode
- For frequent/unique constants (Shift instructions), the model outputs the correct constants
- Mistakes in generating correct values for synthesized constants
  - such as GEP and Alloca operands
  - ends up reproducing the memorized frequent constant values

Type of error	Occurrence
Incorrect Constant	42.3%
Opcode Mismatch	34.9%
Type issue (Sign/Zero extension)	6.7%
Operand swap (canonicalizaton)	1.4%
Others	14.7%

Related Work

- Recent work in applying deep learning techniques to compilers
  - Optimization phase ordering
  - selection of optimization heuristics
  - Compiler cost models

- Building super optimizers for binaries [Bansal 2006]
  - Creating a database of possible optimized sequences from the binaries
  - Limited to X86 binaries
  - Incur high overheads due to huge candidate search space
  - Improving Super Optimizers [Schkufza2013, Bunel 2017]

Open Issues & Future work

- Only 72% of optimization opportunities are realized by NIC
- Correctness checks for NIC generated code sequences
  - IR and CFG Validation Checks
  - Use of ALIVE2 for translation validity checking
  - Automatic NMT post editing techniques/Program repair techniques in future?
- Expanding the dataset for training NIC
  - Currently learning from the traditional IC (behavioral cloning)
  - Leverage super optimizer identified instances in future?