



# **Trabalho Prático 2 - Organização de Computadores**

Eduarda Tessari, Gabriel Tavares,  
Ian Kersz, Nathan Guimarães e  
Maximus Borges



# Índice

- 1. Algoritmos escolhidos**
- 2. Configuração fixa**
- 3. Parâmetros a serem modificados**
- 4. Resultados e Conclusões IPC**
- 5. Resultados e Conclusões Tempo**
- 6. Resultados e Conclusões Cache (extra)**
- 7. Resultados e Conclusões Média IPC**



# 1. Algoritmos Escolhidos



## Primality Test [1]

Algoritmo verifica se um número é primo, testando divisibilidade até a raiz quadrada do número.

## SHA-256 [2]

Algoritmo de hash criptográfico que gera um valor de 256 bits garantindo integridade e segurança dos dados. Ele é amplamente usado em várias aplicações de segurança digital, incluindo assinaturas digitais e certificação de dados.

## Kruskal's Algorithm [3]

Algoritmo para encontrar a árvore geradora mínima de um grafo, adicionando iterativamente as arestas de menor peso que não formam um ciclo.

## 2. Entrada dos algoritmos

### Primality Test

Em um loop que vai de 0 a 40000, testa a “primalidade” de todos os números no intervalo.


### SHA-256

Utilizou-se uma string de tamanho 64kB. Como cada caractere ocupa exatamente 1 byte em C, a string contém 64 mil caracteres ( $64 * 1024 = 64\text{kB}$ ). Foram feitas 15 iterações do algoritmo na mesma string.

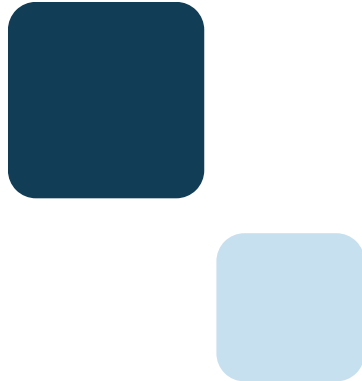
### Kruskal's Algorithm

Para a entrada desse algoritmo, foi determinado quais seriam as dimensões necessárias para a matriz de inteiros ocupar 64kB. Foram feitas 360 iterações do algoritmo na mesma matriz.

```
#define SIZE_IN_KB 64
#define size_in_bytes (SIZE_IN_KB * 1024)
#define num_elements (size_in_bytes / 4)
#define RUNS 360
// Calcula o tamanho da matriz (n x n)
int n = (int)sqrt(num_elements);
```



# 3. Configuração Fixa



## Quantidade de Unidades de Multiplicação e Divisão de Inteiros

Aumento de 1 para 2: como essa unidade é uma parte do processador amplamente utilizada pelos algoritmos, aumentar o número de unidades aproveitaria a superescalaridade do processador.

## Quantidade de Registradores de Número Inteiro

Aumento de 96 para 128: a maioria dos algoritmos manipula números inteiros, tornando vantajoso aumentar o número de registradores.

## Tamanho da Cache L2

Aumento de 256kB para 512kB: queremos garantir que os dados do programa estejam somente na L1 e, em caso de overflow, que estejam no máximo na L2.

## 4. Parâmetros a serem modificados

### Tamanho Cache L1 de dados

A modificação da cache L1 foi pensada especificamente para o algoritmo de Kruskal, que realiza vários acessos à memória (acessos à matriz). Ao aumentar o tamanho da cache, espera-se uma redução no tempo de execução; ao diminuí-la, o tempo de execução deve aumentar.

### Latência IntMult

A modificação deste parâmetro foi pensada em função do algoritmo SHA256. Com base no comportamento do algoritmo, espera-se que a alteração da latência das operações da ULA tenha um impacto significativo no tempo de execução do SHA256.

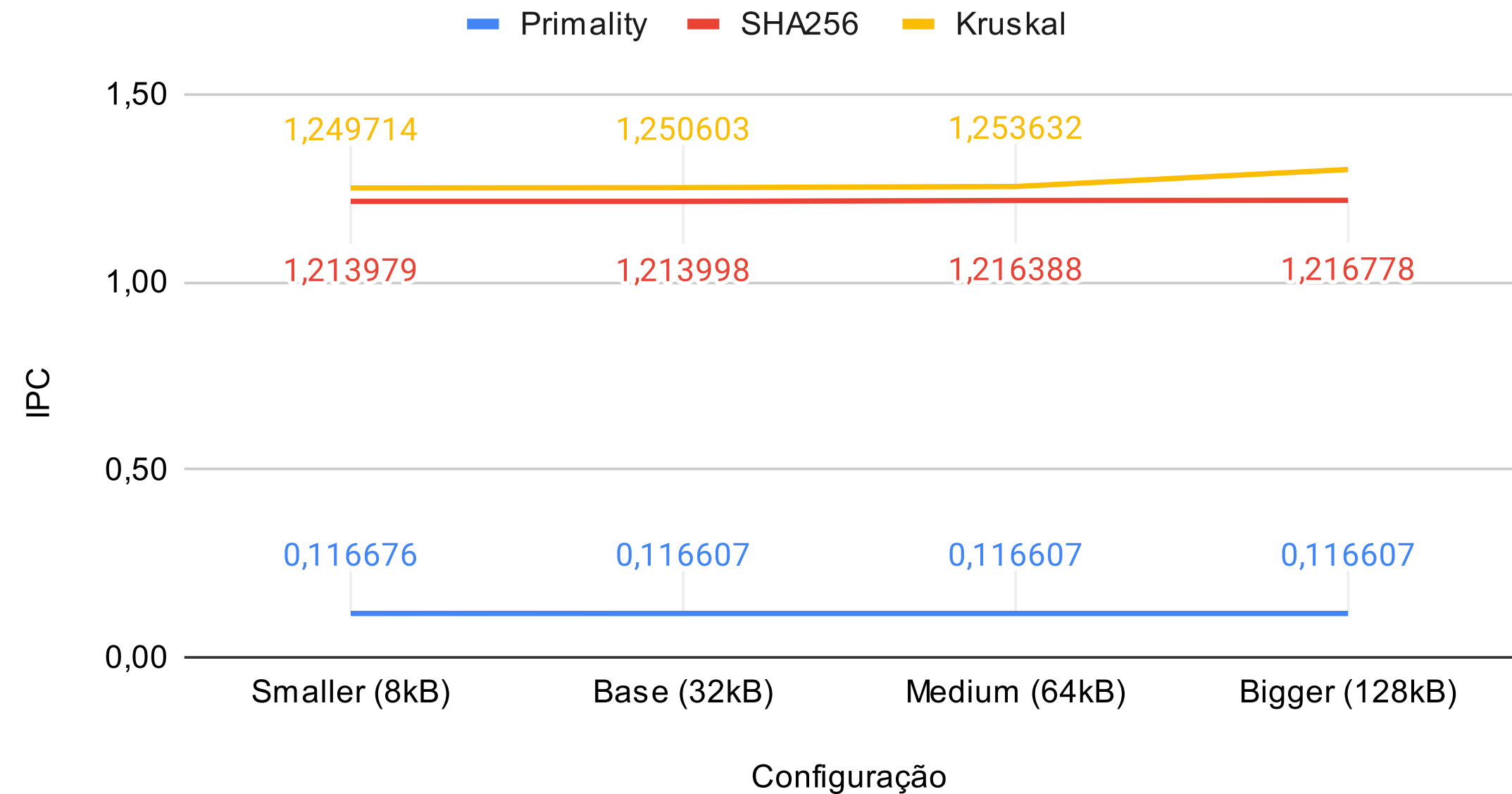
### Latência do SimdFloatSqrt

A alteração deste parâmetro foi planejada com base no algoritmo de Primality Test, pois o código compilado utiliza a função Sqrt no formato SIMD. Aumentar a latência do SimdFloatSqrt deve aumentar o tempo de execução do algoritmo e reduzir o IPC (instruções por ciclo), enquanto diminuir a latência deve ter o efeito contrário.

## 4. Conclusões IPC

- O algoritmo de Kruskal apresenta um leve aumento no IPC à medida que o tamanho da cache L1 de dados aumenta. Isso ocorre porque ele realiza um número significativo de acessos à memória, e uma cache maior permite reduzir esses acessos à memória principal, resultando em uma melhora de desempenho.
- Por outro lado, os algoritmos *Primality* e *SHA256* não sobrescrevem os dados da cache de forma tão intensiva. Por isso, a modificação no tamanho da cache não impacta significativamente o desempenho desses algoritmos.

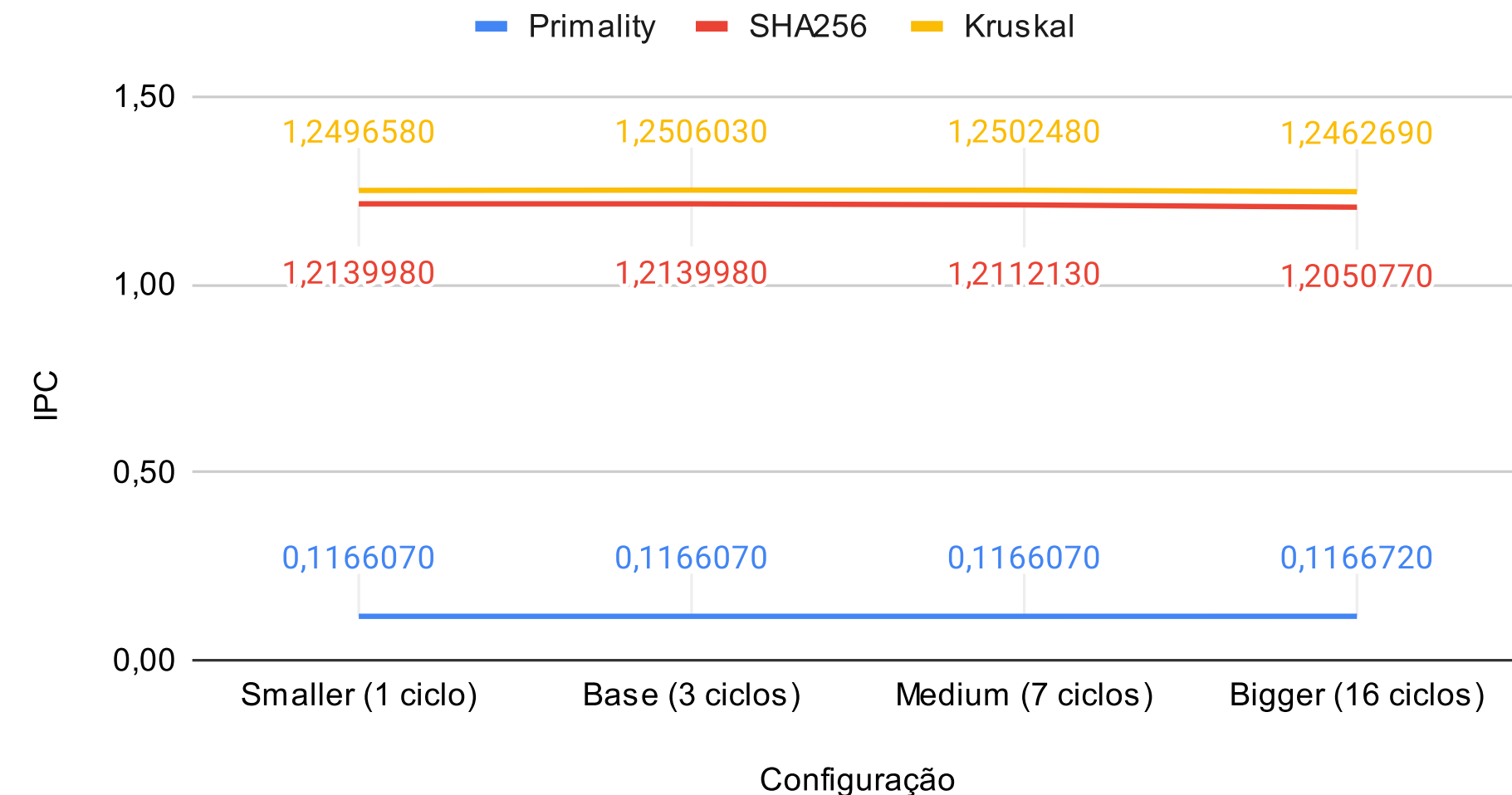
### IPC - Cache L1 Dados



## 4. Conclusões IPC

- A modificação desse parâmetro não teve o impacto significativo que esperávamos, mas ainda assim causou pequenos efeitos nos algoritmos *SHA256* e *Kruskal*, resultando em uma leve queda no IPC à medida que a latência aumentava. Por conta do pipeline interno nesta instrução, acreditamos que sua latência maior esteja sendo compensada, minimizando o impacto total dessa modificação.
- No caso do algoritmo de *Primality*, como ele não faz uso de multiplicações inteiras, já prevíamos que essa mudança não afetaria o IPC.

IPC - Latência da Instrução IntMult

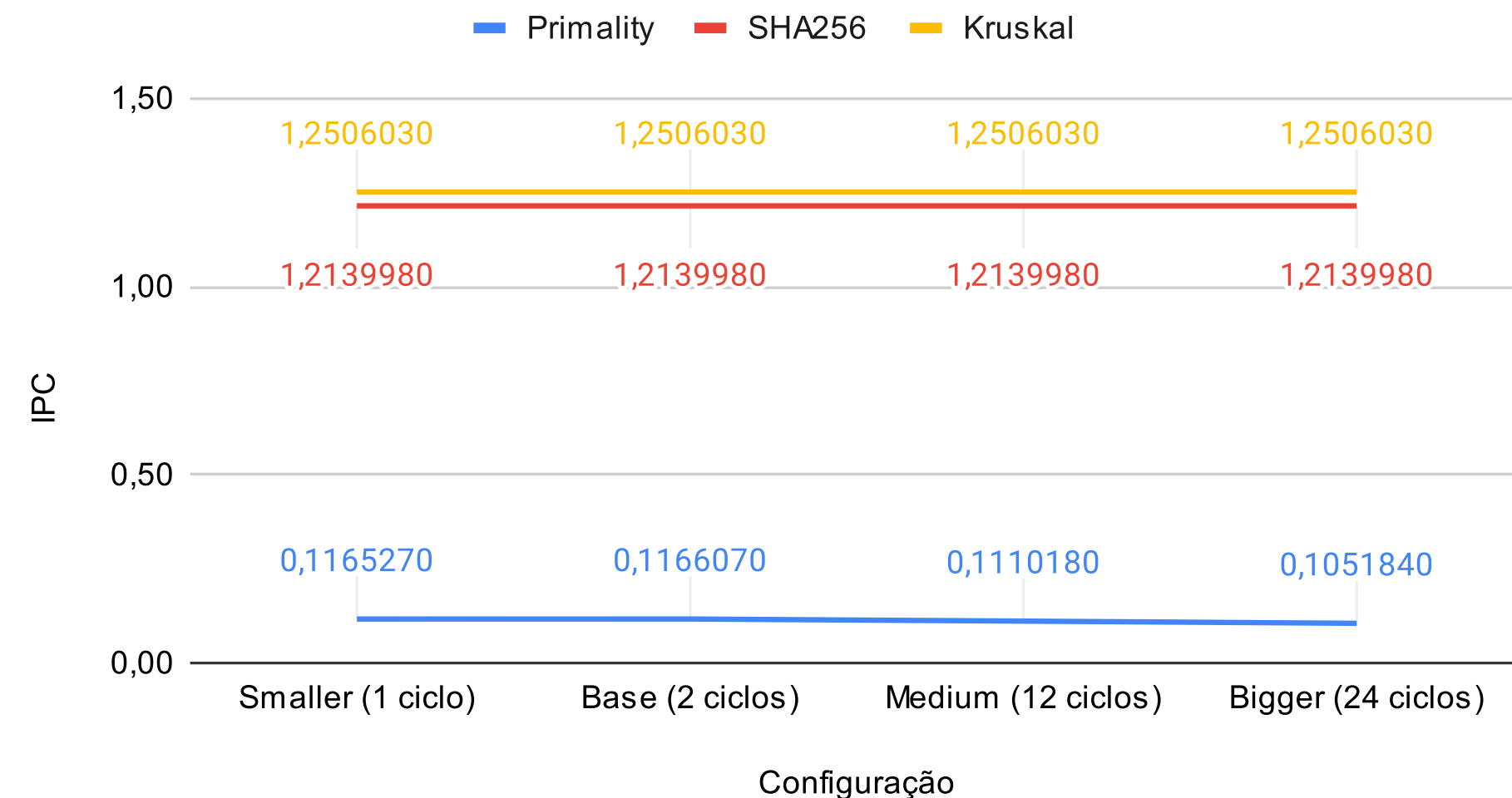




## 4. Conclusões IPC

- Como previsto, a modificação dessa instrução só teria impacto no algoritmo de *Primality Test*, no qual ela é muito utilizada (responsável por 60% das instruções que usam a ULA de *float* do processador). É possível ver no gráfico uma sutil perda de IPC a medida que os ciclos de latência aumentam.
- Para os outros dois algoritmos, os quais não fazem nenhum uso dessa instrução, seus IPCs permanecem os mesmos durante todas as modificações.

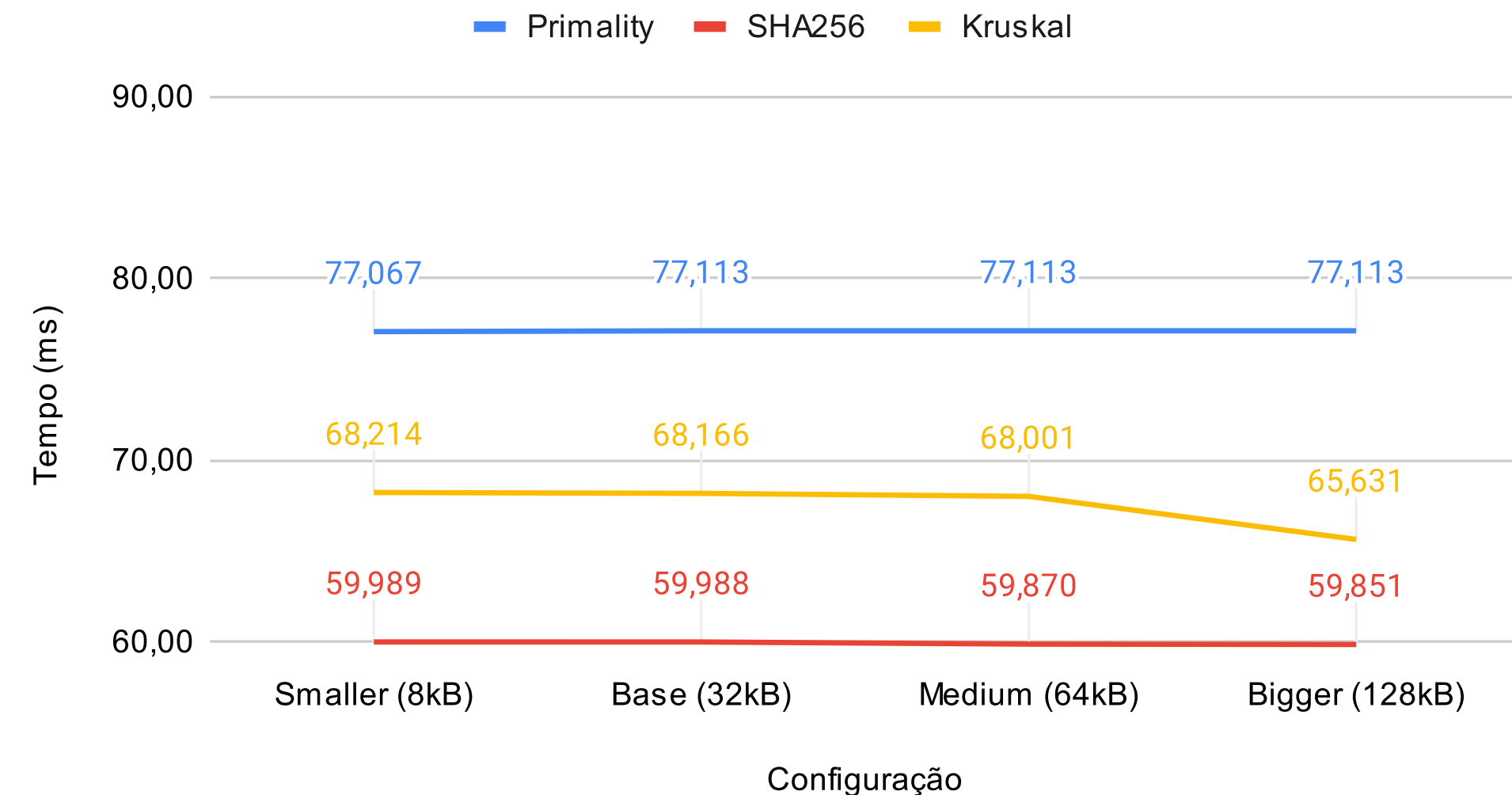
IPC - Latência da Instrução SimdFloatSqrt



## 5. Conclusões Tempo

- Tal como foi visto na análise do IPC, o único algoritmo que se beneficia em grande escala com as mudanças na cache L1 é o Kruskal. Dessa vez, a reação é inversa: quanto maior a cache, menor é o tempo de execução do algoritmo. Novamente, motivado pelo maior uso da cache em detrimento dos demais.
- Os outros dois algoritmos também tiveram melhoras, mas numa escala menor.

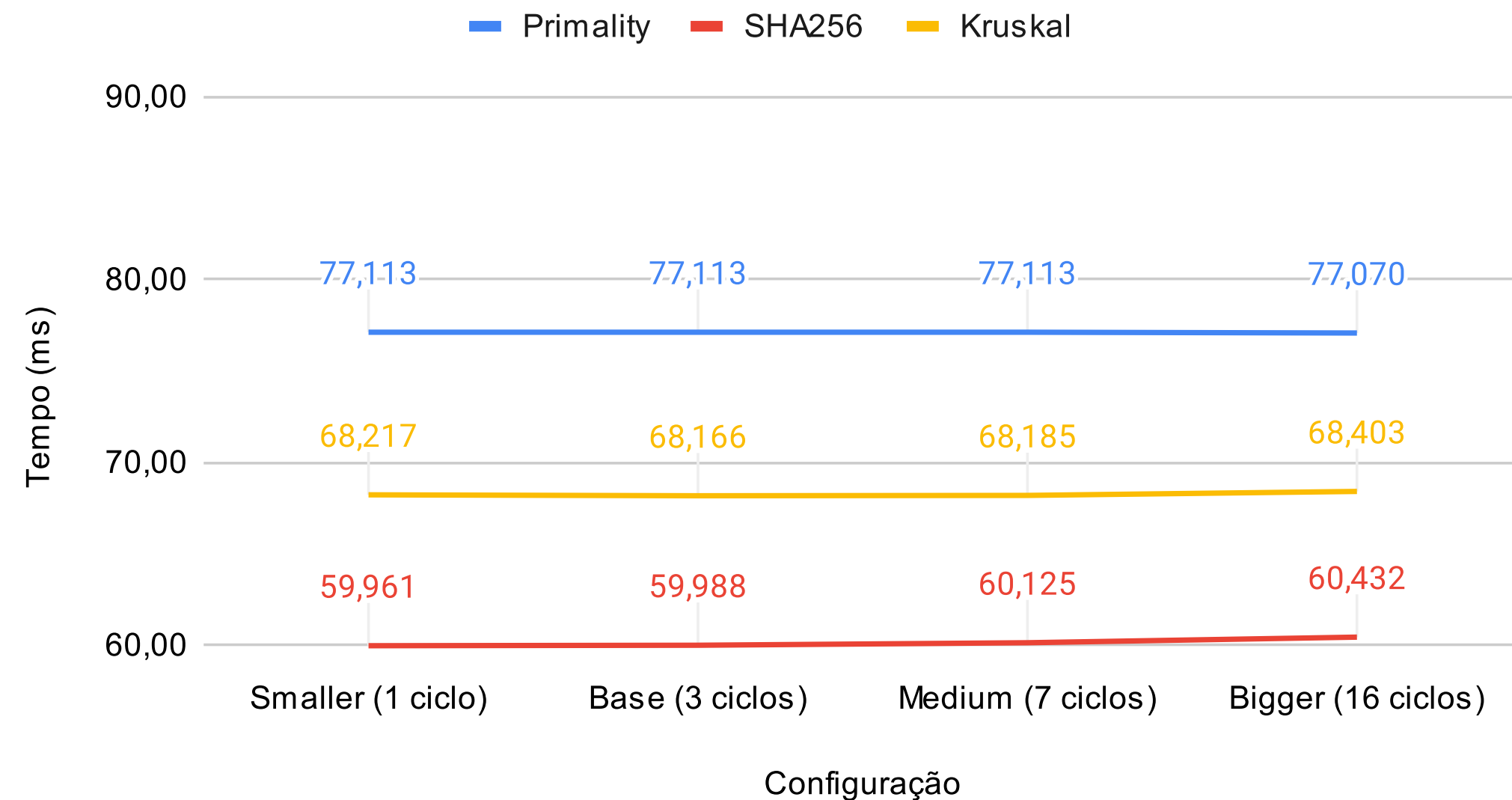
Tempo - Cache L1 Dados



## 5. Conclusões Tempo

- Como mencionado previamente, a variação da latência da instrução IntMult também não terá um grande impacto no tempo de execução dos algoritmos, visto que nenhum deles utiliza ela extensamente.
- Além disso, nos algoritmos onde a instrução era utilizada, o código apresentava grande tendência a fazer bom uso do pipeline disponível dentro da ULA, algo que minimiza mais ainda o impacto das mudanças propostas.

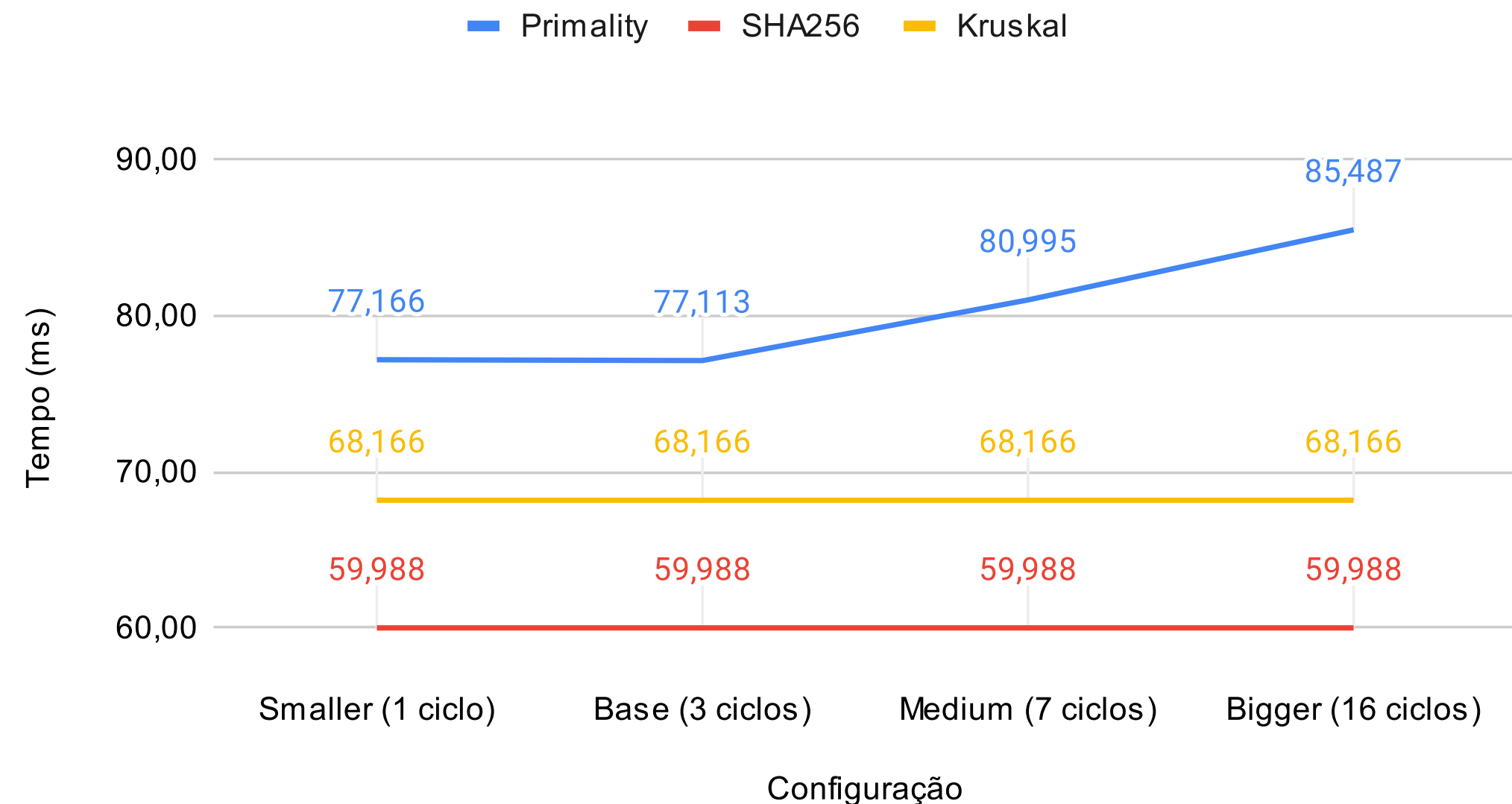
Tempo - Latência da Instrução IntMult



## 5. Conclusões Tempo

- Diferentemente da queda sutil que essa modificação causou no IPC, ela se faz muito presente na análise do tempo de execução do algoritmo de *Primality Test*, com uma diferença de 10% entre as configurações *Smaller* e *Bigger*. Obviamente, a discrepância nos tempos se dá pois esta é a principal instrução do algoritmo.
- Para os demais algoritmos, a curva se manteve exatamente constante, já que não fazem uso da operação de raiz quadrada.

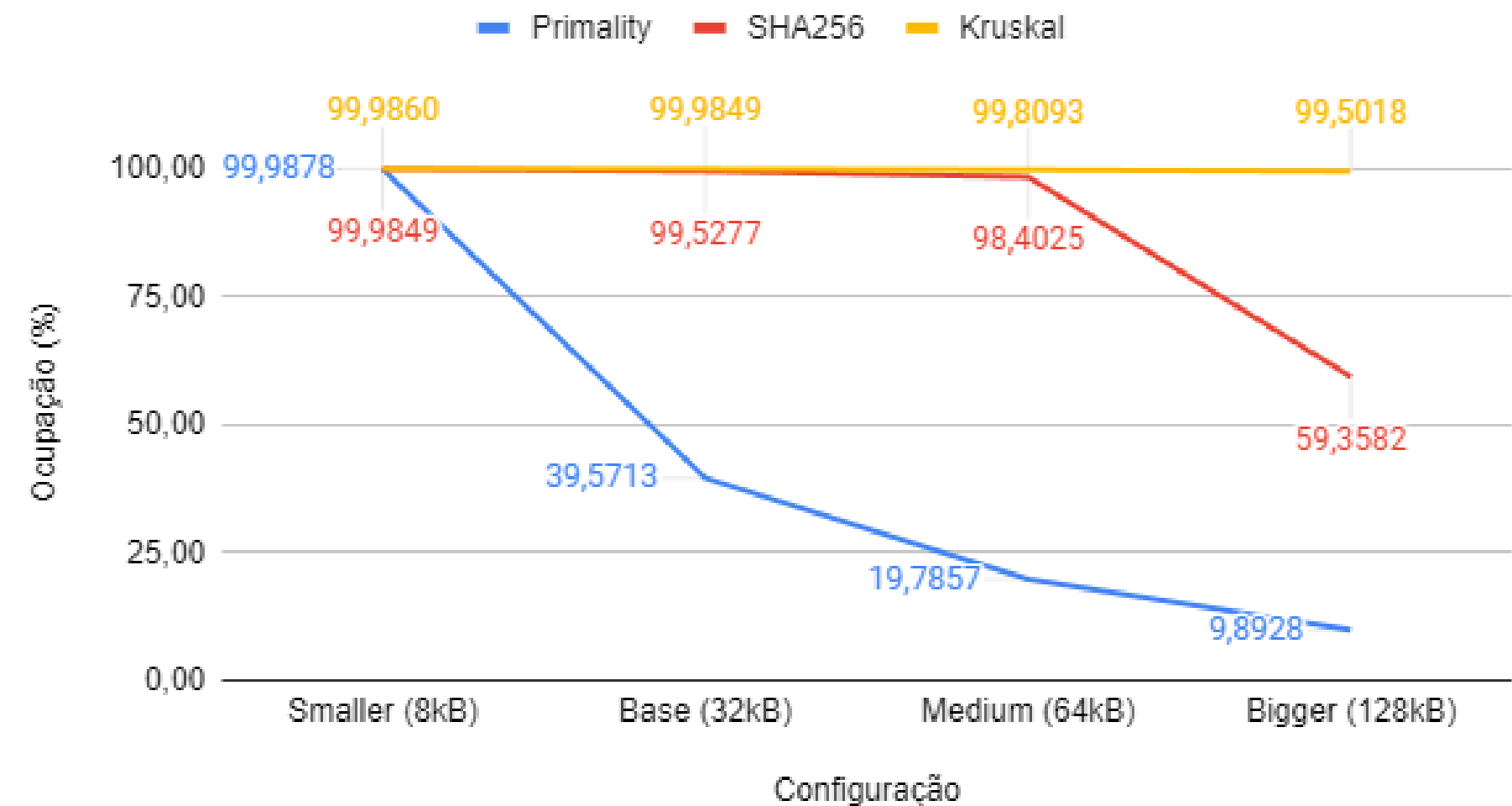
Tempo - Latência da Instrução SimdFloatSqrt



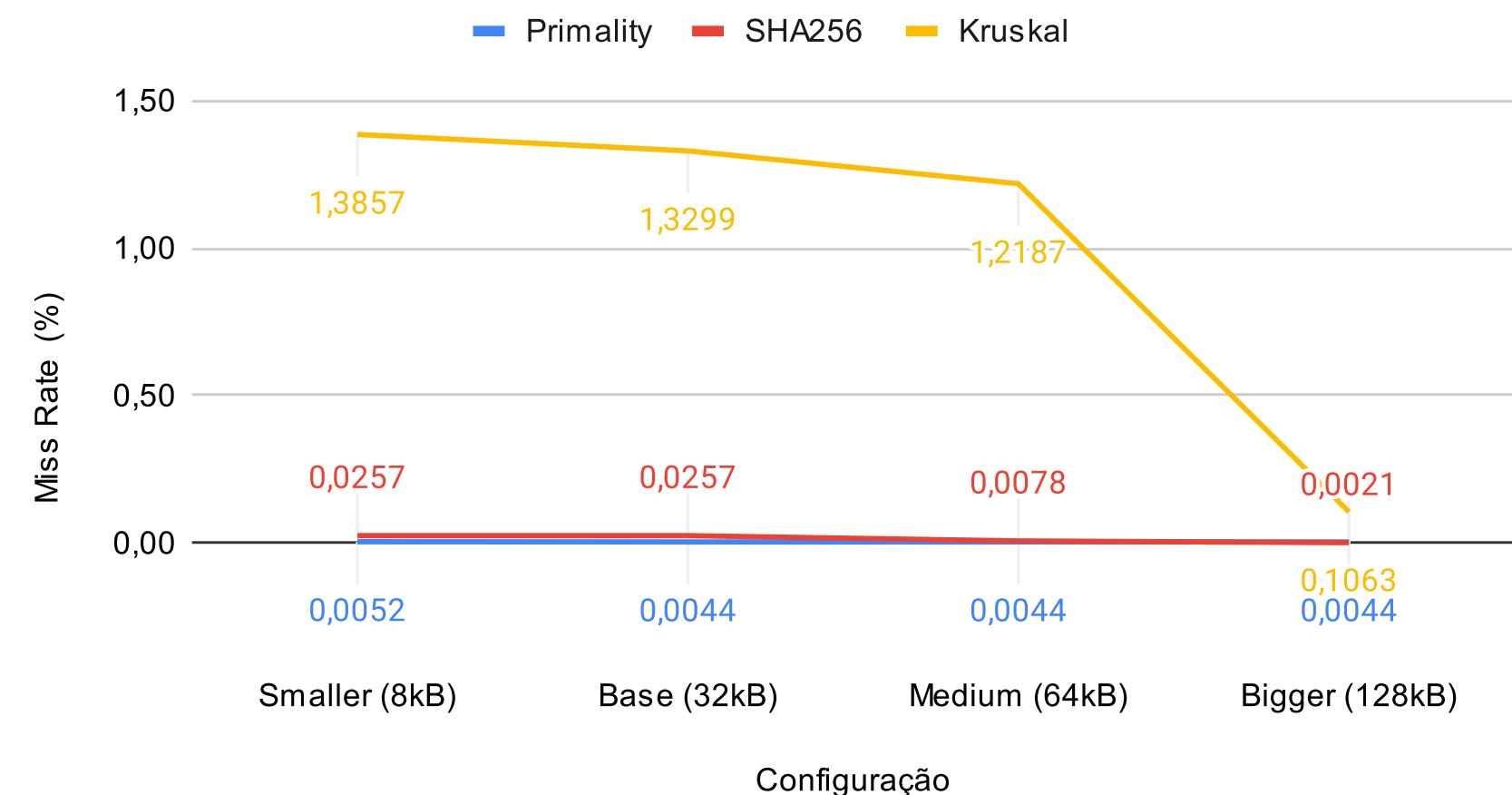
## 6. Conclusões Cache

- Como complemento dos resultados obtidos, ao variar o tamanho da cache de dados, trouxemos dois gráficos adicionais sobre seu uso.
- Apesar de ter a entrada fixa 64kB, o algoritmo de Kruskal utiliza a cache por completo em todas as configurações. A diferença se dá no *miss rate*, que cai abruptamente conforme aumentamos o tamanho da memória.
- O algoritmo *SHA256* utiliza por volta de 64kB de dados, como é possível ver na grande queda da ocupação na configuração Bigger. Apesar disso, ela faz grande reuso dos dados contidos nesse espaço, já que mesmo com ocupação alta, seu *miss rate* é consideravelmente baixo.
- O algoritmo de *Primality Test* tem pouco proveito da memória Cache, visto que já em 32kB existe uma brusca queda em ocupação.

### Ocupação - Cache L1 Dados



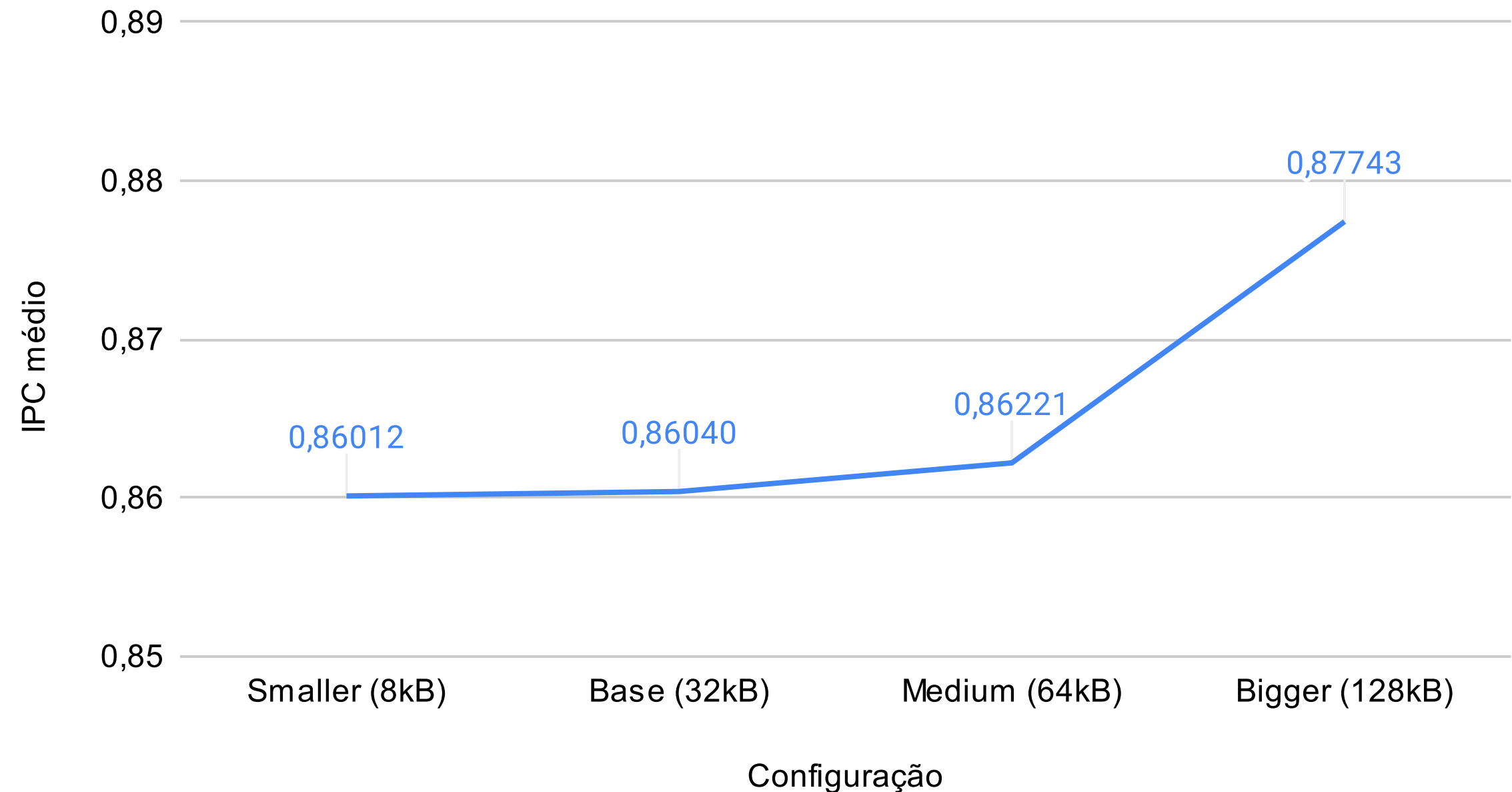
### Miss Rate - Cache L1 Dados



## 7. Média de IPCs

- O aumento do IPC com o aumento do tamanho da cache é bastante evidente, pois todos os algoritmos se beneficiam em algum nível da Cache, mostrando que essa variação é um ótimo custo benefício.
- Nota-se que da configuração Medium para a configuração Bigger há um salto no valor, visto que o algoritmo Kruskal se beneficia fortemente do aumento da cache ao utilizar 128KB.

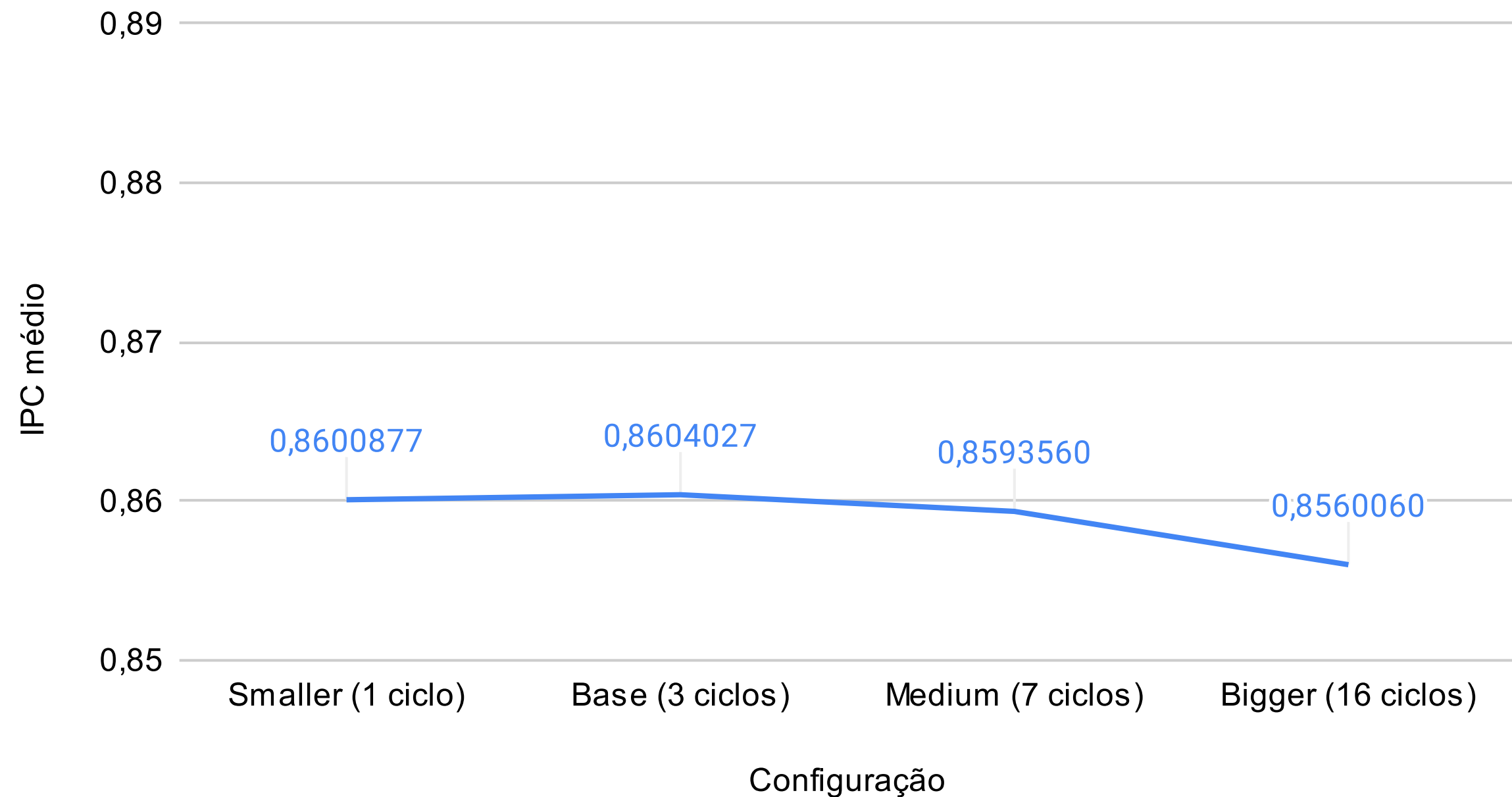
IPC Médio - Cache L1 Dados



## 7. Média de IPCs

- Neste caso, observa-se que o IPC médio diminui com o aumento da latência. Essa mudança é sutil, afetando apenas os algoritmos SHA256 e Kruskal, e de maneira leve, devido à compensação do pipeline interno mencionada anteriormente.

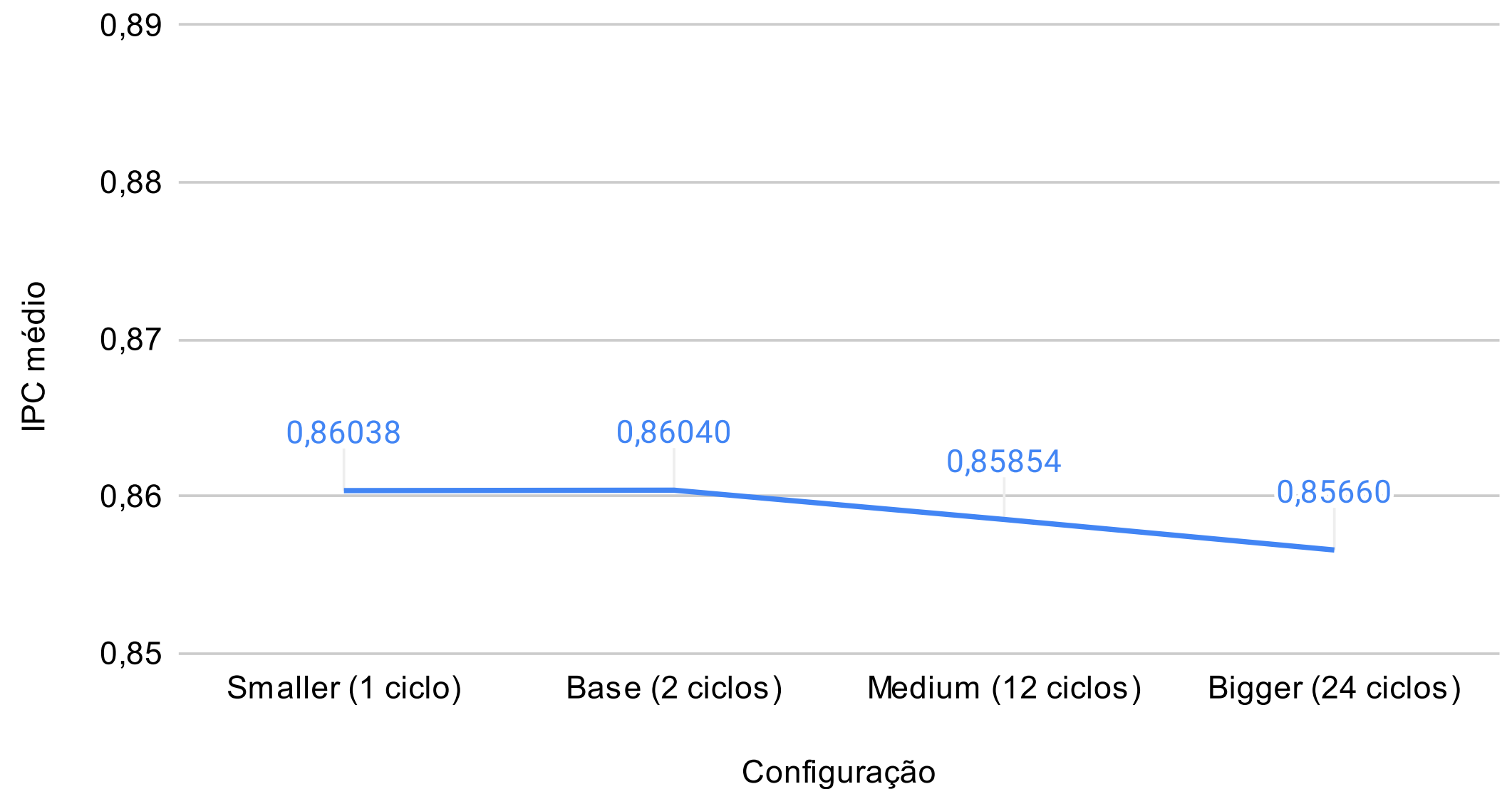
IPC Médio - Latência da Instrução IntMult



## 7. Média de IPCs

- Como esta variação afetou apenas o algoritmo Primality Test, a média dos IPCs não apresentou grandes variações.

IPC Médio - Latência da Instrução SimdFloatSqrt





# Referências

- [1] GeeksforGeeks, “Introduction to Primality Test and School Method,” GeeksforGeeks, Nov. 21, 2015.  
<https://www.geeksforgeeks.org/introduction-to-primality-test-and-school-method/> (accessed Aug. 14, 2024).
- [2] “crypto-algorithms/sha256.c at master · B-Con/crypto-algorithms,” GitHub, 2024. <https://github.com/B-Con/crypto-algorithms/blob/master/sha256.c> (accessed Aug. 14, 2024).
- [3] GeeksforGeeks, “Kruskal’s Minimum Spanning Tree (MST) Algorithm,” GeeksforGeeks, Oct. 30, 2012.  
<https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/> (accessed Aug. 14, 2024).