COC472 - Computação de Alto Desempenho

Trabalho Final

Universidade Federal
do Rio de Janeiro

- Pedro Tubenchlak Boechat - DRE 119065050
  - pedroboechat@poli.ufrj.br

O código-fonte desse trabalho está disponível no meu GitHub
(https://github.com/pedroboechat/UFRJ/tree/main/COC472/Trabalho%20Final).

Rio de Janeiro - RJ
17 de Outubro de 2021 (2021.1)

Introdução

Com um sistema composto por uma CPU Intel i7-10700 (16 threads), 128GB de memória RAM DDR4 e sistema operacional Ubuntu 20.04.3 LTS, foi compilado o programa HPL, que consiste de uma implementação do HPC Linpack Benchmark. Ao longo desse trabalho, será o HPL será profilado e serão testados seus resultados para diferentes parâmetros, com o intuito de buscar o que melhor se aproxima do R_{peak} da máquina. O HPL sempre será executado por meio do comando "mpiexec -n 16 ./xhpl".

Determinação do R_{peak}

É possível calcular o R_{peak} contando o número de adições e multiplicações de pontos flutuantes (com precisão completa) que podem ser realizados durante um período de tempo, geralmente o ciclo de uma máquina[1] ou multiplicando o número de núcleos do processador pela velocidade do clock e pelo número de operações de ponto flutuante que podem ser realizadas em um segundo[2]. Como não encontrei um programa que implemente o primeiro método e não encontrei nas especificações do fabricante do meu processador o número de operações por segundo, estimei para o meu sistema $R_{peak} = 16 \times 2.9 \times 2 = 92.8$ Gflops.

Profilling do HPL

O programa HPL com N = 10000, NB = 256, P = 1 e Q = 16. Ao fim de sua execução (em 30.41 segundos), foi gerado um arquivo *gmon.out*, que foi utilizado com o profilador *Gproof*. Os resultados foram:

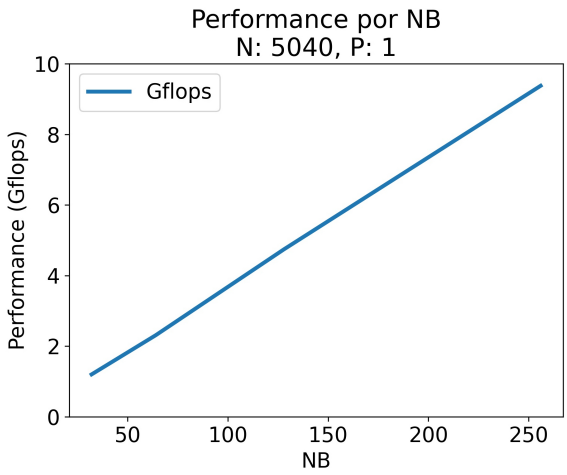
%time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name (s)
25.00	0.15	0.15				HPL_rand
15.00	0.24	0.09				HPL_lmul
13.33	0.32	0.08				HPL_bcast_1rinM
10.00	0.38	0.06				HPL_bcast
6.67	0.42	0.04				HPL_ptimer
5.00	0.45	0.03				HPL_dgemm
5.00	0.48	0.03				HPL_ladd
5.00	0.51	0.03				HPL_pdlange
5.00	0.54	0.03				HPL_pdupdateTT
5.00	0.57	0.03				HPL_ptimer_cputime
1.67	0.58	0.01				HPL_dlaswp00N
1.67	0.59	0.01				HPL_dtrsm
1.67	0.60	0.01				HPL_setran

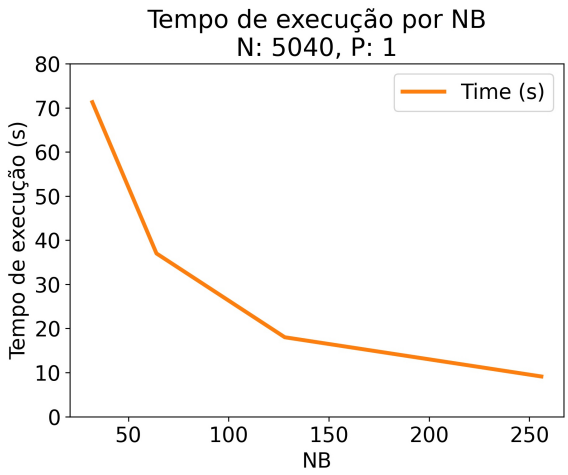
Não foi retornado o número de chamadas de cada função e não foi possível descobrir a causa desse problema, logo pela análise feita a partir dos dados incompletos podemos identificar as funções *HPL_rand* (usada para gerar números aleatórios na etapa de geração da matriz), *HPL_lmul* (usada para multiplicar dois arrays bidimensionais de números inteiros grandes na etapa de geração da matriz) e *HPL_bcast* (usada para transmitir os dados em painel) como possíveis hotspots do programa.[4]

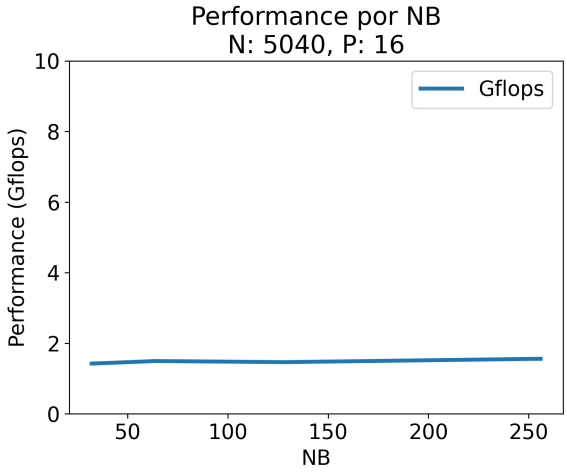
Resultados

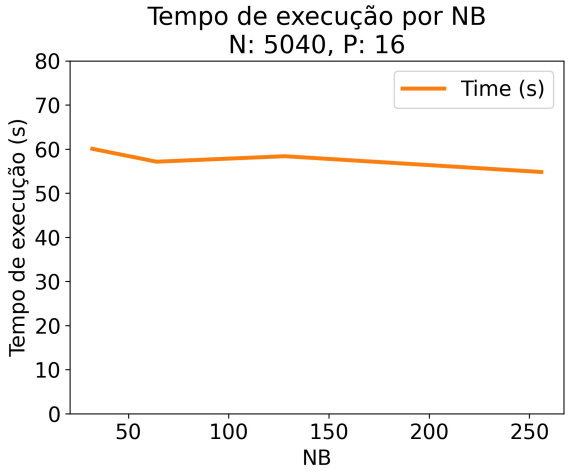
Foi criado um código Python *run.py*, que roda o HPL alternando os valores de N (5040 e 10000), NB (32, 64, 128, 256) e P/Q (tal que P×Q = 16) e salva os resultados em um arquivo CSV. O valor ideal para N no meu sistema era de $N = \sqrt{0.92 \times \frac{128 \times 1024^3}{8}} = 125719$, calculado segundo a equação $N \simeq \sqrt{\beta \times \frac{\texttt{Total Memory Size in bytes}}{\texttt{sizeof(double)}}}$[3], no entanto demorava muito para concluir e então foram utilizados os valores mencionados acima.

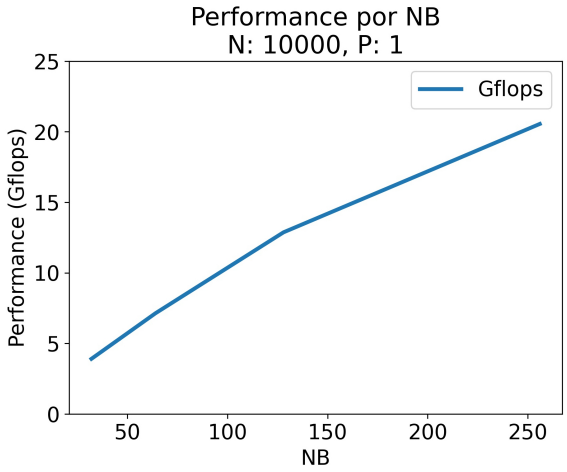
N	NB	P	Q	Time (s)	Gflops
5040	32	1	16	71.29	1.1977
5040	32	2	8	86.64	0.9855
5040	32	4	4	107.91	0.79131
5040	32	8	2	117.08	0.72933
5040	32	16	1	60.06	1.4217
5040	64	1	16	37.0	2.308
5040	64	2	8	56.63	1.5077
5040	64	4	4	72.73	1.1741
5040	64	8	2	81.92	1.0424
5040	64	16	1	57.14	1.4943
5040	128	1	16	18.0	4.7434
5040	128	2	8	34.82	2.4523
5040	128	4	4	54.99	1.5527
5040	128	8	2	60.79	1.4047
5040	128	16	1	58.38	1.4626
5040	256	1	16	9.11	9.3756
5040	256	2	8	21.74	3.9282
5040	256	4	4	39.91	2.1395
5040	256	8	2	48.92	1.7455
5040	256	16	1	54.8	1.5582
10000	32	1	16	170.93	3.9011
10000	32	2	8	211.31	3.1557
10000	32	4	4	276.93	2.4079
10000	32	8	2	271.08	2.4598
10000	32	16	1	131.78	5.0601
10000	64	1	16	93.49	7.1324
10000	64	2	8	139.21	4.7899
10000	64	4	4	192.9	3.4568
10000	64	8	2	191.59	3.4804
10000	64	16	1	125.67	5.3062
10000	128	1	16	51.8	12.873
10000	128	2	8	90.22	7.3907
10000	128	4	4	142.51	4.6792
10000	128	8	2	148.97	4.4762
10000	128	16	1	124.33	5.3631
10000	256	1	16	32.44	20.554
10000	256	2	8	60.68	10.988
10000	256	4	4	110.57	6.0309
10000	256	8	2	123.59	5.3953
10000	256	16	1	122.98	5.4223

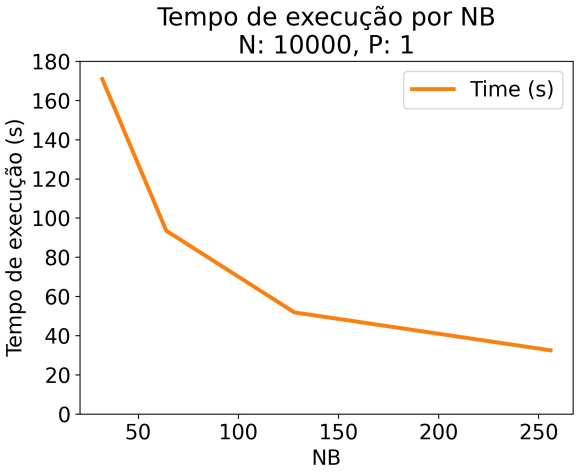

Performance por NB
N: 5040, P: 1
Gflops
Performance (Gflops)
NB

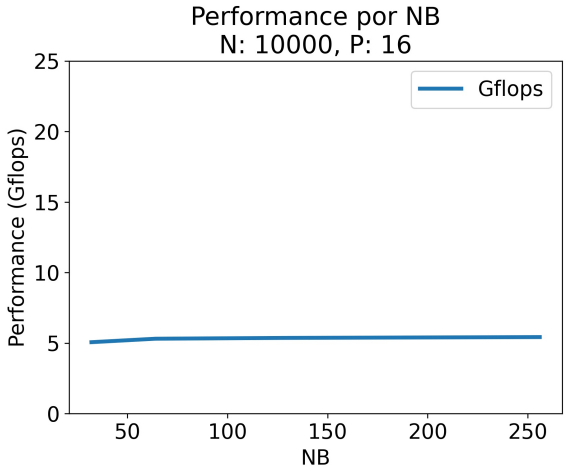

Tempo de execução por NB
N: 5040, P: 1
Time (s)
Tempo de execução (s)
NB

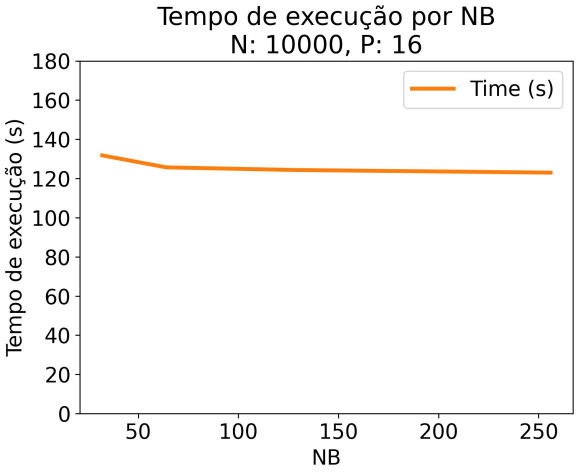

Performance por NB
N: 5040, P: 16
Gflops
Performance (Gflops)
NB


Tempo de execução por NB
N: 5040, P: 16
Time (s)
Tempo de execução (s)
NB


Performance por NB
N: 10000, P: 1
Gflops
Performance (Gflops)
NB

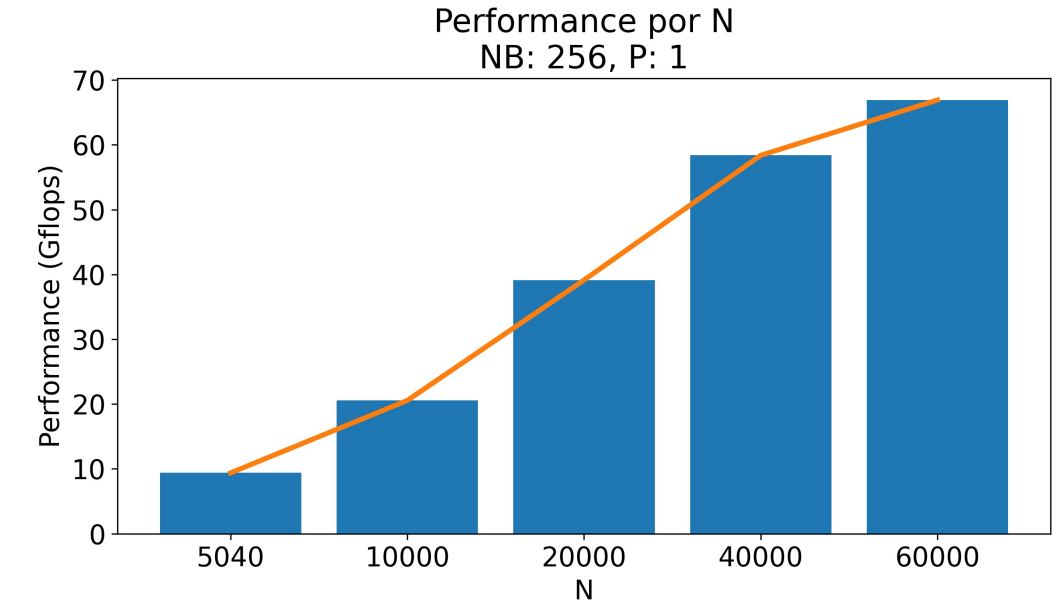

Tempo de execução por NB
N: 10000, P: 1
Time (s)
Tempo de execução (s)
NB

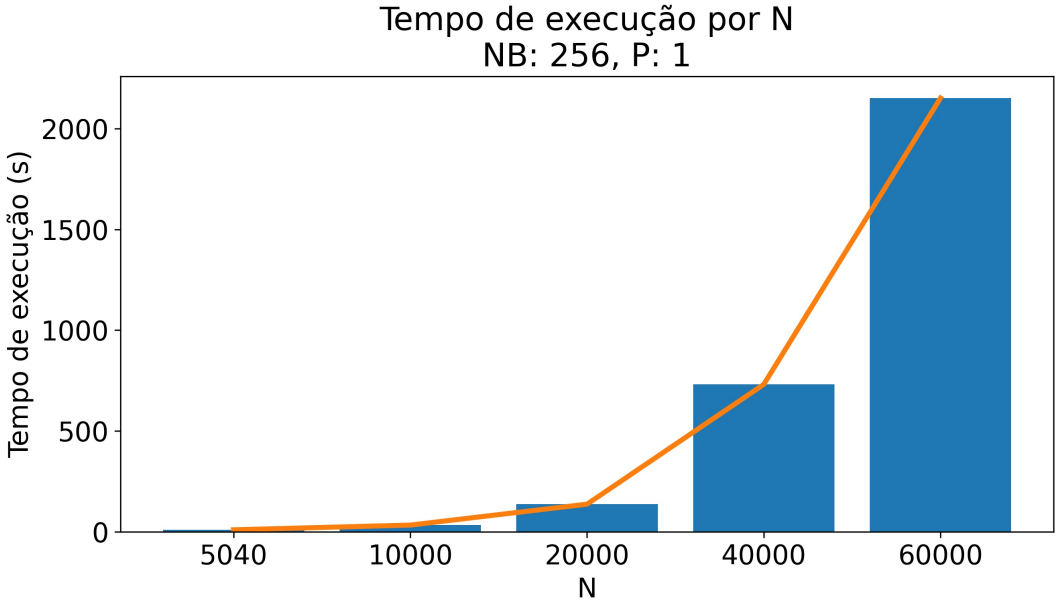

Performance por NB
N: 10000, P: 16
Gflops
Performance (Gflops)
NB


Tempo de execução por NB
N: 10000, P: 16
Time (s)
Tempo de execução (s)
NB

Ao analisar os dados acima, é possível perceber que os menores tempos e maiores desempenhos, independente do N, eram encontrados quando NB = 256, P = 1 e Q = 16. Os valores encontrados para P (número de colunas da matriz) e Q (número de linhas da matriz) não surpreendem, visto que a linguagem de programação C++, na qual o código é escrito e compilado, usa a ordem principal de linha para o armazenamento de arrays multidimensionais. Após essas conclusões, o HPL foi executado mais 3 vezes, fixando os valores de NB, P e Q e alterando os valores de N (20000, 40000 e 60000).

N	NB	P	Q	Time (s)	Gflops
5040	256	1	16	9.11	9.3756
10000	256	1	16	32.44	20.554
20000	256	1	16	136.29	39.136
40000	256	1	16	730.81	58.386
60000	256	1	16	2152.05	66.915





Analisando os novos dados é possível notar que a performance (em Gflops) cresce linearmente, enquanto o tempo de execução cresce exponencialmente.

Conclusão

Por todas as análises feitas, é possível concluir que em um ambiente de computação alto desempenho é necessário estudar o sistema e o código de forma a descobrir as melhores condições de execução, para que o programa desenvolvido atinja a maior performance possível. Também é importante ter a consciência da relação existente entre performance e tempo de execução, assim como de escalabilidade. No caso do *laplace.cxx* no computador utilizado para o estudo fica claro que para uma multiplicação de matrizes é ideal que elas tenham o número de linhas maior ou igual ao número de colunas, além de serem particionadas em um número de blocos alto (*e.g.* NB = 256).

Referências

[1] TOP500 - Frequently Asked Questions
`https://www.top500.org/resources/frequently-asked-questions/`

[2] University Information Technology Services - Understand measures of supercomputer performance and storage system capacity
`https://kb.iu.edu/d/apeq/#measure-flops`

[3] ULHPC Docs - High-Performance Linpack (HPL) benchmarking on UL HPC platform
`https://ulhpc-tutorials.readthedocs.io/en/latest/parallel/mpi/HPL/#MathJax-Span-97`

[4] NetLib - HPL Documentation
`https://www.netlib.org/benchmark/hpl/documentation.html`

[5] NetLib - HPL Algorithm
`https://www.netlib.org/benchmark/hpl/algorithm.html`