Alocação de Registradores utilizando Machine Learning

Pedro Zaffalon da Silva[1], **Wesley Attrot**[1]

[1]Departamento de Computação – Universidade Estadual de Londrina (UEL)
Caixa Postal 10.011 – CEP 86057-970 – Londrina – PR – Brasil

pedro.zaffalon@uel.br, wesley@uel.br

Abstract. *Register allocation is a important phase for compiler optimization, generally mapped to the graph coloring, which is a NP-complete problem. because of its impact for quality code generation, various heuristic algorithms have been proposed. However, heuristics development is a complex process and requires very specialized domain expertise. Recently, several Machine Learning based approaches are being proposed to solve compiler optimization problems. Still, due to the challenge of ensuring correctness, few works applied Machine Learning to register allocation. This works aims to contribute to the research of Machine Lerning usage to optimize register allocation by defining its state of the art, describing methods present in the literature and promissing directions for future works.*

Resumo. *A alocação de registradores é uma etapa com grande impacto na otimização de códigos gerados pelo compilador. Geralmente, sua resolução é realizada através da coloração de grafo, um problema NP-completo. Devido a sua importância, várias heurísticas foram propostas para a sua resolução. Contudo, a criação delas é um processo complexo e altamente especializado. Em um contexto no qual Machine Lerning é cada vez mais aplicado em otimizações de compiladores, sua utilização para melhorar a alocação de registradores se torna uma opção interessante. Porém, devido a necessidade de corretute, não presente em outras formas de otimização realizadas por compiladores, apenas recentemente esse tema foi mais pesquisado. Desta forma, este trabalho propõe a definição do estado da arte da utilização de Machine Learning na alocação de registradores, com objetivos de informar técnicas desenvolvidas e apontar caminhos promissores na área.*

1. Introdução

A alocação de registradores é uma etapa importante da otimização de compiladores, sendo que este problema é estudado há décadas devido a sua importância para a qualidade da geração de código. Geralmente é abstraído para coloração de grafo, um problema de otimização combinatória NP-completo [16, 4], onde cada vértice representa o tempo de vida de uma variável, de forma que uma aresta indica que as variáveis precisam ser armazenadas simultaneamente em algum momento. Desta forma, sendo as cores os registradores que serão alocados, é necessário atribuir cores sem que vértices ligados por uma aresta apresentem a mesma cor.

Para resolver este problema várias heurísticas foram propostas durante os últimos 40 anos [6, 9, 5] e novas continuam sendo pesquisadas [8]. Porém, o desenvolvimento

de heurísticas é um processo complexo, exigindo especialidade em campos bastante específicos, tanto em construção de compiladores, quanto em arquitetura de hardware [28]. Além disso, as heurísticas utilizadas apresentam desempenhos que podem ser melhorados em termos de otimização e muitas vezes precisam ser específicas para arquiteturas [34].

Por outro lado, com o aumento da utilização de *Machine Learning* [27, 11] em diversas áreas, várias abordagens baseadas em redes neurais foram propostas para melhorar as otimizações realizadas pelos compiladores. Geralmente essas abordagens são aplicadas em problemas como *phase ordering* [12, 2], *throughput prediction* [26], ou na criação de heurísticas para otimizações [22, 33, 28]. Observa-se que nos processos citados não é necessário garantir a corretude, visto que erros afetariam apenas o desempenho do código compilado, mas não o seu funcionamento. Devido a maior complexidade, pouco foi explorado em problemas contendo restrições semânticas, como a alocação de registradores, na qual soluções incorretas podem implicar na perda de valores salvos em variáveis, por exemplo. Desta forma, apenas recentemente abordagens baseadas em *Machine Learning* foram aplicadas para resolução da alocação de registradores [10, 34, 24, 20, 31], as quais obtiveram resultados promissores, apesar da pequena quantidade de pesquisas envolvendo esse tema.

Neste contexto, é proposto neste trabalho um estudo sobre as principais técnicas e resultados presentes na literatura, de forma a definir o estado da arte da utilização de *Machine Learning* para otimização da alocação de registradores. Sendo assim, por meio desta trabalho, será possível aprender sobre os métodos que já foram explorados nessa área, além de encontrar caminhos promissores que podem ser abordados em futuras pesquisas.

2. Fundamentação Teórico-Metodológica e Estado da Arte

2.1. Alocação de registradores

Alocação de registradores é um tema que é extensamente pesquisado há décadas devido ao seu impacto no desempenho de códigos gerados por compiladores e por sua complexidade. A abstração mais utilizada para esse problema é o grafo de interferência, no qual os vértices são os tempos de vida das variáveis e as arestas as interferências entre os vértices. Assim, a alocação de registradores é reduzida ao problema de coloração de grafo, onde as cores representam os registradores físicos do processador [7].

Um grafo é k-colorável se é possível atribuir para cada vértice uma das k cores sem que vértices ligados por uma aresta tenham a mesma cor, sendo esse o principal objetivo da alocação de registradores. Se o grafo não for k-colorável é necessário realizar o processo de *spill*, que consiste no uso da memória para armazenar a variável cujo o tempo de vida é representada pelo vértice. Caso não seja possível evitar *spills* o objetivo passa a ser minimizar o seu custo [7].

Considerando que este é um problema NP-completo [16, 4], várias heurísticas foram propostas para a resolução deste problema [6, 9, 5, 8]. Porém, devido a dificuldade de criá-las e a busca por maior eficiência, recentemente foram propostas abordagens baseadas em *Machine Learning* para realizar a alocação de registradores.

2.2. Machine Learning

Machine learning é um ramo da inteligência artificial (IA) que tem como objetivo o desenvolvimento de algoritmos que possam emular a inteligência humana, tomando decisões com base em experiencias acumuladas através da solução bem sucedida de problemas anteriores [27, 11].

Por meio do uso de métodos estatísticos, os algoritmos são treinados, geralmente com grandes quantidade de dados, para fazer classificações ou previsões de acordo com o problema proposto. Essas previsões conduzem a tomada de decisões, de forma a melhorar os resultados do algoritmo [1].

O método de aprendizado pode ser classificado em três formas. O primeiro é o supervisionado, no qual o conjuntos de dados são rotulados para que haja uma resposta para a máquina medir sua precisão. O segundo é o não supervisionado onde os dados não rotulados e o algoritmo deve entender por conta própria. Por fim o semi supervisionado, com apenas parte dos testes rotulados [21].

2.3. Machine Learning na Alocação de Registradores

A utilização de *Machine Learning* para melhorar otimizações de código realizadas por compiladores é uma prática cada vez mais comum, com diferentes técnicas descritas na literatura e utilizadas na indústria [30]. Porém, devido à restrições semânticas, apenas nos últimos anos foi estudado a aplicação de *Machine Learning* dentro do contexto do problema de alocação de registradores. Inicialmente, foram realizados projetos voltados para resolução da coloração de grafo [18, 35, 19, 24, 10], além da geração heurísticas para auxiliar a alocação, assim como é feito para outros problemas de otimização [31].

No trabalho de Khakhulin et al. [19] é proposto um modelo baseado em *Reinforcement Learning* [3] com objetivo de obter heurísticas para resolver o problema de decomposição de árvore, sendo importante para a alocação de registradores. Porém é importante observar que o modelo ainda encontra desafios em ter um desempenho melhor que algoritmos especializados, sendo que este não é o objetivo do trabalho. Similarmente, no artigo de Lemos et al. [24] é apresentado um modelo de *Graph Neural Network* (GNN) [1] para resolver o problema combinatório de coloração de grafo, obtendo resultados satisfatórios. Entretanto, essa abordagem não é suficiente para formar um modelo prático de alocação de registradores [34].

No trabalho de Das et al. [10] é apresentado um *framework* de *Deep Learning* [13] aplicado em coloração de grafos com enfase no problema de alocação de registradores, obtendo um algoritmo aproximado com uma fase de correção de cor após o modelo de aprendizado baseado em *long short-term memory* (LSTM) [17]. O algoritmo obteve resultados próximos ou melhores quando comparado com a abordagem gulosa presente no LLVM [23].

Ainda, no artigo de VenkataKeerthy et al. [34] é proposto uma abordagem *Multi-Agent Hierarchical Reinforcement Learning* [25] como uma solução independente de arquitetura. O modelo oferece solução para diferentes sub-tarefas da alocação de registradores, incluindo coloração, *live range splitting* e *spilling*. Os autores pretendem

[1] *Graph Neural Network* (GNN) é uma classe de redes neurais especializada em processamento de dados descritos em grafos [14]

adicionar outras sub-tarefas como *coalescing*, multi-alocação, e *packing* em trabalhos futuros. Comparado a alocação padrão do LLVM o modelo proposto resultou em melhores alocações e tempo de execução, além de menor acesso de memória.

Por fim, no trabalho de Minsu Kim et al. [20] aborda a alocação de registradores estruturados irregularmente, especialmente com processadores de teste de equipamento automático (ATE) de chips de memória DRAM. Como em ATE não é permitido a realização de *spill*, foi proposta uma abordagem com *deep reinforcement learning (Deep-RL)* [29], com a alocação de registradores abstraída para *Partitioned Boolean quadratic programming* (PBQP) [32, 15]. Sendo treinado com grafos PBQP aleatórios, apresentou resultados competitivos comparado a programas gerais no LLVM.

Esses e outros trabalhos serão estudados para definir o estado da arte conforme proposto por este projeto.

3. Objetivos

Este trabalho tem como objetivo definir o estado da arte da utilização de *Machine Learning* para solucionar o problema de alocação de registradores, de forma a informar o que já foi pesquisado sobre o tema e apontar direções promissoras que podem ser exploradas em futuros trabalhos.

4. Procedimentos metodológicos/Métodos e técnicas

Os objetivos propostos serão alcançados por meios da análise dos métodos utilizados na literatura. Inicialmente será feito um levantamento bibliográfico sobre a utilização de *Machine Learning* para resolução de alocação de registradores ou coloração de grafo. Em seguida, o estudo das técnicas empregadas nos materiais obtidos anteriormente. Ainda, os métodos estudados e seus resultados obtidos serão avaliados de forma que seja possível encontrar caminhos interessantes para futuros trabalhos na área. Por fim, será realizada a escrita do trabalho de conclusão de curso apresentando os resultados do projeto.

5. Cronograma de Execução

Atividades:

1. Realizar o levantamento bibliográfico;
2. Estudar as técnicas utilizados;
3. Analisar os métodos e resultados;
4. Escrever o TCC;

Tabela 1. Cronograma de Execução

	ago	set	out	nov	dez	jan	fev	mar	abr	mai
Atividade 1	x	x								
Atividade 2			x	x	x					
Atividade 3						x	x	x		
Atividade 4							x	x	x	x

6. Contribuições e/ou Resultados esperados

Espera-se que este projeto contribua no melhor entendimento sobre o atual estado da utilização de *Machine Learning* na otimização de alocação de registradores e os principais métodos aplicados, além de indicar caminhos promissores nessa área, incentivando a realização de pesquisas sobre o tema.

7. Espaço para assinaturas

Londrina, *12 de setembro de 2022*.

Aluno	Orientador

Referências

[1] O que é machine learning? https://www.ibm.com/br-pt/cloud/learn/machine-learning. Accessed: 2022-13-08.

[2] Amir H Ashouri, Andrea Bignoli, Gianluca Palermo, Cristina Silvano, Sameer Kulkarni, and John Cavazos. Micomp: Mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning. *ACM Transactions on Architecture and Code Optimization (TACO)*, 14(3):1–28, 2017.

[3] Andrew G. Barto. Chapter 2 - reinforcement learning. In Omid Omidvar and David L. Elliott, editors, *Neural Systems for Control*, pages 7–30. Academic Press, San Diego, 1997.

[4] Florent Bouchez, Alain Darte, Christophe Guillon, and Fabrice Rastello. Register allocation: What does the np-completeness proof of chaitin et al. really prove? or revisiting register allocation: Why and how. In *International Workshop on Languages and Compilers for Parallel Computing*, pages 283–298. Springer, 2006.

[5] Preston Briggs, Keith D. Cooper, and Linda Torczon. Improvements to graph coloring register allocation. *ACM Trans. Program. Lang. Syst.*, 16(3):428–455, may 1994.

[6] Gregory J. Chaitin, Marc A. Auslander, Ashok K. Chandra, John Cocke, Martin E. Hopkins, and Peter W. Markstein. Register allocation via coloring. *Comput. Lang.*, 6(1):47–57, jan 1981.

[7] Gregory J. Chaitin, Marc A. Auslander, Ashok K. Chandra, John Cocke, Martin E. Hopkins, and Peter W. Markstein. Register allocation via coloring. *Computer Languages*, 6(1):47–57, 1981.

[8] Wei-Yu Chen, Guei-Yuan Lueh, Pratik Ashar, Kaiyu Chen, and Buqi Cheng. Register allocation for intel processor graphics. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018, page 352–364, New York, NY, USA, 2018. Association for Computing Machinery.

[9] Fred C. Chow and John L. Hennessy. The priority-based coloring approach to register allocation. *ACM Trans. Program. Lang. Syst.*, 12(4):501–536, oct 1990.

[10] Dibyendu Das, Shahid Asghar Ahmad, and Venkataramanan Kumar. Deep learning-based approximate graph-coloring algorithm for register allocation. In *2020 IEEE/ACM 6th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar)*, pages 23–32, 2020.

[11] Issam El Naqa and Martin J. Murphy. *What Is Machine Learning?*, pages 3–11. Springer International Publishing, Cham, 2015.

[12] Grigori Fursin, Yuriy Kashnikov, Abdul Wahid Memon, Zbigniew Chamski, Olivier Temam, Mircea Namolaru, Elad Yom-Tov, Bilha Mendelson, Ayal Zaks, Eric Courtois, et al. Milepost gcc: Machine learning enabled self-tuning compiler. *International journal of parallel programming*, 39(3):296–327, 2011.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[14] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005.

[15] Lang Hames and Bernhard Scholz. Nearly optimal register allocation with pbqp. In David E. Lightfoot and Clemens Szyperski, editors, *Modular Programming Languages*, pages 346–361, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[16] Juris Hartmanis. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1):90, 1982.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[18] Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Patrick Meyer, Amir Mohammad Karimi-Mamaghan, and El-Ghazali Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022.

[19] Taras Khakhulin, Roman Schutski, and Ivan Oseledets. Graph convolutional policy for solving tree decomposition via reinforcement learning heuristics. 2019.

[20] Minsu Kim, Jeong-Keun Park, and Soo-Mook Moon. Solving pbqp-based register allocation using deep reinforcement learning. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 1–12. IEEE, 2022.

[21] S.B. Kotsiantis, I.D. Zaharakis, and Pintelas. Machine learning: a review of classification and combining techniques. *Artif Intell Rev*, 26(1):159–190, 2006.

[22] Sameer Kulkarni, John Cavazos, Christian Wimmer, and Douglas Simon. Automatic construction of inlining heuristics using machine learning. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 1–12. IEEE, 2013.

[23] C. Lattner and V. Adve. Llvm: a compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86, 2004.

[24] Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 879–885, 2019.

[25] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, page 246–253, New York, NY, USA, 2001. Association for Computing Machinery.

[26] Charith Mendis, Alex Renda, Saman Amarasinghe, and Michael Carbin. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In *International Conference on machine learning*, pages 4505–4515. PMLR, 2019.

[27] Maria Carolina Monard and José Augusto Baranauskas. Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, 1(1):32, 2003.

[28] Raphael Mosaner. Machine learning to ease understanding of data driven compiler optimizations. In *Companion Proceedings of the 2020 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, SPLASH Companion 2020, page 4–6, New York, NY, USA, 2020. Association for Computing Machinery.

[29] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Deep reinforcement learning: An overview. In Yaxin Bi, Supriya Kapoor, and Rahul Bhatia, editors, *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*, pages 426–440, Cham, 2018. Springer International Publishing.

[30] Mircea Namolaru, Albert Cohen, Grigori Fursin, Ayal Zaks, and Ari Freund. Practical aggregation of semantical program properties for machine learning based optimization. In *Proceedings of the 2010 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '10, page 197–206, New York, NY, USA, 2010. Association for Computing Machinery.

[31] Biplab Kumar Saha, Tiffany A. Connors, Saami Rahman, and Apan Qasem. A machine learning approach to automatic creation of architecture-sensitive performance heuristics. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 18–25, 2017.

[32] Bernhard Scholz and Erik Eckstein. Register allocation for irregular architectures. *SIGPLAN Not.*, 37(7):139–148, jun 2002.

[33] Mark Stephenson, Saman Amarasinghe, Martin Martin, and Una-May O'Reilly. Meta optimization: Improving compiler heuristics with machine learning. *ACM sigplan notices*, 38(5):77–90, 2003.

[34] S. VenkataKeerthy, Siddharth Jain, Rohit Aggarwal, Albert Cohen, and Ramakrishna Upadrasta. Rl4real: Reinforcement learning for register allocation, 2022.

[35] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. Reinforcement learning based local search for grouping problems: A case study on graph coloring. *Expert Systems with Applications*, 64:412–422, 2016.