UFPI - Ciências da Computação - Sistemas Distribuídos - 2020.2

Daniel Leite Dantas

Estevam Amorim de Carvalho Alves

Utilizando a Biblioteca MPI

O objetivo deste trabalho é utilizar os recursos fornecidos pela biblioteca MPI para a implementação de diferentes abordagens distribuídas capazes de realizar o cálculo do valor de π, através da divisão das operações em diferentes processos em diferentes nós de um cluster, com a finalidade de testar e comparar as suas respectivas performances (clique aqui para fazer o download do cluster já previamente configurado. Usuário: aluno, Senha: 123456).

Nota: pelo fato de tanto a máquina virtual “OK3” e “OK4” apresentarem problemas, a máquina “OK2” foi clonada e seus clones ajustados para que o cluster, durante os testes, mantivesse a mesma quantidade de nós.

1. **Instruções para Execução**

   As instruções abaixo são voltadas para execução em ambiente **Linux**, utilizando **Python 2 ou 3** com o acréscimo da **Biblioteca MPI**, utilizando as ferramentas **NFS Server** e **SSH:**

   - ➔ Em todas as máquinas crie o usuário **mpiuser** com a mesma senha, através do comando `adduser mpiuser`;
   - ➔ Crie um diretório de nome **cloud** no endereço /home/mpiuser/cloud em todas as máquinas, copie para ele todos os arquivos fonte disponibilizados juntamente com este relatório;
   - ➔ Nas máquinas clientes que irão consumir o diretório compartilhado através da máquina que executa o **servidor NFS** (no cluster indicado anteriormente a máquina “OK” é responsável por essa função), execute `mount -t nfs master:/home/mpiuser/cloud /home/mpiuser/cloud` para realizar a montagem do diretório remoto;
   - ➔ No servidor, estando no diretório **cloud** do segundo passo, utilize o seguinte comando para a execução das implementações: `mpiexec -np N --hostfile`

`maqs python3 code-vx.py`, onde “N” se refere a quantidade de processos em que o cálculo será distribuído, e “x” a versão do código que deverá ser executado.

2. Abordagens e Exibição dos Dados no Terminal

O valor de π pode ser calculado através da seguinte expressão:

$$\frac{\pi}{4} = \int_0^1 \frac{dx}{1+x^2} \approx \frac{1}{N}\sum_{i=1}^{N} \frac{1}{1+\left(\frac{i-\frac{1}{2}}{N}\right)^2}$$

Com base nisto e levando em consideração que a precisão da expressão aumenta proporcionalmente ao valor de N, assumindo o valor como 840, foram propostas quatro abordagens distintas, na primeira e mais simples a expressão é calculada individualmente por processo, na tela então é exibido do lado cada identificador de processo, seu respectivo valor de π encontrado e, assim como em todas as versões, o tempo transcorrido.

Na segunda, cada processo recebe a responsabilidade de calcular o valor da função para uma faixa igualitária de valores de *i* e realizar o somatório desses resultados, é mostrado, para cada processo, o primeiro e último valor de *i* que delimitam a sua faixa, além do resultado do somatório.

Na terceira versão, além disso, todos os processos enviam os resultados dos seus somatórios para um processo mestre, que determina o valor de π ao somar todos os valores recebidos, do mesmo modo que na quarta versão, somente o valor de π e o tempo transcorrido são exibidos na tela.

Já na quarta versão a soma de todos os resultados dos processos é distribuída utilizando o método Reduce da biblioteca MPI, por consequência a quantidade de mensagens enviadas é reduzida.

3. Benchmarks

3.1. Plataforma

Os testes foram executados em um notebook com CPU Intel Core i7-9750H @ 2.60 GHz, 2592 MHz, 6 Núcleos físicos (12 núcleos lógicos) e 16GB de memória RAM utilizando Windows 10 como sistema operacional.

Neste, 4 máquinas virtuais foram montadas com SO Linux Ubuntu 14.4 32 bits utilizando o software Oracle VM VirtualBox, tendo sido designado para cada máquina 1.5GB de memória RAM e 2 processadores lógicos.

Nas máquinas virtuais, as aplicações são executadas com o interpretador Python 3.4.3, utilizando a biblioteca MPI OpenRTE 1.6.5.

3.2. Teste 1: Computando Pi e Sincronizando Resultados

Neste teste, foram comparadas a computação do valor de Pi nas quatro versões do programa:

V1 - sem distribuição de carga: cada processo computa o valor individualmente;

V2 - com distribuição de carga, sem somatório: cada processo computa apenas uma parte do valor de Pi e processos não se comunicam entre si;

V3 - com distribuição de carga e somatório utilizando MPI Send-Recv: cada processo computa uma parte do valor de Pi e comunicam o seu resultado para um nó *root* centralizado;

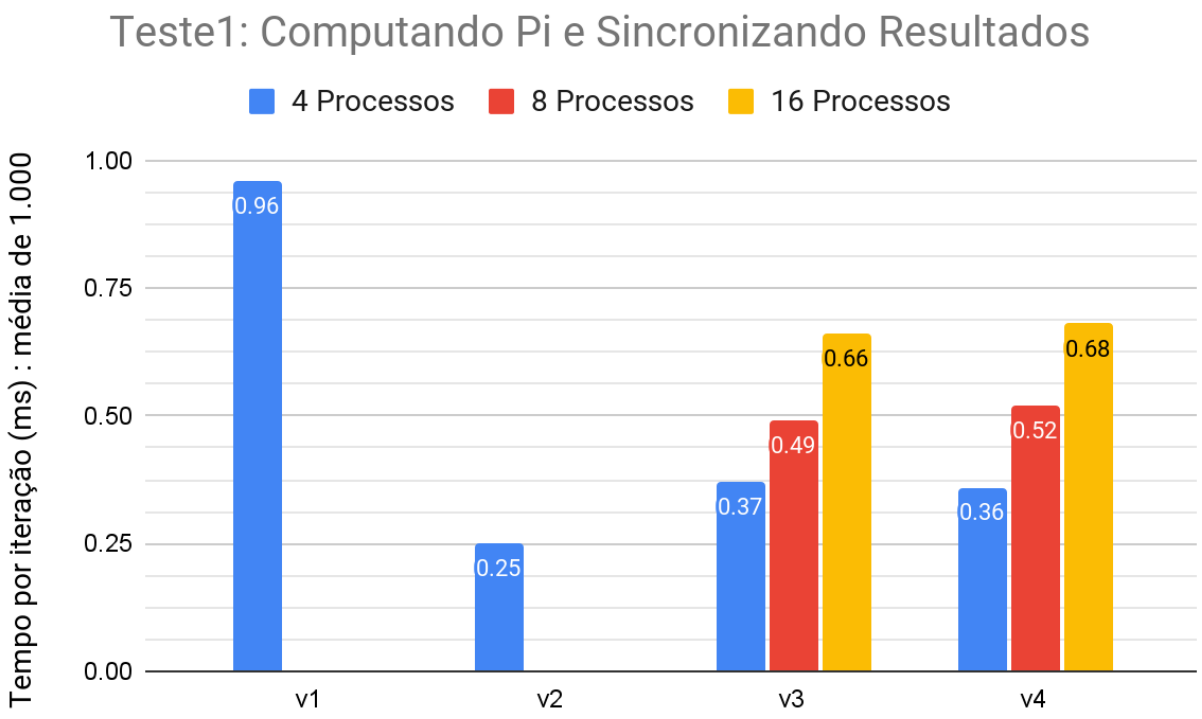
V4 - com distribuição de carga e somatório utilizando MPI Reduce: cada processo computa uma parte do valor de Pi e comunicam o seu resultado para nós intermediários até a redução ao *root*.

Dado o baixo custo computacional dos cálculos envolvidos, a operação foi repetida 1.000 vezes, de forma a acumular um tempo de execução onde flutuações nos recursos do sistema, particularmente instável por conta do uso de máquinas virtuais, produzissem um reduzido ruído nos dados coletados.

Vale ressaltar que, neste primeiro teste, para as versões 3 e 4 do programa, a comunicação com *Send-Recv* ou *Reduce*, respectivamente, é realizada em cada uma das 1.000 iterações. Dessa forma, a média mostrada no gráfico abaixo retrata o custo computacional de uma iteração da computação, comunicação e somatório do valor de Pi.

Para a execução distribuída da tarefa, em todos os testes, foram utilizados 4 Processos MPI, um para cada máquina do cluster. Como esperado, o uso de menos processos resultou em piores resultados para os testes com distribuição de carga, não exibidos nos gráficos. De forma não tão óbvia, entretanto, obteve-se os mesmos resultados utilizando 8 processos, indicando que, mesmo tendo sido atribuídos 2 núcleos lógicos para cada uma das 4 máquinas virtuais, quaisquer ganhos computacionais obtidos com mais processos sendo executados simultaneamente nos 6 núcleos físicos da máquina foram provavelmente perdidos com a necessidade do escalonamento de mais processos pela hypervisor das máquinas virtuais.

Corroborando com o exposto acima, observou-se que, ao serem utilizados mais de 8 processos, os resultados foram proporcionalmente piores para as versões com distribuição de carga, novamente dando sinais do *overhead* necessário para o controle de mais processos que, neste caso, não trazem um real paralelismo. É esperado que, em ambientes com mais nós e/ou núcleos disponíveis sem compartilhamento de hardware, um número maior de processos favoreça a performance das atividades com distribuição de carga.

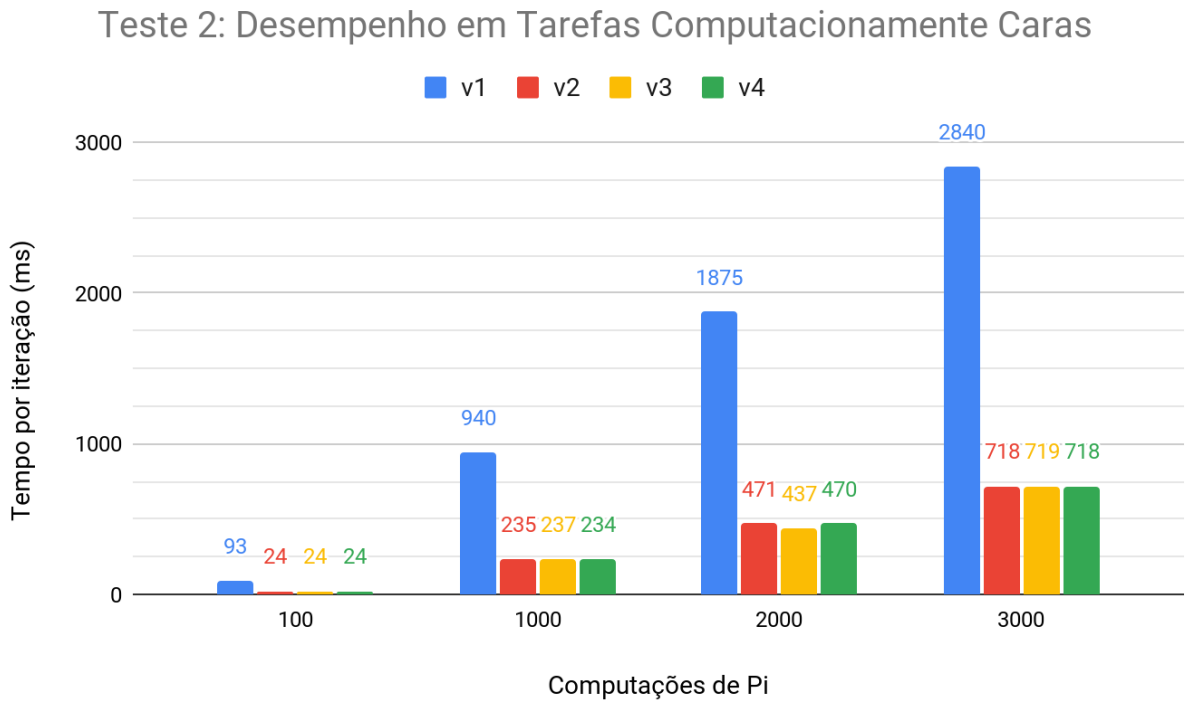


Observa-se que, na versão 2 do programa, onde cada processo computa sua parcela do valor mas não há a comunicação e somatório dos resultados, os processos finalizam aproximadamente 4 vezes mais rápido do que na versão 1, de forma que os ganhos do paralelismo são praticamente ideais.

Nas versões 3 e 4, além da computação dos valores de forma distribuída, é feito também o somatório dos valores. Não obstante, as tarefas ainda são finalizadas aproximadamente 3 vezes mais rápido do que na versão 1, um grande e significativo ganho em performance. Não se observou uma diferença significativa entre o uso das funções *Send-Recv* e *Reduce*, possivelmente por conta do baixo número de processos e nós utilizados.

3.3. Teste 2: Desempenho em Tarefas Computacionalmente Caras

O custo da computação do valor de Pi pode ser considerado barato em relação a aplicações que lidam com muitos dados ou realizam ainda mais operações. Neste segundo teste, portanto, simulou-se uma dessas aplicações computando-se o valor de Pi diversas vezes. No entanto, ao contrário do teste 1, a comunicação e o somatório dos valores computados de forma distribuída nas versões 3 e 4 não é feito toda iteração, mas apenas ao final da computação da carga emulada.



No eixo horizontal estipula-se o custo de uma computação emulada em relação ao custo da computação do valor de Pi, enquanto que o eixo vertical mostra o tempo de execução em milissegundos.

Nesse caso, pode-se observar que o *overhead* adicionado pela comunicação entre os processos e o somatório é desprezível em relação ao custo da tarefa, de forma que os ganhos em paralelismos são praticamente ideais: 4 processos resultam em uma computação 4 vezes mais rápida, aproximadamente. Além disso, mais uma vez pode-se observar que não há uma diferença significativa entre o uso das funcionalidades *Send-Recv* e *Reduce* da biblioteca MPI, dado o número de processos utilizados.

4. Conclusão

Mesmo com um ambiente de testes longe do ideal devido às limitações de hardware, pode-se observar o potencial do uso de bibliotecas de comunicação entre sistemas distribuídos, como a biblioteca MPI.

Conclui-se, ainda, que o custo para comunicação entre esses processos, mesmo em ambientes conectados pela rede, pode ser aceitável ou até desprezível dependendo da complexidade da tarefa desenvolvida.