Análise do desempenho de sistemas computacionais a partir da base SPEC CPU 2017: um estudo introdutório dos resultados de speed-int

Autores: David Kauan Carneiro Pereira (25103306)[1], Gustavo Sonntag Dorow (25100806)[1], Johan Akin Araújo da Silva Rodrigues (25103312)[1], Jonathan Tenório de Lima (25102233)[1].

[1]Universidade Federal de Santa Catarina (UFSC), Departamento de Informática e Estatística (INE), Graduação em Sistemas de Informação, <secdavidka@gmail.com>, <gustavosdorow@gmail.com>, <johanwrodrigues@gmail.com>, <jtlimads@gmail.com>.

Introdução e objetivos

O custo decrescente dos componentes dos sistemas computacionais, em especial os processadores, permitiu uma contínua e intensa evolução no desempenho das máquinas perante algoritmos que resolvem problemas cada vez mais complexos. Os blocos construtores (building blocks) dos computadores modernos são virtualmente os mesmos que deram origem ao IAS, no entanto, as técnicas para extrair o máximo desempenho tornaram-se altamente sofisticadas (STALLINGS, 2021).

A microarquitetura de um sistema computacional compreende o conjunto de implementações internas de um processador que definem como as instruções (ISA) são executadas no nível de hardware e, por isso mesmo, possui papel chave no desempenho dos computadores. Os benchmarks possuem papel fundamental na evolução contínua da microarquitetura dos sistemas e compiladores (LIMAYE; ADEGBIJA, 2018).

Um dos benchmarks mais utilizados globalmente é fornecido pela *Standard Performance Evaluation Corporation* (Standard Performance Evaluation Corporation (SPEC), 2024), cuja última versão (SPEC CPU 2017) avaliou 43 *benchmarks* diferentes conforme a tabela 1.

Tabela 1: Comparação entre SPECspeed®® e SPECrate®®.

Descrição	**SPECspeed®2017**	**SPECrate®2017**
Medição	Desempenho de execução de um programa com núcleo único (CPU)	Quantas tarefas o sistema inteiro consegue executar de uma vez
Foco	Velocidade de uma única tarefa (ou thread)	Taxa de transferência ou performance paralela (*parallel performance*)
Caso de uso	Bom para aplicações que executam um trabalho por vez	Aplicações que executam muitas tarefas simultaneamente
Como é executado	Executa uma cópia de cada programa em um único núcleo	Executa muitas cópias de cada programa usando todos os *cores* disponíveis
Pontuação	Quanto mais elevada, menos tempo de execução é necessário	Quanto maior, mais trabalho pode ser realizado por unidade de tempo
Desempenho	Duração em uma máquina de referência ÷ duração no SUT	Número de cópias × (Duração em uma máquina de referência ÷ duração no SUT)

Fonte: Adaptado de Standard Performance Evaluation Corporation (SPEC) (2024).

O SPEC 2017 é dividido em ***speed***, que mede o desempenho dos computadores na execução de uma tarefa única a partir de um CPU de núcleo *core* singular; enquanto o ***rate*** mede o desempenho para múltiplas tarefas utilizando todos os *cores* disponíveis no CPU. Para cada um destes, existem *benchmarks* específicos para inteiros (*int*) e números de ponto flutuante (*float*). O primeiro caso está fortemente relacionado com a disciplina de "Arquitetura e Organização de Computadores"ministrada para o curso de Sistemas de Informação e, por isso, será utilizado como a base de dados deste trabalho. A tabela 2 apresenta um resumo das aplicações.

Tabela 2: Benchmarks do SPECspeed®®2017 Integer.

Benchmark	**Linguagem**	**KLOC***	**Área de Aplicação**
600.perlbench_s	C	362	Interpretador Perl
602.gcc_s	C	1.304	Compilador GNU C
605.mcf_s	C	3	Planejamento de rotas
620.omnetpp_s	C++	134	Simulação de eventos discretos de redes
623.xalancbmk_s	C++	520	Conversão XML $\rightarrow$ HTML via XSLT
625.x264_s	C	96	Compressão de vídeo
631.deepsjeng_s	C++	10	Inteligência artificial: busca em árvore alpha-beta (xadrez)
641.leela_s	C++	21	Inteligência artificial: busca Monte Carlo em árvore (Go)
648.exchange2_s	Fortran	1	Inteligência artificial: gerador de soluções recursivas (Sudoku)
657.xz_s	C	33	Compressão geral de dados

*KLOC = número de linhas (incluindo comentários e espaços em branco) para os *source files* usados no *build* / 1000.

Fonte: adaptado de Standard Performance Evaluation Corporation (SPEC) (2024).

O desempenho dos sistemas computacionais para os problemas de *speed-int* é medido a partir da relação entre o tempo de execução em uma máquina padrão, definida pela SPEC, e o sistema sob teste como na equação 1.

$$P = \frac{T_{\text{ref}}}{T_{\text{SUT}}} \tag{1}$$

em que P é a pontuação (desempenho), T_{ref} é a duração de execução na máquina de referência e T_{sut} é a duração de execução no sistema sob teste.

Os resultados do SPEC são apresentados como médias geométricas dos testes realizados em cada sistema. Segundo Triola (2021, p. 103), a média geométrica (equação 2) é frequentemente utilizada nas áreas de negócios e economia para calcular a taxa média de mudança de uma variável; sendo, portanto, adequada em estudos de *benchmarking*.

$$\text{MG} = \left(\prod_{i=1}^{n} x_i \right)^{\frac{1}{n}} \tag{2}$$

O objetivo do presente trabalho é realizar uma análise estatística da base de dados do SPEC CPU 2017 para os resultados de testes de desempenho do tipo *speed-int*. Este propósito pode ser detalhado em objetivos específicos como a seguir:

- Calcular as estatísticas descritivas para as variáveis da base de dados e extrair informações pertinentes.
- Comparar os resultados de desempenho (pontuação) para os sistemas computacionais na condição em que são vendidos pelas fabricantes (*base*) com a condição de otimização da microarquitetura (*peak*).
- Avaliar a evolução temporal do desempenho (pontuação) desses sistemas para os principais fabricante/vendedores.
- Avaliar se existe alguma relação potencial entre a pontuação de desempenho e os elementos de arquitetura dos sistemas como tamanho das memórias cache (L1, L2, L3) e a frequência do processador.

Materiais e métodos

Toda a análise estatística conduzida neste trabalho foi realizada com a linguagem R utilizando o software RStudio® versão 2025.09.0+387 (Posit, 2025).

O dataset do SPECspeed Int 2017 é um arquivo .CSV constituído por 55 colunas e 9850 linhas. As variáveis incluem "Hardware vendor"(vendedor do computador/sistema), "System"(descrição do sistema), "# Cores"(número de núcleos), "#Chips"(número de chips), "Processador", dados das caches (L1, L2, L3), "Memory"(configuração da memória principal), "Storage"e as pontuações de desempenho para cada um dos *benchmarks* da tabela 2, nas condições *base* e *peak*. A tabela 3 apresenta um resumo das informações.

Tabela 3: Descrição das variáveis do dataset SPECspeed®2017 Integer.

Variável	**Tipo**	**Unidade**	**Descrição**
Benchmark	Categórica	–	Nome do `suite` individual do conjunto SPECspeed®2017 Integer
Hardware Vendor	Categórica	–	Identifica o fabricante/vendedor do hardware
System	Categórica	–	Contém uma descrição do sistema segundo o vendedor
# Cores	Quantitativa discreta	unidades	Número de núcleos (*cores*) do CPU
# Chips	Quantitativa discreta	unidades	Número de chips do processador
# Enabled Threads Per Core	Quantativa discreta	unidades	Número de tarefas por núcleo

Continua na próxima página

Tabela 3 – continuação da página anterior

Variável	Tipo	Unidade	Descrição
Processador (MHz)	Quantitativa continua	MHz	Frequência de operação do processador
CPU(s) Orderable	Qualitativa ordinal	unidades	Indica o número de chips ordenáveis do CPU
Parallel	Categórica nominal	Yes/No	Informa se a execução paralela é permitida
Base Pointer Size	Quantativa discreta	bits	Refere-se ao tamanho do ponteiro `base`
Peak Pointer Size	Quantativa discreta	bits	Refere-se ao tamanho do ponteiro `peak`
1st Level Cache	Quantitativa discreta	kB	Tamanho da memória cache L1
2nd Level Cache	Quantitativa discreta	kB	Tamanho da memória cache L2
3rd Level Cache	Quantitativa discreta	kB	Tamanho da memória cache L3
Other Cache	Categórica nominal	–	Informa a presença de outra memória cache
Memory	Quantitativa contínua	GB	Tamanho da memória principal
Storage	Quantitativa contínua	GB	Tamanho da unidade de armazenamento não volátil
Operating System	Categórica nominal	–	Descrição do sistema operacional
File System	Categórica nominal	–	Sistema de arquivos
Compilar	Categórica	–	Informa o(s) compilador(es) usado(s)
HW Avail	Categórica	data	Informa a data em que o hardware foi disponibilizado
SW Avail	Categórica	data	Informa a data em que o software foi disponibilizado
Result	Quantitativa contínua	–	Média geométrica de todos os valores de desempenho
Baseline	Quantitativa contínua	–	Média geométrica de todos os valores de desempenho para a condição `base`
600, 602, 605, 620, 623, 625, 631, 641, 648, 657 Base	Quantativa contínua	–	Pontuação de desempenho para cada benchmark na condição `base`
600, 602, 605, 620, 623, 625, 631, 641, 648, 657 Peak	Quantativa contínua	–	Pontuação de desempenho para cada benchmark na condição `peak`

Continua na próxima página

Tabela 3 – continuação da página anterior

Variável	Tipo	Unidade	Descrição
License	Categórica	–	Número/código da licença
Tested by	Categórica	–	Nome da instituição responsável pela execução dos testes
Test Sponsor	Categórica	–	Nome da instituição que patrocinou a execução dos testes
Test Date	Quantitativa discreta	data	Data em que os testes foram realizados
Published	Quantitativa discreta	data	Data em que os testes foram publicados
Updated	Quantitativa discreta	data	Data em que os testes foram atualizados
Disclosure	Categórica	–	Informa o endereço web para consulta

Por simplicidade, os títulos das variáveis da tabela 2 foram abreviados apenas pelo número 9código) indexador, aterior ao ponto de cada string. Essa decisão foi tomada, pois não existe risco de erro, devido a unicidade de cada índice.

Duas variáveis são derivadas das pontuações de desempenho, "Baseline"e "Result". A primeira calcula a média geométrica de todas as pontuações da mesma linha, independentemente do *benchmark*. A segunda calcula a média geométrica de todas as pontuações, incluindo *base* e *peak*.

Os dados ausentes (*missing data*) são representados por *default* como 0.0. Como a variável "Result" possui um escopo mais abrangente, foi utilizada como parâmetro para determinar quais linhas seriam excluídas da base de dados. Esse processo previne a ocorrência de vises não desejados, como a redução das estimativas para variáveis que possuem muitos valores nulos.

Após a remoção dos valores nulos, ainda restaram 6966 instâncias, população mais que suficiente para permitir inferências em relação à população, como propõe Magalhães e Lima (2025, p. 246). Como o novo dataset foram calculadas as medidas-resumo para atender ao objetivo específico 1. Os valores foram calculados com duas casas decimais, respeitando o limite de algarismos significativos reportados (KOUNEV; LANGE; KISTOWSKI, 2020).

Em seguida foram gerados os gráficos de box-plot para as variáveis de desempenho nas condições `base` e `peak`. Também foram produzidos gráficos que mostram a evolução do desempenho (pontuação) ao longo do tempo. Por fim, foram gerados gráficos de dispersão para analisar potenciais relações entre o desempeho e os componentes da arquitetura dos sistemas como memória cache e processador. O script em R completo utilizado pode ser consultado no Anexo.

O impacto produzido pela otimização da microarquitetura no desempenho desses sistemas pode ser calculado supondo que o tempo de duração da execução do algoritmo na máquina de referência seja o mesmo para ambas as condições (`base` e `peak`), então, a relação entre as duas durações pode ser obtida pela equação 3.

$$\frac{T_{SUT}^{O}}{T_{SUT}} = \frac{P}{P_O} \tag{3}$$

em que T_{SUT}^{O} é a duração da execução do algoritmo `i` para o sistema otimizado e P_o é a pontuação de desempenho respectiva.

Resultados e discussões

A tabela 4 resume as estatísticas descritivas e medidas de posição para as variáveis do estudo. Os resultados mostram que não existe um comportamento padrão para as variáveis. Por isso, faz-se necessário analisar caso a caso.

Os valores de média e mediana são próximos, em qualquer *benchmark* para as condições `base` e `peak`, com algumas exceções como '600', '623' e '625', onde ocorre diferença absoluta de até 0.96 (medianas '600').

Em geral, a proximidade entre média e mediana sugere que a média é também representativa da posição da distribuição dos dados. A varância e o desvio-padrão são maiores para *benchmarks* cujas média e mediana são também maiores quando comparadas às outras variáveis de desempenho, como ocorre para '605', '623', '625', '648' e '657'.

Tabela 4: Estatísticas descritivas dos benchmarks do SPECspeed®®2017 Integer.

Benchmark	Média	Mediana	Variância	Desvio-padrão	Q25	Q75	Mín	Máx
600 base	7.66	7.10	4.48	2.12	6.21	9.37	1.34	14.90
600 peak	8.46	8.06	4.21	2.05	7.35	10.10	1.63	15.50
602 base	11.32	10.80	7.70	2.78	9.40	12.70	2.44	22.10
602 peak	11.63	11.30	7.76	2.79	9.60	13.30	2.55	22.10
605 base	18.24	19.50	38.26	6.19	12.00	22.80	4.40	36.40
605 peak	18.44	19.50	39.91	6.32	12.00	22.90	4.40	38.20
620 base	8.94	8.87	8.92	2.99	6.44	11.50	2.32	22.40
620 peak	8.98	8.92	8.75	2.96	6.50	11.50	2.32	22.40
623 base	15.54	13.50	48.68	6.98	9.72	19.00	2.82	35.20
623 peak	16.03	13.60	51.39	7.17	10.30	20.20	2.82	38.90
625 base	17.54	16.90	34.50	5.87	12.30	22.00	3.44	37.60
625 peak	17.93	17.40	37.34	6.11	12.50	22.80	3.56	37.80
631 base	6.22	6.04	1.82	1.35	5.17	7.24	1.27	10.80
631 peak	6.22	6.04	1.82	1.35	5.15	7.24	1.27	10.80
641 base	5.18	4.99	1.17	1.08	4.34	5.96	0.98	8.78
641 peak	5.18	4.99	1.18	1.09	4.35	5.96	0.98	8.78
648 base	20.69	19.40	54.73	7.40	14.10	26.00	3.45	47.10
648 peak	20.68	19.40	55.27	7.43	14.10	26.00	3.45	47.20
657 base	22.30	23.20	27.87	5.28	20.20	25.80	2.42	38.00
657 peak	22.43	23.20	27.60	5.25	20.30	25.80	2.42	39.10
Processador (MHz)	2622.50	2500.00	3.37e+05	5.80e+02	2100.00	3000.00	1700.00	4700.00
Cache L1 (kB)	33.12	32.00	3.46e+01	5.88	32.00	32.00	32.00	64.00
Cache L2 (kB)	1126.57	1024.00	3.33e+05	5.77e+02	1024.00	2048.00	256.00	4096.00
Cache L3 (kB)	97639.45	38400.00	1.96e+10	1.40e+05	22528.00	107520.00	2048.00	1179648.00

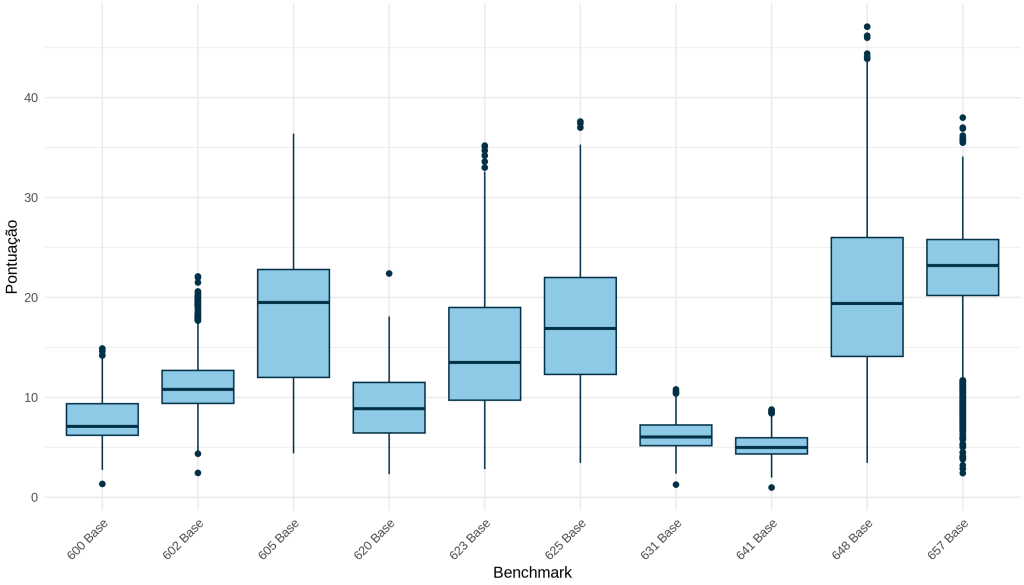
Fonte: os autores

O coeficiente de variação (c.v.) varia desde 21,7% para '631' até 44,9% para '623' e ainda de

18% para a ’Cache L1’ até 143% para a ’Cache L3’. Para além de esprimir como a amplitude dos dados é significativa, esse resultado indica que existe uma diferença significativa no desempenho das máquinas de diferentes fabricantes/vendedores em relação à tarefas unitárias com um único núcleo (*core*) quando executados algoritmos que utilizam números inteiros (`int`).

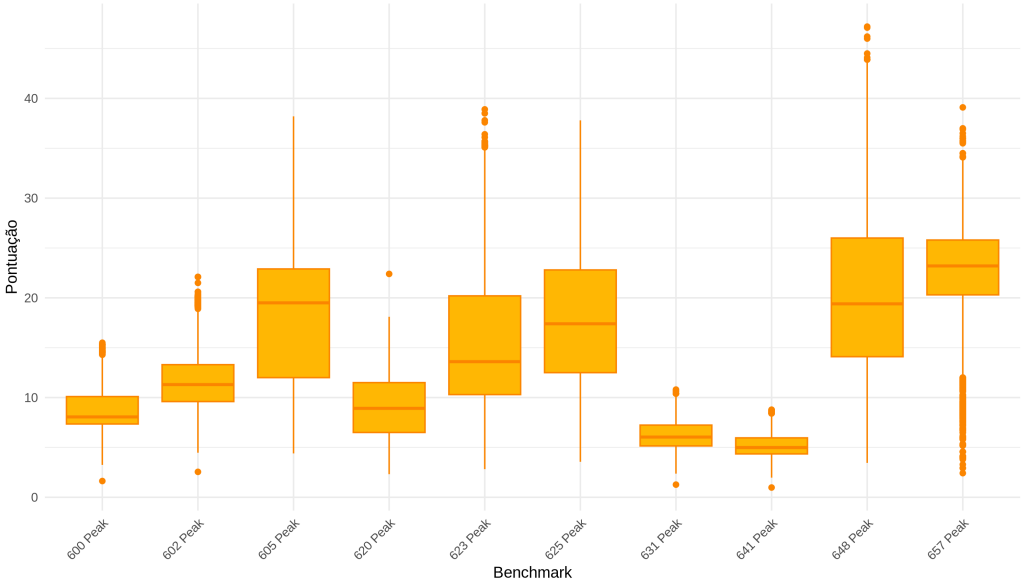
As figuras 1 e 2 corroboram graficamente as observações apresentas anteriormente. É fácil verificar como o comportamento das variáveis espelha-se entre `base` e `peak`. Como esperado, a partir dos resultados da tabela 4, as variáveis ’631’ e ’641’ apresentam a menor amplitude de variação também nos gráficos.

Figura 1: Comparação do desempenho dos *benchmarks* na condição *base*.



Fonte: os autores.

Figura 2: Comparação do desempenho dos *benchmarks* na condição *peak*.



Fonte: os autores.

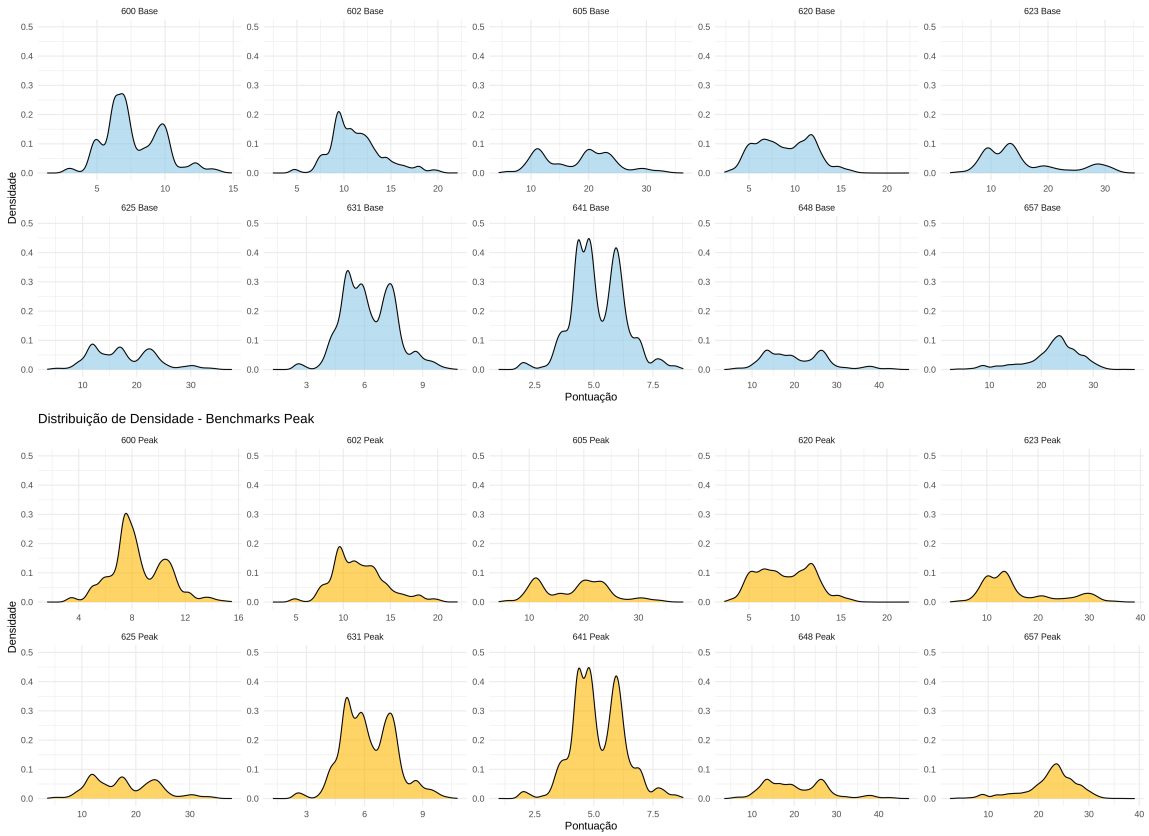
Este último caso possivelmente deve-se a flexibilidade da técnica de Monte Carlo para adaptar-se rapidamente em jogos, conforme pontuam Jung e Hoey (2025). Considerando que os fabricantes ofereceram bons hardwares, pode-se supor que a arquitetura possui um papel menor na execução desse tipo de *benchmark*.

Assumindo que os valores de mediana são representativos das distribuições, verifica-se que não houve ganho no tempo de execução para as variáveis ’605’ e ’631’ a ’657’ (em ordem). O desempenho otimizado é inferior a 5% para ’602’, inferior a 3% para ’625’ e inferior a 1% para ’620’ e ’623’, mas aproximadamente 12% para ’600’.

Foge ao escopo deste trabalho uma análise mais aprofundada das configurações de otimização da microarquitetura, porém, os resultados acima apontam que o impacto é muito pequeno ou até desprezível para a maioria dos *benchmarks*.

As curvas de densidade para as variáveis do estudo são apresentadas na figura 3 e corroboram as inferências realizadas até o momento. Conforme já comentado, não existe uma tendência global, porém, os resultados para `peak` praticamente mimetizam o comportamento das `bases`.

Figura 3: Curvas de densidade para as variáveis do estudo.

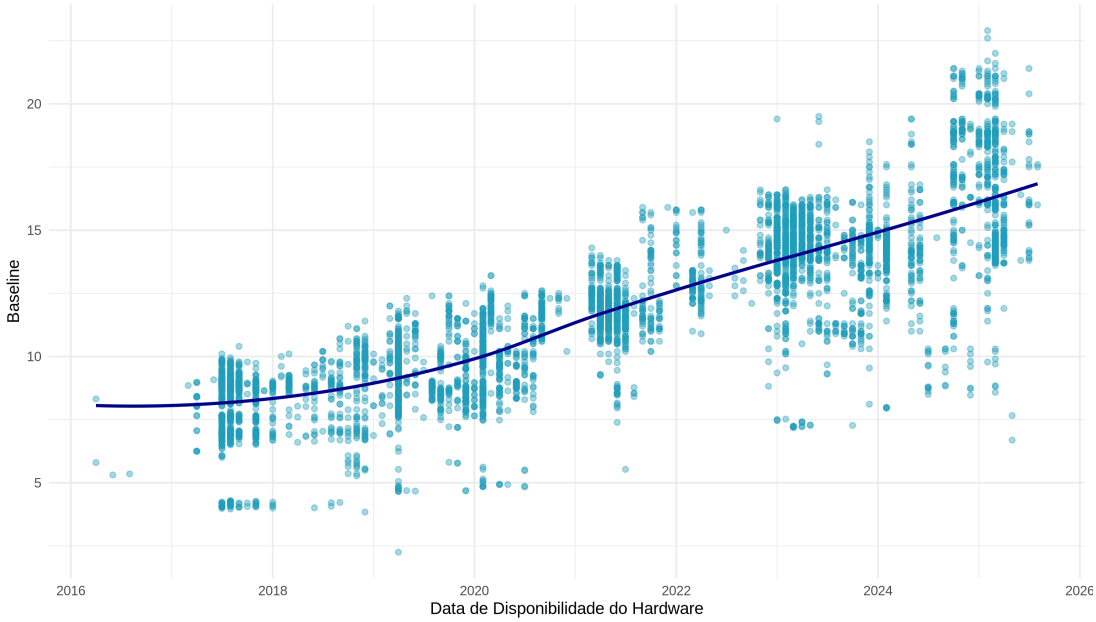


Fonte: os autores.

A figura 4 ilustra a evolução da variável `baseline`, que corresponde ao desempenho médio dos *benchmarks*, ao longo dos anos desde 2016 até 2025 (o último dado é do 3º trimestre de 2025). A linha em azul escuro serve como uma guia e auxilia na identificação de uma tendência crescente. Embora haja uma dispersão significativa a cada ano, representada pelos pontos azuis empilhados, a tendência predominante é de aumento. Esse fenômeno é esperado, sobretudo, em virtude da

evolução contínua dos sistemas computacionais e da redução dos custos com componentes de arquitetura.

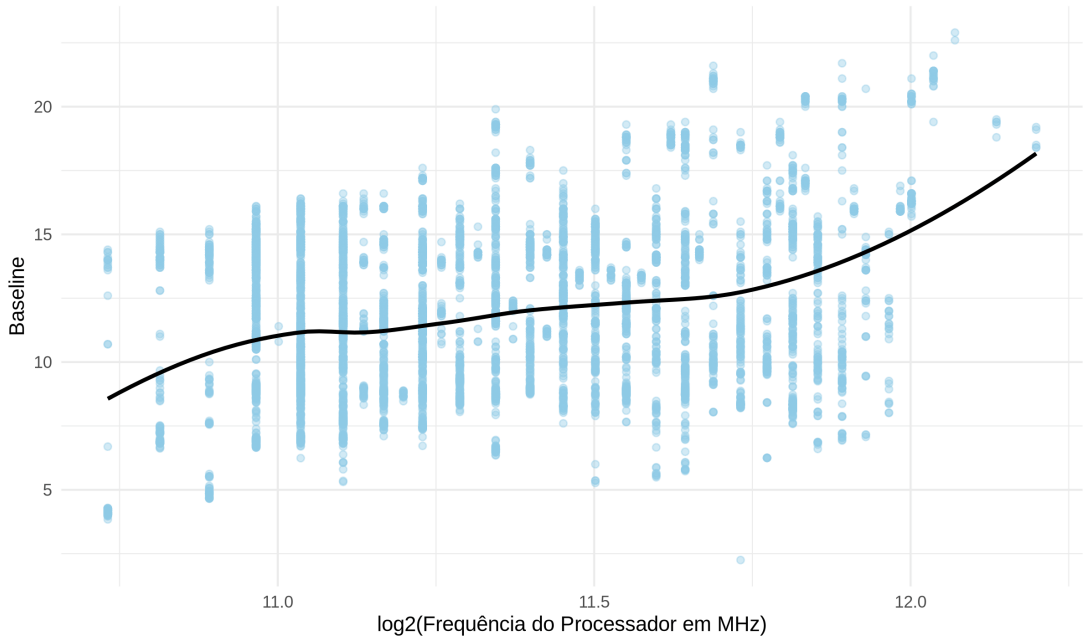
Figura 4: Curvas de densidade para as variáveis do estudo.



Fonte: os autores.

A partir da figura 5 é possível notar que a pontuação do `Baseline` aumenta com a frequência do processador. A frequência representa quantos ciclos de clock a CPU é capaz de executar por segundo. Logo, o resultado do gráfico é esperado, entretanto, cabe ressaltar que o comportamento não é linear. Na faixa de 2 GHz a 3.4 GHz, a pontuação do `Baseline` apresenta um aumento bastante pequeno (1.0 a 1.5 pontos), contudo, a partir deste último ponto a os dados demonstram um tendência crescente intensa (> 5.0 pontos).

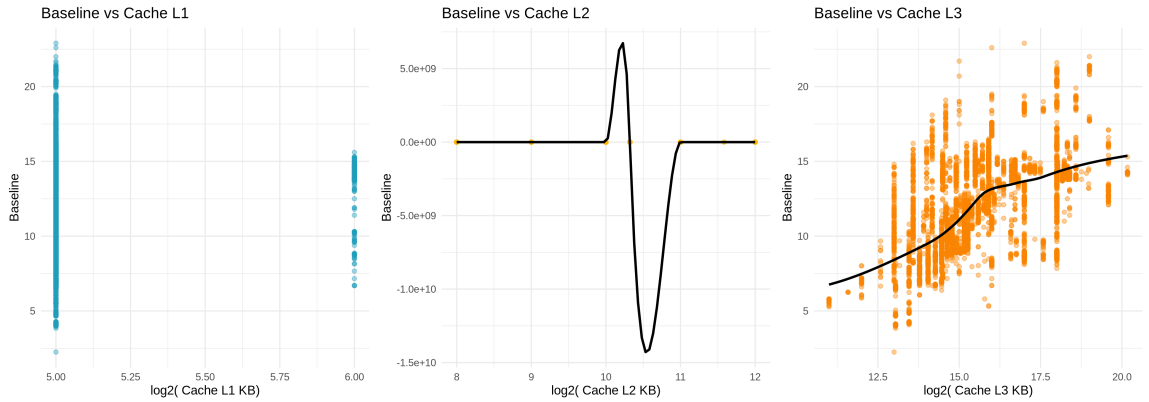
Figura 5: Desempenho x processador.



Fonte: os autores.

Hennessy e Patterson (2017) discutem que outros fatores importantes devem ser considerados junto com a frequência como memória e tamanho da cache. Como pode ser visto na figura 6 as memórias caches L1 e L2 utilizadas possuem pouca amplitude de valores, sobretudo, quando comparadas com a cache L3. Assim, é difícil extrair algumas tendência a partir das L1 e L2. A cache L3, no entanto, parece impactar o desempenho dos `benchmarks`, especialmente na faixa entre 2 MB e 75 MB. Stallings (2021) discute que investir numa cache L2 maior costuma gerar melhores resultados do que na L1 ou na L3, todavia, para as arquiteturas utilizadas nos testes com `speed-int` só é possível inferir mais claramente com respeito à L3 e seu efeito é positivo.

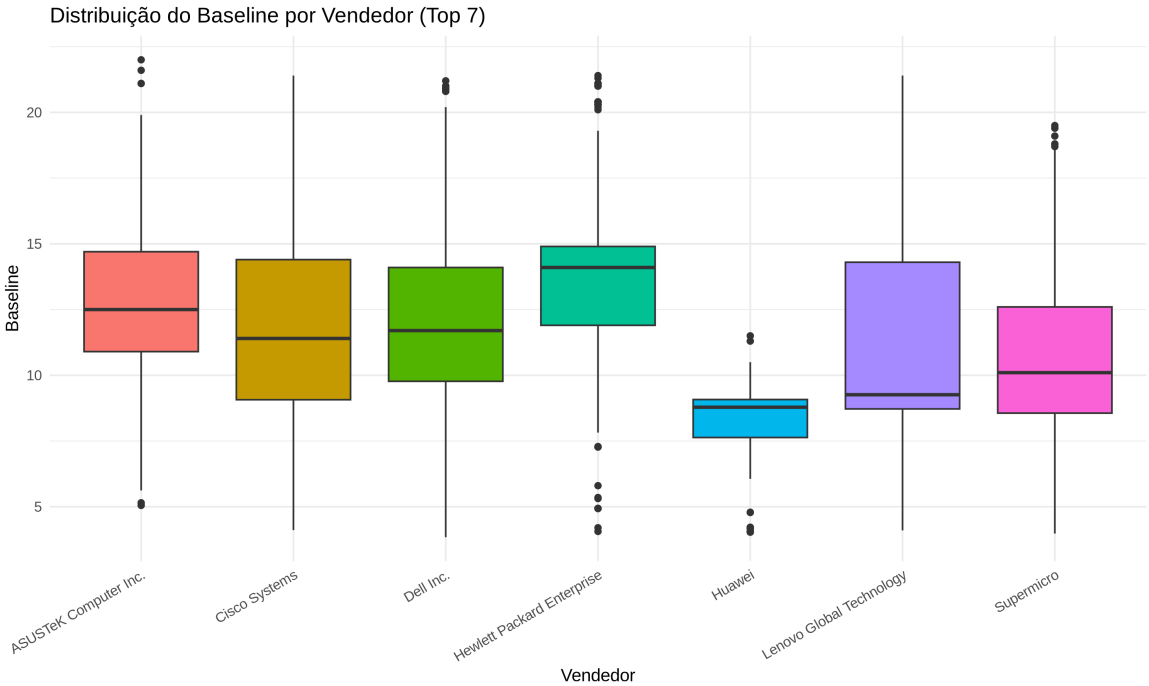
Figura 6: Desempenho x capacidade das memórias cache.



Fonte: os autores.

O dataset estudado é composto por algumas dezenas de vendedores. Dessa forma, foi gerada uma lista desses vendores em ordem descrescente de hardwares fornecidos e selecionados os 07 primeiros para analisar seu desempenho global. Os resultados constam na figura 7.

Figura 7: Comparação do desempenho por fabricante.



Fonte: os autores.

Os servidores disponibilizados pela Hewlett Packard (HP) destacam-se pela proximidade da mediana em relação ao Q75 e também pelas pontuações mais elevadas, ficando a frente de outras fabricantes famosas como ASUS, Cisco e Dell. Na outra ponta, a Huaweii mostra que ainda possui um horizonte de crescimento acentuado em relação às demais empresas.

Considerações finais

Neste trabalho foi realizada uma análise estatística para o dataset da SPEC CPU 2017 no que diz respeito aos algoritmos de `speed-int`. Foram calculadas estatísticas descritivas e os dados de pontuação de desempenho foram analisados em conjunto com dados da arquitetura dos sistemas testados.

Os resultados revelaram que a média é uma boa medida da tendência central para as variáveis analisadas, porém, os valores de pontuação oscilaram em até 44,9%, o que demonstra uma disparidade acentuada entre o desempenho para diferentes hardwares. Além disso, existe pouca diferença entre os resultados para as condições `base` e `peak`.

A análise do ganho de desempenho com a otimização da microarquitetura dos sistemas computacionais mostrou que o aprimoramento é muito pequeno ou desprezível para a maior parte dos algoritmos testados, apresentando ganho real apenas para a execução do compilador Perl (`benchmark` '600').

O desempenho dos sistemas tende a aumenta com o acréscimo na frequência do processador, sobretudo, valores maiores do 3.4 GHz, e também para memórias caches maiores, com destaque para a memória cache L3, mas ainda assim, menos expressivo acima de 75 MB.

Existe uma disparidade entre os resultados dos principais fabricantes/vendedores dos sistemas computacionais fornecidos para os testes que deram origem ao dataset. A Hewlett Packard ocupa posição de destaque, a frente de outras competidoras globais também bem estabelecidas, enquanto a Huaweii ocupa a última posição entre os sete maiores vendedores.

Referências bibliográficas

Referências

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*. 6. ed. Amsterdam, Netherlands: Morgan Kaufmann, 2017.

JUNG, J. D. A.; HOEY, J. Extending heuristic knowledge transfer for general game playing. *IEEE Transactions on Games*, IEEE, p. 1–12, 2025.

KOUNEV, S.; LANGE, K.; KISTOWSKI, J. von. The spec cpu benchmark suite. In: *Systems Benchmarking*. [S.l.]: Springer, Cham, 2020, (Lecture Notes in Computer Science, v. 978). p. 231–250.

LIMAYE, A.; ADEGBIJA, T. A workload characterization of the spec cpu2017 benchmark suite. In: *Proceedings of the 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2018)*. Belfast, Northern Ireland, UK: IEEE, 2018. p. 149–158. Document IEEE 8366949.

MAGALHãES, M. N.; LIMA, A. C. P. *Noções de Probabilidade e Estatística*. 8. ed. São Paulo: edUSP, 2025.

Posit. *RStudio IDE – Open Source*. 2025. Disponível em: <https://posit.co/products/open-source/rstudio/>.

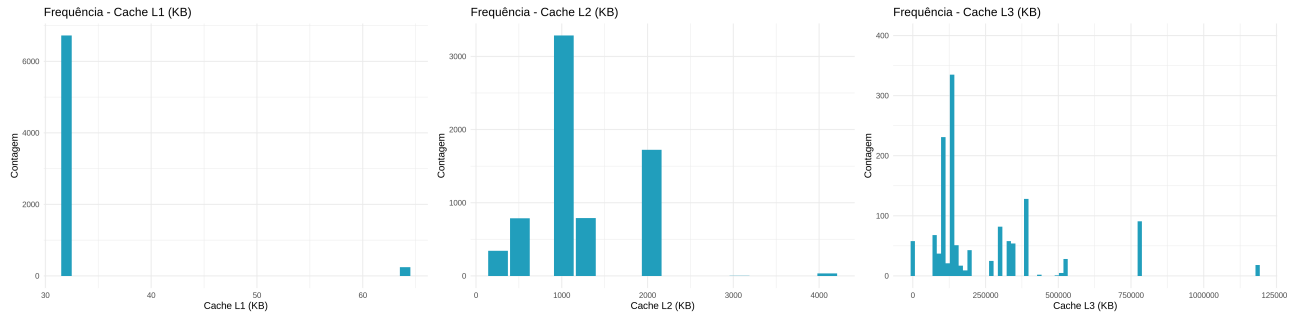
STALLINGS, W. *Computer Organization and Architecture: Designing for Performance*. 11. ed. Harlow, England: Pearson Education, 2021. Global Edition.

Standard Performance Evaluation Corporation (SPEC). *SPEC CPU®2017 Overview / What's New?* 2024. Disponível em: <https://www.spec.org/cpu2017/Docs/overview.html#Q1>.

TRIOLA, M. F. *Elementary Statistics*. 14. ed. Boston: Pearson, 2021.

Anexos

Figura 8: Histogramas das variáveis de cache (L1, L2 e L3).



Fonte: os autores.

```
1
2  ##################################################
3  #PREPARA  O DOS DADOS
4
5  #COLETANDO SOMENTE AS COLUNAS NECESSARIAS E LENDO O SPEED_int.csv
6
7  # Carregar pacotes necess rios
8  library(readr)
9  library(dplyr)
10 library(stringr)
11
12 # Ler o arquivo original
13 speed_raw <- read_csv("SPEED_int.csv")
14
15 # Verificar nomes das colunas
16 colnames(speed_raw)
17
18 # Selecionar colunas de interesse + removendo aqueles Baseline == 0
19 speed_clean <- speed_raw |>
20   filter(Baseline != 0, Result != 0) |>
21   filter(
22     if_all(
23       c(
24         "600 Peak", "602 Peak", "605 Peak", "620 Peak",
25         "623 Peak", "625 Peak", "631 Peak", "641 Peak",
26         "648 Peak", "657 Peak"
```

```r
      ),
      ~ .x != 0
    )
  ) |>

  # Fun  o auxiliar interna para extrair cache em KB
  mutate(
    cache_l1_kb = str_split(`1st Level Cache`, "\\s+") |>
      lapply(function(x) {
        valor <- suppressWarnings(as.numeric(x[[1]]))
        unidade <- toupper(x[[2]])
        if (is.na(valor) || is.na(unidade)) return(NA_real_)
        if (unidade == "MB") return(valor * 1024)
        if (unidade == "KB") return(valor)
        return(NA_real_)
      }) |> unlist(),

    cache_l2_kb = str_split(`2nd Level Cache`, "\\s+") |>
      lapply(function(x) {
        valor <- suppressWarnings(as.numeric(x[[1]]))
        unidade <- toupper(x[[2]])
        if (is.na(valor) || is.na(unidade)) return(NA_real_)
        if (unidade == "MB") return(valor * 1024)
        if (unidade == "KB") return(valor)
        return(NA_real_)
      }) |> unlist(),

    cache_l3_kb = str_split(`3rd Level Cache`, "\\s+") |>
      lapply(function(x) {
        valor <- suppressWarnings(as.numeric(x[[1]]))
        unidade <- toupper(x[[2]])
        if (is.na(valor) || is.na(unidade)) return(NA_real_)
        if (unidade == "MB") return(valor * 1024)
        if (unidade == "KB") return(valor)
        return(NA_real_)
      }) |> unlist()
  ) |>

  # Selecionar colunas finais
  select(
    `Hardware Vendor`, System, `# Cores`, `# Chips`, `# Enabled Threads Per Core`,
    Processor, `Processor MHz`, `CPU(s) Orderable`
    , Storage, `Operating System`, Compiler, `HW Avail`,
    Result, Baseline,
    `600 Peak`, `600 Base`, `602 Peak`, `602 Base`,
    `605 Peak`, `605 Base`, `620 Peak`, `620 Base`,
    `623 Peak`, `623 Base`, `625 Peak`, `625 Base`,
    `631 Peak`, `631 Base`, `641 Peak`, `641 Base`,
    `648 Peak`, `648 Base`, `657 Peak`, `657 Base`,
    `Tested By`, `Test Sponsor`,
    cache_l1_kb, cache_l2_kb, cache_l3_kb
  )

# CLASSIFICANDO AS VARI VEIS
```

```r
speed_clean <- speed_clean |>
  mutate(
    # --- Qualitativas nominais ---
    `Hardware Vendor` = as.factor(`Hardware Vendor`),
    System = as.factor(System),
    Processor = as.factor(Processor),
    `CPU(s) Orderable` = as.factor(`CPU(s) Orderable`),
    Storage = as.factor(Storage),
    `Operating System` = as.factor(`Operating System`),
    Compiler = as.factor(Compiler),
    `Tested By` = as.factor(`Tested By`),
    `Test Sponsor` = as.factor(`Test Sponsor`),

    # --- Qualitativas ordinais (datas) ---
    `HW Avail` = parse_date(`HW Avail`, format = "%b-%Y"),

    # --- Quantitativas discretas ---
    `# Cores` = as.integer(`# Cores`),
    `# Chips` = as.integer(`# Chips`),
    `# Enabled Threads Per Core` = as.integer(`# Enabled Threads Per Core`),

    # --- Quantitativas cont nuas (numeric) ---
    `Processor MHz` = as.numeric(`Processor MHz`),
    Result = as.numeric(Result),
    Baseline = as.numeric(Baseline),

    `600 Peak` = as.numeric(`600 Peak`),
    `600 Base` = as.numeric(`600 Base`),
    `602 Peak` = as.numeric(`602 Peak`),
    `602 Base` = as.numeric(`602 Base`),
    `605 Peak` = as.numeric(`605 Peak`),
    `605 Base` = as.numeric(`605 Base`),
    `620 Peak` = as.numeric(`620 Peak`),
    `620 Base` = as.numeric(`620 Base`),
    `623 Peak` = as.numeric(`623 Peak`),
    `623 Base` = as.numeric(`623 Base`),
    `625 Peak` = as.numeric(`625 Peak`),
    `625 Base` = as.numeric(`625 Base`),
    `631 Peak` = as.numeric(`631 Peak`),
    `631 Base` = as.numeric(`631 Base`),
    `641 Peak` = as.numeric(`641 Peak`),
    `641 Base` = as.numeric(`641 Base`),
    `648 Peak` = as.numeric(`648 Peak`),
    `648 Base` = as.numeric(`648 Base`),
    `657 Peak` = as.numeric(`657 Peak`),
    `657 Base` = as.numeric(`657 Base`),

    cache_l1_kb = as.numeric(cache_l1_kb),
    cache_l2_kb = as.numeric(cache_l2_kb),
    cache_l3_kb = as.numeric(cache_l3_kb)
  )

##testes
summary(speed_clean)
```

```r
glimpse(speed_clean)


################################################

#INICIO DA ANALISE

#CRIACAO E ESCRITA DA TABELA COM ESTATISTICAS DESCRITIVAS NA PASTA tabela_e_graficos

library(dplyr)
library(tidyr)

# Identificar colunas de benchmark (todas terminam com Base ou Peak e come am com 6)
bench_cols <- grep("^6\\d{2} (Base|Peak)$", names(speed_clean), value = TRUE)

# Juntar colunas de interesse: benchmarks + hardware
colunas_estatisticas <- c(
  grep("^6\\d{2} (Base|Peak)$", names(speed_clean), value = TRUE),
  "Processor MHz", "cache_l1_kb", "cache_l2_kb", "cache_l3_kb"
)

# Criar tabela descritiva
resumo_completo <- speed_clean |>
  select(all_of(colunas_estatisticas)) |>
  pivot_longer(cols = everything(), names_to = "variavel", values_to = "valor") |>
  group_by(variavel) |>
  summarise(
    media = mean(valor, na.rm = TRUE),
    mediana = median(valor, na.rm = TRUE),
    variancia = var(valor, na.rm = TRUE),
    desvio_padrao = sd(valor, na.rm = TRUE),
    q25 = quantile(valor, 0.25, na.rm = TRUE),
    q75 = quantile(valor, 0.75, na.rm = TRUE),
    minimo = min(valor, na.rm = TRUE),
    maximo = max(valor, na.rm = TRUE),
    .groups = "drop"
  ) |>
  arrange(variavel)

# Criar pasta se n o existir
if (!dir.exists("tabelas_e_graficos")) {
  dir.create("tabelas_e_graficos")
}

# Salvar a tabela de resumo como CSV
readr::write_csv(resumo_completo, "tabelas_e_graficos/resumo_completo.csv")

#-----------------------------------------------
# Criando dois gr ficos , cada gr fico mostrar  boxplots por benchmark

library(ggplot2)
library(tidyr)
library(dplyr)
library(readr)
```

```r

# Identificar colunas Base e Peak
col_base <- grep("^6\\d{2} Base$", names(speed_clean), value = TRUE)
col_peak <- grep("^6\\d{2} Peak$", names(speed_clean), value = TRUE)

# Preparar os dados em formato longo (long format)
dados_base <- speed_clean |>
  select(all_of(col_base)) |>
  pivot_longer(cols = everything(), names_to = "benchmark", values_to = "valor")

dados_peak <- speed_clean |>
  select(all_of(col_peak)) |>
  pivot_longer(cols = everything(), names_to = "benchmark", values_to = "valor")

# Gr fico Base
grafico_base <- ggplot(dados_base, aes(x = benchmark, y = valor)) +
  geom_boxplot(fill = "#8ecae6", color = "#023047") +
  labs(
    title = "Distribui  o dos Benchmarks - Condi  o Base",
    x = "Benchmark",
    y = "Pontua  o"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Gr fico Peak
grafico_peak <- ggplot(dados_peak, aes(x = benchmark, y = valor)) +
  geom_boxplot(fill = "#ffb703", color = "#fb8500") +
  labs(
    title = "Distribui  o dos Benchmarks - Condi  o Peak",
    x = "Benchmark",
    y = "Pontua  o"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Criar pasta se necess rio
if (!dir.exists("tabelas_e_graficos")) {
  dir.create("tabelas_e_graficos")
}

# Salvar os gr ficos como imagens
ggsave("tabelas_e_graficos/boxplot_base.png", grafico_base, width = 10, height = 6)
ggsave("tabelas_e_graficos/boxplot_peak.png", grafico_peak, width = 10, height = 6)

#------------------------------------------------
#relacionando desempenho do Baseline com frequ ncia, e n veis de cache

library(ggplot2)
library(patchwork) # Para juntar gr ficos

# Criar gr fico para cada vari vel explicativa
p1 <- ggplot(speed_clean, aes(x = `Processor MHz`, y = Baseline)) +
  geom_point(alpha = 0.5) +
```

```r
  geom_smooth(method = "lm", se = FALSE, color = "blue") +
  labs(title = "Baseline vs Processor MHz") +
  theme_minimal()

p2 <- ggplot(speed_clean, aes(x = cache_l1_kb, y = Baseline)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm", se = FALSE, color = "darkgreen") +
  labs(title = "Baseline vs Cache L1 (KB)") +
  theme_minimal()

p3 <- ggplot(speed_clean, aes(x = cache_l2_kb, y = Baseline)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm", se = FALSE, color = "purple") +
  labs(title = "Baseline vs Cache L2 (KB)") +
  theme_minimal()

p4 <- ggplot(speed_clean, aes(x = cache_l3_kb, y = Baseline)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  labs(title = "Baseline vs Cache L3 (KB)") +
  theme_minimal()

# Juntar os 4 gr ficos
(p1 | p2) / (p3 | p4)

ggsave("tabelas_e_graficos/baseline_vs_hardware.png", (p1 | p2) / (p3 | p4), width =
    12, height = 8)

library(tibble)
# Calcular a matriz de correla  o de Pearson
correlacao <- speed_clean |>
  select(Baseline, `Processor MHz`, cache_l1_kb, cache_l2_kb, cache_l3_kb) |>
  cor(use = "complete.obs", method = "pearson")

# Transformar a matriz em data frame para salvar
correlacao_df <- as.data.frame(correlacao) |>
  rownames_to_column(var = "Variavel")

# Garantir que a pasta exista
if (!dir.exists("tabelas_e_graficos")) {
  dir.create("tabelas_e_graficos")
}

# Salvar a matriz como CSV
readr::write_csv(correlacao_df, "tabelas_e_graficos/pearson_baseline_vs_cpu.csv")

#-----------------------------------------------
#adicionando:
#Gr fico 1: mediana do Baseline por semestre (HW Avail)
#Gr fico 2: gr fico de dispers o com Baseline ao longo do tempo

library(lubridate)

# Criar coluna de semestre
```

```r
speed_clean <- speed_clean |>
  mutate(semestre = paste0(year(`HW Avail`), "-S", if_else(month(`HW Avail`) <= 6, "1", "2")))

# Gr fico 1: Mediana por semestre
grafico_mediana_tempo <- speed_clean |>
  group_by(semestre) |>
  summarise(mediana_baseline = median(Baseline, na.rm = TRUE), .groups = "drop") |>
  ggplot(aes(x = semestre, y = mediana_baseline, group = 1)) +
  geom_line(color = "steelblue", linewidth = 1.2) +
  geom_point(color = "steelblue", size = 2) +
  labs(
    title = "Evolu  o da Mediana do Baseline por Semestre",
    x = "Semestre",
    y = "Mediana do Baseline"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Gr fico 2: Dispers o de Baseline ao longo do tempo
grafico_dispersao_tempo <- ggplot(speed_clean, aes(x = `HW Avail`, y = Baseline)) +
  geom_point(alpha = 0.4, color = "#219ebc") +
  geom_smooth(method = "loess", se = FALSE, color = "darkblue") +
  labs(
    title = "Dispers o do Baseline ao Longo do Tempo",
    x = "Data de Disponibilidade do Hardware",
    y = "Baseline"
  ) +
  theme_minimal()

# Salvar os gr ficos
ggsave("tabelas_e_graficos/baseline_mediana_semestre.png", grafico_mediana_tempo, width = 10, height = 6)
ggsave("tabelas_e_graficos/baseline_dispersao_tempo.png", grafico_dispersao_tempo, width = 10, height = 6)

#------------------------------------------------
#baseline vs cache

library(ggplot2)
library(patchwork)

# Fun  o para adicionar log2 no eixo x
cache_plot <- function(cache_var, cache_label, color) {
  ggplot(speed_clean, aes(x = log2(!!sym(cache_var)), y = Baseline)) +
    geom_point(alpha = 0.4, color = color) +
    geom_smooth(method = "loess", se = FALSE, color = "black") +
    labs(
      title = paste("Baseline vs", cache_label),
      x = paste("log2(", cache_label, "KB)"),
      y = "Baseline"
    ) +
    theme_minimal()
}
```

```r

# Criar os 3 gr ficos
g1 <- cache_plot("cache_l1_kb", "Cache L1", "#219ebc")
g2 <- cache_plot("cache_l2_kb", "Cache L2", "#ffb703")
g3 <- cache_plot("cache_l3_kb", "Cache L3", "#fb8500")

# Juntar os tr s gr ficos lado a lado
grafico_caches <- g1 | g2 | g3

# Mostrar e salvar
print(grafico_caches)
ggsave("tabelas_e_graficos/baseline_vs_caches_log2.png", grafico_caches, width = 14,
    height = 5)

#------------------------------------------------
# Gr fico: Baseline vs Processor MHz (com eixo log2)

grafico_frequencia <- ggplot(speed_clean, aes(x = log2(`Processor MHz`), y =
    Baseline)) +
  geom_point(alpha = 0.4, color = "#8ecae6") +
  geom_smooth(method = "loess", se = FALSE, color = "black") +
  labs(
    title = "Baseline vs Processor MHz (Escala log2)",
    x = "log2(Frequ ncia do Processador em MHz)",
    y = "Baseline"
  ) +
  theme_minimal()

# Exibir
print(grafico_frequencia)

# Salvar
ggsave("tabelas_e_graficos/baseline_vs_processor_mhz_log2.png", grafico_frequencia,
    width = 8, height = 5)

#------------------------------------------------
#quantidade de sistemas por Hardware Vendor
#Selecionar os top 7 vendedores
#Filtrar o speed_clean apenas com esses vendedores
#Calcular estat sticas descritivas do Baseline por vendedor
#Criar um boxplot comparativo para analisar varia  es

library(dplyr)
library(ggplot2)
library(readr)

# 1. Identificar os top 7 vendedores
top_vendedores <- speed_clean |>
  count(`Hardware Vendor`, sort = TRUE) |>
  slice_head(n = 7) |>
  pull(`Hardware Vendor`)

# 2. Filtrar dados apenas com os top vendedores
dados_vendedores <- speed_clean |>
```

```r
  filter(`Hardware Vendor` %in% top_vendedores)

# 3. Calcular estat sticas descritivas por vendedor
estatisticas_vendedores <- dados_vendedores |>
  group_by(`Hardware Vendor`) |>
  summarise(
    media = mean(Baseline, na.rm = TRUE),
    mediana = median(Baseline, na.rm = TRUE),
    desvio_padrao = sd(Baseline, na.rm = TRUE),
    q25 = quantile(Baseline, 0.25, na.rm = TRUE),
    q75 = quantile(Baseline, 0.75, na.rm = TRUE),
    minimo = min(Baseline, na.rm = TRUE),
    maximo = max(Baseline, na.rm = TRUE),
    n = n(),
    .groups = "drop"
  )

# 4. Salvar a tabela
readr::write_csv(estatisticas_vendedores,
    "tabelas_e_graficos/estatisticas_top_vendedores.csv")

# 5. Criar boxplot do Baseline por vendedor
grafico_vendedores <- ggplot(dados_vendedores, aes(x = `Hardware Vendor`, y =
    Baseline, fill = `Hardware Vendor`)) +
  geom_boxplot(show.legend = FALSE) +
  labs(
    title = "Distribui  o do Baseline por Vendedor (Top 7)",
    x = "Vendedor",
    y = "Baseline"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))

# Exibir e salvar
print(grafico_vendedores)
ggsave("tabelas_e_graficos/boxplot_baseline_por_vendedor.png", grafico_vendedores,
    width = 10, height = 6)

#------------------------------------------------
# analise de densidade, peak e base, para cada benchmark

library(ggplot2)
library(tidyr)
library(patchwork)

# 1. Selecionar colunas
bench_cols_base <- grep("^6\\d{2} Base$", names(speed_clean), value = TRUE)
bench_cols_peak <- grep("^6\\d{2} Peak$", names(speed_clean), value = TRUE)

# 2. Long format
dados_base <- speed_clean |>
  select(all_of(bench_cols_base)) |>
  pivot_longer(everything(), names_to = "benchmark", values_to = "valor") |>
  mutate(tipo = "Base")
```

```r

dados_peak <- speed_clean |>
  select(all_of(bench_cols_peak)) |>
  pivot_longer(everything(), names_to = "benchmark", values_to = "valor") |>
  mutate(tipo = "Peak")

# 3. Juntar
dados_benchmarks <- bind_rows(dados_base, dados_peak)

# 4. Fun  o para gr fico de densidade individual
plot_density <- function(df, tipo_filtro, y_lim = 0.5) {
  df |>
    filter(tipo == tipo_filtro) |>
    ggplot(aes(x = valor)) +
    geom_density(fill = ifelse(tipo_filtro == "Base", "#8ecae6", "#ffb703"), alpha =
        0.6) +
    facet_wrap(~benchmark, scales = "free", ncol = 5) +
    labs(
      title = paste("Distribui  o de Densidade - Benchmarks", tipo_filtro),
      x = "Pontua  o",
      y = "Densidade"
    ) +
    coord_cartesian(ylim = c(0, y_lim)) +  #      substitui ylim()
    theme_minimal()
}

# 5. Criar os dois pain is
plot_base <- plot_density(dados_benchmarks, "Base")
plot_peak <- plot_density(dados_benchmarks, "Peak")

# 6. Juntar (4 linhas: 2 base + 2 peak)
painel_densidade <- plot_base / plot_peak

# 7. Salvar
ggsave("tabelas_e_graficos/densidade_benchmarks_base_peak.png", painel_densidade,
    width = 16, height = 12)

#------------------------------------------------
# histogramas (em gr ficos de barra) para as vari veis de cache
library(ggplot2)
library(patchwork)

# Histograma L1 - Barras finas
hist_l1 <- ggplot(speed_clean, aes(x = cache_l1_kb)) +
  geom_bar(fill = "#219ebc", width = 1) +
  labs(title = "Frequ ncia - Cache L1 (KB)", x = "Cache L1 (KB)", y = "Contagem") +
  theme_minimal()

# Histograma L2 - Padr o
hist_l2 <- ggplot(speed_clean, aes(x = cache_l2_kb)) +
  geom_bar(fill = "#219ebc") +
  labs(title = "Frequ ncia - Cache L2 (KB)", x = "Cache L2 (KB)", y = "Contagem") +
  theme_minimal()

```

```r
# Histograma L3 - Base mais larga e barras mais grossas, limite do eixo y
hist_l3 <- ggplot(speed_clean, aes(x = cache_l3_kb)) +
  geom_histogram(binwidth = 15000, fill = "#219ebc", width = 150000) +
  ylim(0, 400) +
  labs(title = "Frequ ncia - Cache L3 (KB)", x = "Cache L3 (KB)", y = "Contagem") +
  theme_minimal()

# Juntar e salvar
(hist_l1 | hist_l2 | hist_l3)

ggsave("tabelas_e_graficos/histogramas_cache_final.png", (hist_l1 | hist_l2 |
    hist_l3), width = 20, height = 5)
```