=, =,
=, =,
=, =, =,

UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Projeto de Dissertação de Mestrado

ANDRÉ FELIPE ZANELLA

A definir

Maringá
2020

ANDRÉ FELIPE ZANELLA

A definir

Projeto de dissertação apresentado ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Anderson Faustino da Silva

Maringá
2020

A definir

RESUMO

lipsun.

Palavras-chave: words.

A definir

ABSTRACT

lipsun.

Keywords: words.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ANN: *Artificial Neural Network*

SUMÁRIO

1

Introdução

Compiladores são peças centrais na ciência da computação, uma vez que estes funcionam como pontes entre o software e o hardware. Compiladores modernos apresentam otimizações que podem ser aplicadas em diferentes fases da compilação. Tais otimizações tem como objetivo gerar código semanticamente equivalente ao código fonte original, porém, maximizando ou minimizando alguns atributos do código gerado, com o objetivo de produzir código de melhor qualidade.

Alguns compiladores consolidados, como o GCC (GNU Compiler Collection) 2 e LLVM 3 (Low Level Virtual Machine), disponibilizam otimizações pré-definidas pelo projetista do compilador em forma de diretivas de compilação, chamados, O0, O1, O2, O3, onde cada um aplica uma sequência de otimizações em uma ordem definida. No entanto, nem sempre tais planos geram o melhor código possível, ocasionando na necessidade de criação por humanos de planos de compilação especializados. Tal fato deu origem a técnica de compilação iterativa. Nesta abordagem, sequências de transformações são sucessivamente aplicadas a um programa e sua performance é medida através de sua execução a fim de determinar o melhor plano.

Um sistema de *autotuning*, é um sistema que emprega compilação iterativa acompanhado de um algoritmo gerador de sequências. Estes modelos apesar de encontrarem boas sequências, são custosos, pois necessitam de compilação e execução. Além disso, o grande espaço de busca de transformações torna a tarefa mais complexa. Sistemas modernos de autotuning utilizam diversas técnicas de aprendizagem de máquina para minimizar este problema, selecionando de forma inteligente os pontos que mais otimizam a tarefa de busca. Redes neurais artificiais são uma classe técnicas de aprendizagem que tem se mostrado uma solução generalizada para problemas de classificação, tendo

exito em diversas áreas clássicas de inteligencia computacional. Em sistemas de alto desempenho redes neurais conseguem ser mais eficientes que sistemas de autotuning, pois eliminam a necessidade de executar a aplicação e são adaptáveis para qualquer problema de otimização.

Aprendizagem de máquina pode auxiliar na geração de um código de qualidade superior ao código que o compilador tradicionalmente é capaz de produzir, uma vez que algoritmos são capazes tomar decisões que tornam o processo de otimização eficiente. Este fato ajudou a estabelecer uma direção de pesquisa voltada para aplicações destas técnicas na construção de compiladores. Outra questão que impulsionou o campo é o fato de exclusão de expertes na criação de planos de compilação, substituindo a intuição e experimentação manual destes, pela avaliação empírica de dados guiada por heurísticas.

A tarefa essencial da compilação consiste na escolha de uma boa sequência de transformações. Com este foco, o processo de criar e avaliar um sistema de compilação guiado por aprendizagem de máquina pode ser estruturado em fases. A fase inicial consiste em definir as características qualitativas que irão representar o programa, estas são chamadas de *features*. Uma boa engenharia de *features* é essencial para determinar a qualidade do sistema, uma vez que o modelo de aprendizagem faz uso destes dados para determinar o comportamento do programa. A próxima fase é o treinamento do modelo utilizando um algoritmo de aprendizagem. Existem diversos algoritmos para esta tarefa, cada um especializado para uma classe de problemas. Por exemplo, algoritmos de clusterização são utilizados para agrupar programas considerados similares, enquanto que técnicas de aprendizagem supervisionada, como redes neurais, são adequadas para problemas de classificação, como estimativas de performance. Por fim, o modelo deve ser capaz de tomar boas decisões de otimização para novos programas não observados.

Diversos estudos posteriores se dedicaram a melhoria de sistemas de *autotuning* incorporando técnicas de aprendizagem de máquina a fim de minimizar a busca por pontos no espaço. Muitos sub-campos de aprendizagem de máquina existem e podem ser utilizados no contexto de geração de código e otimização. Surveys especializados são apresentas por Ashouri *et al* (Ashouri et al., 2018) e Wang (Wang e O'Boyle). Nesta dissertação, vamos abordar o sub-campo de aprendizagem supervisionada, mais especificamente, redes neurais profundas.

Redes neurais artificiais (ANNs) são modelos de aprendizagem supervisionada desejáveis para problemas de classificação, pois apresentam bons resultados aplicadas à problemas de regressão não linear, além disso, o poder de representação destas redes é capaz de aprender relações complexas entre as variáveis, enriquecendo assim o modelo.

Teoricamente, qualquer função pode ser aproximada com precisão arbitrária por uma ANN (Goodfellow et al., 2016).

Redes neurais demonstram seu poder através da capacidade automática de aprender a partir de dados. Este paradigma deixou de ser utilizado exclusivamente para problemas especializados a partir da descoberta das técnicas de aprendizagem profunda em 2006, revolucionando áreas como visão computacional, processamento de linguagem natural e mais recentemente vem ganhando popularidade dentro das áreas de pesquisa de linguagens de programação e compiladores (Nielsen, 2018).

O uso de redes neurais tem impulsionado a pesquisa em ambientes de alta performance; estas tem sido utilizadas como modelos de predição para tarefas como determinação de granularidade de *threads* (Magni et al., 2014), predição de *speedup* (Rosário et al., 2020), criação de sequências de otimizações (Xavier e Silva, 2018) (Cummins et al., 2017a), mapeamento de hardware e síntese de *benchmarks* (Cummins et al., 2017b). Os tipos de redes adotadas também varia, abrangendo redes de alimentação contínua, redes recorrentes e redes de grafos.

1.1 Desafios

A pesquisa por estratégias de compilação dividida em dois principais problemas: o *Problema de Seleção de Fase* (PSF)[1], que busca encontrar o melhor conjunto de transformações para um dado programa, sem levar em consideração a ordem em que são aplicadas, e o *Problema de Ordenação de Fase* (POF), que busca, além de se encontrar o melhor conjunto de otimizações, encontrar a melhor ordem com que estas são aplicadas. O restante dessa seção discorre a respeito dos desafios existentes para responder estas duas questões.

Escassez de Dados e Inferência de programas. O desafio mais eminente que limita a performance de modelos de otimização de programas é qualidade dos dados de treino. Apesar de existirem inúmeros conjuntos de *benchmarks* disponíveis estes não são comparáveis com a diversidade de programas que um compilador usualmente tem que lidar. Este fato tem efeitos diretos no treinamento de modelos, uma vez que o conjunto de treino não é capaz representar significativamente o espaço de programas, levando a questões de como prever o comportamento de um programa baseado em um certo conjunto utilizado pelo modelo.

[1]Fase é um sinônimo de sequência, porém, neste caso a palavra *sequência* não indica uma ordem.

Aprendizagem por transferência é um método de aprendizagem de máquina que consiste em armazenar conhecimento adquirido ao solucionar um problema e utiliza-ló para solução de outro problema. Está técnica foi utilizada por Cummins *et al* (Cummins et al., 2017a) para solucionar dois problemas de otimização de domínios diferentes: mapeamento de hardware e granularidade de threads. Redes neurais profundas tem a habilidade de criar abstrações e relações representativas de programas podendo ser capaz de criar abstrações de um domínio ou conjunto de programas e generalizar para outro, no entanto, um modelo que possa ser utilizado para todos os domínios de otimização e programas contínua um desafio de pesquisa.

Representação de Programas. Algoritmos de aprendizagem de máquina aprendem a correlacionar variáveis que alimentam o motor de aprendizagem. Para uma caracterização precisa de um programa é necessário um conjunto de características qualitativas que representem o seu comportamento semântico. Existem inúmeras features diferentes que podem ser utilizadas. Estas geralmente se enquadram em duas categorias: *estáticas*, que são estruturas de dados extraídas do código fonte ou representação intermediária do programa, e *dinâmicas*, obtidas durante a execução execução da aplicação.

Os problemas enfrentados na tarefa de engenharia de features para otimização de código são diversos; alguns deles são: Escolha da representação estática, dinâmica ou uma combinação das mesmas. Além disso, estas podem ser representadas por estruturas de grafo ou vetores. Redução de dimensionalidade. vetores podem ter alta dimensionalidade e por isso, conter informações irrelevantes. Analise de componentes principais (PCA) ou coeficientes de similaridade podem auxiliar na exclusão de características irrelevantes.

Além dos problemas de engenharia de features, outra questão levada em consideração é se a escolha das features deve ser guiada por expertes ou pelo próprio modelo, isto é, a partir de uma representação sintática do código, como *tokens* da linguagem fonte ou representação intermediária por exemplo, o modelo de aprendizagem descobre quais são as features mais representativas, automatizando tarefas como identificação de features muito similares e não sendo dependente do problema em questão. Esta técnica só é possível graças a redes neurais profundas que são capazes de identificar relações entre códigos de programação (Allamanis et al., 2014).

Combinação de modelos Talvez um dos maiores desafios da compilação guiada por aprendizagem de máquina seja a combinação de diferentes modelos de aprendizagem. Em aprendizagem por reforço o algoritmo tenta aprender como maximizar a recompensa de uma função de valor, isto é, dado um input, o algoritmo precisa aprender quais as melhores decisões a serem tomadas. Este modelo de aprendizagem é muito intuitivo e se assemelha com o aprendizado por seres humanos, podendo criar modelos com excelente

precisão, no entanto, aprendizagem por reforço depende do ambiente e se estive possuir muitas possibilidades de decisão pode tornar o treinamento extremamente custoso. Esta abordagem diverge de redes neurais em que os dados de entrada e saída são conhecidos. Combinar aprendizagem por reforço com aprendizagem profunda para resolver o problema de compilação é uma questão em aberto.

Aprendizagem ativa é um método de aprendizagem que pode ser utilizado para minimizar o custo do treinamento. Técnicas de aprendizagem profunda necessitam de grandes volumes de dados (LECUN et al., 2015) tornando o processo de compilação e execução grandes consumidores de tempo. Trabalhos como de Rosário *et al* (Rosário *et al., 2020*) utilizam 30 mil planos de compilação diferentes e 300 programas para treinamento. Aprendizagem ativa foi explorado no trabalho de (Ogilvie et al., 2017) para minimizar o custo da compilação iterativa, no entanto, nenhum trabalho que combine aprendizagem ativa e aprendizagem profunda para atacar o problema de compilação foi encontrado.

1.2 Objetivos e Contribuições

...

1.3 Organização do texto

...

O foco deste trabalho esta no uso de redes neurais profundas para conduzir a criação de código otimizado. As próximas seções descrevem o papel de aprendizagem de máquina na área de compilação, os principais problemas encontrados neste campo de pesquisa, e as contribuições que esta dissertação procura realizar.

2

Revisão de literatura

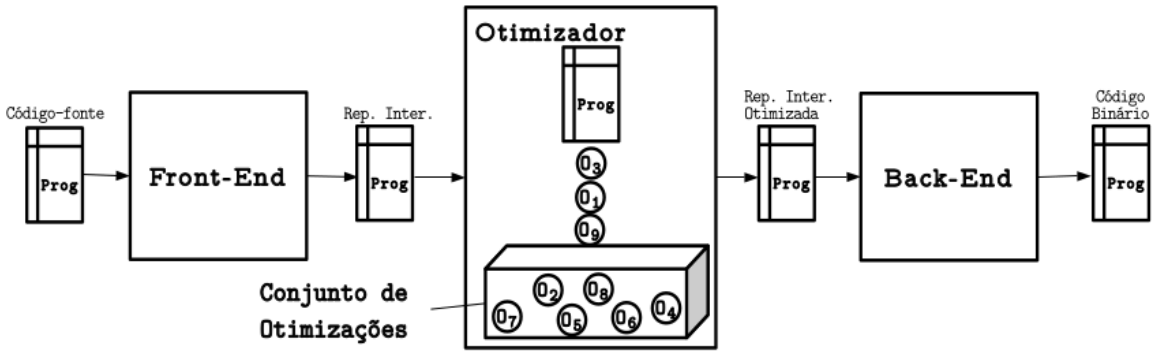
Neste capítulo é fornecida uma visão intuitiva e global do funcionamento de sistemas de compilação otimizantes 2.1, algoritmos de aprendizagem e as técnicas de avaliação utilizadas nesta dissertação 2.2.

2.1 Compilador Otimizante

Segundo Souza (2014) uma *transformação* ou *otimização* é um algoritmo aplicado pelo compilador, que transforma o programa fonte em um programa alvo semanticamente equivalente, porém mais eficiente. Transformações podem ser classificadas de duas formas: transformações de *análise* e *otimizantes*. A primeira tem como objetivo apenas coletar e armazenar informações a respeito do programa, enquanto que a segunda modifica o código e pode fazer uso das informações armazenadas para otimizar a aplicação. Uma *sequência* ou *fase* é um subconjunto do conjunto de transformações presentes em um compilador.

Um *compilador otimizante* é um compilador que apresenta um conjunto de transformações de análise e otimização para serem utilizadas durante o processo de compilação. O processo de compilação é composto pelas seguintes etapas: *Front-End*, e *Back-End*. O *Front-End* representa a fase de análise do código fonte, enquanto que no *Back-End*, ocorre a síntese do código fonte para código binário executável para a arquitetura-alvo. No entanto, compiladores modernos não realizam essa tradução diretamente. Entre as duas fases descritas acima, existe um gerador de *Código Intermediário* e um *Otimizador* (TORCZON e COOPER, 2011). A Figura - 3.1 apresenta a arquitetura de um compilador otimizante.

Figura 2.1: Arquitetura de um compilador moderno.



Fonte: (XAVIER, 2014).

Uma representação intermediária (IR) é uma estrutura de dados que representa uma abstração da linguagem de máquina, que é independente da arquitetura-alvo em questão, facilitando assim, a portabilidade da linguagem. Um otimizador tem a função de aplicar uma sequência de transformações fornecidas pelo usuário, ou uma estratégia já definida no compilador, a fim de melhorar o desempenho de uma aplicação. Transformações podem ser aplicadas ao programa em qualquer fase de compilação, no entanto, a grande maioria é feita no código IR. Compiladores modernos, como o GCC, e a LLVM, possuem otimizadores e suas próprias representações intermediárias de código (JUNIOR, 2016; XAVIER, 2014).

O espaço de busca de transformações é extremamente grande, uma vez que um compilador como o GCC, por exemplo, possui centenas de transformações. O problema de seleção de fase (PSF) é um problema de otimização, que busca encontrar um conjunto de transformações de compilação que proporcionem o maior desempenho para um programa. O desempenho de um programa pode ser definido de acordo com os objetivos que se deseja minimizar, podendo ser o tempo de execução ou tamanho de código gerado, por exemplo.

Para ilustrar a complexidade do espaço de busca, suponha que O seja o conjunto de todas as transformações de um compilador C, onde $O = \{o_1, o_2, \ldots, o_n\}$ tal que o_i, representa uma transformação. O conjunto de partes (conjunto potência) $P(O)$ contém todos os possíveis subconjuntos de transformações de O e possui cardinalidade 2^n, ou seja, possui complexidade exponencial. A fim de definir a sequência ótima, o compilador precisa avaliar todas as possibilidade. Existem ferramentas que procuram encontrar a forma canônica de uma aplicação, isto é, a sequência ótima. Tais sistemas são chamados de *Superotimizadores* (JUNIOR, 2016; MASSALIN, 1987).

Estudos anteriores mostram que as interdependências e a interação entre habilitar e desabilitar transformações em uma sequência pode alterar drasticamente o desempenho

de um programa (AGAKOV et al., 2006). Quando levada em consideração a ordem de aplicação das transformações em um conjunto, se define o problema de ordenação de fase (POF), que pode ter uma complexidade maior que 2^n, uma vez que selecionado um subconjunto com m transformações, tal que $m \leq n$, existem $m!$ arranjos diferentes para este subconjunto.

Transformações de código permitem que o programa se beneficie de melhorias sem alteração do código fonte original. No entanto, devido aos motivos já apresentados, definir manualmente uma sequência para um programa em específico não é uma tarefa viável. A próxima seção apresenta os métodos utilizados para encontrar as melhores transformações automaticamente.

2.2 Aprendizagem de Máquina Supervisionada

Técnicas de aprendizagem supervisionada e não-supervisionada tem sido exploradas na otimização de compiladores Aprendizagem supervisionada é um modelo de aprendizagem de máquina no qual é estabelecida uma relação entra os valores do vetor de entrada e saída correspondente. Este modelo é muito utilizado na resolução de problemas de regressão e classificação, pois correlações entre os dados de entrada e as decisões de otimização tomadas para se obter a melhor performance são estabelecidas. Em aprendizagem não-supervisionada não se tem conhecimento a respeito do valor de saída de cada entrada.

Deep learning é um poderoso framework para aprendizagem supervisionada que tem mostrando bons resultados nas mais diversas áreas do conhecimento pois é capaz de aprender relações complexas entre os dados. Este modelo de aprendizagem pode ser utilizado como uma ferramenta efetiva para extrair tal conhecimento do espaço de otimização de código. As próximas subseções tratam dos algoritmos utilizados nesta dissertação.

2.2.1 Redes Múltiplas Camadas Alimentadas Adiante

Redes feed-forward também conhecidas como multilayer perceptrons são redes neurais compostas por múltiplas camadas alimentadas adiante, isto é, a rede consiste de uma camada de entrada e uma mais camadas ocultas de nós, onde o sinal ou informação é propagado camada por camada, constituindo um mapeamento de um conjunto de variáveis para uma resposta. A Figura - 2.2 mostra uma arquitetura de uma rede alimentada a diante.

Estas redes são consideradas aproximadores universais, pois definem um mapeamento $y = f(x;\theta)$ que aproxima alguma função f^*, onde θ são os parâmetros de aproximação aprendidos pela rede. Estas servem como base para oturos modelos de redes neurais, devido as seguintes características:

- Cada neurônio na rede inclui uma função de ativação não-linear $\varphi(v)$ para descrever o sinal de cada neurônio.
- Contém uma ou mais camadas de neurônios ocultos que capacitam a rede a aprender tarefas complexas.
- Exibe um alto grau de conectividade, isto é, um neurônio pode mapear sua saída para todos os neurônios da camada subsequente.

Um vetor de entrada n-dimensional $\overline{x}$ pode ser transformado em uma resposta de saída utilizando as seguintes equações:

$$\overline{h}_1 = \varphi(W_1^T \overline{x}) \tag{2.1}$$

$$\overline{h}_{p+1} = \varphi(W_{p+1}^T \overline{h}_p) \tag{2.2}$$

$$\overline{o} = \varphi(W_{k+1}^T \overline{h}_k) \tag{2.3}$$

onde (2.1) é a equação *Entrada para Camada Oculta*, (2.2) *Oculta para Camada oculta* e (2.3) *Oculta para Camada de Saída*, W_p é a matriz de pesos, $\overline{h}_p$ é o vetor coluna de sinais da camada p.

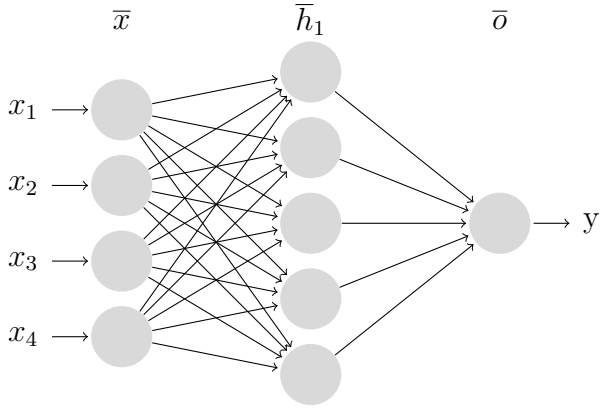


Figura 2.2: Rede neural artificial alimentada adiante composta por uma única camada oculta de 5 neurônios, ou nós, e uma camada de saída com um único neurônio onde. Cada aresta representa a saída de informação de um neurônio que serve como entrada para o neurônio da cada subsequente.

Algumas funções de ativação comumente utilizadas na literatura são as seguintes: Função *Limiar* 2.4

$$\varphi(v) = \begin{cases} 1, & \text{se } v \geq 1 \\ 0, & \text{se } v < 0 \end{cases} \tag{2.4}$$

Função *Sigmóide* 2.5. Esta é definida como uma função estritamente crescente com um comportamento balanceado entre linear e não-linear.

$$\varphi = \frac{1}{1 + exp(-av)} \tag{2.5}$$

Variando o parâmetro a se obtém funções sigmoides com diferentes inclinações.
Função *ReLU (Unidade Linear Retificada)* 2.6.

$$\varphi(v) = max(0, v) \tag{2.6}$$

Esta função largamente utilizada para redes multi-camada devido sua eficiente computação e sua ativação esparsa, por exemplo, para valores aleatórios no intervalo $[-1, 1]$, apenas 50% dos neurônios com ativação ReLU serão ativados.

Aprendizagem Baseada em Descida de Gradiente

Redes neurais usualmente são treinadas utilizando métodos iterativos, baseados em descida de gradiente, pois são capazes de derivar a função de custo a um valor muito pequeno. Funções de custo em redes com multiplas camadas são definidas como composições de funções de peso de camadas anteriores. Para minizar este problema o algoritmo de retropropagação é utilizado extensivamente na literatura (Nielsen, 2018).

Redes alimentadas adiante são inicializadas com pequenos valores de peso aleatórios e com bias nulo ou um pequeno valor positivo. Inicialmente, a computação inicialmente acorre de forma *avante*, isto é, sequencialmente através das camadas até a camada de saída. O resultado final $\hat{y}$ é comparado com o valor da instância de treino y e computado a perda $\mathcal{L}(\hat{y}, y)$. Após este passo, se da inicio a fase *retrograda* que consiste em computar o gradiente da função de custo com respeito aos diferentes pesos de todas as camadas, iniciando o processo pela camada de saída. Os gradientes são utilizados para atualizar os valores dos pesos.

2.2.2 Redes Neurais Recorrentes

Redes neurais recorrentes (RNNs) são redes neurais utilizadas para processamento de dados sequenciais, como series temporais e sequências de texto. Neste caso, a entrada é composta m pontos n-dimensionais $\overline{x}_1 \ldots \overline{x}_m$ com um marco temporal t. Neste caso x_t é o ponto contendo n valores observados no tempo t em uma série temporal.

que permitem conecções recorrentes entre os neurônios artificiais, isto é, um estado oculo atualiza um neurônio formando um ciclo. A figura XXX

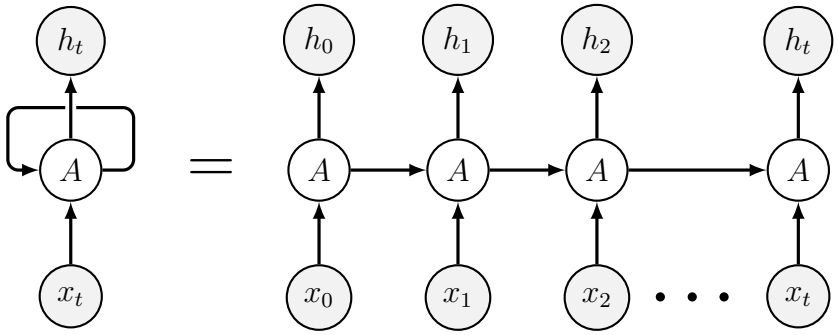


Figura 2.3: Fig02

Células e Portas

GRU e LSTM

Retropropagação Através do Tempo

2.2.3 Reinforcement Learning

Reinforcement Learning is one promising machine learning technique []. RL is a class of decision problems that involves software agents taking actions in an environment in order to maximize the cumulative reward. Markov Decision Problems (MDPs) are a mathematical formalization of the problem of mapping a decision made by the agent based on the environment states to maximizes the expected reward (mathematical framework that models sequential decision making).

An MDP is defined by a 4-tuple $S, A, P(.), R(.)$, where S is the set of the states of the environment, A is the set of actions, $P(s_{t+1} = s^{'}, r_{t+1} = 1 | s_t = s, a_t = a)$ is the state transition probability that specifies the dynamics of the model, and $R(s_t, a_t) = \mathbb{E}[r_t | s_t = s, a_t = a]$ is the reward function. The reward r_t is given by taking the action a in state s.

Given a timestep $t \in T$, the agent receives a subset of states $s_t \in S$ and selects a subset of action $a_t \in A$. The mapping of state s_t to the action is given by a distribution

$\pi(a_t|s_t) = p(A_t = a|S_t = s)$, where π is called *policy*. As a consequence of it's actions, the agent recives an reward $r_{t+1} \in R$ and moves to the next state s_{t+1}. In this way, the agent and the MDP give rise to a sequence $S_0, A_0, R_1, S_1, A_1, R_2 \ldots$ and so on. The *horizon* is defined as the number of time-steps in an episode. If the horizon is finite, we call the MDP is finite.

The *return* is the discounted sum of rewards from time t until horizon and is denoted as $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} \ldots$, where $\gamma \in [0, 1]$ is the discounted factor.

The *values function* $V(s)$ represents the expected return from state s under the policy π, and is defined as follow: $V(s) = \mathbb{E}[G_t|s_t]$.

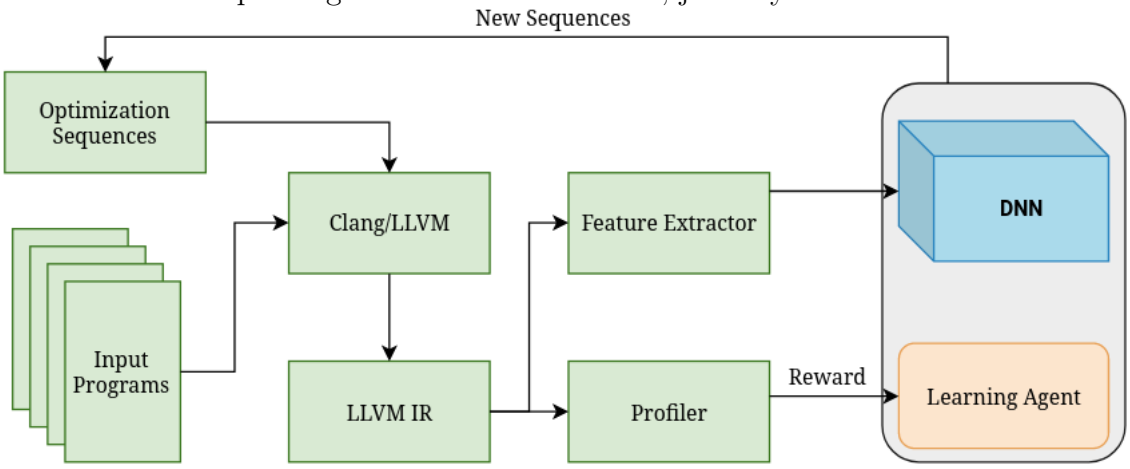
The Reinforcement Learning optimization problem is to improve the policy to maximize the expected return. The optimal policy is given by: $\pi^* = argmax_\pi \mathbb{E}[R|\pi]$.

Q-learning

Since model based RL can be computationally expensive, an alternativally approach that is computattionaly feasible is Q-Learning. Is this case, are not going to learn a model of how the work works.

1. the transitions are unknown
2. the rewards are unknown

here we are not especifing how to tkae the action, just if you do that



Initialize $Q(s, a)$ for all pairs (s, a)
Observe an transition $< s_t, a_t, r_t, s_{t+1} >$
Calculate TD-error $delta(s_t, a_t) = r_t + \gamma max_{a'} Q(s_{t+1}, a') - Q(s, a_t)$
Update estimate of $Q(s_t, a_t)$
$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha\delta(s_t, a_t)$
Drawback: only propagates experience one step

Policy Search

Given π space, search for $argmax_{\pi}V^{\pi}$. Parameterized policy and do stochastic gradient descent.

Deep Reinforcement Learning

Using a neural network in conjunction with RL is called deep RL. Deep RL is gaining popularity due it's success in Robotics, Atari games and superhuman capabilities.

"If the number of steps the RL agent has to take before the environment terminates is one, the problem is called Contextual Bandits; In Contextual Bandits the learner tries to find a single best action in the current state. It involves learning to search for best actions and trial-and-erro"

What distinguishes RF from other supervised machine learning approaches is the presence of self-exploration and exploitation, and the trade-off between them.

In [NeuroVectorizer: End-to-End Vectorization with DeepReinforcement Learning] the authors where able to learn with RL with fewer samples than supervised learning.

"n our case, RL can learn with fewer samples thanthat required in the supervised learning methods and canco-optimize for multiple objectives such as compilation time,code size, and execution time. ... The rewardcan also be defined as a combination of the compilation time,execution time, generated assembly code size,etc."

gradient policy approach

2.2.4 jogando info

MDP planning is when we know how the environment works, that means that we know how the transition dynamics are and we know how the reward model is and we want to compute the optimal behavior:

bellman equation

$Q^*(s,a) = R(s,a) + \gamma$

It means that we know the optimal sum of discounted values of starting in state s and taking the action a

values iteration converges in the limit, doing bellman backups. Since bellman operator is a contraction, it will converge if $\gamma < 1$:

$Q_0 = 0$; $Q_{t+1}(s,a) = R(s,a) + \gamma$

policy iteration converges in finite time:

$Q_0(s,a) = 0$ $\pi_t(s) = argmax_a Q_t(s,a)$ $Q_{t+1}(s,a) = R(s,a) + \gamma$

Model Based RL are computationally expensive.

1. Use experience to estimate model T and R
2. Use estimated models to estimate Q
3. Use estimate Q to estimate π

2.2.5 Propose

Fig [] shows the proposed framework for automatic code reduction. The set of Angha source programs is fed to the framework. The extractor compiles the programs to IR. The agent selects a new transformation to append to the sequence. The agent them compiles the program with clang/LLVM to gather code size improvements, which are used as rewards to the RL agent []

"if the agent accidentally injected bad pragmas, the compiler will ignore it"[]

"Once the model is trained it can be plugged in as-is for inference without further retraining ... It can still be beneficial to keep online training activated so that when completely new loops are observed, the agent can learn how to optimize them too."

Novas alternativas (code2vec, inst2vec)

The RL environment

I must define the Action Space, Observation Space and Reward

"Applying Multiple Passes per Action: An alternative to the action formulation above is to evaluatea complete sequence of passes with lengthNinstead ofa single actionaat each RL iteration. [AUTOPHASE]"

o learn a good policy, it is necessary to appropriately defineactions, rewards, and states. We define the reward as follows: $reward =$

One action picks one sequence from: $seq \in seq_1, seq_2, \ldots, seq_n$

3

Proposta

3.1 Prevendo comportamento com redes neurais

Proposta 01 - Mais segura - (Recurrent Neural Networks for Performance and Power Prediction

Esta idéia é baseada nos seguinte artigos:

1. Predicting HPC parallel program performance based on LLVM compiler
2. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction
3. Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores
4. Less is More: Exploiting the Standard Compiler Optimization Levels for Better Performance and Energy Consumption
5. Using LSTM and GRU Networks Methods for Traffic Flow Prediction.

Redes GRU são relativamente novas em comparação a redes LSTM: foram introduzidas por Kyunghyun Cho *et al* em 2014 (). Seu desempenho tem se mostrado equivalente a redes LSTM em áreas como processamento de sinais, linguagem natural, predição de tráfego e em alguns casos, melhores resultados em tarefas com conjuntos de dados menores

e menos frequentes. No entanto, segundo o conhecimento do autor, redes GRU não foram exploradas em nenhum contexto de compilação.

Além da pesquisa por modelos de aprendizagem de máquina que sejam eficientes e precisos para compiladores, a busca por uma representação generalizada para programas impulsiona atualmente os pesquisadores e basicamente é dividida em duas frentes: características extraídas manualmente ou representações baseadas em PLN. Com esta motivação, nesta pesquisa, representações de código serão empiricamente avaliadas com redes neurais recorrentes, são estas:: Tokens C, tokens IR-LLVM[1] e *features* de Namolaru e Loop[2]. diferentes

No escopo da literatura especializada em compiladores, os trabalhos estado da arte que prevem ganho de desempenho e energia focam em aplicações especializadas, isto é, aplicações paralelas 1) 3), sistemas embarcados 4), simulação microarquitetural 2). Dentre estes, os autores que utilizam redes neurais 1),2),3) criam seus modelos de previsão utilizando apenas MLPs, o que motiva o estudo de diferentes redes neurais profundas para um modelo geral de predição de energia e performance que não foque apenas em aplicações específicas.

Uma vez que redes neurais são aproximadores universais de funções, e com disponibilidade de um conjunto de treino heterogêneo, podemos obter avanços em questões como prever um conjunto de dados baseado em outro distinto e prever programas de modo geral. (Brauckmann et al., 2020) utilizam validação cruzada em 7 conjuntos de benchmarks distintos e avaliam 3 trabalhos estado da arte na tarefa de predição de *threads* e mapeamento de hardware. Na abordagem tanto o autor com sua metodologia como os trabalhos comparados não foram capazes de obter resultados satisfatórios quando treinados com a técnica de validação cruzada; na maioria dos casos o lançar de uma moeda foi mais satisfatório, o que mostra que as redes falharam na aprendizagem. Com o conjunto de benchmarks disponível em YaCoS, conjecturamos que é possível contornar este problema e obter uma generalização satisfatória. Além disso, como serão criados modelos com diferentes formatos de programas, podemos ter *insights* a respeito da representatividade de programas.

Ainda no trabalho de Brauckmann *et al* (Brauckmann et al., 2020) apesar da performance baixa, modelos sequenciais LSTM com representação mais proxíma da semântica do programa foram mais satisfatórios na tarefa de granularidade de *threads* do que MLP e Redes de Grafo. Os modelos foram treinos com técnicas de tokenização; nenhum trabalho com redes recorrentes ou de grafos utilizou features manuais. O cénario muda

[1]https://github.com/tud-ccc/compy-learn
[2]https://gitlab.com/andrefz/feature-extractor

quado abordado o mapeamento de hardware de programas OpenCL. Neste caso GNNs apresentaram melhor desempenho com uma representação baseada em Árvore Sintática Abstrata (AST), ou seja, uma representação mais sintática. Esta divergência mostra que existem modelos que desempenham uma tarefa melhor que outro quando se trata de compilção, mesmo quando estes são técnicas da mesma família de aprendizagem supervisionada, neste caso, redes neurais. Porém, ainda pouco se sabe sobre o impacto dos diferentes formatos de programas para redes profundas em diversas tarefas, portanto, é necessário mais investigação.

Outra questão é: **Como utilizar Aprendizagem por Transferência ?**

Proposta 02 - Incerta - Prediction of Polyhedral Optimizations using Deep Learning

Esta proposta é baseada nos seguintes artigos:

- Predictive Modeling in a Polyhedral Optimization Space - 2013
- Energy Auto-tuning using the Polyhedral Approach - 2014

Vantagens: O extrator de loops é uma ótima representação para programas focados em transformações de loop.

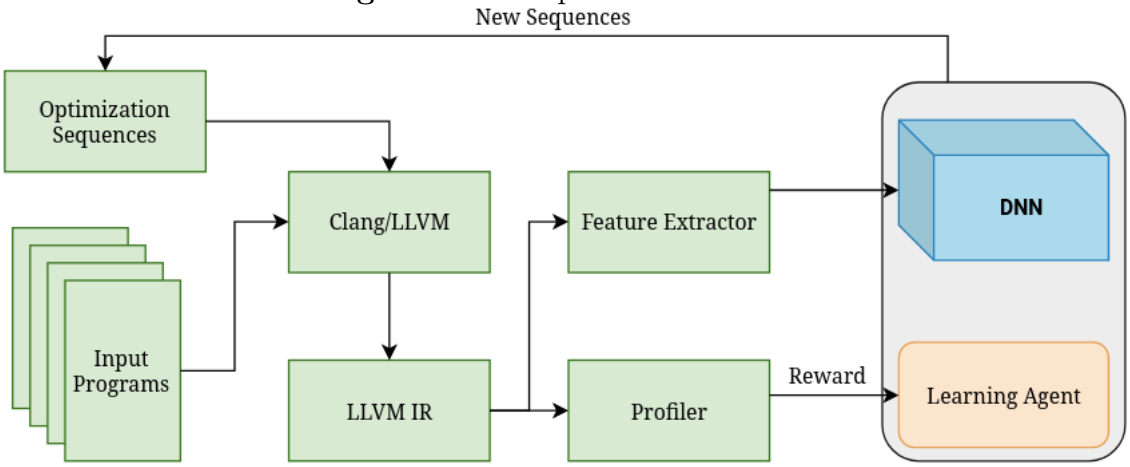
Polly

Proposta 03 - Mais improvável

Esta proposta é baseada nos seguintes artigos:

- Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning
- Compiler Phase Ordering as an Orthogonal Approach for Reducing Energy Consumption.
- Preliminary results for neuro evolutionary optimization phase order generation for static compilation.

Figura 3.1: Proposed framework.



Fonte: (XAVIER, 2014).

4

Trabalhos Relacionados

Este capítulo discute trabalhos relacionados com relação aos temas abordados nesta dissertação. Seção 4.1 apresenta trabalhos prévios em otimização para compiladores guiada por aprendizagem de máquina. Na seção 4.1.3 é discutido um duas classes de aplicações, predição de desempenho e criação de heurísticas de compilação. Finalmente, a seção 4.2 apresenta a principal literatura em Aprendizagem Profunda para compiladores.

4.1 Autotuning para compiladores de alto desempenho

Sistemas de autotuning exploram o espaço de buscam de possíveis implementações para uma aplicação para geração e otimização automática de código para diferentes cenários e arquiteturas. Um algoritmo de busca guiado por diversas estratégias navega neste espaço a fim de encontrar pontos que maximizem o ganho do sistema; este processo é conhecido como compilação iterativa (). A pesquisa muitas vezes envolve compilar, executar e medir o desempenho da aplicação para o hardware em questão, portanto, este processo pode ser muito custoso dependendo do objetivo em questão. A fim de minimizar o custo de busca, técnicas de aprendizagem de máquina são amplamente utilizadas na literatura especializada. Esta seção providencia uma introdução aos sistemas de autotuning.

4.1.1 Compilação Iterativa Inteligente

Compilação Iterativa se provou uma estratégia eficiente para investigação empírica do comportamento de programas no espaço de transformações []. Chen *et al* (Chen et al.,

2010) mostram que a metodologia também é aplicavel para qualquer domínio de programas, avaliando 10,000 benchmarks e obtendo resultados com 2x de ganho de speedup com relação ao plano -O3 do GCC, consolidando ainda mais a técnica.

Com eficâcia comprovada, muitas pesquisas focam em minimizar o custo da compilação iterativa. O trabalho de Bodin *et al* (Bodin et al., 1998) foi pioneiro ao utilizar compilação iterativa de forma inteligente com busca aleatória, que explora menos que 2% do espaço. Agakov*et al* (Agakov et al., 2006) também inova ao utilizar modelagem preditiva parar otimizar o espaço de busca de transformações. Filho *et al* (J. F. Filho, 2018) apresentam um sistema com diferentes estratégias para redução do espaço de busca, guiado por aprendizagem contínua e um motor que emprega raciocínio baseado em casos. Ashouri *et al* (Ashouri et al., 2016) propõe um sistema de autotuning misto de aprendizagem de máquina e compilação iterativa para o problema de seleção de transformações. Outro trabalho que otimiza a tarefa de compilação iterativa é apresentado por Ogilvie *et al* (Ogilvie et al., 2017). Os autores utilizam técnicas de aprendizagem ativa para minimizar o custo de compilação iterativa. Sendo capaz de reduzir até aproximadamente pela metada o número de execuções de uma aplicação.

4.1.2 Infraetruturas de Exploração

Os benefícios proporcionados por aprendizagem de máquina motivou a construção de infraestruturas de compilação que permitem explorar o espaço de otimizações, integração de estratégias e reprodução de experimentos. *Milepost GCC* (Fursin et al., 2011) foi o primeiro projeto que visou a criação de um compilador de produção guiado por aprendizagem de máquina. Este possui uma interface de extração de features estáticas e controle de sequencias de transformações. O sistema de autotuning utiliza 32 programas do conjunto *cBench*. *OpenTuner* é um framework de autotuning multi-objetivo com técnicas de busca independente do domínio. (Ansel et al., 2014). *Mystery* () é uma infraestrutura para exploração e design de sequencias de transformações, assim como Milepost, porém mais robusta. O sistema possui um motor de busca que disponibiliza diversos algoritmos de aprendizagem e também possibilita a caracterização de programas estaticamente ou dinamicamente e diversas métricas para medir a distância entre programas. Mystery utiliza LLVM como base de compilação e disponibiliza um conjunto de mais de 5,000 mil C benchmarks. Um frameworks para compilação iterativa que oferecem abstrações para o processo de compilação iterativa para programas OpenCL é apresentado por Cummins *et al* (Nugteren and Codreanu, 2017).

Uma classe de otimizadores automatizados que também recebe grande atenção são os *Superotimizadores*, que buscam a forma canônica para blocos do programa, isto é, a sequência ótima. Superotimizadores usualmente utilizam busca exaustiva (Bansal and Aiken, 2006), aleatória (Schkufza et al., 2012) ou síntese (Sasnauskas et al., 2017).

4.1.3 Sistemas de predição em HPC

Modelos de aprendizagem de máquina tem a habilidade de prever um resultado para um conjunto de dados baseado em dados anteriores. No contexto de sistemas de alto desempenho, predição de desempenho significa compreender o comportamento que uma aplicação pode obter em um cenário específico. Este comportamento pode ser formulado como um problema de classificação, ou regressão, atuando em valores contínuos ou discretos, respectivamente.

Modelos de predição para valores contínuos encontram aplicações em tarefas como determinação de *speedup* e consumo de energia por exemplo. Em geral preditores recebem como entrada uma tupla (P, T) onde T é a sequencia de transformações, e P o programa de teste e como saída o valor de predição desejado. Diversos trabalhos utilizam algoritmos de ajuste de curva para estimar ganho de *speedup* programas sequências (Vaswani et al., 2007) (Lee and Brooks, 2006) e OpenMP (Curtis-Maury et al., 2008). Luk *et al* (Luk et al., 2009) estima o tempo de execução para CPU e GPU de acordo com a entrada de programas OpenCL. (Wen et al., 2014) extedem o trabalho anterior para agendamento de tarefas em tempo de execução. O trabalho anterior de Brewer *et al.* (Brewer, 1995) propõe um modelo de regressão baseado em um *solver* de equações diferencias parciais para prever tempo de execução de *layout* de dados. Para predição de energia Benedict (Benedict et al., 2015) utilizam a abordagem de floresta aleatória para estudar o comportamento de programas OpenMP, Rejitha(R.S. et al., 2017) preveem o consumo de energia de código CUDA para GPGPU por métodos de regressão dinâmicos.

No domínio de predição de valores discretos tarefas como mapeamento de hardware, desdobramento de loops, granularidade de threads são domínios amplamente estudados e de alta complexidade. Para predição de desdobramento de loops ténicas supervisionadas como, árvores de decisão (Monsifrot et al., 2002), florestas aleatórias (Zacharopoulos et al., 2018), K viznhos próximos (KNN) e máquinas de vetores de suporte (SVM) (Stephenson and Amarasinghe, 2005) são utilizadas. Quando programas OpenCL são o foco da otimização, dois principais problemas são estudados: mapeamento de CPU/GPU que consiste em classificar um problema como CPU ou GPU dependendo em qual plataforma

de hardware este tem execução mais rápida e fator de granularidade de threads, similar a desdobramento de loops porém no contexto de ambientes paralelos

. Wen *et al* (Wen et al., 2014) utilizam SVM para a tarefa. Ardalani *et al* (Ardalani et al., 2015) utilizam regressão *stepwise* seguido pela meta-heurística de agregação via *bootstrap*. Moren e Göhringer (Moren and Göhringer, 2018) utilizam regressão por floresta aleatória. Fan *et al* (Fan et al., 2019) prevem o speedup que diz a respeito da frequência entre número de núcleos e memória para programas OpenCL utilizando diversos regressores lineares como LASSO e SVR.Sarkar e Mitra (Sarkar and Mitra, 2014) propõe um método não baseado em aprendizagem capaz de prever o tempo gasto por um programa em diferentes partes do seu código. A vantagem de tal metodologia é que esta proporciona a capacidade de identificar quais partes do código consomem mais recursos, e portanto limitam o desempenho em plataformas com GPU. Redes neurais artificiais apresentam resultados promissores para o problema de threads (Magni et al., 2014) e (Cummins et al., 2017a).

Predição de sequências de compilação para mitigação do PSF e POF também são problemas classificação. Estratégias que buscam encontrar bons planos de compilação para o PSF utilizando uma abordagem probabilistica baseada em regressão logística pode ser visto no trabalho de Cavazos et al (Cavazos et al., 2007) (Cavazos and O'Boyle, 2006). Ashouri *et al* (Ashouri et al., 2014) também faz uso de probabilidades com um rede Bayesiana para encontrar planos de compilação que maximizam a performance da aplicação alvo. Por outro lado, Xavier e Silva (Xavier e Silva, 2018) utilizam técnicas não supervisionadas de clusterização e meta-heurísticas para além de encontrar bons planos de compilação e também a melhor ordem de aplicação. Ashouri *et al* (Ashouri et al., 2016) fazem uso de utilizadas redes Bayesianas para derivar uma distribuição das melhores transformações e então aplicam compilação iterativa sob esta distribuição. Os autores relatam melhores resultados em comparação a utilização exclusiva de compilação iterativa ou aprendizagem de máquina. Ashouri *et al* (Ashouri et al., 2017) utilizam redes neurais e diversos outros algoritmos de regressão linear para mitigação do POF. Park (PARK, 2015) utiliza SVMs em uma predição denominada como predição por torneio, onde uma classificação binária é realizada entre duas sequências.

4.1.4 Seleção de Features

Modelos de aprendizagem de máquina utilizam features para representar as características dos dados a serem processados; Levando em consideração otimização de código, e o número

ilimitado de representações possíveis para um programa a escolha das características mais representativas passa a ser um problema difícil.

Diferentes tipos de características para representar programas tem sido utilizadas, usualmente classificadas entre estáticas, geralmente extraídas da representação itermediária do programa e dinâmicas, obtidas a partir da execução. Cavazos et al (Cavazos et al., 2007) propõem o uso de contadores de performance, um conjunto compacto de informações que representam o comportamento dinâmico da aplicação. Nestes dados, são incluídos chace misses e utilizações de unidade de ponto flutuante, por exemplo. Por outro lado, Namolaru *et al* (Namolaru et al., 2010) avaliam um conjunto de 56 features estáticas obtidas através de programação lógica em relações. Estas sumarizam informações a respeito de funções, instruções, operandos, variáveis, tipos, constantes, blocos básicos e loops. Este conjunto de características tem se provado uma representação superior quando comparado com os dados estatísticos fornecidos pelo passe *-stats* presente no otimizador LLVM ().

Métodos que automatizam a extração de features incluem o sistema *Hercules* (Park et al., 2014). Os autores desenvolvem uma metodologia de caracterização estática de programas guiada por um motor que utiliza programação lógica para busca de padrões em código a fim de efetuar à extração de características de loops. Muitas destas informações dizem a respeito de dependências de dados e dependências carregadas por loops. Leather *et al* (Leather et al., 2014) apresentam um método automático de selecionar features através programação genética em um espaço de características definido por uma gramática. Devido ao tamanho do espaço de busca, uma gramática de 160kB foi escrita.

Usualmente, técnicas de aprendizagem de máquina utilizam vetores para estruturar características. Park *et al* (Park et al., 2012) apresentam uma abordam única baseada em estruturas de grafo para representatividade de programas ao invés de estruturas de vetores. Resultados superiores a representações estáticas utilizando vetores e contadores de performance foram obtidas utilizando Máquinas de Vetor Suporte (SVM). No entanto, o treinamento se torna mais custoso uma vez que é necessário realizar travessias em grafos.

Todas os trabalhos e técnicas apresentadas possuem um fator em comum: todas as features são selecionadas manualmente. Este processo é sujeito a má representabilidade, pois depende da escolha do designer a respeito das características que mais afetam o desempenho e requer conhecimento especializado do sistema mesmo quando a técnica é automatizada. Técnicas que buscar contornar este problemas são apresentadas na seção 4.2.1.

4.2 Redes neurais para Compiladores

Esta seção providencia um sumário a respeito dos trabalhos realizados em computação de alto desempenho que fazem uso de redes neurais como proposta de solução. A seção 4.2.1 apresenta algumas representações de programas utilizadas em redes neurais. Estas diferem em sua maioria das representações apresentadas na seção 4.1.4 pois são inspiradas pelas técnicas de processamento de linguagem natural.

Redes de Alimentação Adiante. O modelo base de redes multicamadas é muito utilizado para criação de modelos de predição em ambientes de alta performance. Curtis-Maury *et al* (Curtis-Maury et al., 2007) fazem uso destas redes para prever o consumo de energia de aplicações OpenMP em sistemas multinúcleo. Por sua vez Magni *et al* (Magni et al., 2014) utilizam o modelo cascada de redes neurais para prever o melhor fator de granularidade de threads para GPUs. Kulkarni e Cavazos (Kulkarni and Cavazos, 2012) empregam redes neurais para mitigar o problema de ordenação de fase (POF); os autores preveem a performance que a ordem das transformações pode proporcionar para o código sendo otimizado. Ïpek *et al* (Ïpek et al., 2006) utilizam redes neurais para previsão de performance no contexto de exploração do espaço de design arquitetural. Os autores reduzem o número de pontos necessários para simulação. O cálculo de desempenho ocorre através da simulação de um subconjunto de pontos previstos a partir de valores de parâmetros de microarquitetura. Lee *et al* (Lee et al., 2007) realizam um estudo comparativo entre modelos de regressão não linear, como clusterização e ANNs na tarefa de predição de performance no contexto de parâmetros de entradas variáveis. Os resultados indicam que cada método possui vantagens em determinados cenários, no entanto, o processo de treinamento é significativamente mais simplificado quando utilizado ANNs. Dubach *et al* (Dubach et al., 2007) prevem a performance de um programa otimizado utilizando características estáticas e dinâmicas coletadas a partir da execução da aplicação com uma única thread.

Redes Neurais Recorrentes. Diferente de redes multicamadas regulares, RNNs são capazes estimar distribuições baseadas em n observações anteriores, tornando esta classe de redes adequada para problemas de predição de séries temporais. Cummins *et al* (Cummins et al., 2017a) utilizam redes recorrentes LSTM para criar um modelo de predição de granularidade de threads para programas OpenCL, como também o mapeamento em CPU ou GPU. Os autores também propõe um método para representar programas baseado no apenas nas características sintáticas do programa, inspirado por técnicas de processamento de linguagem natural. Ben-Nun *et al* (Ben-Nun et al., 2018) também modelam a própria representação, porém ao contrário de Cummins *et al*, a representação

dos autores ocorre com código intermediário LLVM. Redes LSTM foram avaliadas em duas tarefas de predição similar ao trabalho anterior: mapeamento de hardware e fator de granularidade ótimo de threads. Ambos os trabalhos obtiveram melhores resultados com suas representações em comparação a features extraídas manualmente, e resultados similares quando comparados entre si.

Redes Neurais Baseadas em Grafo. Esta classe de redes neurais utilizam estruturas de grafos como entrada. A representação em forma de grafo, em contraste com o modelo sequencial de vetor, é mais enriquecedora para o modelo, uma vez que podem ser quantificadas informações a respeito do fluxo de controle de um programa, e não somente instruções. Brauckmann *et al* (Brauckmann et al., 2020) atacam os mesmos problemas do trabalho de Cummins e Ben-Nun e demonstram que o ganho de desempenho em utilizar grafos é similar ou melhor a abordagem sequencial. O trabalho de Rosário *et al* () é similar ao trabalho de Brauchmann ao utilizar GNNs como mecanismo de aprendizagem, porém, o autor utiliza estas para prever o tempo de execução de programas dado uma sequência de otimizações.

4.2.1 Representação de Programas

Redes neurais não necessitam de conhecimento preliminar fornecido pela base de conhecimento. Nesta seção discutimos alguns trabalhos que não utilizam features pré-definidas e permitem que o próprio modelo aprenda e selecione as features mais apropriadas.

Cummins *et al* (Cummins et al., 2017a) como já mencionado na seção 4.2 utilizam uma representação sintática, esta se da por meio de uma sequência de tokens representada por números inteiros em um vetor. Ben-Nun *et al* (Ben-Nun et al., 2018) por outro lado, constroem uma representação independente do código fonte, em cima da representação intermediária em cima da representação intermediária da infraestrutura LLVM. Este modelo além de quantificar informações a respeito dos dados também leva informações sobre o fluxo de controle do programa. Ambas as propostas apresentam resultados similares quando testados em problemas de classificação no sentido de que em um certo domínio ou certos benchmarks um modelo é mais performático que outro.

Deixando de lado vetores e adotando estruturas de grafos podemos citar o trabalho de Cvitkovic *et al* (Cvitkovic et al., 2018) que propõe um modelo baseado em árvores sintáticas abstratas porém também carregando informação semática através de anotações. Cada elemento do vocabulário é representado como um nó de um grafo; o resultado final é AST anotada com informações a respeito dos dados e fluxo de controle pronta para ser

processada utilizando Redes Neurais de Grafos. Brauckmann *et al* (Brauckmann et al., 2020) definem duas representações para código:

AST+DF: Uma AST enriquecida com informações de fluxo de dados, onde os nós são rotulados como declarações, instruções e tipos e as arestas são rotuladas como tipo AST, representando a relação "filho-pai", e fluxo de dados, representando cadeias *use-def* de variáveis.

CDFG+CALL+MEM: Um grafo de fluxo de controle (CFG) enriquecido com informações de chamada e nós de memória rotulados com instruções. Arestas do tipo controle e dataflow definem a base da representação CDFG. Arestas CALL com base em dependências para retornar valores de funções. Arestas MIM representam dependências *store-load* de memória.

Os dois modelos são processados por Redes Neurais de Grafos e comprados com DeepTune (Cummins et al., 2017a) e inst2vec (Ben-Nun et al., 2018). Os resultados a respeito da melhor representação foram inconclusivos.

Cummins *et al* () apresentam *ProGraML*, uma representação de grafo construída em cima da representação intermediária LLVM e XLA...

A tabela 4.2.1 apresenta um sumário de todos os trabalhos abordados nesta seção.

Modelo	**Representação de código**	**Extração**	**Arquitetura**
Curtis-Maury *et al*	Estática	Manual	Multilayer Perceptron
Magni	-	Manual	Multilayer Perceptron
Kulkarni e Cavazos *et al*	-	Manual	Multilayer Perceptron
Ïpek *et al*	-	Manual	Multilayer Perceptron
Lee *et al*	-	Manual	Multilayer Perceptron
Dubach *et al*	-	Manual	Multilayer Perceptron
Wen *et al*	Estática & Dinâmica	Manual	Multilayer Perceptron
DeepTune	Estática (C Tokens)	Automática	Recurrent neural network
inst2vec *et al* (inst2vec)	Estática	Automática	Recurrent neural network
Cvitkovic *et al*	Estática	Automática	Graph neural network
Brauckmann *et al*	Estática	Automática	Graph neural network
Rosário *et al*	Estática	Automática	Graph neural network
ProGraML	-	Automática	-

Tabela 4.1: Arquitetura de rede, representação de código, método de extração dos trabalhos presentas na seção 4.2

5

Conclusão

REFERÊNCIAS

AGAKOV, F., BONILLA, E., CAVAZOS, J., FRANKE, B., FURSIN, G., O'BOYLE, M. F. P., THOMPSON, J., TOUSSAINT, M., and WILLIAMS, C. K. I. (2006). Using machine learning to focus iterative optimization. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '06, pages 295–305, Washington, DC, USA. IEEE Computer Society.

Agakov, F., Bonilla, E., Cavazos, J., Franke, B., Fursin, G., O'Boyle, M. F. P., Thomson, J., Toussaint, M., and Williams, C. K. I. (2006). Using machine learning to focus iterative optimization. In *International Symposium on Code Generation and Optimization (CGO'06)*, pages 11 pp.–305.

Allamanis, M., Barr, E. T., Bird, C., and Sutton, C. (2014). Learning natural coding conventions. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, page 281–293, New York, NY, USA. Association for Computing Machinery.

Ansel, J., Kamil, S., Veeramachaneni, K., Ragan-Kelley, J., Bosboom, J., O'Reilly, U., and Amarasinghe, S. (2014). Opentuner: An extensible framework for program autotuning. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 303–315.

Ardalani, N., Lestourgeon, C., Sankaralingam, K., and Zhu, X. (2015). Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 725–737, New York, NY, USA. Association for Computing Machinery.

Ashouri, A. H., Bignoli, A., Palermo, G., Silvano, C., Kulkarni, S., and Cavazos, J. (2017). Micomp: Mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning. *ACM Trans. Archit. Code Optim.*, 14(3).

Ashouri, A. H., Killian, W., Cavazos, J., Palermo, G., and Silvano, C. (2018). A survey on compiler autotuning using machine learning. *ACM Comput. Surv.*, 51(5).

Ashouri, A. H., Mariani, G., Palermo, G., Park, E., Cavazos, J., and Silvano, C. (2016). Cobayn: Compiler autotuning framework using bayesian networks. *ACM Trans. Archit. Code Optim.*, 13(2).

Ashouri, A. H., Mariani, G., Palermo, G., and Silvano, C. (2014). A bayesian network approach for compiler auto-tuning for embedded processors. In *2014 IEEE 12th Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*, pages 90–97.

Bansal, S. and Aiken, A. (2006). Automatic generation of peephole superoptimizers. ASPLOS XII, page 394–403, New York, NY, USA. Association for Computing Machinery.

Ben-Nun, T., Jakobovits, A. S., and Hoefler, T. (2018). Neural code comprehension: A learnable representation of code semantics. *CoRR*, abs/1806.07336.

Benedict, S., Rejitha, R. S., Gschwandtner, P., Prodan, R., and Fahringer, T. (2015). Energy prediction of openmp applications using random forest modeling approach. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 1251–1260.

Bodin, F., Kisuki, T., Knijnenburg, P., O' Boyle, M., and Rohou, E. (1998). Iterative compilation in a non-linear optimisation space. In *Workshop on Profile and Feedback-Directed Compilation*, Paris, France.

Brauckmann, A., Goens, A., Ertel, S., and Castrillon, J. (2020). Compiler-based graph representations for deep learning models of code. In *Proceedings of the 29th International Conference on Compiler Construction*, CC 2020, page 201–211, New York, NY, USA. Association for Computing Machinery.

Brewer, E. A. (1995). High-level optimization via automated statistical modeling. In *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '95, page 80–91, New York, NY, USA. Association for Computing Machinery.

Cavazos, J., Fursin, G., Agakov, F., Bonilla, E., O'Boyle, M. F. P., and Temam, O. (2007). Rapidly selecting good compiler optimizations using performance counters. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '07, page 185–197, USA. IEEE Computer Society.

Cavazos, J. and O'Boyle, M. F. P. (2006). Method-specific dynamic compilation using logistic regression. *SIGPLAN Not.*, 41(10):229–240.

Chen, Y., Huang, Y., Eeckhout, L., Fursin, G., Peng, L., Temam, O., and Wu, C. (2010). Evaluating iterative optimization across 1000 datasets. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10, page 448–459, New York, NY, USA. Association for Computing Machinery.

Cummins, C., Petoumenos, P., Wang, Z., and Leather, H. (2017a). End-to-end deep learning of optimization heuristics. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 219–232.

Cummins, C., Petoumenos, P., Wang, Z., and Leather, H. (2017b). Synthesizing benchmarks for predictive modeling. In *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 86–99.

Curtis-Maury, M., Shah, A., Blagojevic, F., Nikolopoulos, D. S., de Supinski, B. R., and Schulz, M. (2008). Prediction models for multi-dimensional power-performance optimization on many cores. In *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 250–259.

Curtis-Maury, M., Singh, K., McKee, S. A., Blagojevic, F., Nikolopoulos, D. S., de Supinski, B. R., and Schulz, M. (2007). Identifying energy-efficient concurrency levels using machine learning. In *2007 IEEE International Conference on Cluster Computing*, pages 488–495.

Cvitkovic, M., Singh, B., and Anandkumar, A. (EasyChair, 2018). Deep learning on code with an unbounded vocabulary. EasyChair Preprint no. 466.

Dubach, C., Cavazos, J., Franke, B., Fursin, G., O'Boyle, M. F., and Temam, O. (2007). Fast compiler optimisation evaluation using code-feature based performance prediction. In *Proceedings of the 4th International Conference on Computing Frontiers*, CF '07, page 131–142, New York, NY, USA. Association for Computing Machinery.

Fan, K., Cosenza, B., and Juurlink, B. (2019). Predictable gpus frequency scaling for energy and performance. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP 2019, New York, NY, USA. Association for Computing Machinery.

Fursin, G., Kashnikov, Y., Memon, A. W., Chamski, Z., Temam, O., Namolaru, M., Yom-Tov, E., Mendelson, B., Zaks, A., Courtois, E., Bodin, F., Barnard, P., Ashton,

E., Bonilla, E., Thomson, J., Williams, C. K. I., and O'Boyle, M. (2011). Milepost GCC: Machine Learning Enabled Self-tuning Compiler. *International Journal of Parallel Programming*, 39:296–327.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

İpek, E., McKee, S. A., Caruana, R., de Supinski, B. R., and Schulz, M. (2006). Efficiently exploring architectural design spaces via predictive modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, page 195–206, New York, NY, USA. Association for Computing Machinery.

J. F. Filho, L. G. A. Rodriguez, A. F. d. S. (2018). Yet another intelligent code-generating system: A flexible and low-cost solution. *Journal of Computer Science and Technology*, 33(5):940.

JUNIOR, N. L. Q. (2016). *Uma solução hibrida para mitigação do problema de seleção de otimizações*. Mestrado, Universidade Estadual de Maringa.

Kulkarni, S. and Cavazos, J. (2012). Mitigating the compiler optimization phase-ordering problem using machine learning. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, page 147–162, New York, NY, USA. Association for Computing Machinery.

Leather, H., Bonilla, E., and O'boyle, M. (2014). Automatic feature generation for machine learning–based optimising compilation. *ACM Trans. Archit. Code Optim.*, 11(1).

Lee, B. C. and Brooks, D. M. (2006). Accurate and efficient regression modeling for microarchitectural performance and power prediction. *SIGOPS Oper. Syst. Rev.*, 40(5):185–194.

Lee, B. C., Brooks, D. M., de Supinski, B. R., Schulz, M., Singh, K., and McKee, S. A. (2007). Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '07, page 249–258, New York, NY, USA. Association for Computing Machinery.

Luk, C., Hong, S., and Kim, H. (2009). Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 45–55.

Magni, A., Dubach, C., and O'Boyle, M. (2014). Automatic optimization of thread-coarsening for graphics processors. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, page 455–466, New York, NY, USA. Association for Computing Machinery.

MASSALIN, H. (1987). Superoptimizer: A look at the smallest program. *SIGARCH Comput. Archit. News*, 15(5):122–126.

Monsifrot, A., Bodin, F., and Quiniou, R. (2002). A machine learning approach to automatic production of compiler heuristics. In *Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, AIMSA '02, page 41–50, Berlin, Heidelberg. Springer-Verlag.

Moren, K. and Göhringer, D. (2018). Automatic mapping for opencl-programs on cpu/gpu heterogeneous platforms. In Shi, Y., Fu, H., Tian, Y., Krzhizhanovskaya, V. V., Lees, M. H., Dongarra, J., and Sloot, P. M. A., editors, *Computational Science – ICCS 2018*, pages 301–314, Cham. Springer International Publishing.

Namolaru, M., Cohen, A., Fursin, G., Zaks, A., and Freund, A. (2010). 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. Association for Computing Machinery.

Nielsen, M. A. (2018). *Neural Networks and Deep Learning*. Determination Press.

Nugteren, C. and Codreanu, V. (2017). Cltune: A generic auto-tuner for opencl kernels. *CoRR*, abs/1703.06503.

Ogilvie, W. F., Petoumenos, P., Wang, Z., and Leather, H. (2017). Minimizing the cost of iterative compilation with active learning. In *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 245–256.

Park, E., Cavazos, J., and Alvarez, M. A. (2012). Using graph-based program characterization for predictive modeling. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization*, CGO '12, page 196–206, New York, NY, USA. Association for Computing Machinery.

Park, E., Kartsaklis, C., and Cavazos, J. (2014). Hercules: Strong patterns towards more intelligent predictive modeling. In *2014 43rd International Conference on Parallel Processing*, pages 172–181.

PARK, E. J. (2015). *Automatic selection of compiler optimizations using program characterization and machine learning*. Phd, University of Delaware.

R.S., R., Benedict, S., Alex, S., and Infanto, S. (2017). Energy prediction of cuda application instances using dynamic regression models. *Computing*, 99.

Sarkar, S. and Mitra, S. (2014). Execution profile driven speedup estimation for porting sequential code to gpu. In *Proceedings of the 7th ACM India Computing Conference*, New York, NY, USA. Association for Computing Machinery.

Sasnauskas, R., Chen, Y., Collingbourne, P., Ketema, J., Taneja, J., and Regehr, J. (2017). Souper: A synthesizing superoptimizer. *CoRR*, abs/1711.04422.

Schkufza, E., Sharma, R., and Aiken, A. (2012). Stochastic superoptimization. *CoRR*, abs/1211.0557.

Stephenson, M. and Amarasinghe, S. (2005). Predicting unroll factors using supervised classification. In *International Symposium on Code Generation and Optimization*, pages 123–134.

Vaswani, K., Thazhuthaveetil, M. J., Srikant, Y. N., and Joseph, P. J. (2007). Microarchitecture sensitive empirical models for compiler optimizations. In *International Symposium on Code Generation and Optimization (CGO'07)*, pages 131–143.

Wen, Y., Wang, Z., and O'Boyle, M. F. P. (2014). Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms. In *2014 21st International Conference on High Performance Computing (HiPC)*, pages 1–10.

XAVIER, T. C. d. S. (2014). *Solução Integrada para os Problemas de Seleção e Ordenação de Fase*. Mestrado, Universidade Estadual de Maringa.

Zacharopoulos, G., Barbon, A., Ansaloni, G., and Pozzi, L. (2018). Machine learning approach for loop unrolling factor prediction in high level synthesis. In *2018 International Conference on High Performance Computing Simulation (HPCS)*, pages 91–97.