Análise do Desempenho de Aplicações Paralelas geradas por Ferramentas de Inteligência Artificial

João Vitor M. Dias, Antonio Carlos S. Beck, e Arthur F. Lorenzon[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{joao.dias, caco, aflorenzon}@inf.ufrgs.br

Resumo. *Este trabalho compara dois modelos de IA, um da Google e outro da OpenAI, na paralelização de aplicações em C++ usando OpenMP. Através da análise da corretude do código gerado e tempo de execução em uma arquitetura de 40 núcleos de processamento, mostramos que o modelo da OpenAI teve um desempenho superior, paralelizando mais aplicações que o modelo da Google.*

1. Introdução

A utilização de recursos de *Large Language Models* (LLMs) para a realização de tarefas de programação de diferentes níveis de complexidade popularizou-se com a ferramenta ChatGPT pela OpenAI. Desde então, diversas organizações têm disponibilizado seus próprios *softwares* de inteligência artificial (IA), utilizando diferentes arquiteturas de modelo que buscam melhores resultados. Tais *softwares* são amplamente utilizados na produção automática de códigos, na aceleração do ciclo desenvolvimento de softwares e na correção automática de códigos já existentes. Com a popularização desses softwares de IA, a programação paralela pode ser aprimorada e desenvolvida com a geração de códigos paralelos de modo automatizado, facilitando o processo de otimização de algoritmos [Navaux et al. 2023].

Assim, em um contexto de ampla oferta de softwares, surge a necessidade de comparar o desempenho dessas ferramentas para diferentes atividades de programação paralela em arquiteturas de alto desempenho. Análises similares já foram realizadas, como [Godoy et al. 2023], que analisou a produção de código paralelo por LLMs para diferentes linguagens de programação; [Valero-Lara et al. 2023], que comparou Llama-2 e GPT-3 para produção de códigos de alto desempenho. Diferentemente destes trabalhos, os quais requisitaram a produção total de código por modelos generativos ou analisaram as IAs junto a outros escopos de desenvolvimento, este artigo avalia a implementação paralela de códigos previamente escritos de maneira sequencial.

Portanto, neste trabalho, exploramos o escopo de computação de alto desempenho de dois modelos de IA, utilizando a linguagem C++. Foram analisadas a IA Code-Bison-001, da família Palm da Google, e a GPT-3.5-turbo-16k, da família GPT da OpenAI. A partir da paralelização de dez aplicações em C++ com OpenMP para os dois modelos e da execução das aplicações em uma arquitetura com 40 núcleos de processamento, mostramos que o modelo GPT conseguiu paralelizar corretamente um número maior de aplicações, obtendo desempenho melhor que o modelo Code-Bison-001.

2. Fundamentação Teórica

Large Language Models são modelos treinados em grandes quantidades de dados para realizar tarefas de processamento de linguagem natural, que usualmente utilizam a arquitetura de rede neural *transformer* [Vaswani et al. 2023]. Esta arquitetura se diferencia

das demais por usar mecanismos de atenção ao invés de redes neurais convolucionais ou recorrentes. O fluxo de dados em uma arquitetura *transformer* para processamento de linguagem natural inicia com a *tokenização* das palavras, seguida pela vetorização dos *tokens* gerados para formar *word embeddings*. Após a codificação do *input* estar pronta, os vetores passam por uma camada de atenção, a qual atribui pesos da importância de cada vetor em relação aos outros vetores do *input* para o cálculo do *output token*. Essa camada é baseada em *multiheaded self attention*, na qual cada cabeça de atenção do modelo é um espaço vetorial do modelo e está relacionada a uma possível relação entre os vetores. Após, a previsão dos *outputs tokens* é feita a partir de redes neurais diretas.

Os dois modelos utilizados neste trabalho fazem uso dessa arquitetura. Os modelos da família GPT foram introduzidos pela *OpenAI Research* a partir da ideia de que, um pré-treinamento de aprendizado não supervisionado, combinado com um *fine tuning* de aprendizado supervisionado, produziria grandes melhorias para tarefas de *natural language processing* [Radford et al. 2018]. Com essa maneira de treinamento o modelo aprende previamente relações de linguagens para apenas na etapa de *fine tuning* aprender as relações de um domínio específico. No caso dos modelos da Família Palm [Chowdhery et al. 2023], é utilizada a arquitetura *transformer* com uma configuração (*setup*) composta apenas pelo *decoder*, realizando o processamento da camada de atenção em paralelo, de forma semelhante ao trabalho de [Wang and Komatsuzaki]. A *Google Research* investiu no treinamento de um modelo com 540 bilhões de parâmetros em 6144 TPU v4 chips. Esse treinamento foi possível através do uso do sistema de *Pathways*, introduzido também pela *Google Research* [Barham et al. 2022].

3. Metodologia

A comparação dos modelos Code-Bison-001 e GPT-3.5-turbo-16k foi realizada a partir do uso de dez aplicações retiradas de [Lorenzon and Beck Filho 2019]: CCN (*Clenshaw Curtis Nested*), CCR (*Clenshaw Curtis Rule*), DFT (*Discrete Fourier Transform*), FFT (*Fast Fourier Transform*), GS (*Gram-Schmidt*), LU (*LU-Decomposition using Doolittle Method*), MC (*Simulação Monte Carlo*), MM (*Multiplicação de Matriz*), OE (*Ordenação Odd-Even*) e PO (*Equação de Poisson*). Para cada aplicação, foi requisitado aos dois modelos para paralelizar o código sequencial com o uso da interface de programação paralela OpenMP, através do seguinte prompt: `<Without omitting any code, Parallelize this code using OpenMP : + <Code>`. Para analisar o código gerado pelas IAs, classificamos os erros de acordo com os seguintes tipos:

- ***CE-1***: O código paralelo gerado na definição do *pragma* utilizou variáveis que ainda não haviam sido declaradas no código.
- ***CE-2***: O modelo produziu o código incompleto, devido a resposta inferida ultrapassar o número máximo de palavras por resposta que o modelo suporta. A inferência do modelo é encerrada abruptamente.
- ***CE-3***: O modelo produziu o código incompleto, com algumas funções não implementadas, sendo essas omitidas na inferência através de comentários, ou apenas declaradas no cabeçalho do código.

De modo similar, durante a execução da aplicação paralela, a mesma pode apresentar erros devido a geração incorreta do código paralela. Assim, os erros de execução foram classificados de acordo com:

- ***EE-1***: O resultado da execução foi incorreto pois a IA paralelizou uma parte incorreta do código.

Tabela 1. Resultados das execuções

Aplicação	GPT-3.5-turbo-16k		Code-Bison-001	
	Compilação	Execução	Compilação	Execução
CCN	CE-1	X	✓	EE-3
CCR	CE-3	X	CE-2	X
DFT	✓	EE-1	✓	EE-2
FFT	CE-1	X	CE-2	X
GS	CE-2	X	✓	EE-1
LU	✓	EE-1	✓	EE-1
MC	✓	✓	✓	EE-3
MM	✓	✓	✓	✓
OE	✓	✓	✓	✓
PO	CE-2	X	✓	EE-1

- ***EE-2***: O resultado foi incorreto pois faltou uma *cláusula* de paralelismo no *pragma*.
- ***EE-3***: O código não foi paralelizado. Isto é, a IA não foi capaz de gerar um código paralelo e a execução se deu de maneira sequencial.

O ambiente de execução utilizado consiste de um processador 2 x Intel Xeon E5-2650 v3 Haswell (Q3'14), 2.3 GHz com sistema operacional Linux 12 e 128 GB de memória RAM. Para compilação das aplicações, foi utilizado o compilador GNU Compiler Collection versão 12.2.0. Cada aplicação gerada foi executada com o número de *threads* igual ao número de núcleos disponíveis na arquitetura (isto é, 40 *threads*). Para comparar o resultado obtido pelas IAs, o código gerado foi comparado com uma solução *gold* que contém a implementação ideal fornecida pela suíte de aplicações utilizada. O desempenho das versões geradas pelas IAs foi comparado com a execução da aplicação com a solução *gold*. Para cada conjunto de testes (aplicação, IA e número de *threads*), foram realizadas 10 execuções, e os resultados consideram a média aritmética.

4. Resultados

Nós começamos apresentando e discutindo os resultados da etapa de geração do código paralelo através de cada IA, conforme mostrado na Tabela 1. Conforme observado, a IA *Code-Bison-001* foi capaz de gerar mais códigos que foram capazes de compilar (identificados pelo *checkmark*): 8 aplicações comparado a 5 aplicações quando usado o *GPT-3.5-turbo-16k*. Todos os erros de compilação do Code-Bison-001 foram causados por exceder o número máximo de palavras que o modelo suporta no *output*, por outro lado o modelo da família GPT obteve erros variados para compilação.

Os modelos produziram a execução correta da paralelização apenas para aplicações mais simples, como *MM*, *MC* e *OE*. Por fim, o modelo da família GPT paralelizou uma aplicação a mais (*MC*) enquanto o modelo da família Palm omitiu parte do código. Na Tabela 2, são mostrados os tempos de execução dos códigos analisados, sendo importante destacar que caso o modelo não tenha conseguido paralelizar ou tenha errado os resultados foram atribuídos os tempos de execução da aplicação sequencial. A partir da análise das aplicações paralelizadas pelos modelos identificamos um tempo de execução significativamente diferente do tempo obtido pela solução *gold* na aplicação OE. Nessa aplicação a solução *gold* foi executada 7.06% mais rápida que a solução produzida pelo Code-Bison-001 e 0.3% mais rápida que a solução produzida pelo GPT-3.5-turbo-16k. Para a aplicação MM não houve diferença significativa no tempo de execução nas três soluções paralelas, quando comparadas entre si. Na aplicação MC, o modelo da família

Tabela 2. Tempos das execuções

Aplicação	GPT-3.5.16K	Code-Bison-001	Sequencial	Paralelo *Gold*
CCN	139.10	139.10	139.10	5.4
CCR	94.41	94.41	94.41	4.55
DFT	41.15	41.15	41.15	1.97
FFT	152.45	152.45	152.45	66.04
GS	70.78	70.78	70.78	5.2
LU	169.92	169.92	169.92	7.21
MC	32.79	68.87	68.87	32.79
MM	1.11	1.11	22.90	1.11
OE	13.06	14.01	100.72	13.02
PO	82.56	82.56	82.56	5.88

GPT alcançou o mesmo tempo de execução obtido pela solução *gold* enquanto o modelo da família Palm não conseguiu paralelizar essa aplicação.

5. Conclusões e trabalhos futuros

Neste trabalho, comparamos dois modelos de processamento de linguagem natural no escopo de programação paralela, com o objetivo de analisar o desempenho no processo de paralelização de códigos. Constatamos que os modelos foram capazes de paralelizar corretamente apenas as aplicações mais simples, sendo importante destacar que o modelo GPT-3.5-turbo-16k conseguiu paralelizar mais aplicações que o Code-Bison-001. Nesse sentido, trabalhos futuros incluem analisar um número maior de modelos considerando outras variáveis além das que foram utilizadas no presente trabalho.

Referências

Barham, P., Chowdhery, A., Dean, J., Ghemawat, S., Hand, S., Hurt, D., Isard, M., Lim, H., Pang, R., Roy, S., et al. (2022). Pathways: Asynchronous distributed dataflow for ml. *Proceedings of Machine Learning and Systems*, 4:430–449.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2023). Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Godoy, W., Valero-Lara, P., Teranishi, K., Balaprakash, P., and Vetter, J. (2023). Evaluation of openai codex for hpc parallel programming models kernel generation. In *Proceedings of the 52nd International Conference on Parallel Processing Workshops*, ICPP-W 2023. ACM.

Lorenzon, A. F. and Beck Filho, A. C. S. (2019). *Parallel computing hits the power wall: principles, challenges, and a survey of solutions*. Springer Nature.

Navaux, P. O. A., Lorenzon, A. F., and da Silva Serpa, M. (2023). Challenges in high-performance computing. *Journal of the Brazilian Computer Society*, 29(1):51–62.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Valero-Lara, P., Huante, A., Lail, M. A., Godoy, W. F., Teranishi, K., Balaprakash, P., and Vetter, J. S. (2023). Comparing llama-2 and gpt-3 llms for hpc kernels generation.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

Wang, B. and Komatsuzaki, A. Gpt-j-6b: A 6 billion parameter autoregressive language model. https://github.com/kingoflolz/mesh-transformer-jax. Accessado: 2024-02-16.