Universidade Federal Rural do Rio de Janeiro

Nicolas Magalhães Silva
Sérgio Felipe Rezende do Nascimento
Yan Carlos de Figueiredo Machado
Thiago Brum Ferreira

Análise do desempenho de algoritmos de ordenação utilizando PAPI

P2 - Arquitetura 2

magalhesnicolas23@gmail.com
sergiorezende2001@gmail.com
yancfm@gmail.com
th.brumferreira@gmail.com

1 Introdução

O seguinte relatório foi criado para explicar e apresentar os resultados da avaliação da execução de algoritmos de ordenação de vetores, QuickSort e MergeSort, aplicados na linguagem C++, o código-fonte está hospedado no github[1]. O projeto tem objetivo de entender como o processador e a cache estão se comportando durante a execução do pior caso de cada algoritmo, assim, podendo avaliar o comportamento da CPU e comparar as vantagens e desvantagens em cada caso.
As duas funções utilizadas foram reaproveitadas do trabalho de 'Estrutura de Dados I', sortlib, dos alunos Yan Figueiredo, Nicolas Magalhães e Sérgio Rezende, tal trabalho foi desenvolvido com o objetivo de reutilização em outros ambientes, logo, foi visado como boa opção para o projeto. Optamos por não usar pivô aleatório no QuickSort, pois acreditamos que a aleatoriedade poderia influenciar nos testes.

2 Desenvolvimento

A base do desenvolvimento, avaliação da execução, foi desenvolvida em C++ usando o profiler PAPI e a automatização dos testes foi feita usando bash, já a geração de gráficos feita em Julia utilizando Jupyter Notebook.

2.1 Dificuldades do Projeto

Foram realizadas tentativas, sem sucesso, de utilizar o Intel® VTune™ Profiler, a ferramenta apresentou uma complexidade elevada e baixa automatização, tais características corroboraram para o abandono do uso, pois não se adequou ao escopo do projeto.
Ao utilizarmos o PAPI[2] também encontramos diversas dificuldades que foram mais facilmente contornadas, dentre essas temos, por exemplo:

- Dificuldade para utilizar no Windows 10
- Processador Ryzen 5 3600 não possuía os eventos necessários
- Processador FX-8350 não possuía parte dos eventos necessários (L3 Cache miss)
- Eventos falsos positivos(constavam na lista de eventos disponíveis porem não podiam ser utilizados)

2.2 Scripts utilizados

Um script bash foi criado para automatizar execuções individuais do algoritmo, sendo necessário definir o tamanho do vetor e o algoritmo que será executado em linha de comando, observe o exemplo abaixo:

```
1    echo "[WARNING] papi needs to get permission to read system events"
2    sudo sh -c 'echo 0 >/proc/sys/kernel/perf_event_paranoid'
3
4    for counter in {2048..128000..128};
5    do
6    ./test.bin $counter MergeSort;
7    ./test.bin $counter QuickSort;
8    done
```

Basicamente, as primeiras linhas servem para garantir que seja permitido escutar os eventos a partir do PAPI, após isso é executado um loop para iterar sobre o tamanho, até a execução estar finalizada.

3 Testes

Os testes foram realizados utilizando os algoritmos de ordenação QuickSort e MergeSort em tamanho crescente, partindo de 16KB e chegando em 1MB em incrementos de 1KB. Os vetores gerados foram do tipo Inteiro Sem Sinal de 64 bits sendo preenchidos com valores decrescentes, visando simular o pior caso, e todos os testes foram rodados em diferentes versões do Ubuntu (20.0.4 LTS e 18.04 LTS).

3.1 Máquinas Utilizadas

A configuração dos processadores usados variou bastante, o que enriqueceu a variedade das amostras, inclusive processadores de marcas, arquiteturas e gerações diferentes foram utilizados, logo, corroborou para analisar pontos positivos e negativos de cada modelo.

	FX8350	**I5-7200U**	**I5-7400**	**I5-8265U**
Clock	4.0GHz	2.5GHz	3.0GHz	1.6GHz
Clock Turbo	4.2GHz	3.1GHz	3.5GHz	3.9GHz
L1 Cache Dados	8x16kB	2x32kB	4x32kB	4x32kB
Associatividade L1 Dados	4	8	8	8
L1 Cache Instruções	4x64kB	2x32kB	4x32kB	4x32kB
Associatividade L1 Intruções	2	8	8	8
L2 Cache	4x2MB	2x256kB	4x256kB	4x256kB
Associatividade L2	16	4	4	4
L3 Cache	8MB	3MB	6MB	6MB
Associatividade L3	64	12	12	12
RAM	DDR3	DDR4	DDR4	DDR4

Tabela 1: Configurações das CPUs

3.2 Método

Tendo em vista que o objetivo do projeto é analisar o comportamento da cache em diferentes processadores, foram escolhidos tais eventos:

- PAPI_L1_TCM: Evento que pega o cache miss da L1.
- PAPI_L2_TCM: Evento que pega o cache miss da L2.
- PAPI_L3_TCM: Evento que pega o cache miss da L3.
- PAPI_TOT_INS: Evento que pega o número de instruções executadas no código.

Outros eventos foram testados, um exemplo foi o PAPI_TOT_CYC que pegaria o número de clocks durante a execução, porém a maioria das máquinas encontrou erros. Além disso, é importante comentar que o processador da AMD não teve acesso ao evento da cache L3, por isso não estará presente nos gráficos envolvendo tal evento.

4 Resultados

Diversos gráficos relacionando ao número de bytes foram gerados para visualizar os pontos positivos e negativos de cada processador, no caso, vamos tentar entender o motivo de cada resultado, dada as características individuais e os resultados obtidos.

4.1 Cache miss

É possível perceber uma grande diferença na quantidade de cache miss entre as máquinas AMD vs Intel, isso se deve principalmente à diferença de seus tamanhos. Para exibir nossos resultados vamos separar a análise em cada nível de cache.

4.1.1 Cache L1

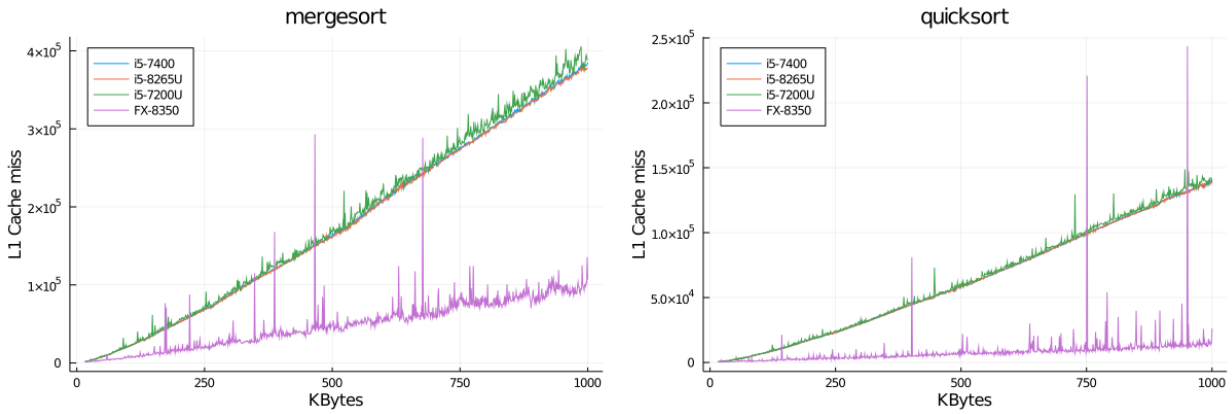

Figura 1: Quantidade de Cache Miss na Cache L1

A principal discrepância do gráfico fica no processador FX-8350, que possui um número bem reduzido de cache miss, isso se dá pelo fato da CPU possuir uma cache L1 de dados menor que a dos outros(16kB enquanto os outros tem 32kB), porém uma cache L1 de instruções significativamente maior(64kB enquanto os outro tem 32kB). Podemos observar então, que para algorítimos de ordenação o número de cache miss está mais ligado à quantidade de instruções do que ao tamanho do próprio vetor, visto que no total as caches tem praticamente o mesmo tamanho.

Outro fator interessante fica na proximidade entre os processadores da Intel, ainda que haja diferença entre gerações. O que nos mostra que a arquitetura da Cache L1 não fez tanta diferença.

4.1.2 Cache L2

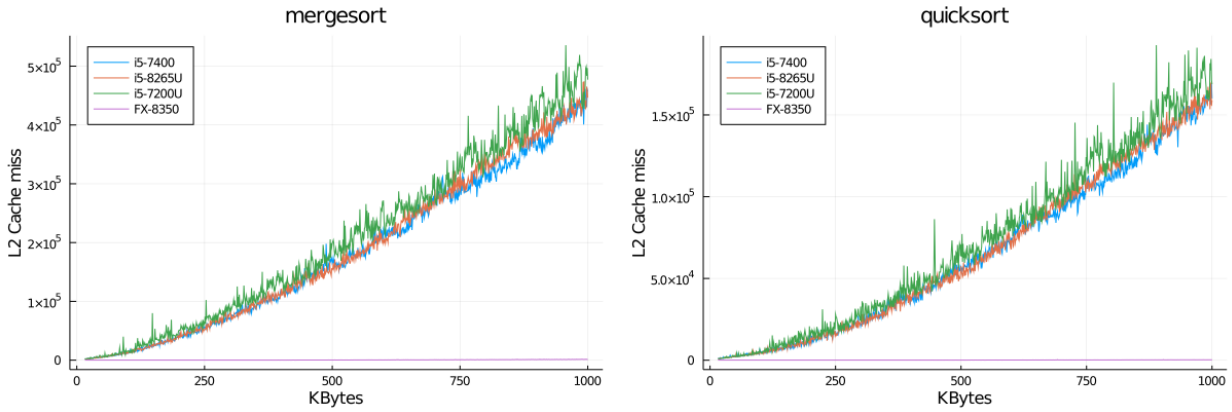

Figura 2: Quantidade de Cache Miss na Cache L2

Ao analisarmos o L2 cache miss, essa diferença entre o FX-8350 e os demais processadores da Intel se intensifica, o tamanho da cache L2 do processador da AMD é de 4x2MB, dessa forma, é possível armazenar todo o vetor, já que na maior iteração o tamanho do vetor é de 1MB.

Enquanto isso os processadores Intel se aproximam no número de cache miss tendo em vista que todos possuem o mesmo tamanho e associatividade (256KB e associatividade 4). Visto isso, podemos concluir que a diferença mínima entre o número de erros entre esses processadores se deve às diferenças na arquitetura e eficiência de cada modelo.

4.1.3 Cache L3

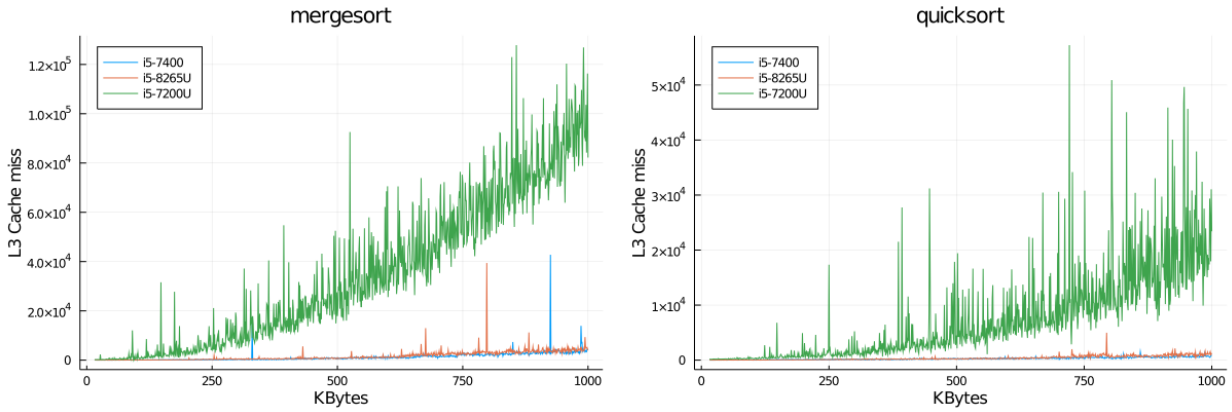

Figura 3: Quantidade de Cache Miss na Cache L3

Visto que não tivemos os dados relacionados com o FX para esse parâmetro, podemos ver agora que o processador que causa uma diferença é o I5-7200U, o processador menos potente entre os três. Vemos um número de cache miss bem maior nessa CPU, além de uma inconsistência muito grande entre os diferentes tamanhos de vetor, isso acontece devido ao tamanho reduzido do L3 que possui 3MB, metade dos outros concorrentes.

4.2 Tempo de execução

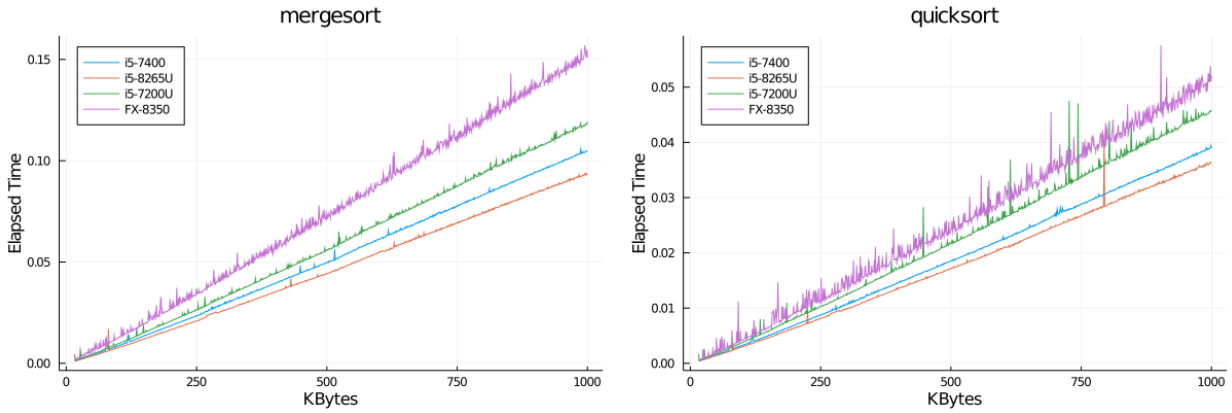

Figura 4: Tempo decorrido da execução dos algoritmos

A análise do tempo decorrido é bastante previsível, as novas tecnologias se provaram mais efetivas do que as antigas. Mesmo com o Clock superior a todas as máquinas usadas nos testes, o FX-8350 não conseguiu ter o melhor tempo, como, na verdade, teve o pior.

Os processadores da Intel tiveram o melhor desempenho, o I5-8265U foi o mais veloz junto ao i5-7400, seguindo a lógica da evolução dos processadores da Intel. É importante perceber a instabilidade do i5-7200U e do FX-8350, que mesmo rodando em um ambiente mais limitado(apenas terminal) para evitar discrepâncias e anomalias nos dados, não foi possível minimizar o ruído da curva.

4.3 Instruções por segundo

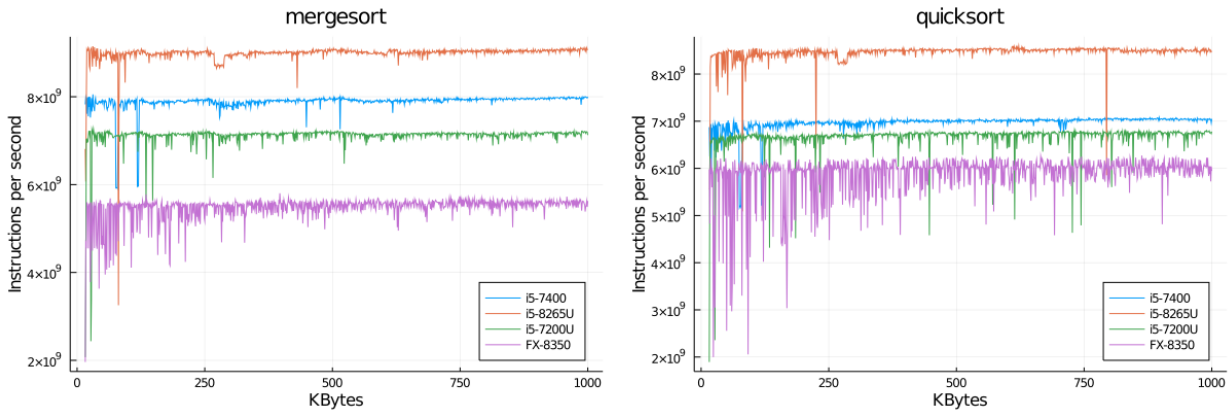

Figura 5: Instruções por segundo em cada processador

Ao dividirmos as instruções pelo tempo demorado em cada iteração conseguimos obter um gráfico que denota perfeitamente as capacidades de cada processador, mostrando com clareza o máximo de instruções que cada CPU consegue realizar por segundo.
Dessa forma é possível comparar diretamente o desempenho de cada processador como é mostrado na tabela abaixo:

QuickSort	**FX8350**	**I5-7200U**	**I5-7400**	**I5-8265U**
Mediana de I/s	6.0102e9	6.7203e9	7.0047e9	8.4973e9
Desempenho	100%	112%	116%	141%

Tabela 2: Análise de desempenho das instruções por processador (Base FX-8350)

[a] I/s = Instruções por Segundo.

MergeSort	**FX8350**	**I5-7200U**	**I5-7400**	**I5-8265U**
Mediana de I/s	5.5496e9	7.1514e9	7.9202e9	9.0161e9
Desempenho	100%	128%	142%	162%

Tabela 3: Análise de desempenho das instruções por processador (Base FX-8350)

Através dessa análise conseguimos perceber que o QuickSort diminui a diferença entre os processadores, enquanto o MergeSort aumenta essa diferença. Isso acontece, pois, o algoritmo MergeSort necessita de mais alocação e mais instruções no geral, enquanto o QuickSort possui menos alocação e a melhor localidade. Dessa forma, concluímos que o QuickSort exige menos da máquina e consegue trazer uma discrepância menor entre os processadores.

4.4 Diferenças entre algoritmos

Os dois algoritmos operam de forma bem diferente, o MergeSort aloca espaço extra, já o QuickSort não exige nenhuma memória extra e possui melhor localidade na cache, pois observa sempre a vizinhança do seu pivô, assim desfrutando o máximo da localidade. Levando isso em consideração, é possível perceber a diferença de desempenho entre os dois algoritmos que chega a ser 3x mais rápido nas instâncias de maior tamanho no QuickSort.
É Importante comentar que embora a complexidade de pior caso do MergeSort(O(nlogn)) seja melhor que a do QuickSort($O(n^2)$), essa vantagem apenas se põe em prática em valores de n bem maiores, onde a alocação extra pode valer mais a pena, os casos de 1 MByte ainda não conseguem desfrutar integralmente dessa particularidade. Existem muitos detalhes importantes para comparar ambos os algoritmos, por exemplo, o QuickSort faz poucas instruções em comparação ao MergeSort, como é possível perceber no gráfico abaixo:

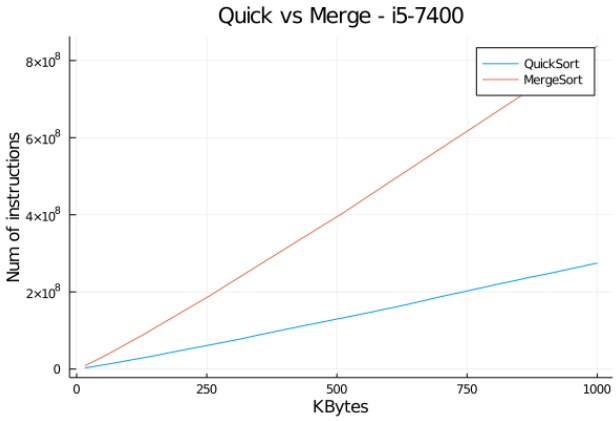

Figura 6: Comparativo entre o número de instruções executadas.

Como é possível perceber, no pior caso, o MergeSort faz cerca de $8x10^8$ instruções enquanto o QuickSort faz cerca de $2x10^8$, isso nos permite concluir que o QuickSort será superior com poucas instâncias, pois não existe necessidade de reescrever nem instruções, nem dados em excesso.

5 Conclusão

Portanto, após a análise dos dados obtidos foi possível entender como as diferenças de implementação conseguem impactar o desempenho do algoritmo na máquina. A simplicidade do QuickSort deixou nítida a vantagem sobre o MergeSort, que por sua alta taxa de instruções evidencia a diferença entre os hardwares. Seguindo os testes, ficou evidente que a quantidade de cache miss é proporcional a quantidade de alocações dinâmicas e instruções, chegando até 36% a mais de cache miss na L1, o MergeSort saiu perdendo por conta de alocar dinamicamente vetores auxiliares em cada recursão, assim esse novo vetor alocado estava sempre competindo a cache com o vetor original.

Sobre os processadores avaliados, foi possível entender que mesmo com uma cache enorme, o FX-8350 não consegue se comparar aos processadores mais recentes, seu tempo foi muito inferior ao restante dos processadores, ficando até 40% mais lento que o i5-8265U no MergeSort, mesmo assim, é louvável a baixa quantidade de cache miss que o processador apresentou. Ainda é possível supor que essa diferença foi causada pela tecnologia DDR3 de memória, não sendo uma competição justa com os novos modelos DDR4.

Por fim, ficamos felizes com os resultados obtidos e com todo o entendimento atrelado a essa pesquisa, conseguimos entender as diferenças relevantes para analisar o desempenho de um algoritmo e de um processador, percebendo a influência da cache e da tecnologia do processador no desempenho dos algoritmos.

Referências

[1] Thiago Brum Ferreira Nicolas Magalhães Silva Sérgio Felipe Rezende do Nascimento e Yan Carlos de Figueiredo Machado. *Análise do desempenho de algoritmos de ordenação utilizando PAPI*. URL: https://github.com/yaansz/ADDAOU-PAPI/.

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