RL4ReAl: Reinforcement Learning for Register Allocation

S. VenkataKeerthy¹, Siddharth Jain¹, Anilava Kundu¹, Rohit Aggarwal¹, Albert Cohen², Ramakrishna Upadrasta¹

IIT Hyderabad¹, Google²

LLVM Performance Workshop
25th February 2023
Register allocation

- Registers are scarce!
  Unbounded set of variables $\rightarrow$ Finite set of registers

- One of the classic NP-Hard problems
  Reducible to graph coloring

- Solutions
  - Constraint-based: ILP and PBQP formulations
  - Heuristic approaches

- LLVM - 4 register allocators
  - Constraint-based: PBQP
  - Heuristic: Greedy, Basic, Fast
LLVM’s Register Allocation Strategies and Heuristics

- No single best allocator
  Greedy performs better in general

- Greedy Allocator Heuristics - Splitting, Coalescing, Eviction and Spilling
- PBQP Allocator Heuristics - Coalescing and Spilling
What makes ML based Register allocation difficult?

- Complex problem with multiple sub-tasks
  - Splitting, Spilling, Coalescing, etc.
- ML schemes should ensure correctness
  - Register type constraints
  - Live range constraints
- Integration of ML solutions with compiler frameworks
  - Python ↔ C++

Proposal - RL4ReAl: Reinforcement Learning for Register Allocation
RL4ReAl: Objectives

Objectives: Machine Learning Framework for Register Allocation

- End-to-end application of Reinforcement Learning for register allocation
- **Semantically correct code generation**
  - Without resorting to a correction phase
  - Correctness constraints imposed on action space
- Multi architecture support

Can an ML model match/outperform *half-a-century old* heuristics?
Constraints in Register Allocation
Register Allocation: Correctness constraints

 Registers are complicated!

 1. Register Constraints
 2. Type constraints
 3. Congruence constraints
 4. Interference constraints
Register Constraints

- Architectural constraints
  - Eg: IDIV32 → Divides contents of $eax; stores result in $eax and $edx

- Register allocation ⇒ Allocating left out virtual registers

```c
1    // Source
2    i = 0
3    x = 10
4    y = 20
5    print x
6    z = y / x
7    i++
8    z = z + 10
9    i++
10   print y
11   print z
12   print i

MOV32ri 0, %i:gr32
MOV32ri 10, %x:gr32
MOV32ri 20, %y:gr32
<call print on %x>
$eax = COPY %y:gr32
<clear $edx>
IDIV32r %x:gr32, implicit-def $eax, implicit-def $edx
%z:gr32 = COPY $eax
%i:gr32 = ADD32ri %i:gr32, 1
...
<call print on %y, %z, %i>
```
Type constraints

- Different types of registers in a register file
  - General purpose registers
  - Floating point registers
  - Vector registers, ...

- Variable type compatibility with the register type

```
1   i = 0
2   x = 10
3   y = 20
4   print x
5   z = y / x
6   i++
7   z = z + 10
8   i++
9   print y
10  print z
11  print i
```

```
MOV32ri 0, %i:gr32
MOV32ri 10, %x:gr32
MOV32ri 20, %y:gr32
<call print on %x>
$eax = COPY %y:gr32
<clear $edx>
IDIV32r %x:gr32, implicit-def ← $eax, implicit-def $edx
%z:gr32 = COPY $eax
%i:gr32 = ADD32ri %i:gr32, 1
<call print on %y, %z, %i>
```
Congruence constraints

- Real-world ISAs have hierarchy of register classes
  - Congruent classes
Interference constraints

Register allocation ⇒ Graph coloring problem

\[
\begin{align*}
x &= 10 \\
y &= 20 \\
\text{print } x \\
z &= 20 + y \\
\text{print } y \\
z &= z + 10 \\
\text{print } z
\end{align*}
\]

Available Registers: R1(Green), R2(Blue)
RL4ReAl: Reinforcement Learning for Register Allocation

- **RL Framework**
  - Node Embeddings
  - Selected Node
  - Split Info
  - Splitting Agent
  - Coloring Agent
  - RL Framework

- **LLVM Environment**
  - Embeddings
  - MLRegAlloc
  - Interference Graph
  - Register Assignment and spilling

- **gRPC Stub**
  - gRPC
  - Update
  - Split Node

- **Source code**
  - Binary
  - Lowering & Optimizations
  - MIR Function

- **MIR2Vec**
  - Node Selector Agent
  - Task Selector Agent
  - Pick Next node
  - OR
  - Color
  - Color Map for all nodes
Interference graphs

Edges: \{\text{phy reg - vir reg}, \text{vir reg - vir reg}\}

Vertices
- MIR instruction representations in the live range of a variable
- Instruction $\rightarrow \mathbb{R}^n$ MIR2Vec embeddings
- Final representation: $\mathbb{R}^{m \times n}$

MIR2Vec representations
- $n$ dimensional vector representation
- Opcode and operand information form the entities in MIR
  $W_o \cdot [O] + W_a \cdot ([A_1] + [A_2] + \cdots + [A_n])$, $W_o > W_a$
Grouping opcodes

- MIR has specialized opcodes
- Based on width, source and destination types
  - 200 different MOV instructions
  - MOV32rm, MOVZX64rr16, MOVAPDrr, etc.
- 15.3K opcodes in x86; 5.4K opcodes in AArch64
  - {build dir}/lib/Target/X86/X86GenInstrInfo.inc
  - {build dir}/lib/Target/AArch64/AArch64GenInstrInfo.inc
- Generic opcodes
  - Specialized opcodes are grouped together
  - {MOV32rx, MOVZX64rr16, MOVAPDrr, ...} → MOV
Representing Interference graphs

- **GGNNs - Gated Graph Neural Networks**
  - Processing graph structured inputs
- **Message passing**
  - Information propagated multiple times across nodes
- **Annotations on nodes → Current state**
  - Visited
  - Colored
  - Spilled
- $\mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{k}$
Hierarchical Reinforcement Learning

● Environment - MLRegAlloc pass in LLVM
  ○ Generates interference graphs + representations
  ○ Register allocation, splitting and spilling as per the prediction

● Multi-agent hierarchical reinforcement learning
  ○ Sub tasks of register allocation → Low level agents

● Agents
  ○ Node selection
  ○ Task selection
  ○ Splitting
  ○ Coloring
**Agents**

**Node Selection Agent**
- Selects the vertex to process next
- **Action space**: Vertices that are not colored
- **Reward**: Based on low-level agents

**Task Selection Agent**
- Selects between split and color
- **Action space**: Split or Color - Split is allowed only if #Uses > k (k = 2)
- **Reward**: Based on low-level agents

**Splitting Agent**
- Predicts the split point in live range of a variable
- **Action space**: Set of valid use points to split
- **Reward**: Difference in spill weights before and after splitting

**Coloring Agent**
- Picks an appropriate color for a given vertex
- **Action space**: Set of Legal registers, if available. Otherwise, spill
- **Reward**: +Spill weight, if colored; -Spill weight, if spilled
Materialization of splitting

- Involves inserting move instructions
- Dataflow problem
  - Similar to \( \text{phi} \) or copy placement
- Use dominance frontier

---

**Algorithm 1:** move-placement in live range splitting

**Parameter:** Virtual register \( v \), Split point \( k \)

Rename \( v \rightarrow v' \)

At use point \( k \) do: \( v'' \leftarrow \text{move}(v') \)

Basic block \( B \leftarrow \text{block}(v_k) \)

**for** \( i \in \text{DominanceFrontier}(B) \) **do**

\( v' \leftarrow \text{move}(v'') \), after last use\( (v') \) in \( i \)

Renamed \( v' \rightarrow v'', \forall \text{use}(v') \) between \( B \) and \( i \)
Global Rewards

- Based on the throughput ($Th$) of the generated function
- Use LLVM MCA
  - Machine Code Analyzer of LLVM
  - Static model to estimate throughput

$$R_G = \begin{cases} +10, & Th_{RLAReAl} \geq Th_{Greedy} \\ -10, & Otherwise \end{cases}$$
Integration with LLVM

- RL4ReAI - to-and-fro communication
  - Decisions/Actions by Python model
  - Materialization of decisions in C++ compiler

- LLVM-gRPC - gRPC based framework
  - Seamless connection between LLVM and Python ML workloads
    - Works as an LLVM library
    - Easy integration
      - As simple as implementing a few API calls
  - Support for any ML workload
    - Not just limited to RL
    - With both training and inference flow
Training phase

- Involves RL model (Python) requesting C++ (LLVM)
- Model takes decisions on splitting and coloring
- C++ (LLVM) generates code for the decision and returns the reward accordingly
Inference

**Inference phase**
- For any input code C++(LLVM) sends a request to the trained model for splitting decision
- As a reply, the trained model returns the decision it took and code is generated.
Experiments

- MIR2Vec representations
  - 2000 source files from SPEC CPU 2017 and C++ Boost libraries
  - 100 dimensional embeddings; trained over 1000 epochs

- Evaluation
  - x86 - Intel Xeon W2133, 6 cores, 32GB RAM
  - AArch64 - ARM Cortex A72, 2 cores, 4GB RAM

- RL models - PPO policy with standard set of hyperparameters

- Register allocations
  - General purpose, floating point and vector registers

| Arch.   | Registers                                                                 |
|---------|---------------------------------------------------------------------------|
| x86     | [A-D]L, [A-D]X, [E,R][A-D]X, [S,I,D]L, [E,R][S,I,D], S, D, R[8-15][B,W,D], FP[0-7], [X,Y,Z]MM[0-15] |
| AArch64 | [X,W][0-30], [B,H,S,D,Q][0-31]                                           |
### Runtime improvements on x86

| Benchmarks   | Runtime Basic | PBQP | Greedy | RL4ReAl | Difference from Basic (Basic - x) |
|--------------|---------------|------|--------|---------|-----------------------------------|
|              |               | L    | G      |         |                                   |
| 401.bzip2    | 360.6         | -7.3 | 7.5    | -1.1    | 10.8                             |
| 429.mcf      | 233.8         | 1.4  | -2.9   | 2.7     | -3.6                             |
| 445.gobmk    | 322.3         | -3.3 | 6.4    | 2.4     | 1.7                              |
| 456.hmmer    | 284.3         | 1.8  | 6.1    | 5.0     | -37.6                            |
| 462.libquantum | 256.4        | -10.1| -1.1   | -2.2    | -6.7                             |
| 471.omnetpp  | 305.7         | 0.7  | 0.4    | 1.2     | 1.2                              |
| 433.milc     | 349.1         | -16.6| 0.1    | -13.8   | -7.0                             |
| 470.lbm      | 184.0         | -7.9 | 3.0    | 2.3     | 1.4                              |
| 482.sphinx3  | 366.0         | -37.5| 1.6    | -3.1    | -2.7                             |

| Benchmarks   | Runtime Basic | PBQP | Greedy | RL4ReAl | Difference from Basic (Basic - x) |
|--------------|---------------|------|--------|---------|-----------------------------------|
|              |               | L    | G      |         |                                   |
| 505.mcf_r    | 344.9         | 4.5  | 6.4    | 6.4     | 8.6                              |
| 520.omnetpp_r| 475.7         | 6.4  | 6.4    | 2.4     | 2.8                              |
| 531.deepsjeng_r | 299.9   | 4.6  | 16.0   | 9.9     | 12.8                             |
| 541.leela_r  | 439.5         | 1.6  | 7.1    | 0.4     | 1.9                              |
| 557.xz_r     | 371.5         | 11.9 | 12.1   | 1.9     | 2.8                              |
| 508.namd_r   | 236.5         | 3.5  | 23.5   | 9.1     | 23.8                             |
| 519.lbm_r    | 261.8         | 1.4  | 57.7   | 50.9    | 58.1                             |
| 538.imagick_r| 479.3         | 16.9 | 115.5  | 118.8   | 118.4                            |
| 544.nab_r    | 417.5         | 5.8  | 132.1  | 131.3   | 134.4                            |

- RL4ReAl shows speedups over Basic in 14/18 benchmarks
- Runtimes very close to Greedy
- Only 1 show more than 4% slow-down
## Analysis of Hot functions

|                    | SPEC CPU 2006 | SPEC CPU 2017 |
|--------------------|---------------|---------------|
|                    | Greedy | RL4REAL | Greedy | RL4REAL |
|--------------------|--------|---------|--------|---------|
| Average            | -1.5   | -2.1    | 6.2    | 7.3     | 4.8     |
| # (val>0)          | 16     | 17      | 13     | 23      | 23      | 17      |
| # (val<0)          | 19     | 18      | 22     | 8       | 8       | 14      |
| Max                | 12.7   | 10.4    | 6.2    | 44.0    | 44.4    | 41.3    |
| Min                | -51.4  | -52.5   | -13.1  | -7.7    | -4.4    | -10.8   |

%Difference in runtime with Basic as baseline on hot functions
# Analysis of Hot functions

| B/M | Functions                  | GREEDY | RL4REAL | Diff. |
|-----|----------------------------|--------|---------|-------|
|     | **Top 5 functions with highest % speedup (over GREEDY)** |        |         |       |
| 401 | BZ2_compressBlock          | -51.3  | -5.2    | 46.1  |
| 445 | do_get_read_result         | -12.0  | -0.5    | 11.5  |
| 482 | mgau_eval                  | -6.0   | 0.3     | 6.3   |
| 429 | price_out_impl             | -0.8   | 2.3     | 3.2   |
| 445 | subvq_mgau_shortlist       | -9.8   | -6.9    | 2.9   |
| 538 | GetVirtualPixelsFromNexus  | 8.3    | 28.8    | 20.4  |
| 538 | SetPixelCacheNexusPixels   | 4.7    | 21.9    | 17.2  |
| 505 | cost_compare               | -7.7   | 8.1     | 15.8  |
| 557 | lzma_mf_bt4_skip          | -1.8   | 3.63    | 5.5   |
| 525 | biari_decode_symbol        | -2.7   | 2.7     | 5.4   |
# Analysis of Hot functions

| B/M  | Functions          | Greedy | RL4ReAl | Diff. |
|------|--------------------|--------|---------|-------|
| Top 5 functions with highest % slow-down (over Greedy) |
| 456  | P7Viterbi          | 2.2    | -13.1   | -15.3 |
| 482  | vector_gautbl_eval_logs3 | 11.9 | -2.5 | -14.4 |
| 401  | mainGtU            | 0.3    | -9.6    | -10.0 |
| 401  | fallbackSort       | 12.6   | 6.2     | -6.4  |
| 445  | fastlib            | 4.8    | -1.1    | -5.9  |
| 557  | lzma_mf_bt4_find   | 1.5    | -10.7   | -12.3 |
| 531  | feval              | 26.4   | 17.7    | -8.6  |
| 505  | primal_bea_mpp     | 0.9    | -7.6    | -8.5  |
| 541  | FastBoard::self_atari | 3.7 | -0.1 | -5.8  |
| 541  | qsearch            | 6.6    | 1.5     | -5.0  |
## Runtimes on AArch64

| Benchmarks   | Runtime | Diff. from |            |            |            |
|--------------|---------|------------|------------|------------|------------|
|              | BASIC   | (BASIC- x) | PBQP       | GREEDY     | RL4REAL    |
| 401.bzip2    | 1366.9  | -41.1      | 15.6       | 12.8       |            |
| 429.mcf      | 1320.5  | -12.7      | -7.5       | 1.6        |            |
| 445.gobmk    | 992.8   | 15.6       | 26.1       | 14.5       |            |
| 462.libquantum | 1627.6 | -8.7       | 4.5        | 9.6        |            |
| 433.milc     | 1251.1  | 59.2       | 70.9       | 45.4       |            |
| 444.namd     | 855.3   | 2.7        | 21.8       | 18.8       |            |
| 470.lbm      | 1604.3  | -6.4       | -16.6      | 16         |            |
| 505.mcf_r    | 1535.1  | 25.9       | 1.9        | -12.8      |            |
| 508.namd_r   | 845     | 0.4        | 34.5       | 40.1       |            |
| 523.xalancbmk_r | 979.1 | 8.1        | -3.4       | 4.4        |            |
| 531.deepsjeng_r | 777.2 | 10.0       | 30.5       | 4.5        |            |
| 541.leela_r  | 1067.9  | -11.3      | -0.1       | -19.5      |            |
| 557.xz_r     | 1163.2  | 3.7        | 22.2       | 21.3       |            |
| 519.lbm_r    | 1657    | 50.9       | -1.6       | 39.8       |            |
| 538.imagick_r | 1244.5 | -3.9       | 75.8       | 65.6       |            |
| 544.nab_r    | 1170.7  | -7.7       | 31.5       | 32.4       |            |

Average: 5.3, 19.1, 18.4
Policy Improvement on Regression cases

- Regression in performance
  - Identify $\rightarrow$ Refine heuristics $\rightarrow$ Evaluate

- MLGO’s policy improvement cycle
  - Fine-tuning of learned RL policy on regression cases

- Identify and Refine
  - Poorly performing benchmarks from each configuration
  - RL4Real-L
    - milc (-13.8s $\rightarrow$ -0.8s)
  - RL4Real-G
    - Hmmer (-37.6s $\rightarrow$ -26s), xz (-8.5s $\rightarrow$ -2.5s)

- Strong case for online learning and domain specialization

Trofin et al, MLGO: a machine learning guided compiler optimizations framework - arXiv, 2021
Summary

- RL4ReAl: Architecture independent Reinforcement Learning for Register Allocation
- Multi agent hierarchical approach
- Generates semantically correct code: constraints imposed on the action space
- Allocations on par or better than the best allocators of LLVM
- New opportunities for compiler/ML research
- Framework will be open-sourced
- [https://compilers.cse.iith.ac.in/publications/rl4real](https://compilers.cse.iith.ac.in/publications/rl4real)
Thank You!

https://compilers.cse.iith.ac.in/publications/rl4real/

| S. VenkataKoorthy | Siddharth Jain | Anilava Kandi |
|-------------------|----------------|---------------|
| IT Hyderabad      | IT Hyderabad   | IT Hyderabad  |
| India             | India          | India         |

| Rohit Aggarwal    | Albert Cohen   | Ramakrishna Upadrasa |
|-------------------|----------------|----------------------|
| IT Hyderabad      | Google         | IT Hyderabad         |
| India             | France         | India                |

Abstract
We aim to automate decades of research and experience in register allocation, leveraging machine learning. We tackle this problem by embedding a multi-agent reinforcement learning algorithm within LLVM, training it with the state of the art techniques. We formulate the constraints that precisely define the problem for a given instruction-set architecture, while ensuring that the generated code preserves semantic correctness. We also develop a PyTorch based framework providing a modular and efficient compiler interface for training and inference. Our approach is architecture-independent.

problem is reducible to graph coloring, which is one of the classical NP-Complete problems [5, 2]. Register allocation as an optimization involves additional sub-tasks, more than graph coloring itself [5]. Several formulations have been proposed that return exact, or heuristic-based solutions.

Readily, solutions are often formulated as constraint-based optimizations [34, 38], ILP [3, 3, 12, 42], PQQP [31], game-theoretic approaches [45], and are tied to a variety of solvers.
In general, these approaches are known to have scalability issues. On the other hand, heuristic-based approaches have been widely used owing to their scalability: reasonable solv-