**UNIVERSIDADE FEDERAL DA PARAÍBA UFPB – CAMPUS IV**

**DEPARTAMENTO DE CIÊNCIAS EXATAS - DCX**

**B. SISTEMAS DE INFORMAÇÃO - BSI**



**Avaliação de Desempenho de Sistemas**

**Análise de desempenho e Medição Prática 1**

José Carlos Jr.; [jose.carlos@dcx.ufpb.br](mailto:jose.carlos@dcx.ufpb.br)

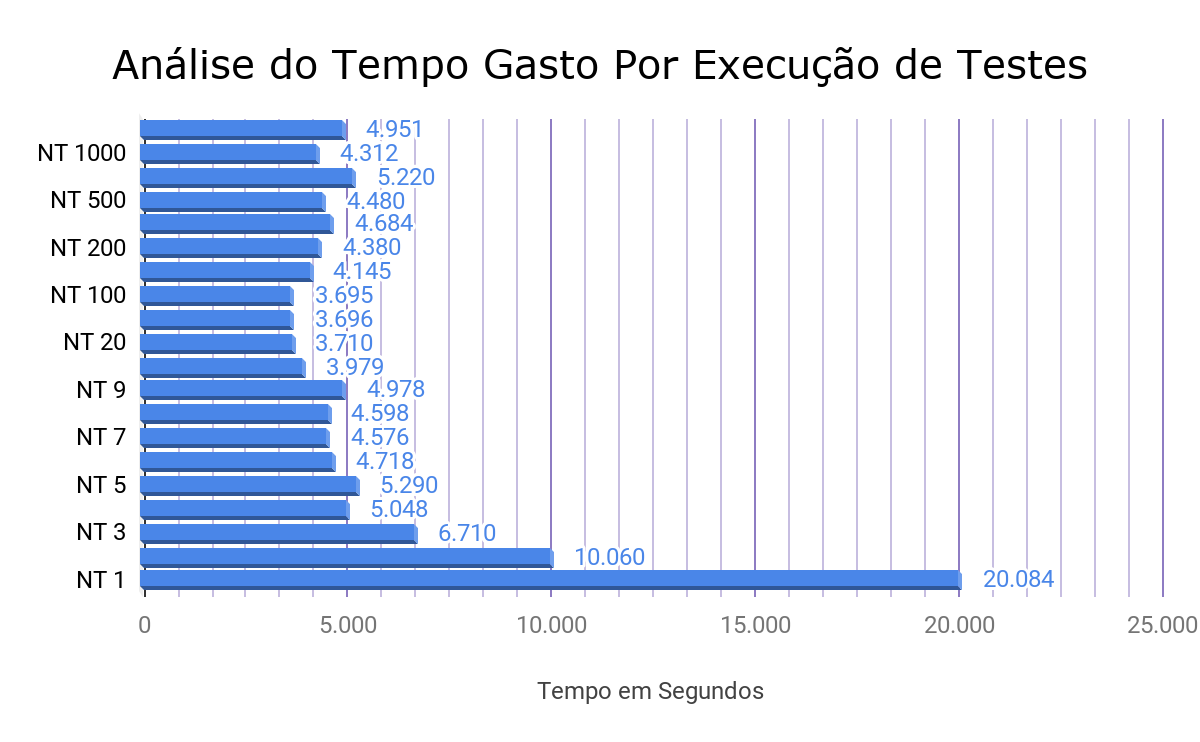
Mat. 2016046363

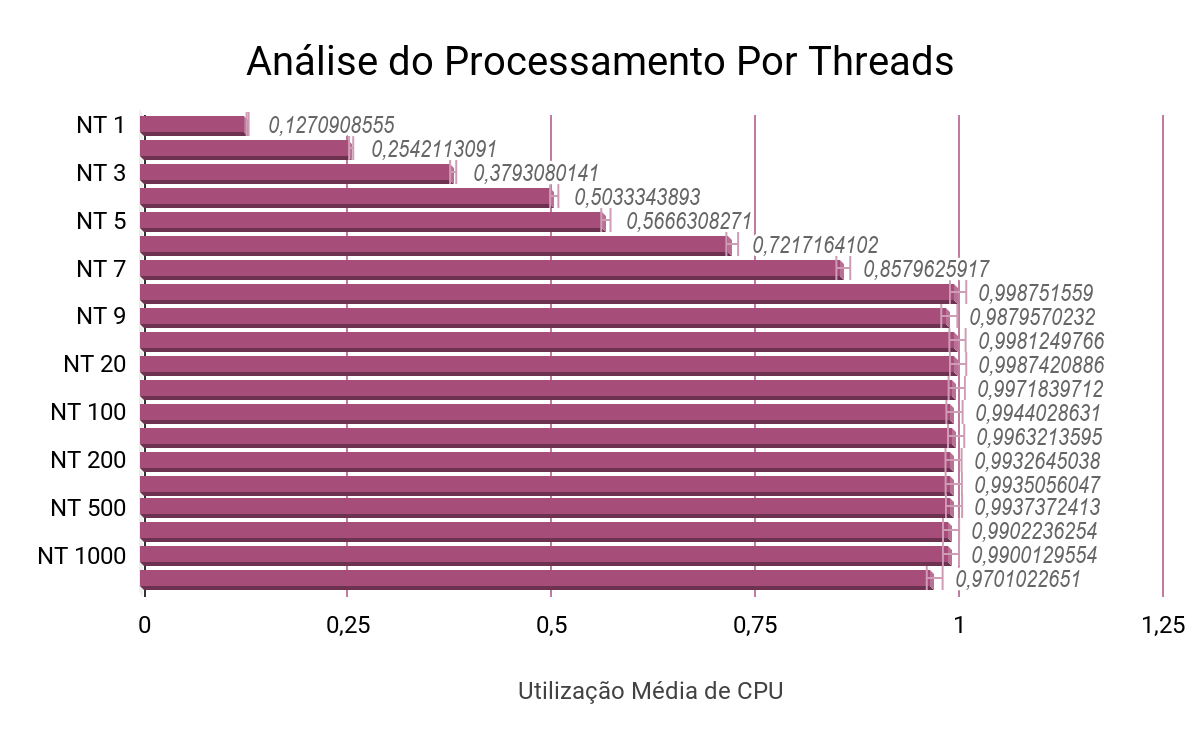
**Rio Tinto - PB**

**Novembro - 2019**

**Problema 1: Procurando números primos**

Estudos e análises dos testes





**Questões**

0. Informe a configuração da máquina na qual você está executando esta avaliação de desempenho, com informações sobre a CPU, Memória e Disco da mesma.

**Processador: Intel ® Core(TM) i5-8250U; CPU 1.60GHz x 8.**

**Memória: 3.74GB**

**Disco: 931.51GB**

1. Primeiro, queremos saber o impacto do uso de multithreading no desempenho. Execute o programa para valores diferentes de número de threads (NumThreads). Por exemplo, para número de threads = {1, 2, 3, ..., 10, ...} (não se restrinja a estes valores). Para esta questão, execute o programa para buscar a quantidade de primos com o valor de número máximo = 95000 (valor default). Faça as seguintes análises:

a. Como o aumento da quantidade de threads afeta a utilização de CPU? Analise os resultados e discuta possíveis causas deste comportamento.

**A execução de muitos processos simultaneamente pode acarretar na lentidão da CPU. Isso ocorre porque, mesmo um processador tendo dois ou mais núcleos, existe um limite para ele, tanto é que quando fiz o teste com 20.000 threads o processador deu erro, nem sempre quanto mais threads e melhor e tampouco quanto menos, temos que ter uma quantidade ideal para processar tal ação com melhor desempenho.**

b. Como o aumento da quantidade de threads afeta o tempo total de execução? Analise os resultados e discuta possíveis causas deste comportamento.

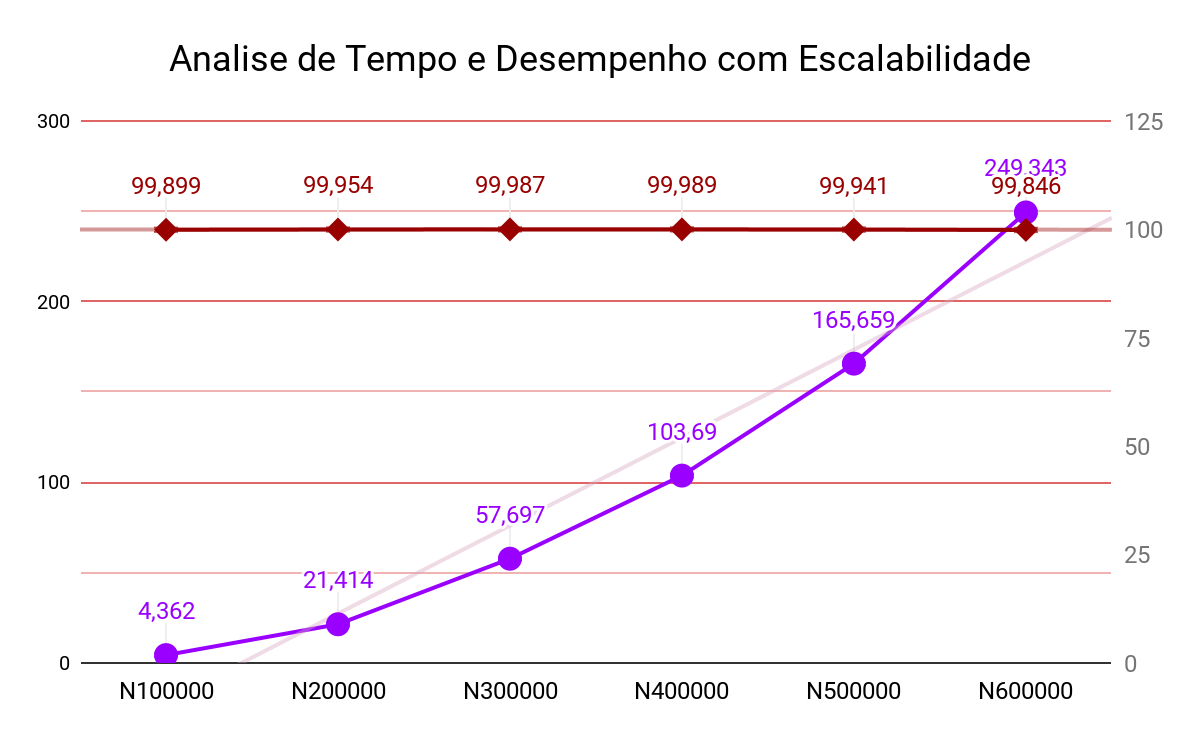
**Como os processos são linhas de códigos, módulos na verdade, onde serão executados, onde cada processo são instruções que informa ao processador suas ações. E quanto mais processamento estiver ocorrendo, maior será o tempo gasto para realizar a execução de um determinado processo. Além disso, quanto maior for o número de threads (Acima de 1000) o desempenho da CPU vai perdendo sua eficiência de acordo com os testes realizados em minha máquina.**

c. Qual número de threads você acha mais adequado para executar este programa? Você acha que sua escolha poderia ser diferente caso você executasse o mesmo programa em uma máquina diferente? Justifique.

**O número mais ideal de threads de acordo com os testes realizado na minha CPU fou *20 threads*, pois de todos os testes realizados, foi o segundo melhor desempenho e o terceiro menor tempo, seus resultados foram: *3,710s para o tempo de processamento e 0.9987420886 = 99.874% de desempenho.* A diferença para o teste com 100 threads em que teve o menor tempo foi de apenas 0,015s, porém o seu desempenho foi de apenas 0.9944028631 MeanUtilization = 99.44%, e já comparando com o melhor desempenho foi o teste realizado com com 8 threads que registou um desempenho de CPU de 0.998751559 = 99.875%, ficando apenas 0.001% com melhor desempenho, porém o tempo que foi gasto para realizar tal processamento o tornou ineficiente pois seu tempo foi de 4,598s.**

**Se os testes fossem realizados em outras máquinas com diferentes configurações e suportes, os resultados seriam diferentes, pois processadores com a quantidade de núcleos e versoẽs diferentes dariam números divergentes.**

2. Queremos agora analisar a escalabilidade deste programa. Então, vamos analisar como o desempenho é impactado quando aumentamos a carga de trabalho do sistema. Neste problema, a carga aumenta quando aumentamos o número máximo indicado para a busca de números primos (NumeroMaximo). Execute o programa para valores diferentes do número máximo para a busca dos primos. Exemplo: 100000, 200000, 300000, 400000 ... (não se restrinja a esses valores). Para esta análise, mantenha fixo o número de threads, usando o número escolhido na letra (c) da questão 1.



***Analise:***

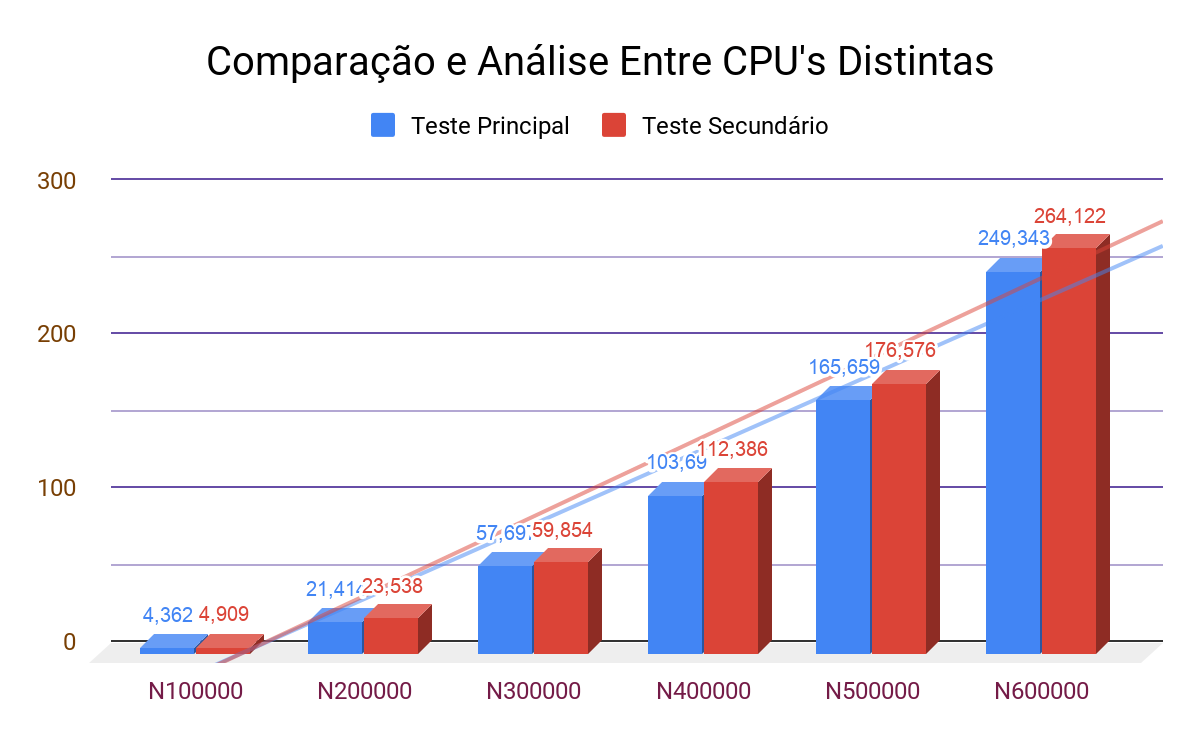
a. Como o aumento da carga do sistema (o número máximo) afeta o tempo total de execução? Analise os resultados e discuta possíveis causas deste comportamento.

**Afeta o tempo total de execução pois como está aumentando o número de processamento, está aumentando a quantidade de tarefas e consequentemente sobrecarregando os fios ou encadeamento de execução (Threads). Mesmo o processo sendo dividido por 20 fios simultaneamente, houve de fato uma longanimidade em realizar o andamento da aplicação do processo.**

b. Com base nos resultados de desempenho, você considera este programa escalável? Justifique sua resposta e dê sua opinião sobre possíveis motivos dele ser ou não ser.

**Para afirmar se este programa é escalável, entramos em um alto grau de relatividade, porém sempre que o programa for operável com uma quantidade acima de números primos máximos acima de 200000 se tornaŕa inviável por conta do tempo total gasto para realizar tal processamento. Em uma outra máquina melhor, talvez esse número poderia ser bem maior, mas de acordo com os testes realizados na minha CPU esta é a minha conclusão.**

***3. [EXTRA]*** Faça outras análises com o programa e tente extrair informações interessantes sobre desempenho. Você pode alterar o código para responder esta questão. Você também pode executar o programa em máquinas com hardware diferentes e comparar os resultados, discutindo as diferenças de desempenho.

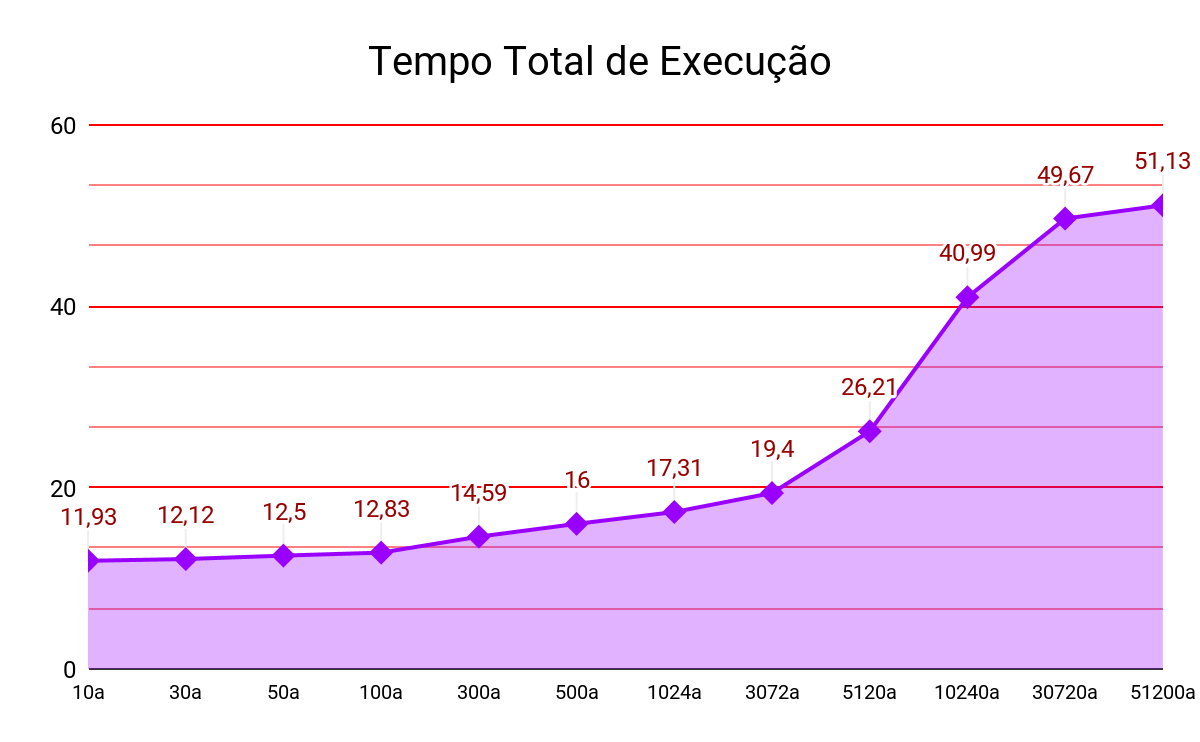


**Fiz o teste em outro computador sem alterar o código do programa com o exemplo e modelo descrito e utilizado na questão anterior para poder compará-los. Podemos ver que os testes realizados na CPU2, tiveram um tempo mais elevado para concluir suas execuções, pois a máquina na qual foi utilizada, é uma processador com uma versão mais antiga que a minha, por isso, o mesmo demorou um pouco mais.**

**Problema 2: Fazendo acessos à memória**

**Questões**

1. Queremos saber qual o impacto do tamanho do array no desempenho dos acessos feitos à memória. Podemos ter um problema parecido quando tratamos de um sistema banco de dados, por exemplo. Analise o desempenho das buscas feitas no array para uma grande variedade de tamanhos de array. Por exemplo, você pode executar tanto para tamanhos de algumas unidades de KB (1, 2, ...), quanto para algumas dezenas (10, 20, ...), centenas (100, 200, ...), da ordem de MB (1024, 2048, ...) até quanto for possível no seu sistema (provavelmente perto de 1GB = 1024 \* 1024 KB). Com essas medições, você deve ser capaz de realizar as seguintes análises:



a. Como o tempo de busca no array varia quando se aumenta o tamanho do array? Analise os resultados e discuta possíveis causas deste comportamento.

**Quando aumentamos o tamanho dos arrays, temos uma constante progressão crescente no tempo. A medida em que os acessos são feitos nos níveis de memória, tende de certa forma impactar no desempenho da mesma, resultando de forma indubitável no quesito desempenho de tempo.**

b. O aumento do tempo de busca é bem comportado quando se aumenta o tamanho do array? O sistema é escalável, ou em alguns momentos existem mudanças mais bruscas no tempo de busca ao variar o tamanho do array? Analise esta questão e discuta possíveis causas deste comportamento.

**Sim, quando aumentamos o tamanho dos array de forma padronizada, o resultado tende a crescer também de forma estandardizada. Sim, o sistema é escalável, porém até certo ponto limite da própria memória. Em momento algum demonstrou anomalias ou surpresas em seus respectivos resultados, talvez tal ação se de pelo fato do programa que foi desempenhado para tais testes.**

***2. [EXTRA]*** Faça outras análises com o programa e tente extrair informações interessantes. Você pode alterar o código para esta tarefa.

**Quando eu fiz os testes e refiz usando alguns novos parâmetros, pude observar que ao usar um tamanho de Array de 5MB (5120 KB) mas com 1 bilhão de acessos em posições aleatórias do array, consegui realizar em um tempo aproximadamente de 27s, e quando quando fiz o mesmo teste porém com uma quantidade menor de posições de arrays que foi com 100 milhões de acessos em posições aleatórias do array, gastei um tempo equivalente a 2,7s. Resumindo, há de certa forma uma proporção escalável por casas decimais de acordo com os testes nos quais realizei.**