Relatório P2 — Arquitetura de Computadores II

Eduardo Cabral, Mateus da Silva, Romulo Menezes, and Victor Lopes

Departamento de Ciência da Computação, Universidade Federal Rural do Rio de Janeiro, Rio de Janeiro, Brasil.

Data: 22 de agosto de 2021

Professor: Marcelo Panaro de Moraes Zamith

1 Introdução

O experimento consistiu em comparar dois algoritmos de ordenação, quicksort e merge sort, em diferentes computadores e utilizando uma ferramenta de profile para extrair dados de desempenho. Para realizar os experimentos, foram definidos alguns parâmetros de padronização e métricas relevantes a serem coletadas.

Ferramenta de profile

A ferramenta de profile escolhida para a coleta das métricas foi o PAPI[1] (Performance Application Programming Interface). Houve tentativas de utilizar o Intel® VTune™ Profiler, porém, a extração dos dados resultantes da execução dos comandos da CLI[1] não é trivial, e se os dados fossem coletados pela GUI[2], resultaria em grande esforço manual, visto que o número de iterações era aproximadamente 1000.

[1]CLI - Command-line interface (Interface de Linha de Comando)
[2]GUI - Graphical User Interface (Interface Gráfica do Usuário)

Configuração dos computadores

CPU	i7-8550U	i7-2600	i5-5200U	i7-2600k
Cores	4 (HT)*	4 (HT)*	2 (HT)*	4 (HT)*
Clock base	1.80 GHz	3.40 GHz	2.20 GHz	3.40 GHz
Clock turbo (ST)**	4.00 GHz	3.80 GHz	2.70 GHz	4.40 GHz
RAM	8 GB (DDR4)	16 GB (DDR3)	12 GB (DDR3L)	12 GB(DDR3)

Tabela 1: *HT - Hyperthreading **ST - Single thread.

	i7-8550U	i7-2600	i5-5200U	i7-2600k
Cache L1 Dados	4x32 KB	4x32 KB	2x32 KB	4x32 KB
Cache L1 Instruções	4x32 KB	4x32 KB	2x32 KB	4x32 KB
Total Cache L1	256 KB	256 KB	256 KB	256 KB
Associatividade L1	8	8	8	8
Total Cache L2	4x256 KB	4x256 KB	2x256 KB	4x256 KB
Associatividade L2	4	8	8	8
Cache L3	1x8192 KB	1x8192 KB	1x3072 KB	1x8192 KB
Associatividade L3	16	16	12	16

	i7-8550U	i7-2600	i5-5200U	i7-2600k
OS	Ubuntu 20.04	Ubuntu 20.04	Ubuntu 20.04	Mint 19.3
Python	Python 3.7.6	Python 3.8.5	Python 3.5.1	Python 3.6.9
g++	g++ 9.3	g++ 9.3	g++ 9.3	g++ 7.5
PAPI	PAPI 6.0.0.1	PAPI 6.0.0.1	PAPI 6.0.0.1	PAPI 6.0.0.1

Método

Para a realização dos testes foram definidos parâmetros para a padronização dos mesmos em diferentes máquinas. Esses parâmetros consistiam em compilar e executar os códigos C++ a partir de um código escrito em Python. Foi definido também que os códigos seriam executados no text-mode do sistema, sem GUI ou quaisquer programas abertos pelo usuário, para obter ter um resultado mais preciso. Ao decorrer da execução dos códigos, as métricas coletadas eram escritas em arquivo para posteriormente ser realizada a criação dos gráficos. Os códigos foram compilados da seguinte forma:

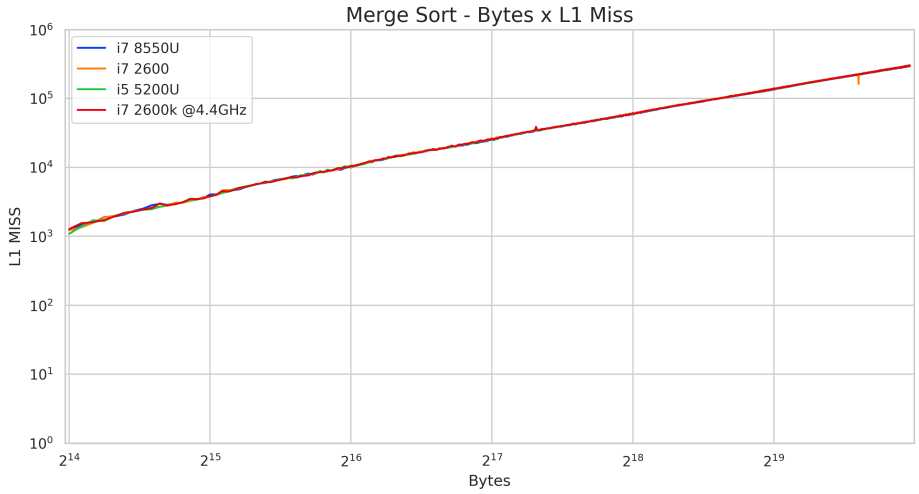
```
g++ ./lib/System.cpp algorithms/source_code.cpp -o ./bin/
output_code -lpapi -O0 -I./lib/ -I. -lm
```

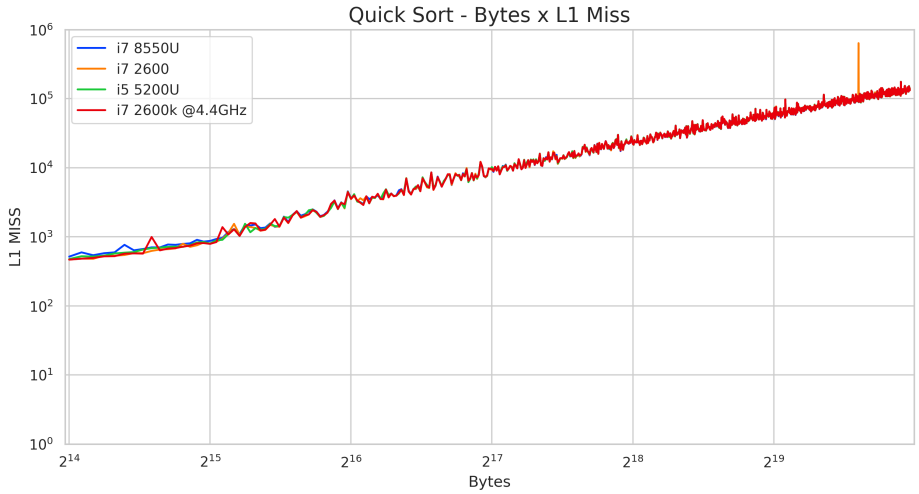
As métricas utilizadas foram cache miss nos níveis L1, L2 e L3, o total de instruções executadas, e o tempo de execução, todas coletadas pelo PAPI.

2 Resultados

2.1 Cache Miss

2.1.1 Tamanho do vetor em bytes x L1 Miss

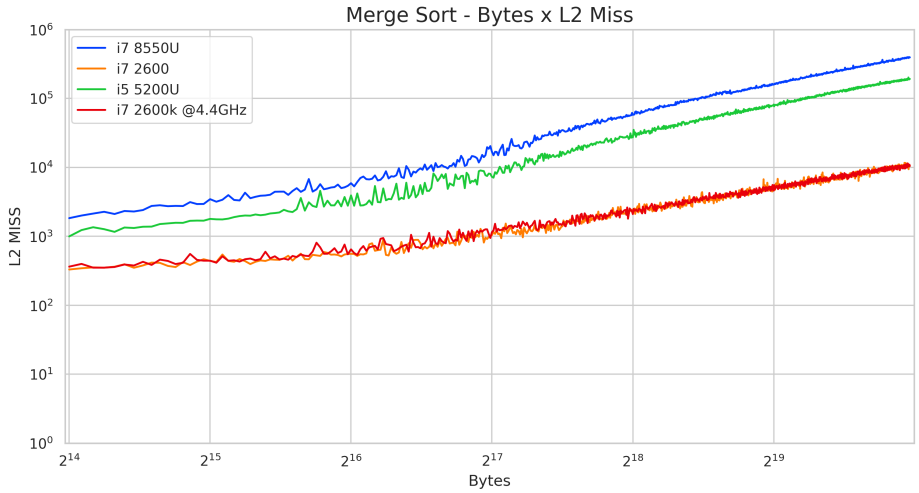


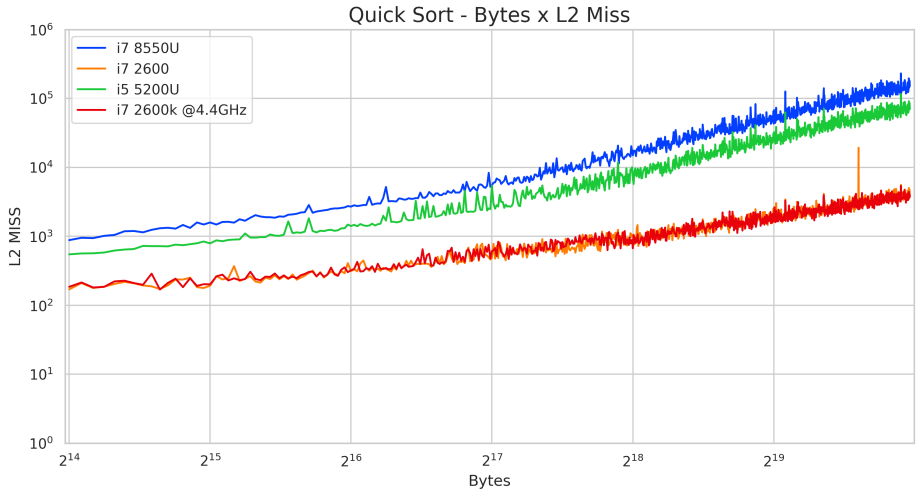


Como os processadores possuem o mesmo tamanho e associatividade de cache L1, a quantidade de L1 cache misses permaneceu semelhante entre todas CPUs para um dado algoritmo. Contudo, é possível perceber que o merge sort apresentou, em média, 126% (±1%) mais L1 cache miss que o quicksort em todos os processadores. O motivo da diferença expressiva pode ser atribuído ao fato de que o merge sort necessita de vetores adicionais para armazenar os valores ordenados das metades do vetor da etapa

anterior, o que geraria mais acessos à memória cache, que pode ser a causa do maior número de cache misses.

2.1.2 Tamanho do vetor em bytes x L2 Miss

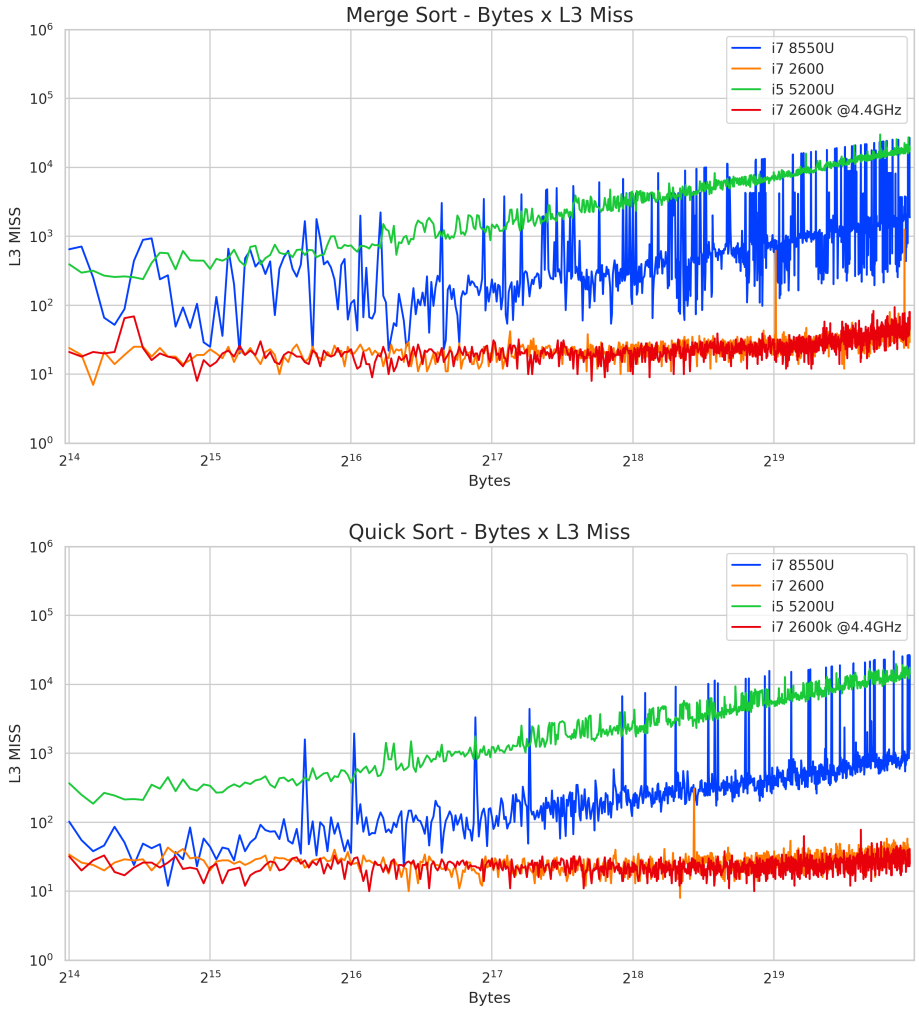




Nos resultados de L2 cache miss para ambos algoritmos, obtivemos uma anormalidade repetível, ambos i7 de segunda geração apresentaram níveis de cache miss muito inferiores às outras CPUs testadas, apesar de não haver nenhum motivo aparente que justifique tamanha disparidade. Deve-se notar que ambos notebooks utilizam arquiteturas mais novas que os desktops. O i5-5200U apresentou cerca de 1580% mais L2 cache miss no merge sort e 1340% mais L2 cache miss no quicksort quando

comparado ao i7-2600, que possui especificações idênticas de cache L2. Contudo, uma possível explicação para o i7-8550U ter apresentado 50% mais L2 cache miss que o i5-5200U pode ser porque o i7 apresenta cache L2 de mesmo tamanho, porém com nível de associatividade 4, enquanto o i5 possui nível de associatividade 8. A diferença entre as duas CPUs mobile nos indica que ambos algoritmos fazem uso massivo de apenas parte das entradas do cache, causando conflitos que sobrescrevem dados frequentemente, mesmo sem usar toda sua capacidade, então mais vias aparentam ser mais úteis que maior capacidade para estes algoritmos. Também foi encontrado que o algoritmo merge sort apresentou, em média, 168% (±15%) mais L2 cache miss que o quicksort.

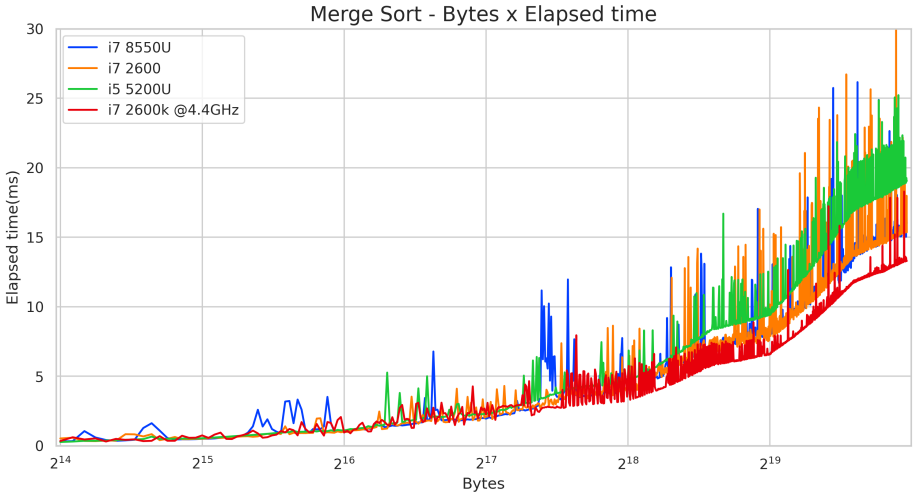
2.1.3 Tamanho do vetor em bytes x L3 Miss

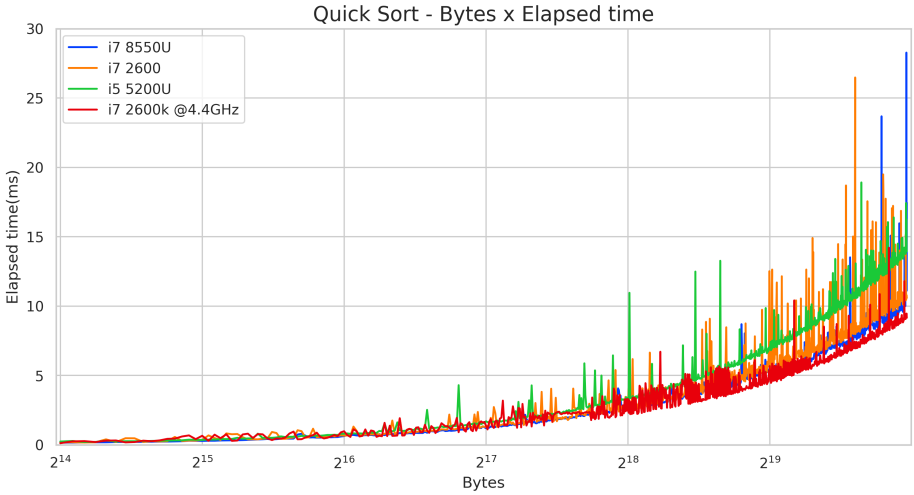


O cache L3 apresentou a menor diferença de cache misses entre o merge sort e quicksort, com o merge sort tendo, em média, apenas 3,6% a 82,8% mais L3 cache miss que o quicksort, o que representa maior intervalo entre as CPUs dentre todos resultados de diferença média entre cache miss do merge sort para o quicksort, com o i7-2600 e o i7-8550U apresentando 3,6% e 82,8% mais misses, respectivamente.

As anormalidades percebidas nos resultados de L2 cache miss para ambos i7 de segunda geração permanecem nos resultados de L3 cache miss, com o i7-8550U apresentando 2800% mais L3 cache miss no merge sort e 1550% mais L3 cache miss no quicksort quando comparado ao i7-2600. Outra anormalidade é a grande amplitude de cache miss no i7-8550U entre cada tamanho de vetor, o que destoa consideravelmente das curvas mais lineares das outras CPUs. Este fenômeno ocorreu em maior frequência no teste do merge sort e é repetível. Além disso, podemos perceber nesta análise que o i5-5200U apresentou 990% e 1440% mais L3 cache miss, no merge sort e quicksort, respectivamente, quando comparado ao i7-8550U, e isso pode ser largamente atribuído ao maior cache e nível de associatividade do i7-8550U (8MB, 16 vias) comparado ao i5-5200U (3MB, 12 vias).

2.2 Tempo de execução





Como pode ser observado nos gráficos, o i7-8550U e o i7-2600 obtiveram resultados muito próximos, apesar de grandes diferenças entre as CPUs, isso pode ser explicado pelos seguintes fatores: o i7-8550U apresenta uma arquitetura mais nova, e consequentemente, maior IPC[3], além de maior clock single core, porém, é prejudicado pela menor associatividade do cache L2, fazendo com que resultasse em um desempenho similar ao mais antigo i7-2600, que foi largamente auxiliado pela inferior taxa de cache miss, que permanece incompreendida. É possível também observar que o i7-8550U consegue uma pequena vantagem quando o quicksort é usado, por conta da

[3]IPC - Instruções por clock

menor dependência de cache do algoritmo, que é uma de suas desvantagens, porém, não é suficiente para superar o i7-2600k, que é idêntico ao i7-2600, mas com clock single core 15,8% superior devido ao seu overclock[4].

Diferença ***média*** de cache miss do merge sort para o quicksort				
Cache	**i5-5200U**	**i7-2600**	**i7-2600k**	**i7-8550U**
L1	+126,5%	+126,2%	+127,1%	+125,3%
L2	+182,9%	+154,1%	+165,4%	+180,5%
L3	+21,9%	+3,6%	+14,2%	+82,8%

Diferença ***mediana*** de cache miss do merge sort para o quicksort				
Cache	**i5-5200U**	**i7-2600**	**i7-2600k**	**i7-8550U**
L1	+128,9%	+130,8%	+131,7%	+128,5%
L2	+206,1%	+158,6%	+171,5%	+206,8%
L3	+18,7%	-3,8%	+4,1%	+73,5%

[4]overclock - Aumento da frequência de fábrica da CPU

3 Conclusão

O i5-5200U apresentou o pior desempenho das CPUs testadas, e isso se dá por ter o menor clock turbo single core e cache L3. Os i7-8550U e i7-2600 apresentaram resultados intermediários em ambos algoritmos, como explicado na análise dos resultados de tempo de execução, enquanto o i7-2600k @4.4GHz apresentou o melhor desempenho por ter as mesmas características do i7-2600, que apresentou bons resultados, mas com clock maior.

Sobre a comparação entre os dois algoritmos, é possível observar que o quicksort foi executado mais rapidamente que o merge sort em todas as CPUs, embora o algoritmo do merge sort tenha complexidade O(nlogn) e o quicksort tenha complexidade $O(n^2)$. Contudo, vale ressaltar que todos testes foram executados com a função 'posix_memalign' para alocar os dados dinamicamente em múltiplos do alinhamento informado, no caso 64 bytes - tamanho da linha de cache de todas CPUs -, e foi posteriormente descoberto que esta função não trazia nenhum ganho de desempenho aos algoritmos, pelo contrário, o overhead resultante de sua execução pode ser a causa de uma execução mais lenta de ambos algoritmos, em especial o merge sort, que foi mais afetado. Mas como todos os testes já haviam sido conduzidos, foi decidido por manter os resultados encontrados.

Todos os dados, algoritmos e gráficos informados no relatório podem ser consultados no repositório algorithms-comparison[2] do GitHub.

Referências

[1] You H. Terpstra D. Jagode H. e Dongarra J. "Collecting Performance Data with PAPI-C, Tools for High Performance Computing". Em: (2009).

[2] Romulo Morais Menezes Eduardo de Almeida Ferro Cabral Mateus Campello da Silva e Victor Lopes Machado. *algorithms-comparison*. URL: `https://github.com/FerroEduardo/algorithms-comparison`.