Paradigmas de Computação Paralela

Relatório de Desenvolvimento

11 de Janeiro de 2020

Conteúdo

0.1 Introdução

Este trabalho, realizado no âmbito da Unidade Curricular de Paradigmas da Computação Paralela, tem como principal objetivo avaliar a aprendizagem do paradigma de programação baseado em memória distribuīda (MPI).
O algoritmo a ser estudado em concreto é o da multiplicação de matrizes esparsas por vetores, matrizes essas que se diferem das restantes pelo facto de possuírem uma grande quantidade de elementos cujo valor é nulo. Para além da implementação do algoritmo baseado em memória distribuída, tem-se como objetivo a comparação dos resultados do mesmo com os resultados obtidos na implementação do algoritmo em memória partilhada(OpenMP), com o fim de perceber o comportamento deste em diferentes ambientes, tentando obter a melhor escalabilidade.

0.2 Descrição do Problema

O caso de estudo, em particular, tem como base matrizes quadradas (N x N) e, como referido anteriormente, matrizes esparsas são caracterizadas pela grande quantidade de valores nulos que possuem,tendo surgido várias alternativas para a sua representação, sendo uma delas o Formato COO (Coordinate List Format), a qual será também aqui usada.

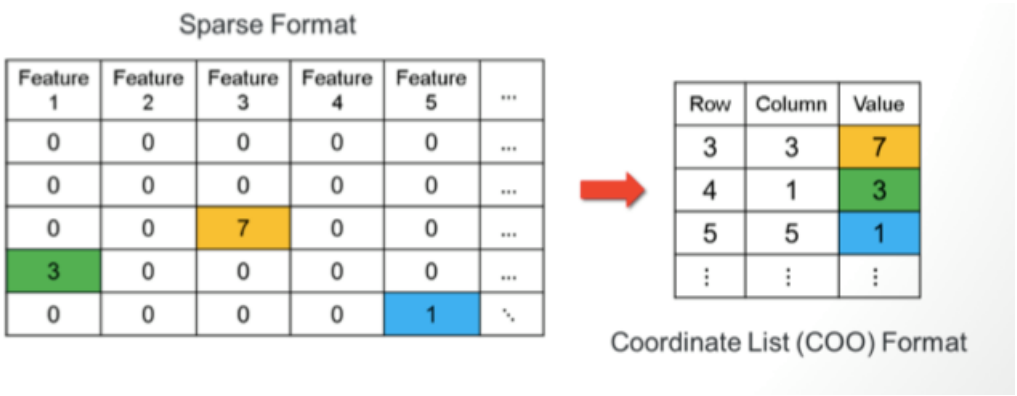


Figura 1: Representação em COO de matriz esparsa

Neste tipo de representação são utilizados 3 vetores: val, col e row, em que para cada posição i, o elemento val[i] corresponde ao elemento da posição (rows[i], cols[i]), sendo descartados os elementos cujo valor é nulo. Num ambiente em que se tem vários processos a trabalhar sobre um conjunto de dados, é necessário garantir que não existe mais do que 1 processo a trabalhar numa mesma linha da matriz, e, para tal, é essencial um planeamento antes da implementação do algoritmo.
O método usado para o problema em questão, foi o da metodologia de desenvolvimento de aplicações paralelas de Foster, também conhecido como metodologia PCAM (partição, comunicação, aglomeração e mapeamento), seguindo uma estratégia "master - slave", onde o master é responsável pela gestão das tarefas a realizar e os slave são responsáveis por executar essas tarefas.

0.2.1 Partição

```
for (unsigned i = 0; i < t; i++){
    result[rows[i]] += v[cols[i]] * val[i];
}
```

Nesta fase são identificadas as tarefas que são geridas pelo master. Da análise deste algoritmo pode verificar-se que este se baseia num único ciclo que executa a mesma operação sobre vários elementos de dados, fator que leva a que haja uma decomposição no domínio dos dados. Para evitar o problema apresentado inicialmente, em que se pode ter vários processos a trabalhar na mesma linha da matriz, a divisão dos dados pelos processos é baseado em blocos de linhas, evitando-se , assim, conflitos.

0.2.2 Comunicação

Para que a distribuição de trabalho pelos processos seja possível é necessário comunicação entre o master e os slaves. Essa comunicação é responsável pelo envio do master para os slaves da parte dos arrays sobre qual a multiplicação de matrizes opera. No final do cálculo do vetor resultado o slave devolve a parte do array sobre a qual operou ao master, concluindo o processo com a reunião de todas as partes do vetor resultado. Assim, podemos concluir que se trata de uma comunicação local, estruturada e estática pois cada tarefa apenas comunica com o master recebendo deste os vetores e devolvendo os resultados, podendo também definir a comunicação como síncrona.

0.2.3 Aglomeração

Visando a redução dos custos de comunicação, são enviadas a cada slave várias tarefas em cada mensagem (bloco de linhas de matriz).

0.2.4 Mapeamento

O mapeamento é efectuado através da distribuição dos slave pelos processadores, distribuição essa que deve ter em conta a redução dos custos de comunicação entre processos. Para tal é essencial dar prioridade aos cores da mesma máquina, usando-se o **map-by-core**, que faz uma distribuição round robin dos processos pelos cores físicos da máquina.
Em relação à distribuição de carga de trabalho, estando esta diretamente relacionada com a densidade de uma parte da matriz, foi usada uma distribuição dinâmica para fazer face ao problema do **load balancing** desproporcionado.

0.3 Implementação do algoritmo em MPI

Para o desenho do algoritmo em MPI foi decidido implementar uma versão de alocação de tarefas entre processos dinâmica.

O **load balacing**(balanceamento de carga) é uma técnica para distribuir a carga de trabalho uniformemente com o objetivo de otimizar a utilização de recursos, maximizar o desempenho, minimizar o tempo de resposta e evitar sobrecarga.

A principal finalidade desta versão é combater o **load balancing** desproporcionado entre processos o que conduz a um aumento do tempo que a *master* fica à espera que o último processo acabe, em oposição a todos os outros que já terminaram.

Para a implementação foi fixado um número de linhas a enviar aos processos,**num_lines**. Para cada um dos blocos é guardado o índice onde se inicia,**work_batch**, e o data size do mesmo,**data_size**.

Primeiramente é enviado um bloco de dados para cada processo que desenvolve os respetivos cálculos e transfere o resultado à *master*. Após receber os resultados de um *slave*, a mesma envia um novo bloco de dados para esse slave. Este processo é repetido até todos os processos serem enviados e tratados.

Para a finalização do algoritmo, os *slaves* recebem uma tag,**End**, enviada pela *Master* para concluírem a sua execução.

De seguida pode observar-se a estratégia "master-slave":

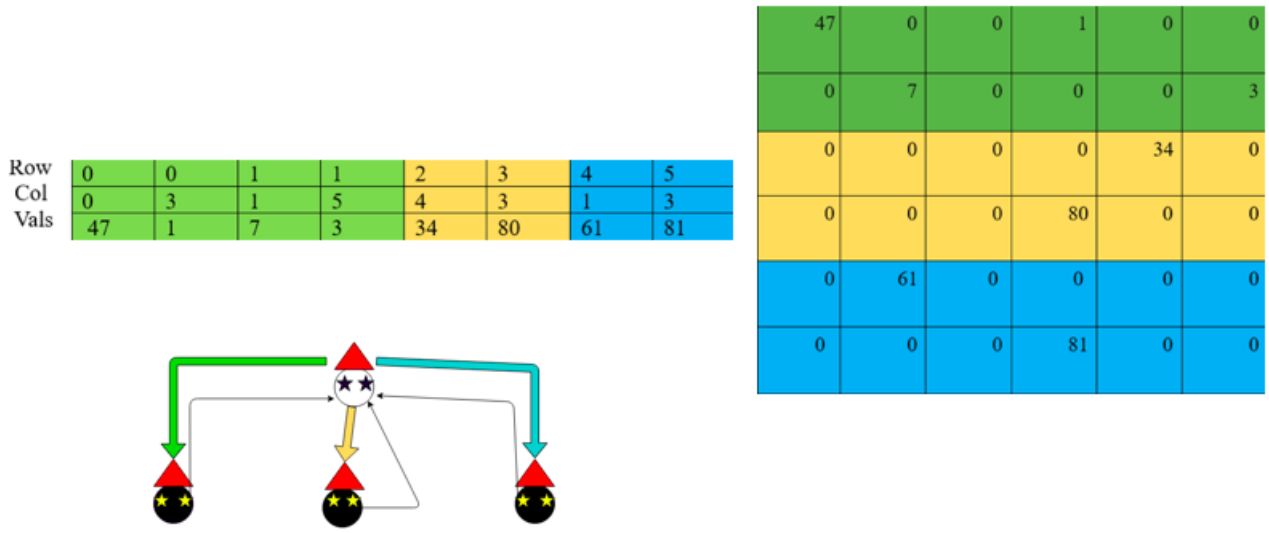


Figura 2: Master em cima a distribuir tarefas pelos slaves

0.4 Demonstração e Análise de Resultados

Após o desenvolvimento da solução, foram efetuados vários testes nos nós 641 do SeARCH, cada um composto por duas máquinas Intel E5-2650V2 cada uma com 8 cores físicos. Para a compilação foi usada a versão 4.9.0 do GCC e a versão 1.8.4 do OpenMPI.

Para a realização dos testes levou-se em consideração alguns aspetos:

- A Cache foi limpa e,deste modo, correu-se os testes em cold miss.
- A dispersão da matriz é de aproximadamente 60%

- Para cada tamanho da matriz e cada processo(2,4,8 e 16) foram realizados 5 testes, sendo o resultado usado a mediana dos mesmos.
- Os tempos foram obtidos com a ajuda da função **MPI_Wtime()** e o tempo dos resultados utilizado foi em milisegundos, numa máquina com precisão de medição de microsegundos.
- É retirada a parte que efetua a multiplicação para se obterem os tempos de comunicação.

0.4.1 Apresentação de Resultados

De seguida encontram-se os tempos dos ganhos em 2 máquinas:

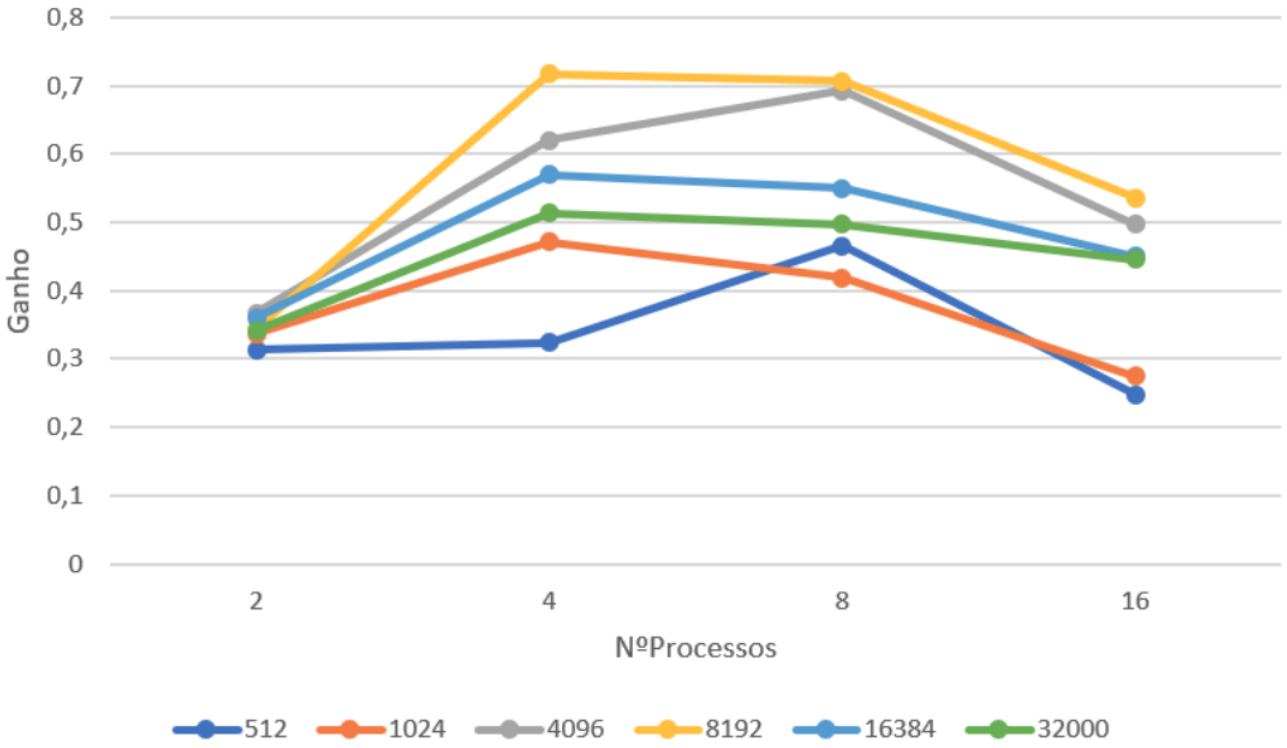
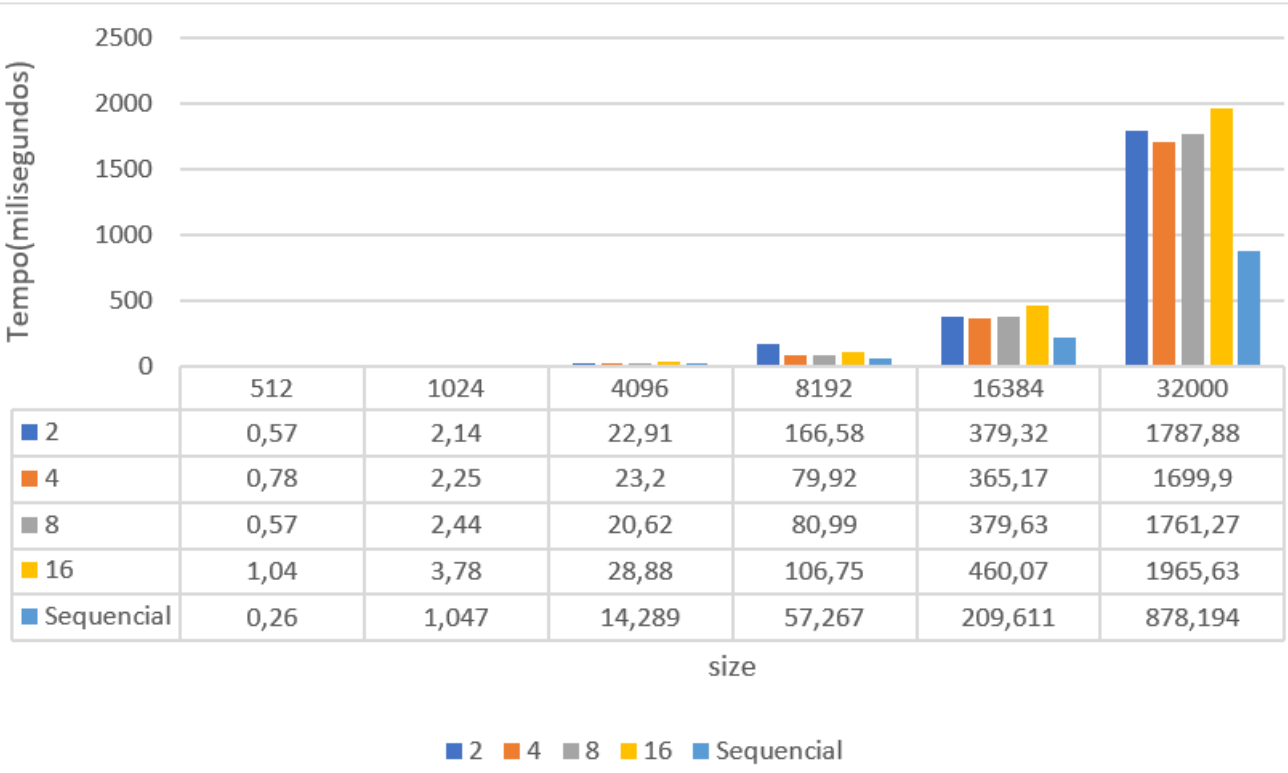


Figura 3: Ganho para mapeamento bycore

Como podemos observar é obtida pior performance para os tamanhos de 512 e 1024.Existe uma melhoria para os restantes tempos mesmo que com perdas face à versão sequencial. Conseguimos verificar que a relação entre computação e tempo de comunicação será o factor de perda na versão de tamanho 512 e 1024. As perdas nas versões paralelas indicam que existe um bottleneck na comunicação entre processos ou na distribuição de carga entre processos.



	512	1024	4096	8192	16384	32000
2	0,57	2,14	22,91	166,58	379,32	1787,88
4	0,78	2,25	23,2	79,92	365,17	1699,9
8	0,57	2,44	20,62	80,99	379,63	1761,27
16	1,04	3,78	28,88	106,75	460,07	1965,63
Sequencial	0,26	1,047	14,289	57,267	209,611	878,194

Figura 4: Tempos de comunicação

Nos tempos de comunicação apresentados em cima verificamos que o aumento do número de processos, com exceção do uso de 2 processos, conduz a uma subida do tempo de comunicação pois é necessário estabelecer e gerir comunicações com mais entidades.

Podemos assim excluir a hipótese de obtermos melhoria de performance em relação à versão sequencial devido ao elevado custo de comunicação entre processos.

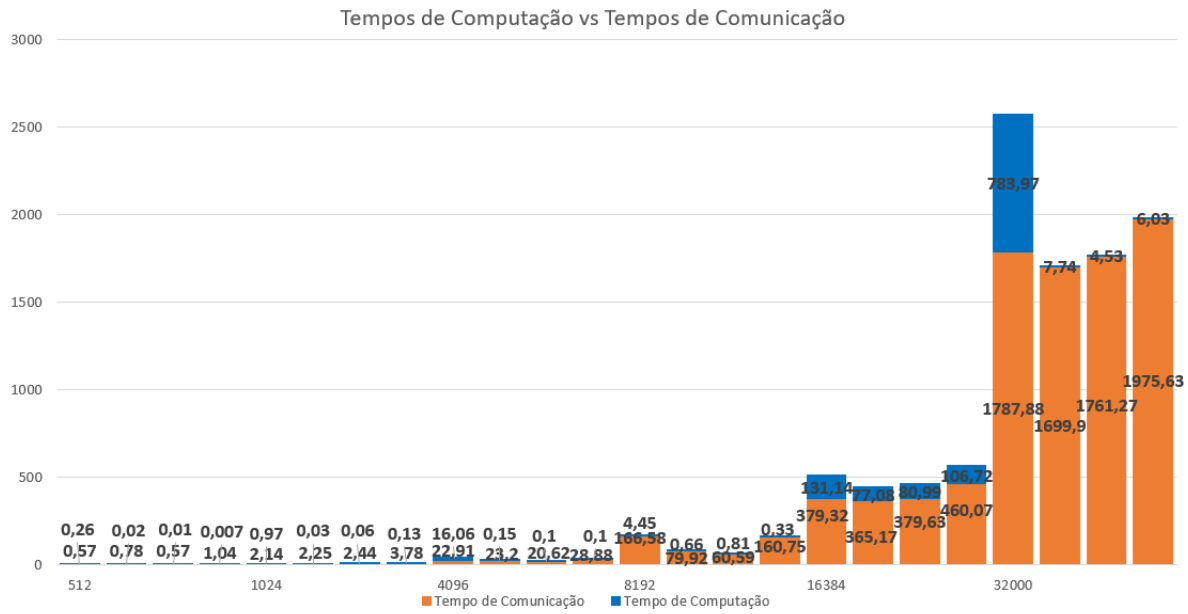


Figura 5: Tempos de comunicação vs computação

Como se pode observar os tempos de comunicação são bastante superiores aos de computação à medida que se aumenta o número de processos(2,4,8 e 16 para cada tamanho da matriz). No gráfico acima,os tempos de computação apresentam uma melhor escalabilidade com o aumento do número de processos pois existe uma melhor localidade dos dados em cache de cada processo face à versão sequencial que não tem capacidade para alocar a totalidade dos dados em cache(pode ver se versão sequencial no gráfico de cima).

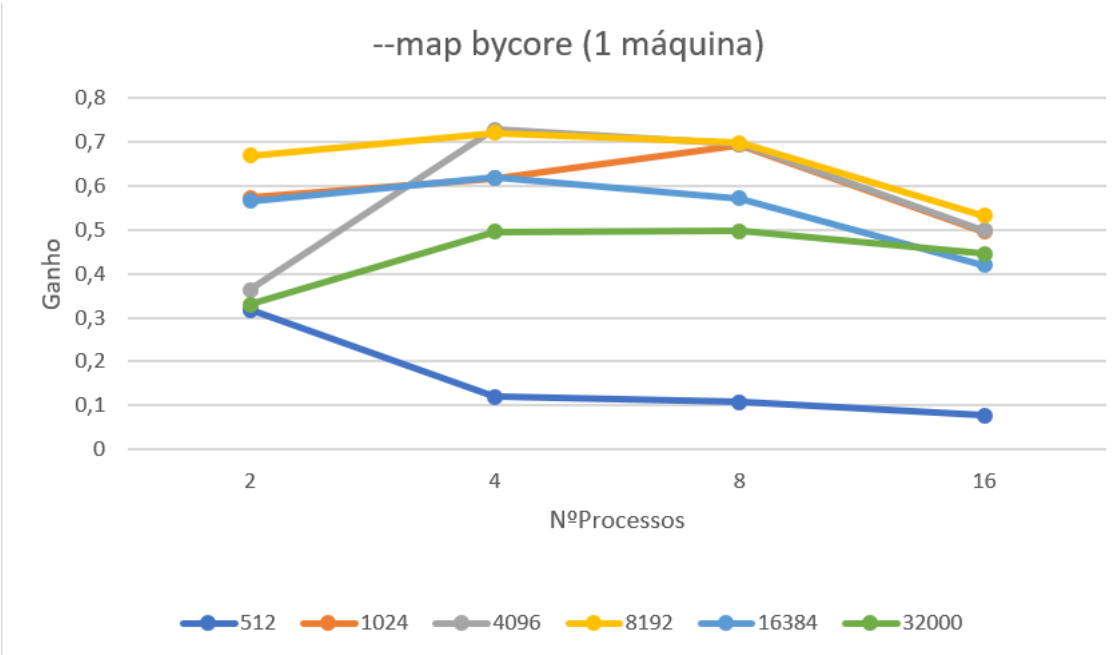


Figura 6: Ganho para uma máquina

Para uma máquina obtém-se melhor performance pois, usando duas máquinas,a ligação entre nós pode ser

uma rede utilizada para mais tráfego que não apenas o nosso.

0.5 Conclusão

Após a análise de resultados obtidos, pode concluir-se que para obtenção de uma melhor performance, num ambiente onde é explorado o paralelismo e, mais concretamente neste algoritmo em específico, o OpenMPI não é de todo a melhor solução. Isto deve-se principalmente aos tempos de comunicação demasiado elevados em comparação ao ganho obtido na computação dos dados distribuīdos pelos vários processos.

Apesar da tentativa de combater o balanceamento de carga, usando uma versão dinâmica, o preço a pagar na comunicação entre os processos foi demasiado elevado levando a que não houvesse ganhos em relação à versão sequencial.

Em suma, consideramos que adquirimos os conceitos base de computação paralela no paradigma de memória distribuída usando MPI sendo inviável para o nosso algoritmo de multiplicação de matrizes esparsas por um vetor. Seria mais praticável uma solução híbrida com OpenMP.

0.6 Autores

- Adriana Meireles a82582
- Shazhod Yusupov a82617

Capítulo 1

Anexos

		512				
Medições/NºProcessos		Sequential:	2	4	8	16
1		0,259766	0,820801	0,830078	0,548096	1,081787
2		0,258789	0,826904	0,856201	0,557129	1,080811
3		0,273926	0,833984	0,802246	0,556885	1,047852
4		0,260254	0,831787	0,792969	0,552002	1,047119
5		0,25678	0,828857	0,781982	0,557129	1,026367
	Mediana	**0,259766**	**0,828857**	**0,802246**	**0,556885**	**1,047852**
	Ganho	1	0,31340267	0,32379844	0,46646256	0,24790333

		1024				
		Sequential:				
Medições/NºProcessos			2	4	8	16
1		1,047119	3,131348	2,219971	2,501953	3,806885
2		1,044922	3,109131	2,189209	2,519043	3,82666
3		1,050049	3,088867	2,223145	2,47583	3,822998
4		1,053711	3,087158	2,283203	2,489014	3,750977
5		1,045898	3,115234	2,218018	2,516846	3,793945
	Mediana	1,047119	3,109131	2,219971	2,501953	3,806885
	Ganho		0,33678832	0,47168139	0,41852065	0,27505927

		4096				
Medições/NºProcessos		Sequential:	2	4	8	16
1		16,434082	39,054199	23,01416	20,733887	28,750244
2		16,447998	38,916992	22,985107	20,62793	28,598145
3		14,275879	38,840088	23,049072	20,573975	28,769043
4		14,283936	39,001953	23,194824	20,601074	28,795166
5		14,28833	38,974121	23,116943	20,61499	28,785156
	Mediana	14,28833	38,974121	23,049072	20,61499	28,769043
	Ganho	1	0,36661071	0,61990912	0,6931039	0,49665642

		8192				
Medições/NºProcessos		Sequential:	2	4	8	16
1		66,0979	171,033691	79,919922	80,750977	107,081787
2		57,177979	166,583984	79,088867	80,986084	106,752197
3		57,26709	166,226807	78,917969	81,800781	106,598877
4		57,154053	165,827881	80,584961	81,209961	107,127686
5		57,328857	169,978027	80,197998	80,408936	106,049072
	Mediana	57,26709	166,583984	79,919922	80,986084	106,752197
	Ganho	1	0,34377308	0,71655588	0,7071226	0,53644882

		16384				
Medições/NºProcessos		Sequential:	2	4	8	16
1		52,051025	131,142822	78,799072	80,64917	107,529053
2		56,624268	130,569092	77,083252	81,041748	106,726074
3		57,207031	131,382812	76,243164	82,145996	106,065918
4		52,197021	131,016846	77,028076	80,999756	106,302002
5		57,281006	134,520996	77,386963	80,780762	107,12915
	Mediana	56,624268	131,142822	77,083252	80,999756	106,726074
	Ganho	1	0,43177558	0,73458587	0,69906714	0,53055702

		32000				
Medições/NºProcessos		Sequential:	2	4	8	16
1		802,103027	2570,15283	1712,881	1770,363	1972,318
2		852,653809	2572,52588	1707,644	1760,552	1966,673
3		880,175781	2567,455	1716,201	1765,794	1969,7028
4		879,094727	2571,845	1707,316	1765,994	1978,235
5		878,194092	2572,03	1704,558	1762,754	1968,958
	Mediana	878,194092	2571,845	1707,644	1765,794	1969,7028
	Ganho		0,34146463	0,51427235	0,49733666	0,44585107

De seguida, segue-se os dados relativos aos tempos de comunicação

		512				
Medições/NºProcessos		Sequential:	2	4	8	16
1		0,258789	0,578125	0,769775	0,566895	1,082031
2		0,263184	0,571045	0,777832	0,559082	1,00415
3		0,261963	0,576904	0,76001	0,62793	0,99096
4		0,260254	0,565918	0,761963	0,555908	1,040039
5		0,262939	0,570068	0,768066	0,566162	1,041992
	Mediana	0,261963	**0,571045**	**0,768066**	**0,566162**	**1,040039**
	Ganho	1	0,4587432	0,3410683	0,4626997	0,2518781

		1024				
		Sequential:				
Medições/N°Processos			2	4	8	16
1		1,055176	2,145996	2,244873	2,5	3,712891
2		1,043945	2,13403	2,238037	2,440918	3,736084
3		1,046875	2,166016	2,25	2,436035	4,782715
4		1,044922	2,140869	2,282959	2,493896	5,0979
5		1,052979	2,117188	2,266113	2,406982	4,88208
	Mediana	1,046875	2,140869	2,25	2,440918	4,782715
	Ganho	1	0,4889954	0,4652778	0,4288858	0,2188872

		4096				
Medições/N°Processos		Sequential:	2	4	8	16
1		16,435059	22,996094	23,345947	20,748047	28,975098
2		16,441162	22,866943	23,267822	20,622803	28,776855
3		16,427002	23,461182	23,165039	20,725098	28,952148
4		16,437012	22,664062	23,204102	20,60791	28,883057
5		16,437012	22,912109	23,044922	20,891113	28,863037
	Mediana	16,437012	22,912109	23,204102	20,725098	28,883057
	Ganho	1	0,7173941	0,7083667	0,7930969	0,5690884

		8192				
Medições/N°Processos		Sequential:	2	4	8	16
1		66,147949	97,929199	233,48999	79,844971	105,09985
2		66,137939	97,35791	77,916016	246,4812	104,53589
3		66,057129	97,500244	78,946045	79,611084	104,69312
4		57,318115	97,493896	243,2168	79,309814	104,5398
5		57,198975	97,906006	77,690674	80,053955	104,37793
	Mediana	66,057129	97,500244	78,946045	79,844971	104,5398
	Ganho	1	0,6775073	0,8367377	0,8273173	0,631885

		32000				
Medições/N°Processos		Sequential:	2	4	8	16
1		801,12329	1778,4819	1699,8979	1759,0591	1965,625
2		800,51221	1791,8782	1698,9492	1769,4397	1964,2742
3		800,50684	1787,8821	1699,5552	1761,4661	1969,054
4		852,44165	1781,4568	1712,4539	1761,272	1967,7271
5		802,37598	1808,0491	1703,4309	1760,0339	1964,9758
	Mediana	801,12329	1787,8821	1699,8979	1761,272	1965,625
	Ganho	5408,8388	0,4480851	0,4712773	0,454855	0,4075667