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 → 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
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
// Source
i = 0
x = 10
y = 20
print x
z = y / x
i++
z = z + 10
i++
print y
print z
print i
```

```c
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

```plaintext
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
%z:gr32 = ADD32ri %i:gr32, 1
<call print on %y, %z, %i>
```
Real-world ISAs have hierarchy of register classes
  ○ Congruent classes

Figure source: Wikipedia
Interference constraints

Register allocation $\Rightarrow$ Graph coloring problem

```
x = 10
y = 20
print x
z = 20 + y
print y
z = z + 10
print z
```

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

Edges: \{phy \ reg - \ vir \ reg, \ 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 \( \# \text{Uses} > k \) \((k = 2)\)
- **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 phi or copy placement
- Use dominance frontier

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

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

1. Rename \( v \rightarrow v' \)
2. At use point \( k \) do: \( v'' \leftarrow \text{move}(v') \)
3. Basic block \( B \leftarrow \text{block}(v_k) \)
4. **for** \( i \in \text{DominanceFrontier}(B) \) **do**
   - \( v' \leftarrow \text{move}(v'') \), after last use(\( v' \)) in \( i \)
   - Rename \( v' \rightarrow v'' \), \( \forall \text{use}(v') \) 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

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

- **RL4ReAl - 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

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 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

<table>
<thead>
<tr>
<th>Arch.</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch64</td>
<td>[X,W][0-30], [B,H,S,D,Q][0-31]</td>
</tr>
</tbody>
</table>
**Runtime improvements on x86**

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

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Runtime</th>
<th>Difference from Basic (Basic - x)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Basic</td>
<td>PBQP</td>
<td>Greedy</td>
<td>RL4ReAl</td>
<td>L</td>
<td>G</td>
</tr>
<tr>
<td>505.mcf_r</td>
<td>344.9</td>
<td>4.5</td>
<td>-1.6</td>
<td>8.6</td>
<td>-4.7</td>
<td></td>
</tr>
<tr>
<td>520.omnetpp_r</td>
<td>475.7</td>
<td>6.4</td>
<td>6.4</td>
<td>2.4</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>531.deepsjeng_r</td>
<td>299.9</td>
<td>4.6</td>
<td>16.0</td>
<td>9.9</td>
<td>12.8</td>
<td></td>
</tr>
<tr>
<td>541.leela_r</td>
<td>439.5</td>
<td>1.6</td>
<td>7.1</td>
<td>0.4</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td>557.xz_r</td>
<td>371.5</td>
<td>-0.6</td>
<td>11.9</td>
<td>12.1</td>
<td>-8.5</td>
<td></td>
</tr>
<tr>
<td>508.namd_r</td>
<td>236.5</td>
<td>2.5</td>
<td>23.5</td>
<td>9.1</td>
<td>23.8</td>
<td></td>
</tr>
<tr>
<td>519.lbm_r</td>
<td>261.8</td>
<td>1.4</td>
<td>57.7</td>
<td>50.9</td>
<td>58.1</td>
<td></td>
</tr>
<tr>
<td>538.imagick_r</td>
<td>479.3</td>
<td>-16.9</td>
<td>115.5</td>
<td>118.8</td>
<td>118.4</td>
<td></td>
</tr>
<tr>
<td>544.nab_r</td>
<td>417.5</td>
<td>5.8</td>
<td>132.1</td>
<td>131.3</td>
<td>134.4</td>
<td></td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th></th>
<th>SPEC CPU 2006</th>
<th>SPEC CPU 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Greedy</td>
<td>RL4REAL</td>
</tr>
<tr>
<td>Average</td>
<td>-1.5</td>
<td>-2.1</td>
</tr>
<tr>
<td># (val&gt;0)</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td># (val&lt;0)</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>Max</td>
<td>12.7</td>
<td>10.4</td>
</tr>
<tr>
<td>Min</td>
<td>-51.4</td>
<td>-52.5</td>
</tr>
</tbody>
</table>
## Analysis of Hot functions

%speedups obtained by Greedy and RL4ReAI over Basic

<table>
<thead>
<tr>
<th>B/M</th>
<th>Functions</th>
<th>Greedy</th>
<th>RL4ReAI</th>
<th>Diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>401</td>
<td>BZ2_compressBlock</td>
<td>-51.3</td>
<td>-5.2</td>
<td>46.1</td>
</tr>
<tr>
<td>445</td>
<td>do_get_read_result</td>
<td>-12.0</td>
<td>-0.5</td>
<td>11.5</td>
</tr>
<tr>
<td>482</td>
<td>mgau_eval</td>
<td>-6.0</td>
<td>0.3</td>
<td>6.3</td>
</tr>
<tr>
<td>429</td>
<td>price_out_impl</td>
<td>-0.8</td>
<td>2.3</td>
<td>3.2</td>
</tr>
<tr>
<td>445</td>
<td>subvq_mgau_shortlist</td>
<td>-9.8</td>
<td>-6.9</td>
<td>2.9</td>
</tr>
<tr>
<td>538</td>
<td>GetVirtualPixelsFromNexus</td>
<td>8.3</td>
<td>28.8</td>
<td>20.4</td>
</tr>
<tr>
<td>538</td>
<td>SetPixelCacheNexusPixels</td>
<td>4.7</td>
<td>21.9</td>
<td>17.2</td>
</tr>
<tr>
<td>505</td>
<td>cost_compare</td>
<td>-7.7</td>
<td>8.1</td>
<td>15.8</td>
</tr>
<tr>
<td>557</td>
<td>lzma_mf_bt4_skip</td>
<td>-1.8</td>
<td>3.63</td>
<td>5.5</td>
</tr>
<tr>
<td>525</td>
<td>biari_decode_symbol</td>
<td>-2.7</td>
<td>2.7</td>
<td>5.4</td>
</tr>
</tbody>
</table>
Analysis of Hot functions

<table>
<thead>
<tr>
<th>B/M</th>
<th>Functions</th>
<th>Greedy</th>
<th>RL4ReAl</th>
<th>Diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top 5 functions with highest % slow-down (over Greedy)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>456</td>
<td>P7Viterbi</td>
<td>2.2</td>
<td>-13.1</td>
<td>-15.3</td>
</tr>
<tr>
<td>482</td>
<td>vector Gäutbl_eval_logs3</td>
<td>11.9</td>
<td>-2.5</td>
<td>-14.4</td>
</tr>
<tr>
<td>401</td>
<td>mainGtU</td>
<td>0.3</td>
<td>-9.6</td>
<td>-10.0</td>
</tr>
<tr>
<td>401</td>
<td>fallbackSort</td>
<td>12.6</td>
<td>6.2</td>
<td>-6.4</td>
</tr>
<tr>
<td>445</td>
<td>fastlib</td>
<td>4.8</td>
<td>-1.1</td>
<td>-5.9</td>
</tr>
<tr>
<td>557</td>
<td>lzma_mf_bt4_find</td>
<td>1.5</td>
<td>-10.7</td>
<td>-12.3</td>
</tr>
<tr>
<td>531</td>
<td>feval</td>
<td>26.4</td>
<td>17.7</td>
<td>-8.6</td>
</tr>
<tr>
<td>505</td>
<td>primal_bea_mpp</td>
<td>0.9</td>
<td>-7.6</td>
<td>-8.5</td>
</tr>
<tr>
<td>541</td>
<td>FastBoard::self_atari</td>
<td>3.7</td>
<td>-0.1</td>
<td>-5.8</td>
</tr>
<tr>
<td>541</td>
<td>qsearch</td>
<td>6.6</td>
<td>1.5</td>
<td>-5.0</td>
</tr>
</tbody>
</table>
### Runtimes on AArch64

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Runtime <strong>BASIC</strong></th>
<th>Diff. from <strong>BASIC</strong> (<strong>BASIC</strong> - x)</th>
<th><strong>PBQP</strong></th>
<th><strong>GREEDY</strong></th>
<th><strong>RL4REAL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>401.bzip2</td>
<td>1366.9</td>
<td>-41.1</td>
<td>15.6</td>
<td>12.8</td>
<td></td>
</tr>
<tr>
<td>429.mcf</td>
<td>1320.5</td>
<td>-12.7</td>
<td>-7.5</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>445.gobmk</td>
<td>992.8</td>
<td>15.6</td>
<td>26.1</td>
<td>14.5</td>
<td></td>
</tr>
<tr>
<td>462.libquantum</td>
<td>1627.6</td>
<td>-8.7</td>
<td>4.5</td>
<td>9.6</td>
<td></td>
</tr>
<tr>
<td>433.milc</td>
<td>1251.1</td>
<td>59.2</td>
<td>70.9</td>
<td>45.4</td>
<td></td>
</tr>
<tr>
<td>444.namd</td>
<td>855.3</td>
<td>2.7</td>
<td>21.8</td>
<td>18.8</td>
<td></td>
</tr>
<tr>
<td>470.lbm</td>
<td>1604.3</td>
<td>-6.4</td>
<td>-16.6</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>505.mcf_r</td>
<td>1535.1</td>
<td>25.9</td>
<td>1.9</td>
<td>-12.8</td>
<td></td>
</tr>
<tr>
<td>508.namd_r</td>
<td>845</td>
<td>0.4</td>
<td>34.5</td>
<td>40.1</td>
<td></td>
</tr>
<tr>
<td>523.xalancbmk_r</td>
<td>979.1</td>
<td>8.1</td>
<td>-3.4</td>
<td>4.4</td>
<td></td>
</tr>
<tr>
<td>531.deepsjeng_r</td>
<td>777.2</td>
<td>10.0</td>
<td>30.5</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>541.leela_r</td>
<td>1067.9</td>
<td>-11.3</td>
<td>-0.1</td>
<td>-19.5</td>
<td></td>
</tr>
<tr>
<td>557.xz_r</td>
<td>1163.2</td>
<td>3.7</td>
<td>22.2</td>
<td>21.3</td>
<td></td>
</tr>
<tr>
<td>519.lbm_r</td>
<td>1657</td>
<td>50.9</td>
<td>-1.6</td>
<td>39.8</td>
<td></td>
</tr>
<tr>
<td>538.imagick_r</td>
<td>1244.5</td>
<td>-3.9</td>
<td>75.8</td>
<td>65.6</td>
<td></td>
</tr>
<tr>
<td>544.nab_r</td>
<td>1170.7</td>
<td>-7.7</td>
<td>31.5</td>
<td>32.4</td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>5.3</strong></td>
<td><strong>19.1</strong></td>
<td><strong>18.4</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Policy Improvement on Regression cases

- Regression in performance
  - Identify → Refine heuristics → 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 → -0.8s)
    - RL4Real-G
      - Hmmer (-37.6s → -26s), xz (-8.5s → -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.iitb.ac.in/publications/rl4real/