Multiplicação de Matrizes MPI

Luis Felipe Risch e Paulo Mateus Luza Alves

Departamento de Informática – Universidade Federal do Paraná (UFPR)

pmla20@inf.ufpr.br (GRR20203945) e lfr20@inf.ufpr.br (GRR20203940)

Implementação do algoritmo

O algoritmo consiste na execução de uma multiplicação de matrizes de forma paralelizada utilizando as funções fornecidas pelo MPI (Message Passing Interface). Logo, recebe duas matrizes A e B, e retorna a matriz C como resultado, tendo como objetivo realizar esta multiplicação, de tal forma que o gasto de tempo seja inferior à versão sequencial deste processo e por consequência gerando um aumento de operações em ponto flutuante por segundo (FLOPS).

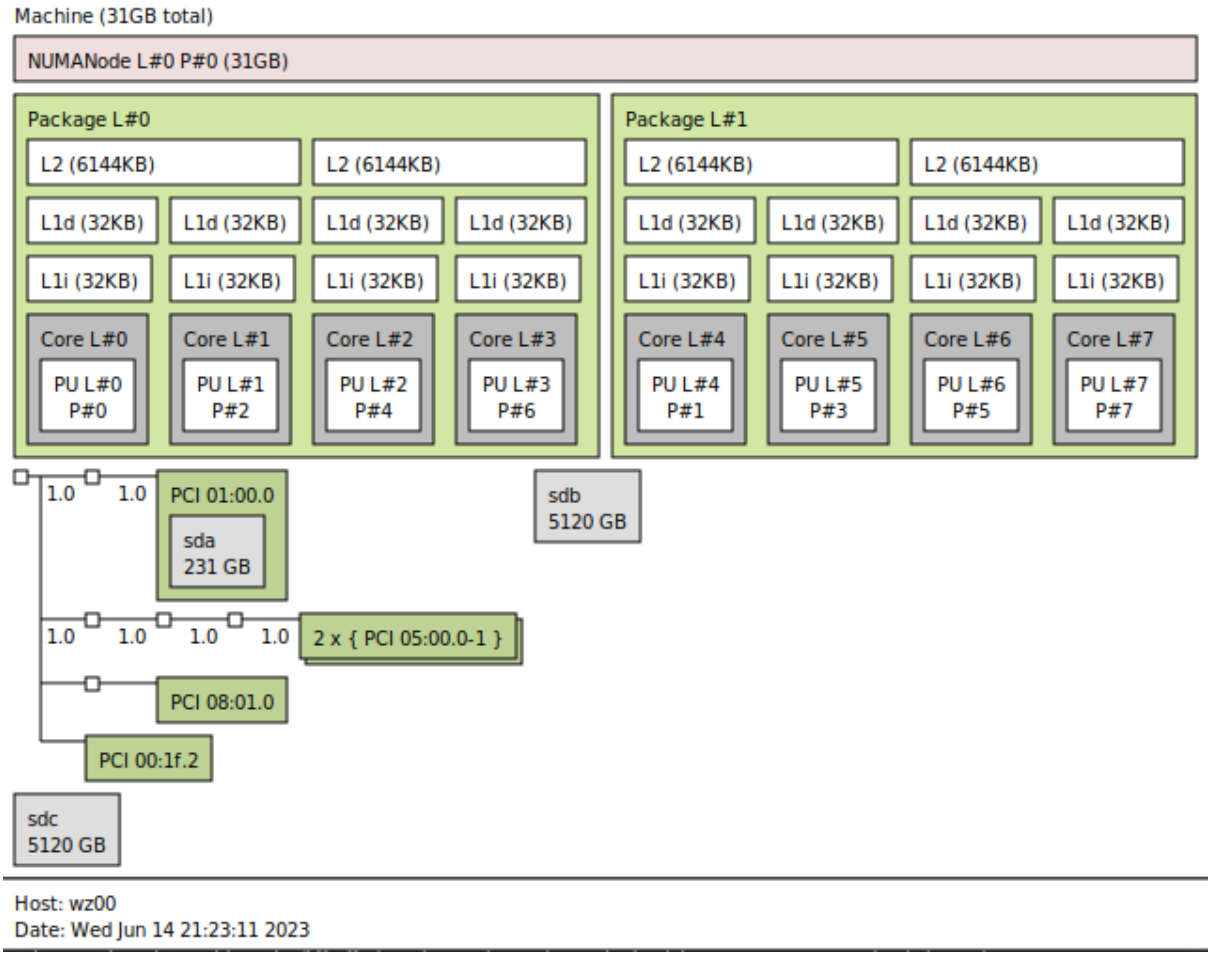
Para realizar tal tarefa, o algoritmo, como mencionado acima, utiliza as ferramentas disponibilizadas pelo MPI, sendo elas as funções *MPI_Bcast*, *MPI_Scatter* e *MPI_Gather*.

Logo abaixo, explicaremos o funcionamento da função *parallel_mult_matrix*, que recebe as matrizes como parâmetro e executa a multiplicação de forma paralela. Portanto, o algoritmo construído funciona da seguinte forma:

1. Cada *host* recebe uma cópia da matriz B, utilizando *MPI_Bcast,* passado pelo raíz;
2. Cada *host* recebe uma faixa horizontal (linhas) da matriz A, utilizando *MPI_Scatter*, passadas pela raíz;
3. Cada host, após o recebimento da matriz B e da sua respectiva faixa de A, executa a multiplicação da mesma forma que o algoritmo sequencial, porém apenas para sua região;
4. Cada host, após executar o cálculo, realiza um *MPI_Gather* para a raíz, enviando a sua faixa calculada para a matriz C.

Descrição do processador usado

Segue a imagem do processador obtido com o programa *lstopo*.



Para complementar a imagem acima, segue as informações obtidas ao rodar o programa *lscpu*:

Hostname: wz00;

Processador: Intel(R) Xeon(R) CPU E5462 @ 2.80GHz

CPU MHz: 2793,018;

L1d cache: 256 KiB;

L1i cache: 256 KiB;

L2 cache: 24 MiB;

flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ht tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts rep_good nopl cpuid aperfmperf pni dtes 64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 lahf_lm pti tpr_shadow vnmi flexpriority vpid dtherm

Descrição dos experimentos

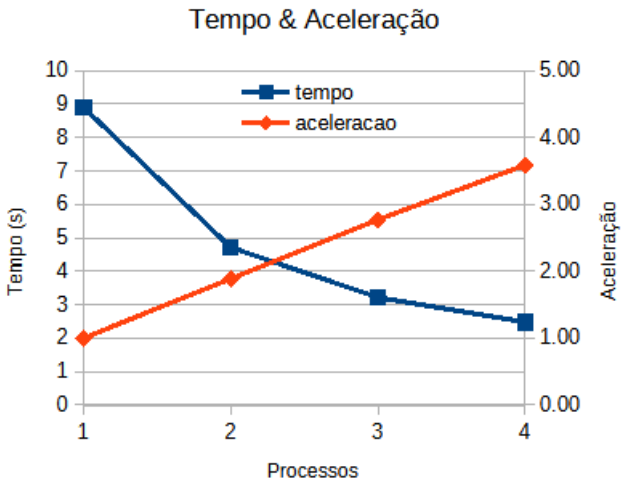
Para realizar os experimentos solicitados pela especificação do trabalho, seguimos os seguintes passos:

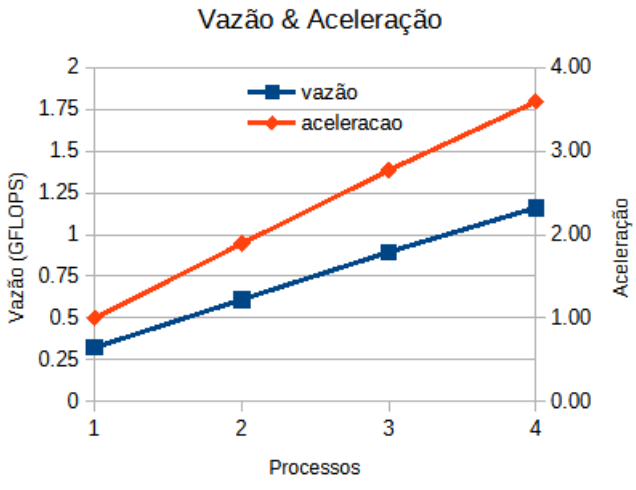
1. Agendamos 10 jobs, de forma exclusiva, para rodar o programa *mmul* em 1 nodo com os seguintes tamanhos de matrizes: *nla* = 1800; *m* = 1000; *ncb* = 800. Aguardamos a conclusão das execuções e extraímos as informações necessárias(tempo e vazão) para preencher uma planilha de dados.
2. Agendamos mais 10 jobs, de forma exclusiva, para rodar o programa *mmul* em 2 nodos com os seguintes tamanhos de matrizes: *nla* = 1800; *m* = 1000; *ncb* = 800. Novamente, esperamos que os resultados fossem computados e extraímos as informações para a planilha de dados.
3. Em seguida, agendamos 10 jobs, de forma exclusiva, para rodar o programa *mmul* em 3 nodos com os seguintes tamanhos de matrizes: *nla* = 1800; *m* = 1000; *ncb* = 800. Aguardamos os resultados e adicionamos as informações na planilha de dados.
4. Finalmente, agendamos mais 10 jobs, de forma exclusiva, para rodar o programa *mmul* em 4 nodos com os seguintes tamanhos de matrizes: *nla* = 1800; *m* = 1000; *ncb* = 800. Após a conclusão dos experimentos, extraímos as principais informações e as adicionamos à planilha de dados.

Finalizando os experimentos, obtivemos os dados e gráficos apresentados abaixo.

Tabelas de dados e gráficos - Sumarizando tempo médio, vazão média e aceleração

	Valores Médios			
Processos	**1**	**2**	**3**	**4**
Tempo(s)	**8.904562**	**4.7153876**	**3.2171709**	**2.481519**
Vazão (GFLOPS)	**0.3235311**	**0.6108311**	**0.8953664**	**1.1608208**
Aceleração	**1.00**	**1.89**	**2.77**	**3.59**





Conclusões

Como pedido na especificação, quando o algoritmo é executado com apenas um processo a versão sequencial é utilizada. Logo, para todas as demais quantidades de *hosts* temos a versão paralela em execução.

Sendo assim, podemos verificar pelas tabelas e pelos gráficos que ao paralelizar o processo obtemos um ganho expressivo de performance, ou seja, o tempo necessário é reduzido e consequentemente as medidas de vazão e aceleração aumentam, pois estas são inversamente proporcionais ao tempo.

Por exemplo, ao adicionar apenas 1 processo, ou seja, executar com 2 *hosts*, o tempo de execução diminuiu aproximadamente pela metade, e também conseguimos verificar o aumento da vazão e da aceleração na mesma proporção.

Para os casos posteriores, ou seja, 3 e 4 *hosts*, podemos verificar que o aumento não é tão expressivo, isso pode ser resultado de diversos fatores, como gargalo de processamento em cada *host*, por exemplo.