Estudo com o Benchmark NPB em sua forma serial

Mathaus C. Huber, Thiago Reis Porto

[1]Universidade Federal de Pelotas (UFPEL) – Discente do Curso Superior de Ciência da Computação
R. Gomes Carneiro, 1 - Centro – 96075-630 – Pelotas – RS – Brazil

mchuber@inf.ufpel.edu.br, trporto@inf.ufpel.edu.br

Abstract. *This article describes the practical project in programming languages developed in the subject Programming Language Concepts, given to the fifth semester of the Computer Science course, at the Federal University of Pelotas, which refers to the comparison of program performance using different options of GCC compiler optimization.*

Resumo. *Este artigo descreve o projeto prático em linguagens de programação desenvolvida na disciplina de Conceitos de Linguagem de Programação, conferida ao quinto semestre do curso de Ciência da Computação, da Universidade Federal de Pelotas, no qual se refere a compararação do desempenho dos programas utilizando diferentes opções de otimização do compilador GCC.*

1. Introdução

Neste trabalho foi realizado um estudo dos efeitos das otimizações do GCC -O2 e -O3, comparando seus efeitos na execução dos programas em relação a tempo de execução e faltas em memórias cache. Também foi utilizado na análise dados do GCC sem estas otimizações com a flag -O0(default do GCC). Foram utilizados os seguintes kernels do Benchmark NPB disponibilizado pela NASA: IS, EP, CG, FT e MG. Porém, foi utilizado uma versão modificada onde os programas possuem apenas 1 thread(serial). Esta versão foi disponibilizado pelo professor que ministra a disciplina de Conceitos de Linguagem de Programação(CLP) na UFPEL, Gerson Cavalheiro. As próximas seções descrevem a metodologia adotada no trabalho, discussões dos resultados e a conclusão.

2. Metodologia

Os kernel dos benchmarks possuem classes de problemas, que escalam em complexidade(tamanho) na ordem S, W, A, B, C. Sendo S a classe de menor complexidade e C a maior. De modo a facilitar as análises deste trabalho e devido a limitações de hardware, escolhemos trabalhar com as classes S e W. Então para cada kernel(IS, EP, CG, FT e MG) foi executado um programa nas classes S e W. Para adquirir um tempo de execução mais confiável do que é oferecida pelo NPB foi utilizado a função de C gettimeofday(), disponível pelo header <sys/time.h> para obter o tempo decorrido na execução do programa em microssegundos. Cada execução foi realizada por meio da ferramenta Valgrind utilizando sua funcionalidade cachegrind para adquirir dados quantitativos em relação a faltas na cache. Com o Valgrind foram coletados as faltas nas caches I1(cache nível 1 de instruções), D1(cache nível 1 de dados) e LL(cache l3 unificada reconhecida pelo Valgrind e tratada como last level). Para cada teste, foi realizada 30 execuções e então obtida as médias dos dados. Para agilizar este processo foi utilizado um script bash, que compila

e executa os programas com valgrind, além de salvar um arquivo texto log com as faltas nas caches e resultado do bechmark. Bastando alterar nas configurações de makefile do benchmark a otmização desejada para executar, o código abaixo mostra um exemplo do script.

```
#!/bin/bash
make mg CLASS=S
make mg CLASS=W
make is CLASS=S
make is CLASS=W
make ep CLASS=S
make ep CLASS=W
make cg CLASS=S
make cg CLASS=W
make ft CLASS=S
make ft CLASS=W

for i in $(seq 1 30);
do
        valgrind --log-fd=9 9>>mg_S_O3.log --tool=cachegrind bin/mg.S >> mg_S_O3.log
        valgrind --log-fd=9 9>>mg_W_O3.log --tool=cachegrind bin/mg.W >> mg_W_O3.log
        valgrind --log-fd=9 9>>is_S_O3.log --tool=cachegrind bin/is.S >> is_S_O3.log
        valgrind --log-fd=9 9>>is_W_O3.log --tool=cachegrind bin/is.W >> is_W_O3.log
        valgrind --log-fd=9 9>>ep_S_O3.log --tool=cachegrind bin/ep.S >> ep_S_O3.log
        valgrind --log-fd=9 9>>ep_W_O3.log --tool=cachegrind bin/ep.W >> ep_W_O3.log
        valgrind --log-fd=9 9>>cg_S_O3.log --tool=cachegrind bin/cg.S >> cg_S_O3.log
        valgrind --log-fd=9 9>>cg_W_O3.log --tool=cachegrind bin/cg.W >> cg_W_O3.log
        valgrind --log-fd=9 9>>ft_S_O3.log --tool=cachegrind bin/ft.S >> ft_S_O3.log
        valgrind --log-fd=9 9>>ft_W_O3.log --tool=cachegrind bin/ft.W >> ft_W_O3.log
done
```

Após os logs estarem disponíveis foi realizado um pequeno script em Python para processá-los, coletar os dados relevantes para o trabalho e gerar graficos para a discussão dos resultados. Todos os logs das execuçoes assim como o código em python(colab/jupyter notebook) estão disponíveis no seguinte diretório no github: `https://github.com/Thiago-Reis-Porto/Trab_2_CLP-`. Também foi necessário um pequeno estudo na documentação do benchmark e GCC, para saber um pouco sobre os kernels, classes e otimizações.

Em resumo:

- IS(Integer Sort) - Esse kernel realiza uma operação de ordenação com grandes inteiros, testa tanto a velocidade de computação de inteiros como o desempenho na comunicação. Acessos de memória aleatório.

- EP(Embarrassingly Parallel) - Fornece uma estimativa do limite superior do desempenho de ponto flutuante. Isto é o desempenho sem comunicação significativa entre processadores.

- CG(Conjugate Gradient) Um método de gradiente conjugado é usado para calcular uma aproximação para o menor autovalor de uma matriz positiva grande, esparsa e simétrica. Comunicação e acesso de memória irregular.

- MG(Multi-Grid) Kernel multigrid simplificado, realiza analises multgrid em malhas. Requer comunicação altamente estruturada de longa distância, e testa comunicação de dados tanto de longa quanto curta distância. Uso de memória intenso.

- FT(Fourier Transform) Soluciona equações diferencias parciais com Transformada Fourier discreta em 3D, comunicação all-to-all.

- O2 - Adiciona algumas flags a mais que -O1(e.g. usa schedule de instruções), o compilador tenta aumentar o desempenho sem comprometer o tempo de compilação

e tamanho do código. Não utiliza otimizações que troquem espaço por tempo.

- O3 - Mais agressiva que -O2 ativa mais flags, opta por trocar espaço por tempo. Aumentando o tamanho do código e o tempo de compilação, em alguns casos até piora o desempenho devido ao alto custo de memória.

3. Discussão dos resultados

A análise dos resultados foi realizada com base nas tabelas e gráficos. As tabelas foram geradas com os dados das médias, foi possível verificar que a distribuição dos dados da execução não se diferenciaram o suficiente, portanto pertencem a distribuições normais. Os gráficos abaixo mostram funções densidades de algumas médias.

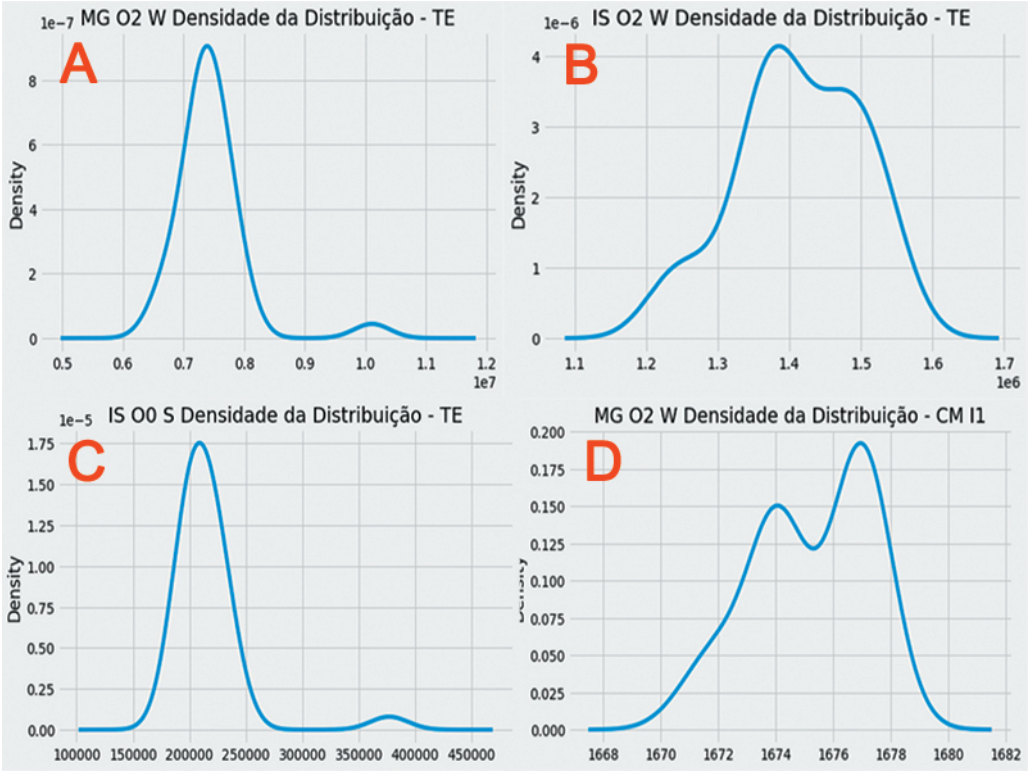

Figura 1. A - Distribuição do tempo de execução kernel MG, Classe W e -O2
B - Distribuição do tempo de execução kernel IS, Classe W e -O2
C - Distribuição do tempo de execução kernel IS, Classe S e -O0
D - Distribuição das faltas na cache l1 kernel MG, Classe W e -O2

A seguir estão apresentadas as tabelas individuais com as médias de valores das 30 execuções, de cada kernel para as classes S e W, com as flags -O2, -O3,- O0(Sem otimização).

Classe	O	Tempo(us)	I1(Faltas)	D1(Faltas)	LL(Faltas)
S	O0	216319	1385	142431	12404
W	O0	3229275	1389	11834623	1746171
S	O2	103042	1376	142372	12390
W	O2	1412177	1368	11833295	1746182
S	O3	119217	1374	188146	12396
W	O3	1694930	1367	12297290	2487741

Tabela 1. Tabela Kernel IS

Classe	O	Tempo(us)	I1(Faltas)	D1(Faltas)	LL(Faltas)
S	O0	24002562	1481	8421582	20724
W	O0	47166336	1482	16823246	20724
S	O2	21160132	1450	8421326	20698
W	O2	42725592	1450	16822734	20698
S	O3	20127388	1456	8421583	20703
W	O3	40966186	1456	16823247	20703

Tabela 2. Tabela Kernel EP

Classe	O	Tempo(us)	I1(Faltas)	D1(Faltas)	LL(Faltas)
S	O0	4342181	1542	6572919	45390
W	O0	29807258	1543	115426233	41444382
S	O2	1498816	1483	6573671	45344
W	O2	10849787	1488	114760670	41444174
S	O3	1460042	1514	6571151	44864
W	O3	10782939	1526	114773724	41442022

Tabela 3. Tabela Kernel CG

Classe	O	Tempo(us)	I1(Faltas)	D1(Faltas)	LL(Faltas)
S	O0	758226	1863	477484	22937
W	O0	37157967	1861	29558960	14448172
S	O2	216755	1673	478510	22775
W	O2	7451259	1675	29560668	14448069
S	O3	196724	1950	477024	23003
W	O3	5099024	1951	29553465	14443667

Tabela 4. Tabela Kernel MG

Classe	O	Tempo(us)	I1(Faltas)	D1(Faltas)	LL(Faltas)
S	O0	12829233	1705	6665490	2544652
W	O0	28808242	1694	43556171	7147731
S	O2	2655946	1541	7336632	2641443
W	O2	6154694	1545	44689762	7668973
S	O3	2246606	1569	7347609	2642023
W	O3	4828684	1576	44658782	7669042

Tabela 5. Tabela Kernel FT

Quanto ao tempo de execução, em geral, houve ganho de desempenho ao ativar as otimizações. Em certos casos é posivel perceber que o ganho do -O3 não é tão alto em relação a -O2. Em apenas um dos casos -O3 piorou o desempenho(embora nesta situação ainda melhor que sem otimização), isso ocorreu com o Kernel IS, esse kernel realiza ordenação de inteiros com grandes valores. É provável que as flags ativadas por -O3 tenham aumentando o tamanho do código e custo de memória neste programa a ponto de piorar o desempenho. O aumento das faltas nas caches com -O3 neste caso além de indicar a perda de desempenho, fortalece essa hipótese.

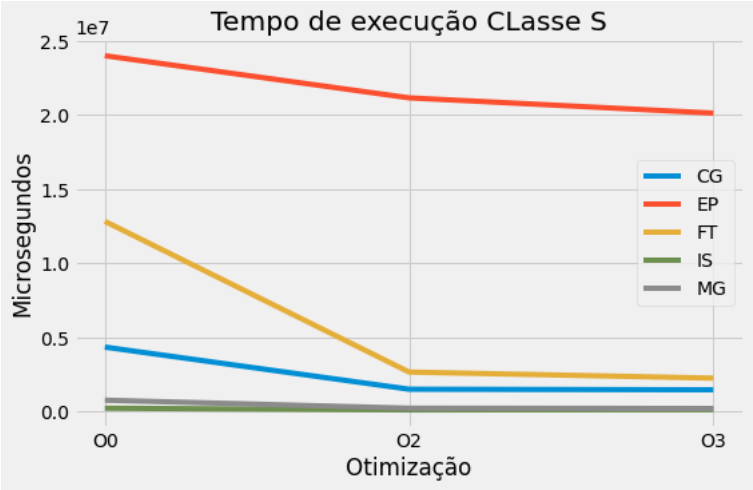

Figura 2. Gráfico tempo de execução Classe S com todos kernels, note que a piora com -O3 em IS é difícil de reconhecer neste gráfico agrupado.

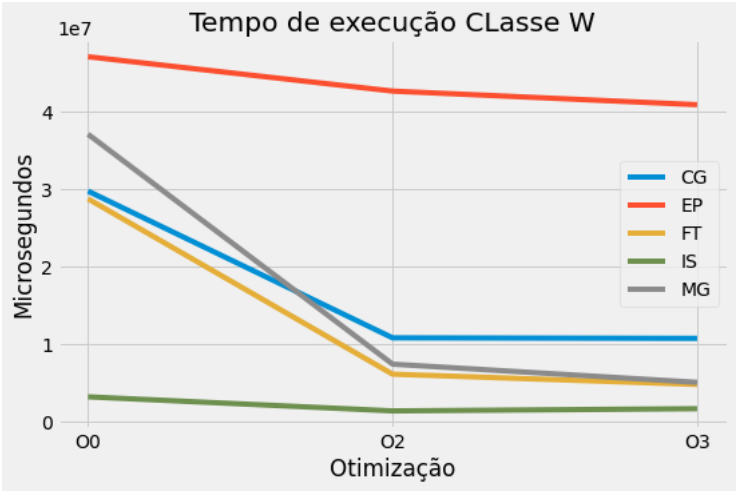

Figura 3. Gráfico tempo de execução Classe W com todos kernels, note que a piora com -O3 em IS é difícil de reconhecer neste gráfico agrupado.

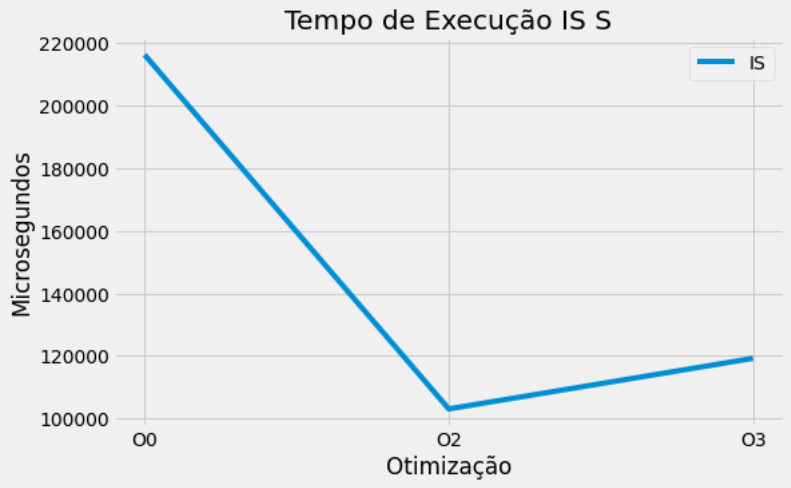

Figura 4. Gráfico tempo de execução Classe S com kernel IS

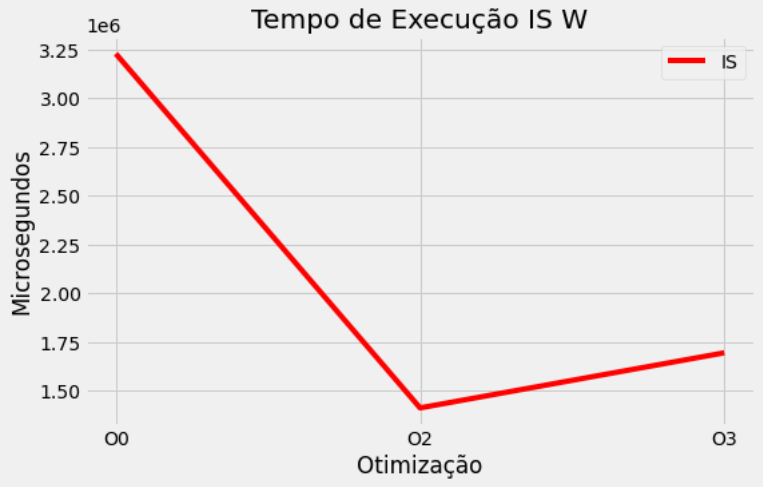

Figura 5. Gráfico tempo de execução Classe S com kernel IS

Os dados mostram que as otimizações na maioria dos caso diminuem as taxas de faltas, o que certamente justifica parte do ganho de desempenho. As faltas por instruções aparecem em valores bem menores do que as faltas por dados, o que é esperado já que são mais fácil de evitar (devido ao uso de loops e sequencialidade aproveitada na exploração da localidade espacial pelos blocos das caches). MG, CG e FT aparecem com os maiores números de faltas é esperado, pois, seus comportamentos são de acesso e comunicação irregulares, ou uso intenso de memória. EP foi o kernel mais lento, uma hipótese é que suas operações com ponto flutuante foram bastante penalizadas ao ser transformado em serial(1 thread). Também é possível identificar um aumento significativo de faltas de instruções com o Kernel MG com a otimização -O3. Esse kernel aplica métodos numéricos multgrid em malhas, é provável que as otimizações de -O3 que geram mais instruções e aumento do código tenham causado mais conflitos na cache e consequentemente esse aumento de faltas em instruções.

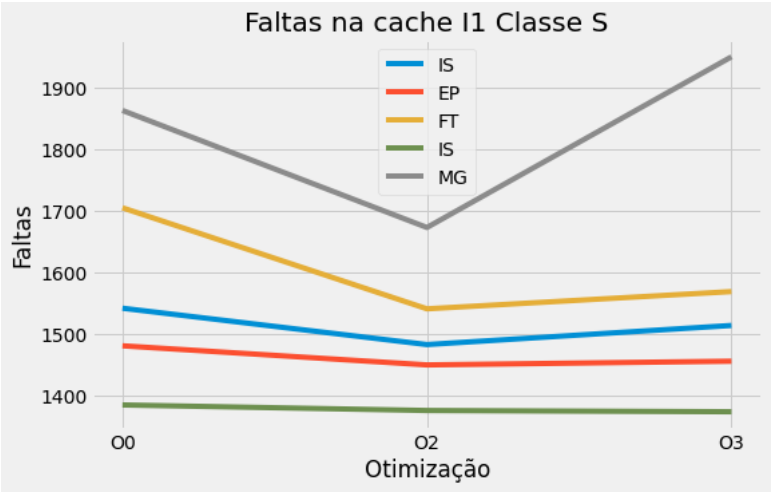

Figura 6. Gráfico faltas na cache I1 Classe S

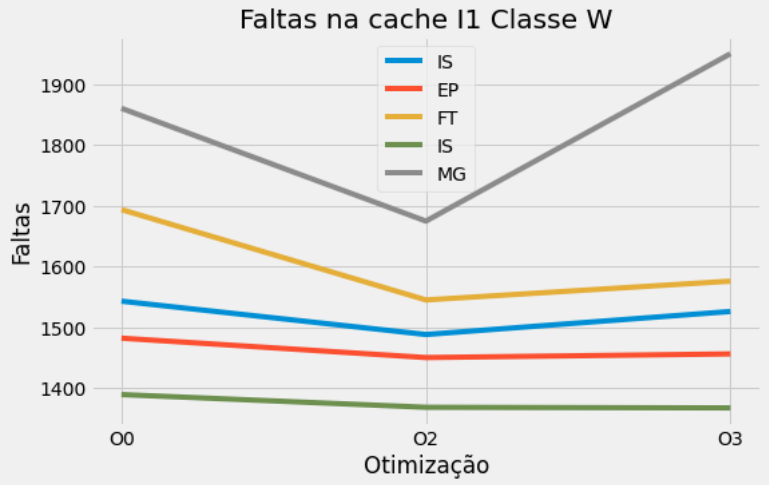

Figura 7. Gráfico faltas na cache I1 Classe W

4. Conclusão

Neste trabalho foram apresentados quantitativos coletados por execuções em uma versão serial modificado do benchmark NPB, onde foram testadas as Classes S e W os kernels IS, EP, CG, MG, FT, perante as flags de otimização do GCC -O2 e -O3. Também foi apresentada uma discussão qualitativa dos resultados, que indicam que ambas otimizações em geral melhoram o desempenho em sua otimização. Porém -O2 se apresentou como uma otimização mais estável e segura, que oferece bons resultados com baixos riscos. -O3 ativa otimizações mais agressivas, realizando trocas de espaço pelo tempo, em alguns casos isso pode ser prejudicial e afetar o desempenho que se está tentando ganhar. Entretando é importante resaltar que é necessario uma análise mais ”profunda”do que a realizada neste trabalho para se retirar melhores conclusões, i.e. realizando testes com as classes de maior tamanho(A, B, C), além de um número maior de execuções e uma maior análise nos algoritmos dos kernel e flags do GCC.

Referências

GCC - Options That Control Optimization. `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`. Acesso em: 12/11/2021.

Gentoo - GCC optimization. `https://wiki.gentoo.org/wiki/GCC_optimization`. Acesso em: 12/11/2021.

Linuxtopia - Optimization Levels. `https://www.linuxtopia.org/online_books/an_introduction_to_gcc/gccintro_49.html`. Acesso em: 12/11/2021.

Bailey, D., Harris, T., Saphir, W., Van Der Wijngaart, R., Woo, A., and Yarrow, M. (1995). The nas parallel benchmarks 2.0. Technical report, Technical Report NAS-95-020, NASA Ames Research Center.