Otimização do Tempo de Execução de um Problema de Colisor de Particulas Usando Yatuner

[1]**Thiago J. S. Rodrigues,** [1]**Marcos T. A. Gonzalez**

[1]Universidade Federal do Pará
Faculdade de Engenharia de Computação, Tucuruí - Pará

tj.vinci97@gmail.com, amaris@ufpa.br

Abstract. *The present work is contained in the multidisciplinary area of Computational Physics, this field of study aims to computationally estimate effects of the physical world, based on computational calculations. The study presented here aims to analyze the gain in execution time of a software code, which mathematically simulates the dispersion of the energy of an impact using grains as dissipators of the energy produced by the collision of spheres, after being subjected to a Autotuner script. Time improvement was done through a tuner called Yatuner, developed by researcher Junyi Mei. The tuner used has a module for selecting the best parameters and optimizers of the compiler used, through statistical mathematical techniques, the tuning tool used delivers results of shorter times in all scenarios experienced for the simulator, when compared with the results of standard configurations of optimizing the GCC compiler, Yatuner achieved results approximately 2.5 times faster, in the best case achieved, running the simulator with the selected parameters was up to 2.8 times faster.*
Keywords: Yatuner, GCC, Computational physics, software.

Resumo. *O presente trabalho está contido na área multidisciplinar da Física computacional, esse campo de estudo visa estimar computacionalmente efeitos do mundo físico, com base em cálculos computacionais. O estudo aqui apresentado destina-se a realizar uma análise do ganho de tempo de execução do código de um software, que simula matematicamente a dispersão da energia de um impacto utilizando grãos como dissipadores da energia produzida pelo choque de esferas, após ser submetido a um script de Autotuner. O Melhoramento do tempo foi feito através de um tuner chamado Yatuner, desenvolvido pelo pesquisador Junyi Mei. O tuner utilizado possui um módulo de seleção dos melhores parâmetros e otimizadores do compilador empregado, através de técnicas matemáticas estatísticas a ferramenta de tunning utilizada entrega resultados de tempos menores em todos os cenários experimentados para o simulador, quando comparado com os resultados das configurações padrões de otimização do compilador GCC, o Yatuner conseguiu resultados aproximadamente 2,5 vezes mais rápidos, no melhor caso alcançado a execução do simulador com os parâmetros selecionados foi de até 2.8 vezes mais rápido.*
Palavras-chave: Yatuner, GCC, Física computacional, software.

1. Introdução

A área da computação de alto desempenho requer necessariamente grande capacidade de processamento de dados pela demanda quase sem fim de poder computacional nas áreas de

ciência e engenharia, apesar das pesquisas para desenvolver hardware específicos e mais avançados, esses estudos devem ser acompanhados de novas tecnologias de software, nesse espaço surge um paradigma chamado autotuning (autoajuste), que são Frameworks que utilizam algoritmos para potencializar o processo de compilação e execução de programas de linguagem compiladas. O simulador de colisões foi escrito em linguagem C, usando principalmente a GNU scientific library para resolver uma série de equações diferenciais que representam as características da interação no mundo físico. O presente trabalho passou por diversos estágios até sua conclusão, de início foi proposto que a aceleração da execução do simulador ocorreria por meio da paralelização do programa, principalmente paralelizando o método Ruggen-Kutta de quarta ordem, usado para resolver as EDOs do simulador.

1.1. Justificativa

No contexto de computação de alto desempenho, eficiência energética e aproveitamento dos recursos disponíveis são conceitos fundamentais a serem sempre levados em conta, a abordagem feita neste artigo visa aproveitar o melhor desempenho do recursos disponíveis para pesquisa e acelerar a execução de softwares que demandam muito tempo, pois nem sempre os pesquisadores possuem recursos para lidar com problemas complexos de computação na academia, indústria ou pesquisa pessoal e afinal eficiência no tempo significa também eficiência em atividades humanas. Autotuners são utilizados até mesmo com o uso de GPUs, pois os compiladores utilizam parâmetros específicos para cada tipo de otimização. Os autotuners são capazes de pesquisar espaços maiores do que seria possível manualmente, o GCC por exemplo possui um espaço de busca maior que 2^{200} combinações de parâmetros, autotuning pode ser usado para compensar o balanço entre desempenho e precisão, ou outros critérios como como consumo de energia e uso de memória. Este trabalho experienciou o desafio de usar um autotuner, o primeiro desafio foi usar a configuração correta, pois o Yatuner é um tuner de multipropósito, isso significa que ele pode ajustar o programa alvo em diversos aspectos como tempo e tamanho do binário compilado, o usuário e o criador de autotuner devem encontrar maneiras de representar as complexas estruturas e restrições de dados específicas do domínio para o qual se destina o script de tuner.

1.2. Objetivos

1.2.1. Objetivo Geral

O objetivo deste trabalho é melhorar o tempo de execução de um simulador de colisões de esferas desenvolvido na Linguagem C, utilizando um framework de ajuste automático de propósito geral, através da escolha de parâmetros de compilação e otimizadores selecionados pelo algoritmo do autotuner usado.

1.2.2. Objetivos Específicos

- Análise e caracterização do software de simulação de sistemas físicos apresentado no artigo [Machado and Sen 2018].
- Selecionar ou desenvolver um processo de tuning automático voltado à escolha de parâmetros de compilação.

- Construir um conjunto de scripts com o Yatuner para obter melhores tempos de execução do programa alvo.
- Análise e interpretação quantitativa e qualitativa dos tempos obtidos pela execução dos scripts de tuning.

1.3. Estrutura do artigo

O documento está dividido em 6 seções principais: Na seção 1 se encontram a justificativa e a introdução ao trabalho desenvolvido, sua origem e uma explicação sobre o contexto na área de física computacional, na seção seguinte, seção 2, estão presentes conceitos teóricos de como o Yatuner funciona, juntamente com uma descrição do artigo que descreve a pesquisa sobre o simulador de interação. Na Seção 3 são apresentados os trabalhos relacionados a esse artigo, correlatos ao tema da pesquisa desenvolvida. Na seção 4, a seção sobre metodologia, estão contidas as informações sobre o ambiente e as configurações nas quais as experiências aconteceram e as quantidades aplicadas nos scripts para execução das tarefas. Na seção 5 estão apresentados os resultados alcançados, para melhor efeito de visualização esta seção contém algumas figuras e tabelas ilustrando os resultados numéricos, a seção 6 contém as ideias de trabalhos futuros, provavelmente estes trabalhos serão desenvolvidos em uma oportunidade posterior pelos autores deste artigo.

2. Conceitos Teóricos

2.1. Sobre Autotuners

A tecnologia de tuning automática de software foi desenvolvida em forte ligação com a computação de alto desempenho. No momento da publicação deste livro [Naono et al. 2010], os supercomputadores mais rápidos do mundo podem alcançar o desempenho de até um exaflop (10^{18}) – o equivalente a um quintilhão de operações de ponto flutuante por segundo, permitindo explorar questões científicas anteriormente inacessíveis em astronomia, dinâmica dos fluídos, física de partículas, biológia e os demais campos da ciência. O crescimento do desempenho dos supercomputadores continuará, impulsionado pela procura ilimitada de maior poder de computação na ciência e engenharia computacional, e apoiado pelo progresso incessante da tecnologia de silício, supercondutores e pelas invenções da arquitetura de hardware de alto desempenho. A evolução da tecnologia de hardware deve ser acompanhada por tecnologias de software inovadoras que disponibilizem a capacidade computacional do hardware da forma mais otimizada possível. O ajuste automático de software é considerado o paradigma mais promissor para atender a essa demanda da tecnologia de software [Naono et al. 2010].

2.1.1. O Framework Yatuner

O Yatuner é um framework desenvolvido por Junyi Mei e Guo Dawei, alunos da Universidade de Nankai, para a competição para OS competition 2022 [Mei Junyi 2022]. Como descrito pelo próprio autor, este é mais um tuner automático para compiladores, por consequência, então, este é um autotuner, mas o que é um autotuner? Autotuners são otimizadores de compiladores para linguagens compiladas (por exemplo GCC/G++ para C/C++) de acordo com o cenário desejado, estes softwares surgiram como uma alternativa para fazer ajustes com os parâmetros e otimizadores mais adequados de acordo com a

análise do tuner, a métrica sinalizada indica o tipo de otimização a ser feita, no caso do tempo a métrica indicada é *duration_time*. Para o Yatuner, pode ser de tempo de execução, redução do tamanho do arquivo, extração de estatísticas do harware e etc. Normalmente, um compilador de código aberto como o GCC possui algumas opções de otimização padrão (por exemplo: -O1, -O2, -O3 ou -Os), que são genéricas [Stallman et al. 2003], no caso de um tuner a melhoria se dá por meio da seleção de parâmetros específicos que otimizam a compilação de acordo com a métrica fornecida, através de algoritmos que o Yatuner utiliza, como otimização Bayesiana e *Linear Upper Confidence Bound*. O simulador de interação física é um sistema dinâmico e portanto não determinístico, para esses sistemas o Yatuner utiliza o algoritmo LinUCB, caso o sistemas fosse determinístico o sistema seria otimizado pelo método Bayesiano.

2.1.2. LinUCB

LinUCB, significa *Linear Upper Confidence bound*, apelidado de "Otimismo em face da incerteza" (OFU), desenvolvido em [Li et al. 2010], é um algoritmo de reforço de aprendizado, a ideia principal do algoritmo é calcular a recompensa esperada de cada branch, onde um ramo é escolhido aleatoriamente e o algoritmo é aplicado, baseado na média observada o ramo com limite estatístico de confiança superior é escolhido. Quando o algoritmo é aplicado e o valor esperado se aproxima da média, a área de confiança diminui. A área é escolhida a partir da combinação linear das recompensas dos braços anteriores, nesse caso temos que pressupor que os braços são distintos, e não compartilham características, são chamados "disjuntos". Para fazer isso, o LinUCB decompõe o vetor de recursos da rodada atual em uma combinação linear de vetores de recursos vistos nas rodadas anteriores e usa os coeficientes e recompensas calculados nas rodadas anteriores para calcular a recompensa esperada na rodada atual.

2.2. Simulador do Problema de Colisor de Particulas

O estudo dos pesquisadores Luís Paulo S. Machado e Surajit Sen, investiga numericamente a propagação de ondas em sistemas granulares cônicos bidimensionais sujeitos a um impulso, de forma a controlar a dispersão de energia em um cristal granular quadrado bidimensional com esferas cônicas de variados tamanhos, quantidades e materiais. O estudo do decaimento da amplitude de onda, em sistemas granulares pode ser usado para realizar projetos em uma variedade de assuntos como abalos sísmicos, coletes à prova de balas, superfícies para controle de explosão, as possibilidades são enormes [Machado and Sen 2018].

O arranjo: os arranjos bidimensionais são compostos de dois tipos de grãos esféricos, os grãos principais e os grãos intersticiais (Veja figura 1).

Os grãos principais formam uma matriz com os raios e massas diminuindo exponencialmente em cada camada, o sistema é assim configurado para ajudar no *Momentum* e na força de decaimento, os grãos intersticiais são posicionados entre os espaços dentro da matriz, a bola de ataque e a primeira camada de grãos são as maiores em tamanho e massa. Todos os grãos em cada camada são idênticos. os grãos principais são responsáveis pela propagação da perturbação.

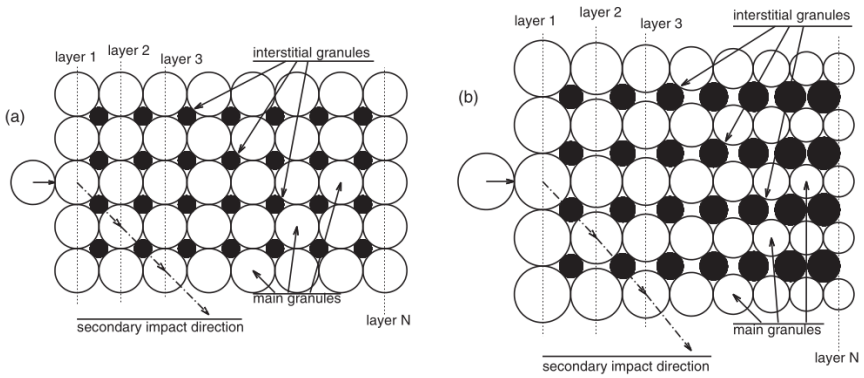

Figura 1. Arranjo de grãos

2.2.1. Sobre o código

O simulador foi escrito pelo professor Machado usando a linguagem de programação C [Machado and Sen 2018], o autor contou com funções da biblioteca científica GNU (GSL) [Gough 2009], essa biblioteca contém ampla gama de ferramentas matemáticas para operações científicas, são mais de 1000 funções, como geradores de números aleatórios, equações diferenciais e constantes físicas. No simulador a GSL é usada principalmente para cálculo de EDOs que dão comportamento de interação ao simulador, o método Runge-Kutta da função foi usado para integrar a equação de movimento dos grãos em contato dado pela seguinte equação

$$m_i \frac{d\mathbf{V}_i}{dt} = -\sum_j \alpha_{ij} (\delta_{ij}^+)^{3/2} \hat{n}_{ij} \tag{1}$$

O programa necessita de parâmetros essenciais para ser executado, são eles em ordem: Tempo total, grãos na vertical, grãos na horizontal e parâmetro de afilamento.

3. Trabalhos Correlatos

O artigo original do Yatuner [Mei Junyi 2022] tem o título traduzido de ”Yatuner: Mais um autotuner”, esse nome dá a leve noção de que este é um entre outros tantos tuners automáticos disponíveis para uso, de fato, quando analisamos a literatura disponível percebemos que o tuner apresentado faz parte de uma variedade de Frameworks disponíveis para os mais diferentes propósitos e com diferentes estruturas.

O mais notável dos tuners é o OpenTuner, autodeclarado o primeiro autotuner de multipropósito a descrever espaços de pesquisa completos, ele produz códigos de domínio específico que suportam a completa customização das configurações para o programa a ser ajustado, o OpenTuner usa diversas técnicas de busca em tempo de execução e ao mesmo tempo, testando cada candidato de configuração e avaliando-os, tais técnicas são capazes de compartilhar os resultados mais promissores em um banco de dados comum para ajudar umas as outras a encontrar um bom resultado, os algoritmos adicionam resultados de outras técnicas como novos membros da sua população, para isso é utilizado uma solução ótima para o problema do *multi-armed bandit*, tal qual no Yatuner. É o mais sofisticado programa

de autotuning para resolver problemas complexos de uma forma que pode ser facilmente adotado por outros projetos. Durante os 16 testes de benchmarks, dos 7 diferentes projetos testados houveram resultados de aceleração que foram de 1.15x (para transformadas de fourier) até maiores que 2.8x (para multiplicação de matrizes), tais resultados foram obtidos (relativamente) com pouco esforço do programador. [Ansel et al. 2014]

Em [Bruel et al. 2017] os pesquisadores utilizam o OpenTuner como autotune para procurar por parâmetros de compilação para o compilador CUDA, o autotune implementado usa a técnica *multi-armed bandit with sliding window, area under the curve credit assignment* para localizar a solução ótima, conhecido simplesmente como AUC bandit. Os resultados mostraram mais de 2x de aceleração para Eliminação Gaussiana (GAU) e quase 2x de aceleração para Heart Wall (HWL) e aceleração de mais de 4x para um algoritmo de multiplicação de matrizes (MMS). Nesse caso a performance alcançada pelo autotuning, se mostrou melhor que as opções de compilação de alto nível para as GPUs NVIDIA de microarquiteturas Kepler e Maxwell utilizadas.

4. Metodologia

O ambiente de execução dos testes é uma máquina virtual Ubuntu Desktop rodando no VirtualBox 7.0.12, a máquina host é o notebook pessoal do autor deste artigo. Apresento na Tabela 1 as principais configurações de software e hardware da máquina usada para os experimentos.

Recurso	Detalhes
SO	Ubuntu 20.04.6 LTS x86_64
Kernel	5.15.0-107-generic
CPU	AMD Ryzen 5 5500U x4
Memory	1963MiB — 1,8 GiB Swap

Tabela 1. Tabela exibindo as configurações da máquina de teste

Atendendo ao requisito de compilação do simulador de partículas, o GCC (v.10) está sendo usado como compilador base da qual estão sendo extraídos e usados otimizadores e parâmetros de compilação para execução do tuner, o programa combina e analisa os 230 otimizadores e 225 parâmetros de compilação contidos no GCC menos os otimizadores do –O3. Após a retirada dos otimizadores do -O3 restam apenas 79 otimizadores para testes do tuner, com exceção do otimizador *-fipa-pta*, que precisou ser removido, pois necessitava de permissão especial do SO para ser executado. Para o desenvolvimento do arquivo de otimização de tempo, foi utilizada a versão mais recente do Python (v3.9) até a data de escrita desse artigo, devido à conflitos de versões mais antigas das bibliotecas com a atual versão do Python foi necessário que adaptações fossem feitas na estrutura original do Yatuner. Para a realização dos testes mantiveram-se as condições físicas iniciais padrões para o simulador, tais quais: constante elástica, quantidade de grãos, aceleração gravitacional, energia cinética, energia e arranjo inicial.

O principal fator principal de aprimoramento de resultados para o tuner é o tempo, de tal forma que é preciso ajustar o número de amostras (tempo) e combinações (epochs) para que as rodadas de testes durem o máximo de tempo, a fim de obtermos interpretações mais assertivas sobre os resultados. Na tabela 2 é possível ver a relação entre as quantidades de amostras e o tempo de execução do tuner.

tuner.run (num_samples)	Tempo de execução
100	5h 1m 34s
200	9h 29m 24s
400	19h 42m 48s

Tabela 2. Tabela exibindo o número de amostras e epochs

Com a intenção de comparar as medidas de tempo dos otimizadores com os resultados do Yatuner, realizou-se a média dos tempos de execução do simulador sem o Yatuner, para melhor visualização e efeitos de comparação.

Quando se trabalha com médias de resultados é importante calcular a variável estatística chamada de *desvio padrão*, que indica o quanto a média se desvia do conjunto de dados, neste trabalho o desvio padrão apresentado para cada média foi obtido usando a função *stdev* da biblioteca *statistics* do Python, pois essa função usa a fórmula padrão, onde cada valor no conjunto de dados, subtrai a média do conjunto e eleva esse resultado ao quadrado, depois calcula a média desses quadrados e tira a raiz quadrada desse resultado.

5. Resultados

Nesta seção serão apresentados os resultados e discussões sobre a performance do tuner e tempo de execução. Para fins de comparação, a tabela 3 exibe os tempos obtidos com execuções manuais sem o uso do Yatuner, estes números são as médias de 10 rodadas para cada otimizador.

Método	Resultado (μs)	Desvio padrão (μs)
O0	3708744.30	13481.93
O1	2707414.23	14034.42
O2	2584479.61	11342.92
O3	2576521.48	20739.53
Ofast	2478604.32	24038.19
Os	2603492.15	14849.22
No optimizer	4881966.84	647525.62

Tabela 3. Média de tempos manuais

5.1. Rodadas de testes com os scripts do Yatuner

Com o objetivo de auferir as menores medidas de tempos para a execução do programa, foram coletadas diferentes medidas de tempo para diferentes quantidades de amostras de tempo para cada otimizador. Nas tabelas abaixo o *score* refere-se a uma métrica de desempenho relativa, onde o fator principal é o tempo, o cálculo do *score* compara a média de cada conjunto de amostras com a menor média entre todos os conjuntos de amostras considerados, portanto o maior valor (100.00) é o menor tempo, o segundo maior valor é o segundo menor tempo, e assim sucessivamente. *Delta* é uma variação percentual entre as pontuações em comparação com a referência da categoria “O2”, O cálculo do *delta* é feito para avaliar a diferença percentual relativa entre as pontuações. *Delta* positivo indica que a pontuação da categoria em questão é maior do que a pontuação da categoria de referência ”O2”. *Delta* negativo indica que a pontuação da categoria em questão é menor do que a

Método	Resultado (μ s)	Score	Delta
O0	3751056.27	39.01	
O1	2675769.66	54.69	
O2	2593443.81	56.43	
O3	2570144.77	56.94	
Ofast	2466395.87	59.34	
Os	2603054.80	56.22	
Optimizers	2414814.76	60.60	7.40%
Parameters	1463436.96	100.00	77.22%

Tabela 4. Tabela de resultados para 100 amostras

O0	O1	O2	O3	Ofast	Os	Opt	Param
x	1.3x	1.43x	1.46x	1.49x	1.42x	1.53x	2.53x

Tabela 5. Tabela de comparação para 100 amostras

pontuação da categoria de referência "O2". os deltas ajudam a entender como as diferentes categorias se comparam em termos de desempenho relativo. A comparação é sempre feita em relação à categoria "O2", que serve como referência.

Primeiro caso: 100 amostras.

A Tabela 5 é usada para comparação e tem o objetivo de mostrar de maneira mais intuitiva e direta as diferenças de tempo entre os métodos. Por padrão o GCC toma o método O0 para compilar o programa, portanto este é o método que será tomado como base para comparar com os tempos dos outros métodos. Observando visualmente o gráfico de comparação de tempos (Figura 2) 01, 02, 03 e Os obtiveram médias de tempos muito parelhos, *Ofast* e *Optimizers* tiveram tempos muito próximos um do outro, com leve vantagem para *Optimizers*, *Parameters* obteve um tempo 2.53 vezes menor que O0 e 1,04x menor comparada com o método *Ofast*, o mais rápido dos otimizadores do GCC.

Segundo caso: 200 amostras.

Para o caso em que o *num_samples* foi de 200 amostras *Parameters* foi expressivamente melhor do que os outros métodos (Tabela 7), foi cerca de 2.8 vezes mais rápido do que O0. O1, O2, O3 e Os tiveram tempos relativamente semelhantes, dessa vez a

Método	Resultado (μs)	Score	Delta
O0	3751198.04	35.81	
O1	2674228.99	50.24	
O2	2592947.50	51.81	
O3	2571315.07	52.25	
Ofast	2465409.13	54.49	
Os	2596073.76	51.75	
Optimizers	2419829.85	55.52	7.15%
Parameters	1343438.30	100.00	93.01%

Tabela 6. Tabela de resultados para 200 amostras

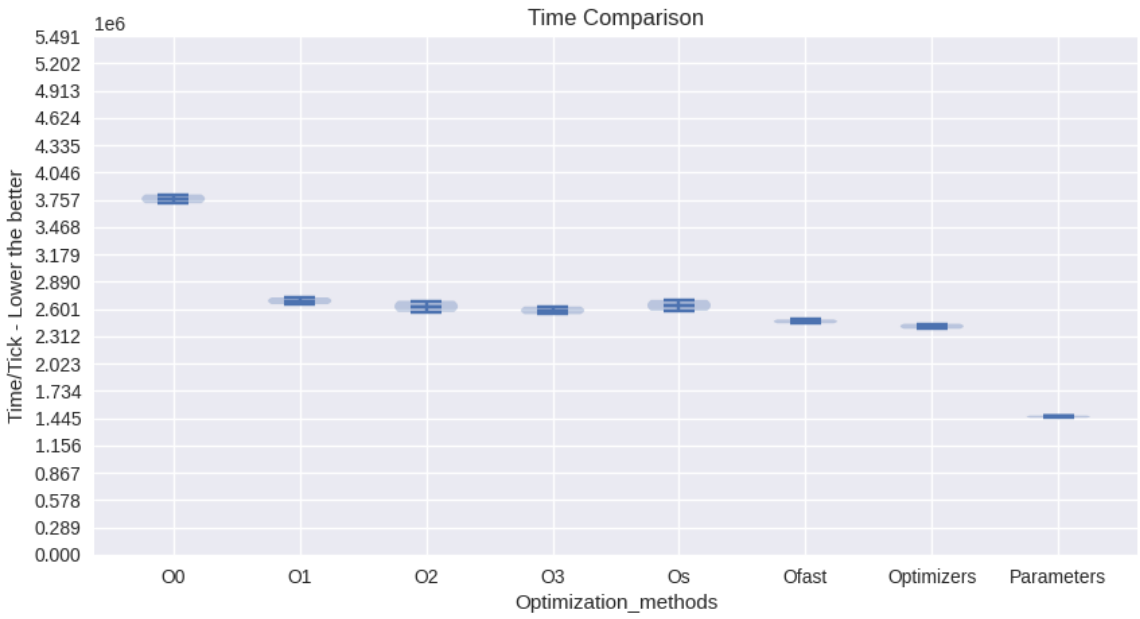

Figura 2. Gráfico de intervalos de tempo para 100 amostras.

O0	O1	O2	O3	Ofast	Os	Opt	Param
x	1.40x	1.44x	1.45x	1.52x	1.44x	1.55x	2.79x

Tabela 7. Tabela de comparação para 200 amostras

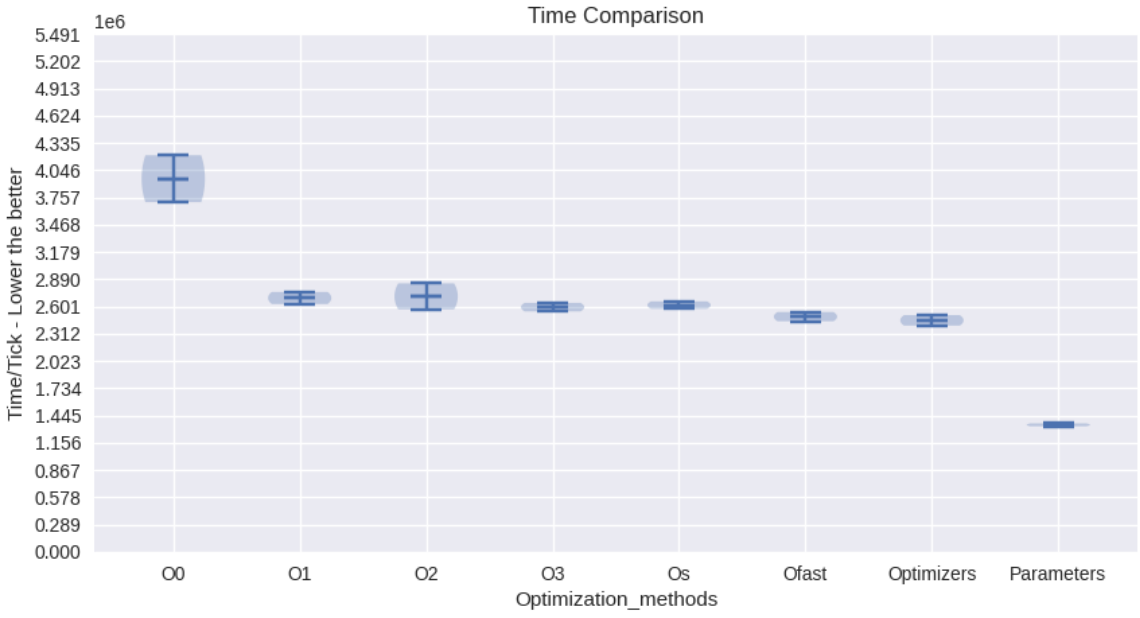

Figura 3. Gráfico de intervalos de tempo para 200 amostras.

Método	Resultado (μ s)	Score	Delta
O0	3749127.52	35.92	
O1	2673986.54	50.36	
O2	2591006.95	51.98	
O3	2566807.14	52.47	
Ofast	2464495.82	54.64	
Os	2597153.11	51.85	
Optimizers	2396301.87	56.20	8.13%
Parameters	1346690.05	100.00	92.40%

Tabela 8. Tabela de resultados para 400 amostras

O0	O1	O2	O3	Ofast	Os	Opt	Param
x	1.40x	1.44x	1.46x	1.52x	1.44x	1.56x	2.78x

Tabela 9. Tabela de comparação para 400 amostras

diferença de *Ofast* e *Optimizers* para os métodos anteriores foi menor do que no primeiro caso (Tabela 4), nota-se também que os extremos no método de O0 e O2 (Figura 4) foram muito maior do que no caso anterior.

Terceiro caso: 400 amostras.

Parameters teve magnitude semelhante ao caso de 200 amostras (Tabela 9), um ganho de 2.8 vezes em relação ao método padrão, o restante dos métodos também seguiu o valor aproximado do caso anterior de 200 amostras (Tabela 7), nestes dois caso a diferença dos extremos para método O0 foi visivelmente maior, resultando num gráfico de aparência mais prolongada (Figura 3 e 4).

5.2. Parâmetros comuns nos experimentos

Parâmetros	Otimizadores
hot-bb-frequency-fraction	-ffast-math
inline-min-speedup	
max-completely-peeled-insns	
max-inline-insns-auto	
max-pending-list-length	
uninlined-function-time	
vect-max-version-for-alias-checks	

Tabela 10. Tabela de parâmetros e otimizador em comum

Como parte do processo de análise dos resultados decidimos separar os parâmetros comuns às 3 execuções, a tabela 10 exibe os parâmetros e o único otimizador em comum encontrado nos arquivos de saída após cada execução. Estes parâmetros controlam diferentes aspectos de como o GCC otimiza o código durante a compilação, eles influenciam decisões de como o código deve ser transformado e melhorado em termos de desempenho, tamanho e eficiência durante a compilação onde esses requisistos de desempenho são críticos. O

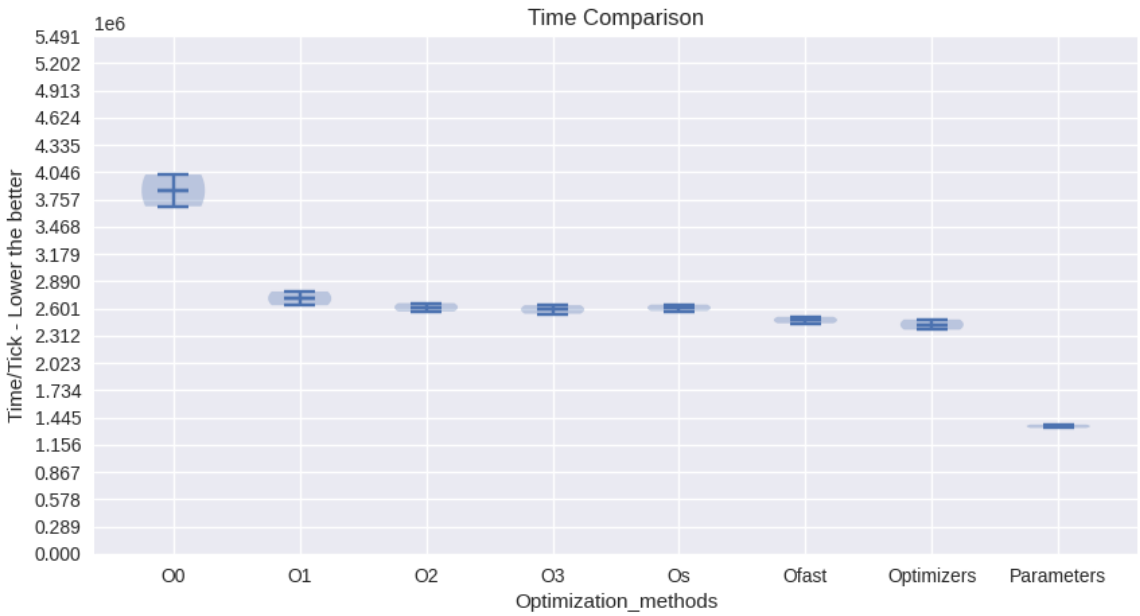

Figura 4. Gráfico de intervalos de tempo para 400 amostras.

único ítem dos arquivos de otimização em comum *-ffast-math* faz com que o compilador aplique uma série transformações matemáticas no código na intenção que ele fique mais rápido.

5.3. Gráficos do teste de hipóteses

O teste de hipóteses apresentou grande aleatoriedade em suas formas (Figura 5), como foi pontuado no artigo original do Yatuner [Mei Junyi 2022], ao compilar o programa uma vez e executá-lo várias vezes foi constatado uma grande aleatoriedade do tempo de execução. mesmo para diferentes números de amostras os tempos não formavam uma sequência da qual pôde-se extrair um padrão.

6. Conclusão

Esse trabalho comprovou a eficácia do uso de softwares de autotuning para a física computacional, obtendo melhor tempo em todos os experimentos, posteriormente os resultados apresentados neste artigo poderão ser comparados com outras abordagens para otimização de resultados, como o uso de programação paralela para obter melhores tempos de execução.

O Yatuner na função de otimizador de tempo para a execução do simulador de contato de partículas foi excelente em todos os casos, a execução com os parâmetros selecionados pelo tuner foi expressivamente menor do que qualquer um dos otimizadores fornecidos por padrão pelo GCC, aproximadamente 2.5 vezes mais rápido do que execuções sem parâmetros e otimizadores. Como esperado os otimizadores padrões do GCC tiveram tempos parecidos quando executados pelo yatuner e comparados as médias dos tempos manuais.

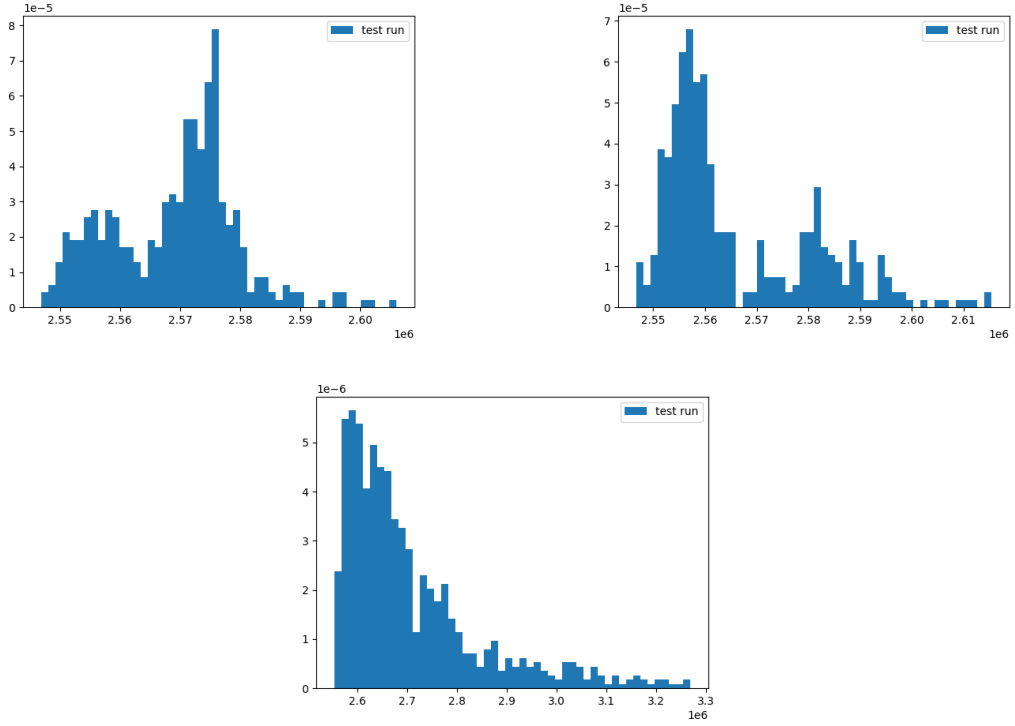

Figura 5. Testes de distribuicao

Trabalhos futuros: Podemos pensar na utilização de placas gráficas paralelizar o processamento de uma maior quantidade de amostras no teste de hipóteses, e no cálculo de limites de confiança superior do método LinUCB que está sendo usado no cálculo de probabilidades do Yatuner.
Outra abordagem posterior a ser feita será retomar a ideia inicial da pesquisa, que seria de paralelizar a execução do simulador de interações físicas, para que o processamento do software seja feito com o auxílio de placas gráficas.

Referências

[Ansel et al. 2014] Ansel, J., Kamil, S., Veeramachaneni, K., Ragan-Kelley, J., Bosboom, J., O'Reilly, U.-M., and Amarasinghe, S. (2014). Opentuner: An extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 303–316.

[Bruel et al. 2017] Bruel, P., Amarís, M., and Goldman, A. (2017). Autotuning cuda compiler parameters for heterogeneous applications using the opentuner framework. *Concurrency and Computation: Practice and Experience*, 29(22):e3973.

[Gough 2009] Gough, B. (2009). *GNU scientific library reference manual*. Network Theory Ltd.

[Li et al. 2010] Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, WWW '10. ACM.

[Machado and Sen 2018] Machado, L. P. S. and Sen, S. (2018). Controlled energy dispersion in two-dimensional decorated granular crystals. *Physical Review E*, 98(3):032907.

[Mei Junyi 2022] Mei Junyi, G. D. (2022). yatuner: yet another auto tuner for compilers. Technical report, OS competition 2022.

[Naono et al. 2010] Naono, K., Teranishi, K., Cavazos, J., and Suda, R. (2010). *Software automatic tuning: from concepts to state-of-the-art results*. Springer Science & Business Media.

[Stallman et al. 2003] Stallman, R. M. et al. (2003). Using the gnu compiler collection. *Free Software Foundation*, 4(02).