UNICAMP - Universidade Estadual de Campinas

Instituto de Computação

MO601 - Arquitetura de Computadores II

Prof. Rodolfo Jardim de Azevedo

# Projeto 3

# Experimentar ferramentas e coletar dados

Fabrício Ferreira da Silva

RA 231900

Campinas – SP

Maio de 2023

# Sumário


1 Objetivo ..... 3

2 Especificação ..... 3

3 Ferramentas ..... 3

3.1 SPEC CPU 2017 benchmark ..... 3

3.1.1 Ambiente de teste ..... 4

3.1.2 Comandos executados ..... 4

3.1.3 Resultados ..... 4

3.1.4 Comparação de métricas com dados do site ..... 5

3.2 Simulador multi-core Sniper ..... 5

3.2.1 Ambiente de teste ..... 5

3.2.2 Comandos executados ..... 6

3.2.3 Slowdown ..... 6

3.2.4 Métricas de Desempenho ..... 7

3.3 Perf profiler ..... 7

3.3.1 Ambiente de teste ..... 8

3.3.2 Comandos executados ..... 8

3.3.3 Métricas de Desempenho ..... 8

3.4 Parsec benchmark ..... 9

3.4.1 Ambiente de teste ..... 9

3.4.2 Comandos executados ..... 9

3.4.3 Métricas de Desempenho ..... 10

3.5 Rodinia benchmark ..... 10

3.5.1 Ambiente de teste ..... 10

3.5.2 Versão e programas executados ..... 11

3.5.3 Resultados ..... 11

3.6 Intel Pin ..... 13

3.6.1 Ambiente de teste ..... 13

3.6.2 Comandos executados ..... 13

3.6.3 Métricas de Desempenho ..... 14

3.7 Dinero cache simulator ..... 14

3.7.1 Ambiente de teste ..... 14

3.7.2 Comandos executados ..... 15

3.7.3 Métricas de Desempenho ..... 15

4 Conclusão ..... 16

# 1 Objetivo

Ampliar o conhecimento de ferramentas úteis para avaliações em arquitetura de computadores executando múltiplas delas

# 2 Especificação

Você deve conhecer algumas das ferramentas e coletar dados sobre a execução delas. Cada ferramenta tem um conjunto de tarefas a realizar.

# 3 Ferramentas

- SPEC CPU 2017 benchmark
- Simulador multi-core Sniper
- Perf profiler
- Parsec benchmark
- Rodinia benchmark
- Intel Pin
- Dinero cache simulator

## 3.1 SPEC CPU 2017 benchmark

O pacote de benchmark SPEC CPU® 2017 contém conjuntos intensivos de CPU da próxima geração, padronizados pelo setor, da SPEC para medir e comparar o desempenho intensivo de computação, enfatizando o processador, o subsistema de memória e o compilador de um sistema.

A SPEC projetou esses conjuntos para fornecer uma medida comparativa de desempenho de computação intensiva na mais ampla gama prática de hardware usando cargas de trabalho desenvolvidas a partir de aplicativos de usuários reais. Os benchmarks são fornecidos como código-fonte e requerem o uso de comandos do compilador, bem como outros comandos por meio de um shell ou janela de prompt de comando.

O pacote de benchmark SPEC CPU 2017 contém 43 benchmarks, organizados em quatro suítes:

As suítes SPECspeed® 2017 Integer e SPECspeed® 2017 Floating Point são usadas para comparar o tempo de um computador para concluir tarefas únicas.

As suítes SPECrate® 2017 Integer e SPECrate® 2017 Floating Point medem a produtividade ou o trabalho por unidade de tempo.

### 3.1.1 Ambiente de teste

Esse benchmark foi executado usando o seguinte ambiente:

- Model : Avell High Performance C62 LIV
- OS: Microsoft Windows 11 Home Single Language 64-bit
- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- L1CacheSize: 64KB per core, L2CacheSize : 256KB per core, L3CacheSize : 12MB
- MaxClockSpeed : 2592
- NumberOfCores : 6
- NumberOfLogicalProcessors: 12
- Memory: 2 x 04CB AM1P26KC8T1-BAAS 8 GB (8589934592) at 2667

### 3.1.2 Comandos executados

Acessar a pasta raiz do SPEC 2017 e executar o arquivo **shrc.bat** que configura as variáveis de ambiente, em seguida para gerar esses resultados foram usados os seguintes comandos:

runcpu -c teste1-VisualStudio-windows.cfg --reportable --iterations 2 --copies=5 intrate

runcpu -c teste1-VisualStudio-windows.cfg --reportable --iterations 2 --copies=10 intrate

runcpu -c teste1-VisualStudio-windows.cfg --reportable --iterations 3 --copies=10 intrate

runcpu -c teste1-VisualStudio-windows.cfg --reportable --iterations 2 --threads=4 intspeed

runcpu -c teste1-VisualStudio-windows.cfg --reportable --iterations 3 --threads=16 intspeed

### 3.1.3 Resultados

Usando os comandos acima foram obtidos os seguintes resultados:

| **Resultados da execução do SPEC CPU 2017** | | | | | |
|---|---|---|---|---|---|
| **Suíte** | **Cópias** | **Threads** | **Nº de Iterações** | **Tempo de execução** | **Métrica Final (base)** |
| Intrate | 5 | X | 2 | 10.690 s – 2,97 hrs | 15.6 |
| Intrate | 10 | X | 2 | 13.334 s – 3,70 hrs | 24.9 |
| Intrate | 10 | X | 3 | 19.628 s – 5,45 hrs | 25.1 |
| Intspeed | X | 4 | 2 | 9.827 s – 2,73 hrs | 5.75 |
| Intspeed | X | 16 | 3 | 13.085 – 3,63 hrs | 6.26 |

### 3.1.4 Comparação de métricas com dados do site

Para comparar as métricas obtidas no meu computador com as métricas do site eu acessei o link https://www.spec.org/cpu2017/results/cpu2017.html e busquei por testes em que o processador também fosse do modelo Intel Core i7, os resultados encontrados foram:

| **Resultados da execução do SPEC CPU 2017 no site** | | | | |
|---|---|---|---|---|
| **Suíte** | **Ambiente** | **Cópias** | **Threads** | **Métrica Final (base)** |
| Intrate | SuperWorkstation 5039C-T (X11SCA, Intel Core i7-9700K) | 8 | X | 44.8 |
| Intrate | ASUS Z170M-PLUS Motherboard (Intel Core i7-6700K) | 8 | X | 23.5 |
| Intspeed | SuperWorkstation 5039C-T (X11SCA, Intel Core i7-9700K) | X | 8 | 10.6 |

Comparado com os dados do site o **SuperWorkstation 5039C-T** obteve uma métrica final bem superior em comparação com o meu computar, porém ele possui configurações de hardware bem superiores. Já comparando com o **ASUS Z170M-PLUS** só foram encontrados testes usando o intrate e as métricas finais ficaram bem próximas, já que ele possui configurações de hardware mais parecidas com meu computador.

## 3.2 Simulador multi-core Sniper

O Sniper é um simulador multi-core desenvolvido para a avaliação de desempenho de processadores multi-core. O Sniper permite simular o desempenho de processadores multi-core considerando detalhes como pipeline de instruções, hierarquia de memória, escalonamento de tarefas, latência de comunicação entre núcleos, entre outros.

### 3.2.1 Ambiente de teste

Esse benchmark foi executado usando o seguinte ambiente:

- Model : Avell High Performance C62 LIV
- OS: Ubuntu 18.04 LTS
- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- L1CacheSize: 64KB per core, L2CacheSize : 256KB per core, L3CacheSize : 12MB
- MaxClockSpeed : 2592
- NumberOfCores : 6
- NumberOfLogicalProcessors: 12
- Memory: 2 x 04CB AM1P26KC8T1-BAAS 8 GB (8589934592) at 2667

### 3.2.2 Comandos executados

Acessar a pasta raiz do Sniper e configurar as variáveis de ambiente usando o diretório de instalação do mesmo com os seguintes comandos:

export SNIPER_ROOT=/home/fabricio/Downloads/Projeto3/sniper-latest/sniper-7.4

export GRAPHITE_ROOT=/home/fabricio/Downloads/Projeto3/sniper-latest/sniper-7.4

export BENCHMARKS_ROOT=$(pwd)

Em seguida para gerar os resultados de tempo nativos foram usados os seguintes comandos:

time ./benchmarks/FFT > FFT.txt

time ./benchmarks/RADIX > RADIX.txt

time ./benchmarks/CHOLESKY benchmarks/inputs/tk14.O > CHOLESKY.txt

Por fim, para gerar as métricas do Sniper foram usados os seguintes comandos:

time ./run-sniper ./benchmarks/FFT > FFT.txt

time ./run-sniper ./benchmarks/RADIX > RADIX.txt

time ./run-sniper ./benchmarks/CHOLESKY benchmarks/inputs/tk14.O > CHOLESKY.txt

### 3.2.3 Slowdown

Usando os comandos acima foram obtidos os seguintes resultados:

| **Resultados de Slowdown da execução do Sniper** | | | | |
|---|---|---|---|---|
| **Programa** | **N° de Instruções** | **Tempo execução simulador (TS)** | **Tempo de execução Nativo (TN)** | **Slowdown (TS/TN)** |
| FFT | 1025064 | 4.90s | 0,003s | 1.633,33 |
| RADIX | 46726412 | 42.37s | 0,020s | 2.118,50 |
| CHOLESKY | 43487004 | 42.19s | 0,005s | 8.438,00 |

### 3.2.4 Métricas de Desempenho

Usando os comandos acima para a geração de métricas, foram obtidos os seguintes resultados:

| Resultados da execução do Sniper | | | |
|---|---|---|---|
| **Métrica** | **FFT** | **RADIX** | **CHOLESKY** |
| **N° de Instruções** | 1.025.064 | 46.726.412 | 43.487.004 |
| **N° de ciclos** | 997.766 | 104.566.728 | 19.613.244 |
| **IPC** | 1.03 | 0.45 | 2.22 |
| **N° de acessos cache L1-I** | 119.822 | 3.705.548 | 4.287.867 |
| **Miss rate** | 1.52% | 0.05% | 0.13% |
| **N° de acessos cache L1-D** | 316.390 | 4.830.459 | 14.619.073 |
| **Miss rate** | 1.93% | 6.09% | 1.35% |
| **N° de acessos cache L2** | 8.015 | 296.310 | 203.821 |
| **Miss rate** | 72.26% | 41.43% | 56.36% |
| **N° de acessos cache L3** | 5.874 | 122.817 | 115.031 |
| **Miss rate** | 97.53% | 30.69% | 36.35% |
| **N° branches** | 111.197 | 1.372.978 | 4.696.489 |
| **Miss rate** | 4.98% | 0.25% | 1.23% |

## 3.3 Perf profiler

O Perf é uma ferramenta de análise de desempenho para sistemas Linux. Ele é baseado na tecnologia de contadores de desempenho do kernel do Linux, que permite monitorar uma ampla variedade de eventos de hardware, como por exemplo ciclos de CPU, número de instruções executadas, cache misses, chamadas de sistema, entre outros. Esses eventos são coletados em nível de sistema operacional.

### 3.3.1 Ambiente de teste

Esse benchmark foi executado usando o seguinte ambiente:

- Model : Avell High Performance C62 LIV
- OS: Ubuntu 18.04 LTS
- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- L1CacheSize: 64KB per core, L2CacheSize : 256KB per core, L3CacheSize : 12MB
- MaxClockSpeed : 2592
- NumberOfCores : 6
- NumberOfLogicalProcessors: 12
- Memory: 2 x 04CB AM1P26KC8T1-BAAS 8 GB (8589934592) at 2667

### 3.3.2 Comandos executados

Foram executados os seguintes comandos para a coleta de métricas do Perf:

time perf stat ./benchmarks/FFT > FFT.txt
time perf stat ./benchmarks/RADIX > RADIX.txt
time perf stat ./benchmarks/CHOLESKY benchmarks/inputs/tk14.O > CHOLESKY.txt

### 3.3.3 Métricas de Desempenho

Foram coletadas, quando disponíveis, as mesmas métricas geradas no Sniper, são elas:

| **Resultados da execução do Perf** | | | |
|---|---|---|---|
| **Métrica** | **FFT** | **RADIX** | **CHOLESKY** |
| **N° de Instruções** | 1.968.421 | 49.286.534 | 45.328.916 |
| **N° de ciclos** | 1.785.176 | 104.398.364 | 19.855.063 |
| **IPC** | 1.10 | 0.47 | 2.28 |
| **N° branches** | 320.137 | 1.833.348 | 5.796.998 |
| **Miss rate** | 3.15% | 0.61% | 2.34% |

Ao comparar o resultado das métricas encontradas no Perf em relação ao Sniper, elas tiveram valores bem próximos, como por exemplo o IPC, porém foi notado um aumento no número de instruções e na Branch Miss Rate. Acredito que o aumento no número de instruções, deve ter ocorrido pois o perf monitora a execução nativa da aplicação, portanto ele é executado em paralelo com o sistema operacional, acarretando instruções adicionais.

## 3.4 Parsec benchmark

O PARSEC Benchmark Suite 3.0 é um conjunto de benchmarks utilizado para avaliar o desempenho de sistemas computacionais em ambientes paralelos e distribuídos. As aplicações são projetadas para aproveitar paralelismo em diferentes níveis, incluindo paralelismo de tarefa, de dados e de instruções.

### 3.4.1 Ambiente de teste

Esse benchmark foi executado usando o seguinte ambiente:

- Model : Avell High Performance C62 LIV
- OS: Ubuntu 18.04 LTS
- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- L1CacheSize: 64KB per core, L2CacheSize : 256KB per core, L3CacheSize : 12MB
- MaxClockSpeed : 2592
- NumberOfCores : 6
- NumberOfLogicalProcessors: 12
- Memory: 2 x 04CB AM1P26KC8T1-BAAS 8 GB (8589934592) at 2667

### 3.4.2 Comandos executados

Foram executados os seguintes comandos para a coleta de métricas do Parsec, alterando apenas o valor após o **-n** pelo número de threads desejadas, para esses testes foram usados os seguintes valores: 1, 2, 4, 8, 16.

```
parsecmgmt -a run -p fluidanimate -i native -n 1
parsecmgmt -a run -p blackscholes -i native -n 1
parsecmgmt -a run -p freqmine -i native -n 1
parsecmgmt -a run -p vips -i native -n 1
parsecmgmt -a run -p dedup -i native -n 1
parsecmgmt -a run -p bodytrack -i native -n 1
parsecmgmt -a run -p facesim -i native -n 1
parsecmgmt -a run -p ferret -i native -n 1
parsecmgmt -a run -p swaptions -i native -n 1
```

Não foi usado o comando **parsecmgmt -a run -p all -i native**, pois apesar de na etapa de build e execução com apenas 1 thread ocorrer tudo bem, nos testes com mais threads a execução ficava estática no benchmark **Raytrace.** Além disso o programa **X264** também não funcionou corretamente, então não foram coletadas métricas de ambos.

### 3.4.3 Métricas de Desempenho

Usando os comandos acima para a geração de métricas, foram obtidos os seguintes resultados:

| Resultados da execução do Parsec | | | | | |
|---|---|---|---|---|---|
| **Programas** | **N° de Threads** | | | | |
| | **1** | **2** | **4** | **8** | **16** |
| **fluidanimate** | 3m 8.77s | 1m 42.45s | 57.30s | 47.76s | 43.78s |
| **blackscholes** | 52.76s | 32.21s | 21.95s | 18.23s | 16.53s |
| **freqmine** | 4m 40,02s | 2m 18.75s | 1m 12,06s | 48.56s | 44.11s |
| **vips** | 50.96s | 26.92s | 14.26s | 9.41s | 10.62s |
| **dedup** | 11.11s | 5.83s | 3.48s | 3.01s | 3.84s |
| **bodytrack** | 1m 12.72s | 38.77s | 22.30s | 16.57s | 14.27s |
| **facesim** | 3m 0.02s | 1m 37.85s | 55.92s | 50.41s | 46.92s |
| **ferret** | 2m 51.18s | 1m 28.72s | 48.11s | 34.22s | 32.43s |
| **swaptions** | 2m 14.59s | 1m 8.77s | 36.89s | 24.47s | 22.58s |

## 3.5 Rodinia benchmark

O Rodinia benchmark é um conjunto de benchmarks desenvolvido pela Universidade da Virgínia. Ele foi desenvolvido para testar o desempenho de hardware, software e algoritmos em tarefas paralelas e computacionalmente intensivas. Os benchmarks dele são escritos em diversas linguagens de programação, como C, C++, CUDA, OpenCL, entre outros; para permitir a avaliação de diferentes ambientes de programação paralela. Ele calcula várias métricas de desempenho, incluindo tempo de execução, taxa de processamento, throughput, acurácia, entre outros.

### 3.5.1 Ambiente de teste

Esse benchmark foi executado usando o seguinte ambiente:

- Model : Avell High Performance C62 LIV
- OS: Ubuntu 18.04 LTS
- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- L1CacheSize: 64KB per core, L2CacheSize : 256KB per core, L3CacheSize : 12MB
- MaxClockSpeed : 2592
- NumberOfCores : 6
- NumberOfLogicalProcessors: 12

- Memory: 2 x 04CB AM1P26KC8T1-BAAS 8 GB (8589934592) at 2667

### 3.5.2 Versão e programas executados

Foi usado para esse teste o Rodinia versão 3.1 e os programas executados foram o Kmeans, LavaMD e Streamcluster na linguagem OpenCL.

### 3.5.3 Resultados

#### *3.5.3.1 LavaMD*

Para o LavaMD foi acessada a pasta raiz do programa e foram usados os comandos a seguir, gerando os dados presentes na tabela abaixo:

./lavaMD -cores 16 -boxes1d 10

./lavaMD -cores 8 -boxes1d 10

./lavaMD -cores 4 -boxes1d 10

./lavaMD -cores 2 -boxes1d 10

./lavaMD -cores 1 -boxes1d 10

| **Resultados da execução do LavaMD no Rodinia** | |
|---|---|
| **N° de Threads** | **Tempo de Execução** |
| 1 | 4.72s |
| 2 | 2.46s |
| 4 | 1.36s |
| 8 | 1.05s |
| 16 | 0.91s |

Analisando os dados foi possível observar que com o aumento do número de threads o tempo de execução do programa foi diminuindo consideravelmente, porém não de uma forma linear.

#### *3.5.3.2 Streamcluster*

Para o Steamcluster foi acessada a pasta raiz do programa e foram usados os comandos a seguir, gerando os dados presentes na tabela abaixo:

./sc_omp 10 20 256 16384 16384 1000 none output.txt 4

./sc_omp 10 20 256 32768 32768 1000 none output.txt 4

./sc_omp 10 20 256 65536 65536 1000 none output.txt 4

./sc_omp 10 20 256 131072 131072 1000 none output.txt 4

| Resultados da execução do Streamcluster no Rodinia | |
|---|---|
| **N° de Pontos** | **Tempo de Execução** |
| 16384 | 2.12s |
| 32768 | 5.24s |
| 65536 | 9.23s |
| 131072 | 19.62s |

O número de pontos foi sendo multiplicado por 2 a cada teste, como esperado o tempo de execução do programa aumentou, mas não foi de uma forma linear.

#### *3.5.3.3 Kmeans*

Para o Kmeans foi acessada a pasta raiz do programa e foram usados os comandos a seguir, gerando os dados presentes na tabela abaixo:

./kmeans_openmp/kmeans -n 16 -i ../../data/kmeans/kdd_cup

./kmeans_openmp/kmeans -n 8 -i ../../data/kmeans/kdd_cup

./kmeans_openmp/kmeans -n 4 -i ../../data/kmeans/kdd_cup

./kmeans_openmp/kmeans -n 2 -i ../../data/kmeans/kdd_cup

./kmeans_openmp/kmeans -n 1 -i ../../data/kmeans/kdd_cup

| Resultados da execução do Kmeans no Rodinia | |
|---|---|
| **N° de Threads** | **Tempo de Execução** |
| 1 | 2.38s |
| 2 | 1.38s |
| 4 | 0.97s |
| 8 | 0.63s |
| 16 | 0.47s |

Analisando os dados foi possível observar que assim como o LavaMD no Kmeans com o aumento do número de threads o tempo de execução do programa foi diminuindo consideravelmente, porém não de uma forma linear.

## 3.6 Intel Pin

O Intel Pin é uma ferramenta para avaliação dinâmica e instrumentação de código desenvolvida pela Intel. Ela é usada para examinar, modificar e avaliar o comportamento de programas em tempo de execução, sem precisar alterar o código fonte original. Ele funciona como um ”pino” que se acopla a um programa permitindo que os desenvolvedores coletem dados detalhados sobre desempenho, comportamento e características do mesmo.

Possuí uma API flexível e extensível que permite a criação de ferramentas personalizadas para coleta de informações ou para realizar análises. Com o Pin é possível realizar uma grande gama de análises, como profiling de desempenho, rastreamento de instruções, monitoramento de acessos a memória, entre outros.

### 3.6.1 Ambiente de teste

Esse benchmark foi executado usando o seguinte ambiente:

- Model : Avell High Performance C62 LIV
- OS: Ubuntu 18.04 LTS
- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- L1CacheSize: 64KB per core, L2CacheSize : 256KB per core, L3CacheSize : 12MB
- MaxClockSpeed : 2592
- NumberOfCores : 6
- NumberOfLogicalProcessors: 12
- Memory: 2 x 04CB AM1P26KC8T1-BAAS 8 GB (8589934592) at 2667

### 3.6.2 Comandos executados

Foram executados os seguintes comandos para a coleta de métricas do Pin, foram usadas as ferramentas **icount** e **dcache** no processo:

```
time ../../../pin -t obj-intel64/icount .so -- benchmarks/RADIX > RADIX.txt
time ../../../pin -t obj-intel64/icount.so -- benchmarks/CHOLESKY
benchmarks/inputs/tk14.O > CHOLESKY.txt
time ../../../pin -t obj-intel64/icount .so -- benchmarks/FFT > FFT.txt
```

```
time ../../../pin -t obj-intel64/dcache.so -- benchmarks/RADIX > RADIX.txt
time ../../../pin -t obj-intel64/dcache .so -- benchmarks/CHOLESKY
benchmarks/inputs/tk14.O > CHOLESKY.txt
time ../../../pin -t obj-intel64/dcache.so -- benchmarks/FFT > FFT.txt
```

### 3.6.3 Métricas de Desempenho

Usando os comandos acima para a geração de métricas, foram obtidos os seguintes resultados:

| Resultados da execução do Pin | | | |
|---|---|---|---|
| **Métrica** | **FFT** | **RADIX** | **CHOLESKY** |
| **N° de Instruções** | 1.001.072 | 46.704.089 | 43.462.742 |
| **N° de loads cache L1-D** | 208.156 | 2.959.767 | 9.762.617 |
| **Miss rate** | 3.58% | 5.66% | 3.00% |
| **N° de store cache L1-D** | 99.553 | 812.097 | 4.559.491 |
| **Miss rate** | 3.65% | 26.82% | 2.27% |
| **N° de acessos cache L1-D** | 307.709 | 3.771.864 | 14.322.108 |
| **Miss rate** | 3.60% | 10.21% | 2.76% |

## 3.7 Dinero cache simulator

O Dinero IV cache simulator é uma ferramenta de simulação de cache desenvolvida para facilitar a análise e estudo do comportamento do cache em sistemas. Ele permite aos usuários examinarem o desempenho e o impacto da hierarquia de cache em programas e algoritmos. Ele simula a hierarquia de cache de um sistema, incluindo cache L1, L2 e, caso possua, L3. Ele permite a definição de diferentes configurações de cache, como tamanho, associatividade, políticas de substituição e de escrita, entre outros. Isso permite explorar diferentes cenários e entender como as configurações de caches afetam o desempenho.

### 3.7.1 Ambiente de teste

Esse benchmark foi executado usando o seguinte ambiente:

- Model : Avell High Performance C62 LIV
- OS: Ubuntu 18.04 LTS
- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- L1CacheSize: 64KB per core, L2CacheSize : 256KB per core, L3CacheSize : 12MB

- MaxClockSpeed : 2592
- NumberOfCores : 6
- NumberOfLogicalProcessors: 12
- Memory: 2 x 04CB AM1P26KC8T1-BAAS 8 GB (8589934592) at 2667

### 3.7.2 Comandos executados

Foi executado o seguinte comando para a coleta de métricas do Dinero, foram testados diversos tamanhos de cache diferentes e o programa escolhido foi o RADIX:

```
./dineroIV -informat p -l1-dsize 8K -l1-isize 8K -l1-ibsize 16 -l1-dbsize 8 -l2-usize 512K -l2-ubsize 32K -l2-uassoc 1 -l3-usize 2M -l3-ubsize 32K -l3-uassoc 1 < radix/radix.din
```

Esse foi o comando que gerou os melhores resultados, foi variado tamanhos de cache L1 de 1 a 16 KB, L2 de 1 a 512KB e L3 de 64KB a 2MB.

### 3.7.3 Métricas de Desempenho

Usando o comando acima com diversas configurações para a geração de métricas, foram obtidos os seguintes resultados:

| **Resultados da execução do Dinero L1** | | | |
|---|---|---|---|
| **Métrica** | **1KB** | **8KB** | **16KB** |
| **l1-dcache Miss rate** | 0.5548 | 0.2782 | 0.2782 |

Escolhido 8KB pois a taxa de Miss rate com 16KB foi idêntica.

| **Resultados da execução do Dinero L2** | | | |
|---|---|---|---|
| **Métrica** | **1KB** | **128KB** | **512KB** |
| **L2-ucache Miss rate** | 0.4479 | 0.2772 | 0.0826 |

Escolhido 512KB pois a taxa de Miss rate foi a mais baixa e aumentando para 1MB não houve muita mudança.

| Resultados da execução do Dinero L3 | | | |
|---|---|---|---|
| **Métrica** | **64KB** | **1MB** | **2MB** |
| **L3-ucache Miss rate** | 0.6643 | 0.4535 | 0.0808 |

Escolhido 2MB pois a taxa de Miss rate foi a mais baixa e aumentando para 4MB não houve muita mudança.

# 4 Conclusão

Esse projeto foi muito interessante, pois ele propôs uma maneira diferente de trabalhar, já que no projeto 1 e 2 foram os alunos que desenvolveram simuladores para receber dados e apresentar resultados, nesse o objetivo foi utilizar ferramentas de benchmarks de terceiros para aprender sobre esse tema, entender como as ferramentas funcionam e interpretar os resultados recebidos e tirar conclusões.

Aprendi diversos tipos de métricas de desempenho de computadores e como interpretá-las e compará-las, o que será muito útil para calcular desempenho de aplicações em projetos futuros, que são ótimos dados para comparação e validação de se uma aplicação é viável ou não.

Além disso, durante a instalação dos pacotes, conheci muitas novas linguagens de programação e ferramentas de suporte para benchmarks, que já que nunca havia usado ferramentas desse tipo, tive que instalar todas as dependências de cada uma delas, assim conhecendo-as e aprendendo como usá-las.

Tive algumas dificuldades durante a configuração das ferramentas devido à falta de expertise com elas, porém felizmente consegui executar todas as ferramentas e obter todas as métricas solicitadas pelo professor, concluindo com êxito esse projeto.

Concluindo, ao fim desse projeto tive uma visão da importância do uso dessas ferramentas de benchmark, já que elas são essenciais para a comparação, avaliação e validação de dispositivos e acredito que no futuro, já possuo uma base suficiente para saber como utilizá-las de maneira correta, restando apenas um aprofundamento nas funcionalidades e programas da ferramenta usada para que seja possível abstrair os melhores resultados delas.