Programação Paralela Híbrida aplicada em um Cluster Computacional com testes em Multiplicação de Matrizes

Leonardo S. Mendes[1], Eduardo Ferreira[1]

[1]Departamento de Engenharias e Ciência da Computação – Universidade Regional Integrada do Alto Uruguai e das Missões (URI) – Câmpus Santiago
Caixa Postal 97700-000 – Santiago – RS – Brasil

lsm.ccomp@gmail.com, eduardo.ferreira@urisantiago.br

Abstract. *This article highlights the use of two parallelism tools, MPI and OpenMP, applied in a computational cluster. Programming is made of a hybrid form, with MPI being used to parallelize the cluster message exchange, and OpenMP to parallelize the number of processors for each node. The tests are applied using a matrix multiplication algorithm, and the results obtained are compared with a computer with twice the processing of the cluster nodes.*

Resumo. *Este artigo evidencia o uso de duas ferramentas de paralelismo, MPI e OpenMP, aplicadas em um cluster computacional. A programação ocorre de forma híbrida, sendo o MPI utilizado para paralelizar a troca de mensagens do cluster, e o OpenMP para paralelizar o número de processadores de cada nó. Os testes são aplicados utilizando um algoritmo de multiplicação de matrizes, e os resultados obtidos são comparados com um computador com o dobro de processamento dos nós do cluster.*

1. Introdução

Com a evolução, novos computadores com grandes poderes de processamento foram sendo desenvolvidos. Contudo, para obter um maior poder ainda de processamento, foram desenvolvidos os *clusters* computacionais, um aglomerado de computadores interligados entre si capazes de gerar um grande poder de processamento.

Além disso, uma forma que permite um grande poder de processamento com o paralelismo, é a utilização de ferramentas como MPI e OpenMP, onde realizam trocas de mensagens utilizando a paralelização das threads.

Diante disso, ao unir as duas ferramentas em uma mesma aplicação, e utilizá-las nos *clusters*, é possível realizar uma programação paralela híbrida. Sendo as trocas de mensagens dos nós do *cluster* realizadas através do MPI, e com o OpenMP, as trocas de mensagens de forma individual em cada nó, utilizando seu número de processadores.

2. *Cluster* Computacional

Cluster é o nome dado a um sistema distribuído que relaciona dois ou mais computadores para que estes trabalhem de maneira conjunta no intuito de processar uma determinada tarefa (ALECRIM, 2013).

No seu desenvolvimento há requisitos a serem cumpridos, ele precisa ter mais de dois nós (*back-end*), sendo que precisa ter ao menos um nó mestre (*front-end*) (BACELLAR, 2010).

3. MPI

MPI (Message-Passing Interface) é uma biblioteca desenvolvida para ambientes de memória distribuída, máquinas paralelas massivas e redes heterogêneas. Oferece um conjunto de rotinas para o desenvolvimento de aplicações através de troca de mensagens, possibilitando que os processos se comuniquem (MPI, 2012, tradução nossa).

MPI possibilita a implementação de programação paralela em memória distribuída em qualquer ambiente, sendo suas principais vantagens a portabilidade e a facilidade de utilização.

4. OpenMP

De acordo com (FAVARETTO, 2018, apud CHANDRA et al, 2011), a ferramenta Open Multi-processing – OpenMP é uma interface de programação que oferece diversos recursos para a criação, desenvolvimento e execução de programas multithread. Uma aplicação com OpenMP utiliza o modelo de programação fork-and-join, com uma thread principal sendo criada no início do programa e a mesma executando sozinha até localizar uma diretiva que permita uma execução paralela, voltando a thread principal a executar sozinha após o termino da região paralela (MARONGIU, 2011, tradução nossa).

Por sua implementação se dar em C/C++, Fortran, entre outras, o OpenMP provê um padrão suportado por quase todas as plataformas ou arquiteturas de memória compartilhada, oferecendo um controle total de manipulação ao escrever o código.

5. Programação Paralela Híbrida

No modelo de programação híbrido MPI/OpenMP utiliza-se MPI para a comunicação entre os nós e OpenMP para paralelização dentro do nó. A ideia básica desse modelo é permitir que qualquer processo MPI possa criar um grupo de threads OpenMP.

6. Metodologia

Ao fazer uma análise aprofundada sobre os conceitos das duas bibliotecas, e ver a sua semelhança, assim como suas diferenças, foi possível utilizar uma abordagem que as usasse em conjunto. Utilizando o *cluster*, mesclar as duas bibliotecas possibilita um maior ganho de tempo.

O método usado foi um paralelismo entre os nós do *cluster* utilizando o MPI, replicando todo o código em cada um dos nós, e definindo cada trecho da multiplicação para que cada nó realize uma parte da multiplicação, foi possível separar por processos e aplicar o OpenMP no paralelismo de cada nó, usando suas threads. Dentro de cada trecho de multiplicação definido pelo MPI, é definido o número de threads que cada nó irá usar.

7. Resultados Parciais

O *cluster* computacional desenvolvido conta com 21 máquinas, todas com processadores Intel Core i5 da quarta geração, possuindo 4 processadores cada. Todas utilizam o sistema operacional Debian Stretch GNU/Linux 9.5 com kernel 4.9. A distribuição do *cluster* se da por um nó mestre e vinte nós escravos, contudo para a obtenção dos resultados, foi utilizado apenas duas máquinas, nó mestre e nó escravo, somente como uma forma de teste.

Após testar o *cluster* utilizando o *benchmark* NPB (*NAS Parallel Benchmark*), para garantir sua eficiência e aumento de performance, foi partido para o princípio da programação paralela híbrida, no qual, após estudar os princípios de sua utilização, foi projetado um algoritmo de multiplicação de matrizes agindo paralelizado.

O algoritmo se baseia na troca de mensagens feita por nós do *cluster* através do MPI, e nas trocas de mensagens individuais realizadas dentro de cada nó através do OpenMP.

Como pode ser visto no código fonte da Figura 1, os testes ainda estão sendo feitos com base em um algoritmo simples de multiplicação de matrizes, para melhor delimitar o uso das threads, quanto a threads de computação e threads de comunicação.

```
1.    if(no==1){ //seleciona o número do processo
2.          #pragma omp parallel private(i,k,j)shared(m1,m2,m3,tid) //define o que irá paralelizar
                                                                    //com o OpenMP
3.          {
4.             #pragma omp for                                      //começa a paralelização
5.             for(i=(i2/3)*2; i<i2; i++){
6.                    for(j=(i2/3)*2; i<i2; i++){
7.                           for(k=i2/3; k<(j1/3)*2; k++){
8.                                  m3[i][j] = m3[i][j] + (m1[i][k] * m2[k][j]);
9.                           }
10.                   }
11.            }
12.         }
13.         MPI_Send(&mult,1,MPI_INT,0,type,MPI_COMM_WORLD); //envia para o processo
                                                              //que irá receber
14.   }else{ //seleciona outro procecsso
15.         MPI_Recv(&mult,1,MPI_INT,1,type,MPI_COMM_WORLD,&status); //o processo
                                                          //recebe as informações enviadas
16.         #pragma omp parallel private(i,k,j)shared(m1,m2,m3,tid) //define o que vai paralelizar
                                                                    //com o OpenMP
17.            {
18.                 #pragma omp for                                //começa a paralelização
19.                 for(i=0; i<i2/3; i++){
20.                        for(j=0; j<i2/3; j++){
21.                               for(k=0; k<j1/3; k++){
22.                                      m3[i][j] = m3[i][j] + (m1[i][k] * m2[k][j]);
23.                               }
24.                        }
25.                 }
26.            }
```

Figura 1. Código fonte para multiplicação de matrizes

Quanto aos resultados obtidos, foi feito uma comparação no tempo de execução entre o *cluster* computacional, e um computador i7 da 8ª geração, para que pudesse haver uma base de melhoria estimada, como mostra a Tabela 1. O computador usado para a comparação conta com 4 núcleos físicos e 4 lógicos, e possui 16 GB de RAM.

Tempo da multiplicação no *cluster em* segundos	Tempo em segundos no computador
0.005832 (seg.)	0,015625 (seg.)
0.005949 (seg.)	0,015625 (seg.)
0.005831 (seg.)	0,015625 (seg.)
0.005817 (seg.)	0,015625 (seg.)
0.005938 (seg.)	0,015625 (seg.)

Tabela 1. Tempo em segundos da multiplicação de matrizes

Como pode ser visto, os resultados obtidos no computador foram todos iguais, tendo sido os testes realizados com os 4 núcleos. O *cluster* então se mostrou superior a um computador possuindo o dobro de processamento dos nós dele, isso se deve ao fato do *cluster* dividir as tarefas em partes menores e então resolver mais rapidamente, resultado em um menor tempo de execução.

8. Conclusão

A programação paralela híbrida pode ser considerada como uma boa forma de resolver cálculos complexos em menos tempo hábil, já que além do grande aumento de performance gerado por um *cluster* normal, apenas com as trocas de mensagens de seus nós uns com os outros, ao paralelizar também os nós de forma individual, o aumento de performance será ainda maior.

Portanto, a utilização dessas ferramentas em conjunto pode significar uma melhora na obtenção de resultados que demandem de grande poder computacional para resolver cálculos, devido as diversas quebras de equações para se chegar a um resultado, e isso se mostrou evidente de acordo com os resultados obtidos.

Referências

Alecrim, E. (2013) “Cluster: conceito e características”, https://www.infowester.com/cluster.php, Janeiro.

Bacellar, H. Viana. (2010) “Cluster: Computação de Alto Desempenho”, http://www.ic.unicamp.br/~ducatte/mo401/1s2010/T2/107077-t2.pdf, Janeiro.

Favaretto, R. M. (2018) “Uma análise de eficiência do uso das ferramentas de programação paralela OpenMP e TBB na remoção de ruídos em imagens”. Revista THEMA, vol, 15. Nº 4, pág. 1234 a 1255.

Marongiu, A., Burgio, P. e Benini, L. Supporting OpenMP on a multi-cluster embedded MPSoC. In: Microprocessors and Microsystems, 8. 2011, Amsterdam. Proceedings at Microprocess Microsyst. Amsterdam: Elsevier Science Publishers, 2011, p. 668-682.

MPI. (2012) “MPI: A Message-Passing Interface Standard Version 3.0”, Message Passing Interface Forum.