# Aplicação de um Cluster de Raspberry Pi no Ensino de Processamento Paralelo: Uma Aplicação Prática

**Ian Andrade Moreira, Carlos Magno Moreira Sales, Esbel Tomás Valero Orellana**

[1]Departamento de Ciências Exatas e Tecnológicas (DCET)
Universidade Estadual de Santa Cruz (UESC)
Campus Soane Nazaré de Andrade, Rodovia Jorge Amado, Km 16
45.662-900 – Bairro Salobrinho, Ilhéus – BA – Brasil.

ianmoreira80@gmail.com, cmmsales78@gmail.com, evalero@uesc.br

***Abstract.*** *One of the biggest difficulties in training programmers for high-performance computing is the high cost of installing, operating, and maintaining HPC clusters. In this context, the option for low cost and consumption equipment has been strengthened to implement parallel architectures. The present work intends to demonstrate how, with the use of this type of tools, the study of important aspects of parallel processing can be approached. It is proposed to present the effect of network communication speed and the granularity of the problem in the performance of applications based on message passing with MPI. We used the implementation of the matrix multiplication routine DGEMM from the BLIS library to construct a case study that can be used in the classroom.*

***Resumo.*** *Um entrave importante na formação de programadores para computação de alto de desempenho está no alto custo da instalação, operação e manutenção dos clusters de **HPC**. Neste sentido tem ganhado força a opção por equipamentos de baixo custo e consumo para implementar arquiteturas paralelas. O presente trabalho se propõe demonstrar como, com a utilização deste tipo de ferramentas, pode ser abordado o estudo de aspectos importante do processamento paralelo. Pretende-se apresentar o efeito da velocidade de comunicação em rede e da granularidade do problema no desempenho de aplicações baseadas em troca de mensagens com **MPI**. Foi utilizada a implementação da rotina **DGEMM**, de multiplicação de matrizes, da biblioteca **BLIS** para construir um caso de estudo que pode ser utilizado em sala de aula.*

## 1. Introdução

A busca contínua por um maior poder de cálculo e processamento computacional tem impulsionado a computação de alto desempenho (*High Performance Computing* - **HPC**). A utilização eficiente de computadores de grande porte, ou simplesmente *clusters* de **HPC**, está gerando uma demanda, cada vez maior, pela formação e capacitação de especialistas na configuração, operação e gestão deste tipo de equipamento, assim como de desenvolvedores na área de processamento paralelo. Um entrave para a realização de ações nesse sentido está na necessidade de disponibilizar a infraestrutura computacional necessária para realizar as mesmas. Os equipamentos disponíveis para **HPC** são caros e sua operação envolve elevados custos de manutenção e de consumo de energia. Trabalhos recentes apontam para o fato de que o custo operacional dos *clusters* de **HPC** supera

rapidamente o custo para aquisição dos mesmos e propõem iniciativas para construir equipamentos com maior eficiência energética [Vasudevan et al. 2009], [Durand et al. 2014], [Filho et al. 2017].

Os avanços recentes na área de computadores de placa única tem fornecido a opção de utilizar equipamentos de baixo custo e baixo consumo, como o caso do **Raspberry Pi** ou do **Parallella**, para aplicações que precisam de grande poder de cômputo, [Matthews et al. 2016]. Na atualidade pode ser encontrada uma vasta documentação sobre a montagem e utilização de *clusters* com computadores deste tipo. Trabalhos nesta linha têm demonstrado a viabilidade de construção e utilização de pequenos *clusters* de **HPC** baseados em computadores de placa única, particularmente o **Raspberry Pi**, para diversas aplicações, [Doucet and Zhang 2017], [Matthews et al. 2018], [Kumar et al. 2019].

Um tipo de aplicação se destaca neste contexto, dado o baixo custo de instalação e operação deste tipo de equipamentos: as aplicações para de técnicas de processamento paralelo. Alguns trabalhos, como o de [Doucet and Zhang 2017], já apontam nesta direção. Contudo ainda o tema não é muito abordado, sobretudo no que se refere a utilizar as próprias limitações dos computadores de placa única para estudar aspectos específicos de computação paralela.

O presente trabalho aborda a viabilidade de utilização de um *cluster* de **HPC**, baseado em **Raspberry Pi**, para o estudo de algumas técnicas e ferramentas de processamento paralelo. Desta forma pretende-se demonstrar como as limitações de velocidade de comunicação em rede dos computadores de placa única permitem estudar o efeito do tempo de comunicação dos processos no desempenho de aplicações paralelas baseadas em troca de mensagens. O estudo de caso abordado permite também apresentar duas métricas importantes utilizadas nas avaliações de desempenho em **HPC**: tempo de execução e granularidade.

O texto está organizado de forma a apresentar, inicialmente, os aspectos do processamento paralelo que serão abordados no trabalho e sua importância, assim como as características principais do **Raspberry Pi** e da rotina de multiplicação de matrizes **DGEMM** que será utilizada como caso de estudo. Posteriormente é descrito o experimento proposto e as ferramentas utilizadas para sua realização. A apresentação e discussão dos resultados são feitas a seguir culminando com as conclusões e trabalhos futuros.

## 2. Contextualização do Problema

O paradigma de troca de mensagens é a base para a maioria das aplicações paralelas de grande porte, [Jin et al. 2011]. A execução de uma aplicação que utiliza, de forma simultânea, vários processos que colaboram para a resolução de um problema via troca de mensagens é possível graças as diversas bibliotecas que implementam o padrão *Message Passing Interface* (**MPI**). Desta forma o estudo dos recursos fornecidos pelo padrão **MPI** faz parte dos currículos da maioria dos cursos de programação paralela.

As bibliotecas **MPI** disponibilizam uma série de funções que permitem, de acordo com as necessidades do problema, utilizar mecanismos de comunicação ponto a ponto, comunicações coletivas, comunicações bloqueadas e não-bloqueadas. A implementação dos algoritmos mais eficientes para estes tipos de comunicação, de acordo com a topologia

de rede utilizada, é feita nas bibliotecas que implementam o padrão **MPI**, ficando seus detalhes ocultos ao programador que utiliza as funcionalidades das mesmas.

Para se aprofundar no uso destes mecanismos de comunicação pode ser utilizado um problema relativamente simples e bastante conhecido: a multiplicação de matrizes. A rotina de multiplicação de matrizes é amplamente difundida e simples de paralelizar. De forma especial pode-se destacar a rotina da *General Matrix Multiplication* (**GEMM**) implementada na biblioteca **BLAS** que calcula o produto.

$$C_{m,n} = \alpha A_{m,p} B_{p,n} + \beta C_{m,n}, \tag{1}$$

onde $C_{m,n}$ é uma matriz de $m$ linhas e $n$ colunas, $A_{m,p}$ é uma matriz de $m$ linhas e $p$ colunas, $B_{p,n}$ é uma matriz de $p$ linhas e $n$ colunas e $\alpha$ e $\beta$ são escalares. A implementação desta rotinas para valores de ponto flutuante de precisão dupla é denominada de **DGEMM**.

As sub-rotinas básicas de álgebra linear (*Basic Linear Algebra Subprograms*) ficaram conhecidas como **BLAS**. Criado na década de 70, o conjunto de rotinas é aplicado no desenvolvimento de inúmeras aplicações [Dongarra et al. 1985]. Atualmente existem diversas bibliotecas que implementam de forma otimizada as rotinas definidas na **BLAS**. Um exemplo importante a ser destacado é a **BLIS** que otimiza os algoritmos de forma a explorar a capacidade de utilizar operações vetoriais dos processadores. A **BLIS** também permite explorar o paralelismo em arquiteturas de memória compartilhada com abordagens que utilizam OpenMP ou Pthreads, [Willenbring and Laboratories 2015].

Uma característica relevante da rotina **GEMM** é que, dado o tamanho das matrizes envolvidas, é possível estimar a quantidade de operações de ponto flutuante envolvidas na operação (*Floating Point Operations* - **FLOP**). Para o caso, por exemplo, de $A$, $B$ e $C$ serem matrizes quadradas ($m = n = p$) temos que a quantidade de operações é da ordem de $n^3$, [Hunger 2007].

As rotinas da **BLIS** podem ser utilizadas então, em sua versão sequencial ou paralela, em conjunto com as funções para troca de mensagens do **MPI**, para implementar uma aplicação paralela que pode ser facilmente escalável [Correia et al. 2018]. Neste contexto é importante definir um conjunto de métricas que possam ser utilizadas para avaliar o desempenho das aplicações paralelas.

Utilizando como base o tempo de execução diversas métricas podem ser exploradas para caracterizar o desempenho. O tempo de execução inclui, de forma geral, o tempo dedicado às operações de cômputo ($T_{cal}$) e o tempo dedicado à comunicação entre os processos ($T_{com}$). No caso das aplicações paralelas o tempo de execução depende do tamanho do problema ($n$) e da quantidade de processadores ($p$) utilizados na execução. Desta forma pode ser definida a granularidade ($G$) como a relação entre o tempo de cômputo e o tempo de comunicação.

$$G(n,p) = \frac{T_{cal}(n,p)}{T_{com}(n,p)}. \tag{2}$$

O tempo gasto com comunicação entre os processos é um dos fatores que sobrecarregam a aplicação paralela em relação à implementação sequencial equivalente. O aumento desta sobrecarga pode prejudicar significativamente o desempenho de uma aplicação paralela comprometendo a escalabilidade da mesma, [Gustafson 1988]. Com

ajuda de um *cluster* de **HPC** baseado em **Raspberry Pi** é possível explorar mais a fundo o conceito de granularidade e o efeito do tempo de comunicação no desempenho da aplicação paralela.

Equipado com um processador ARM Cortex-A53 de quatro núcleos a 1.2GHz e com 1GB de memória RAM (LPDDR2) a 900 MHz, o **Raspberry Pi 3** permite execução de até quatro processos simultâneos para executar a rotina **DGEMM** para matrizes de até 4096 linhas por 4096 colunas. Entretanto, quando se escala a aplicação para utilizar mais do que quatro processos, com a participação de vários nós de cômputo interligados em rede, a comunicação entre eles passa a ser um elemento limitador. Testes feitos com diversos modelos de **Raspberry Pi** mostram que o envio e recepção de tráfego pela rede Ethernet, no caso do modelo **3B**, chega a ser 10 vezes mais lento que num Macbook Air com uma placa Gigabit Ethernet, [Geerling ]. Desta forma pode ser proposto um caso de estudo para avaliar o efeito do tempo de comunicação e da granularidade no desempenho da aplicação paralela que utiliza a rotina **DGEMM**.

## 3. Explorando HPC com Computadores de Placa Única

A rotina **DGEMM**, que implementa uma forma generalizada de multiplicação de matrizes dada por (1), pode ser representada graficamente de forma simples como mostra a Figura 1. Para paralelizar a aplicação é possível dividir o trabalho entre $p$ processos de maneira que cada um calcula a submatriz $C_{\overline{m},n}$, onde $\overline{m} = \frac{m}{n}$, como:

$$C_{\overline{m},n} = \alpha A_{\overline{m},p} B_{p,n} + \beta C_{\overline{m},n}. \tag{3}$$

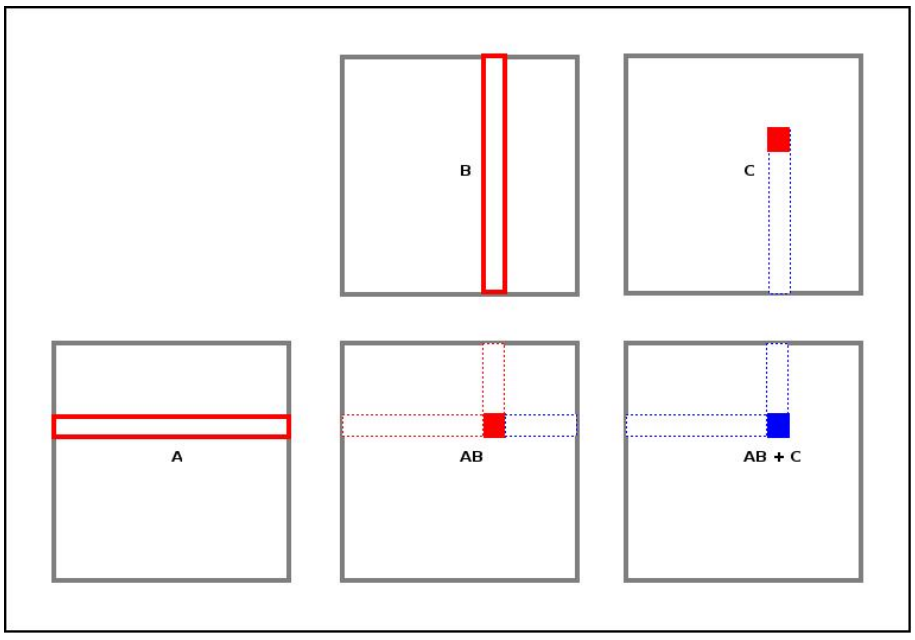


**Figura 1. Representação simplificada da multiplicação de matrizes implementada pela rotina DGEMM**

Utilizando os recursos do padrão **MPI** podem ser criados $p$ processos que são identificados por um índice que varia entre $0$ e $p-1$ e que, no padrão **MPI**, é conhecido como $rank$. Recomenda-se que, neste tipo de aplicações, as operações de entrada e saída

sejam concentradas num único processo que, neste caso, será de $rank = 0$. Este processo é encarregado de:

- Alocar e inicializar as matrizes do cálculo;
- Distribuir a matriz $B_{p,n}$
- Distribuir os fragmentos $C_{\overline{m},n}$ e $A_{\overline{m},p}$ entre todos os processos da aplicação;
- Após todos os processos concluírem sua parte do cálculo, agrupar os resultados parciais na matriz resultante $C_{m,n}$.

Neste ponto pode-se trabalhar os conceitos de comunicações ponto a ponto ou coletivas, tanto para distribuir as matrizes entre os processos quanto para coletar os resultados finais. Pode-se destacar também o fato de que ficam bem caracterizadas nesta sequência tanto as etapas que envolvem comunicação entre processos quanto a etapa de cômputo. Em relação ao cômputo, que se resume à chamada da rotina **DGEMM**, pode-se alterar ”artificialmente” o peso desta etapa simplesmente aumentando o número de chamadas à rotina.

Desta forma é possível explorar a aplicação com diferentes níveis de processamento variando a relação entre tempo de cômputo sem alterar o tempo de comunicação. Pode-se então mensurar os tempos de comunicação e de cômputo para estudar a influência da granularidade no desempenho da aplicação paralela.

## 4. Materiais e Métodos

Para a realização dos testes deste artigo foi utilizado um equipamento construído, como parte de um projeto inicial, que se propôs demonstrar a viabilidade de montar, configurar, gerenciar e operar um *cluster* de **HPC**, utilizando **Raspberry Pi**, [Moreira et al. 2018]. Como requisito para este equipamento foi colocado que ele fosse capaz de reproduzir o ambiente de operação e gerenciamento semelhante ao que é utilizado no *cluster* **CACAU**, atualmente em operação no Núcleo de Biologia Computacional e Gestão de Informações Biotecnológicas (**NBCGIB**) da Universidade Estadual de Santa Cruz (**UESC**), [NBCGIB ]. O resultado é um equipamento altamente portátil que pode ser facilmente montado e utilizado em sala de aula para fins didáticos, veja na Figura 2.


(a) (b)

**Figura 2. *clusters* de HPC utilizando 8 Raspberry Pi 3B: (a) Estrutura do cluster, (b) Cluster em funcionamento**

As aplicações foram desenvolvidas em **C** e compiladas com o GCC versão 6.3.0. A biblioteca **MPI** utilizada foi a **OpenMPI**, versão 3.3.1. Para simplificar a manipulação

de matrizes foi utilizada a biblioteca **GNU** *Scientific Library* (**GSL**) versão 2.4, que disponibiliza uma serie de recursos com esta finalidade. O código foi linkado com a versão da biblioteca BLIS que foi compilada no *cluster* a partir do código fonte disponibilizado no repositório GitHub do projeto (https://github.com/flame/blis).

Foi abordado apenas o uso de mecanismos de comunicação coletiva em cada uma das etapas, como mostra a Tabela 1. Os tempos foram medidos inserindo uma chamada à função que retorna a hora da máquina antes e depois da chamada a cada uma das rotinas de comunicação e cálculo representadas na Tabela 1. Desta forma o tempo de comunicação pode ser medido como a soma dos tempos das etapas 1, 2, 3 e 5. Já o tempo de de cômputo está relacionado à etapa 4.

**Tabela 1. Etapas de comunicação e cômputo da aplicação paralela.**

| Etapa | Ação | Funções Utilizadas |
|---|---|---|
| 1 | Distribuir a matriz $B_{p,n}$ | `MPI_Bcast(...)` |
| 2 | Distribuir os fragmentos $C_{\overline{m},n}$ | `MPI_Scatter(...)` |
| 3 | Distribuir os fragmentos $A_{\overline{m},p}$ | `MPI_Scatter(...)` |
| 4 | Calculando os fragmentos $C_{\overline{m},n}$ | `gsl_blas_dgemm(...)` |
| 5 | Agrupar os resultados na matriz $C_{m,n}$ | `MPI_Gather(...)` |

O experimento proposto então consiste basicamente em duas etapas. Na primeira etapa é executado o algoritmo **DGEMM** para diferentes tamanhos de matrizes quadradas variando de 512 a 3584. Para cada tamanho de matriz foram feitas 3 repetições do cálculo e calculada a média de tempo de cada uma das 5 etapas descritas na Tabela 1. A aplicação foi executada com 4 processos, em uma única estação do cluster, e com 8 processos, utilizando duas estações.

A segunda etapa consistiu em alterar a carga de trabalho da aplicação introduzindo uma estrutura de repetição na etapa 4 do processo. Desta forma conseguiu-se um aumento do tempo de cômputo de forma a alterar a granularidade do problema. Nesta segunda etapa a aplicação foi executada com 8 processos.

Em todos os casos pode ser calculada uma estimativa desempenho ($D(n,p)$) em operações de ponto flutuante por segundo (**FLOP/s**). Com esta finalidade pode se estimar o número de operações de ponto flutuante necessárias para fazer o cálculo com cada tamanho de matriz e dividir pelo tempo total ($T$). O tempo total, neste caso, inclui o tempo de comunicação e de cômputo. Apesar de não fazer parte do processo de cálculo, as operações de comunicação compõem a parte da sobrecarga introduzida na aplicação paralela e devem ser consideradas na hora de estimar o desempenho. Desta forma:

$$D(n,p) = \frac{n^3}{T_{cal} + T_{com}} \tag{4}$$

## 5. Resultados e Discussão

Na primeira etapa do experimento foi executada a aplicação com 4 e 8 processos. Os resultados com 4 processos podem ser observados na Tabela 2. Dos resultados podemos observar como o tempo de comunicação aumenta em menor proporção que o tempo de cômputo, em relação ao tamanho do problema. Desta forma a granularidade aumenta e

junto com ela o desempenho. Outras métricas, como o *speedup* ou a eficiência, poderiam dar uma ideia melhor da evolução do desempenho. Entretanto como estas métricas são calculadas em relação ao tempo da versão sequencial, não foram utilizadas neste trabalho.

**Tabela 2. Resultados da aplicação rodando com 4 processos para diferentes tamanhos de matrizes quadradas. As colunas representam: o tamanho das matrizes quadradas ($n$), o tempo de *broadcast* da matriz $B$ ($T_{BcB}$), o tempo de *scatter* da matriz $A$ ($T_{ScA}$) e da matriz $C$ ($T_{ScC}$), o tempo de cômputo ($T_{Cal}$), o tempo de *gather* da matriz $C$ ($T_{GtC}$), o tempo de comunicação ($T_{Com}$), a granularidade ($G(n,4)$) e o desempenho ($D(n,4)$)**

| $n$ | $T_{BcB}$ (s) | $T_{ScA}$ (s) | $T_{ScC}$ (s) | $T_{Cal}$ (s) | $T_{GtC}$ (s) | $T_{Com}$ (s) | $G(n,4)$ | $D(n,4)$ (GFLOP/s) |
|---|---|---|---|---|---|---|---|---|
| 512 | 0,007 | 0,005 | 0,005 | 0,141 | 0,004 | 0,020 | 6,963 | 1,663 |
| 1024 | 0,029 | 0,017 | 0,018 | 1,160 | 0,024 | 0,087 | 13,322 | 1,723 |
| 1536 | 0,065 | 0,038 | 0,043 | 3,766 | 0,089 | 0,235 | 16,036 | 1,813 |
| 2048 | 0,116 | 0,067 | 0,078 | 8,796 | 0,215 | 0,476 | 18,492 | 1,854 |
| 2560 | 0,186 | 0,104 | 0,129 | 17,203 | 0,431 | 0,850 | 20,250 | 1,859 |
| 3072 | 0,261 | 0,149 | 0,182 | 28,957 | 0,663 | 1,257 | 23,045 | 1,920 |
| 3584 | 0,357 | 0,230 | 0,254 | 46,291 | 1,088 | 1,928 | 24,011 | 1,910 |

Na Tabela 3 podem ser constatados os resultados obtidos quando executada a aplicação com 8 processos. Neste caso as limitações de comunicação em rede do **Raspberry Pi 3B** potencializam os efeitos da comunicação em rede. Como pode ser visto a granularidade cai para menos de 1, o que significa que o tempo de cômputo é inferior ao tempo de comunicação. O desempenho mostra que, mesmo utilizando o dobro de processos, o resultado é quase sempre inferior, atingindo uma melhora pouco significativa apenas para as matrizes maiores.

**Tabela 3. Resultados da aplicação rodando com 8 processos para diferentes tamanhos de matrizes quadradas. As colunas representam: o tamanho das matrizes quadradas ($n$), o tempo de *broadcast* da matriz $B$ ($T_{BcB}$), o tempo de *scatter* da matriz $A$ ($T_{ScA}$) e da matriz $C$ ($T_{ScC}$), o tempo de cômputo ($T_{Cal}$), o tempo de *gather* da matriz $C$ ($T_{GtC}$), o tempo de comunicação ($T_{Com}$), a granularidade ($G(n,8)$) e o desempenho ($D(n,8)$)**

| $n$ | $T_{BcB}$ (s) | $T_{ScA}$ (s) | $T_{ScC}$ (s) | $T_{Cal}$ (s) | $T_{GtC}$ (s) | $T_{Com}$ (s) | $G(n,8)$ | $D(n,8)$ (GFLOP/s) |
|---|---|---|---|---|---|---|---|---|
| 512 | 0,532 | 0,255 | 0,090 | 0,041 | 0,093 | 0,970 | 0,043 | 0,266 |
| 1024 | 0,687 | 0,431 | 0,371 | 0,331 | 0,375 | 1,864 | 0,178 | 0,979 |
| 1536 | 1,710 | 0,877 | 0,840 | 1,291 | 0,861 | 4,288 | 0,301 | 1,300 |
| 2048 | 3,114 | 1,571 | 1,478 | 3,357 | 1,517 | 7,680 | 0,437 | 1,557 |
| 2560 | 4,789 | 2,410 | 2,379 | 6,942 | 2,255 | 11,833 | 0,587 | 1,788 |
| 3072 | 6,967 | 3,304 | 3,326 | 12,260 | 3,502 | 17,098 | 0,717 | 1,976 |
| 3584 | 9,430 | 4,506 | 4,524 | 20,146 | 5,174 | 23,634 | 0,852 | 2,104 |

A segunda etapa do experimento visa explorar como a alteração na granularidade via aumento do tempo de cômputo se reflete no desempenho da aplicação. Neste sentido pode-se introduzir um fator de incremento no tempo de computação ($Up$) que definirá o

número de vezes que será chamada a rotina DGEMM. O número de operações de ponto flutuante é afetado também pelo fator $Up$. Com base nos resultados obtidos na primeira etapa foi estabelecido uma expressão para calcular $Up$ que preserve o valor da granularidade para o melhor caso obtido com 4 processos. Desta forma:

$$Up = D(3584, 4)\frac{T_{com}(3584, 4)}{T_{cal}(3584, 4)} \tag{5}$$

Na Tabela 4 se apresentam os resultados de desempenho com as novas condições de granularidade impostas com o aumento da carga de trabalho. A terceira coluna da tabela mostra a razão entre o desempenho com 4 e com 8 processos. Esta razão pode ser interpretada como um fator de escalabilidade para este caso que tende a ser ótimo na medida que se aproxima de 2. Neste caso as matrizes maiores, novamente, apresentaram os melhores resultados atingindo um desempenho na ordem de 3,7 **GFLOP/s**

**Tabela 4. Resultados da aplicação, com fator de incremento de cômputo, rodando com 8 processos para diferentes tamanhos de matrizes quadradas.**

| $n$ | $D(n, 8)$ (**GFLOP/s**) | $D(n, 8)/D(n, 4)$ |
|---|---|---|
| 512 | 2,561 | 1,540 |
| 1024 | 3.175 | 1,842 |
| 1536 | 3.453 | 1,905 |
| 2048 | 3.559 | 1,920 |
| 2560 | 3.563 | 1,916 |
| 3072 | 3.743 | 1,950 |
| 3584 | 3.683 | 1,928 |

## 6. Conclusões

Os experimentos conduzidos neste artigo permitem explorar, com base nas limitações de velocidade de comunicação das **Raspberry Pi**, a granularidade como uma importante métrica para avaliar as características de uma implementação paralela. A utilização de outras métricas e o uso de diferentes mecanismos de comunicação podem fazer enriquecer ainda mais este experimento, visando uma melhor compreensão das técnicas de processamento paralelo.

A escolha do fator de sobrecarga $Up$ seguiu um critério específico neste exemplo, o que pode ser alterado em outras experiências. A realização do mesmo teste envolvendo 16 ou 32 processos, capacidade máxima do *cluster* atual, podem alterar a escolha deste fator.

***Agradecimentos***

A pesquisa contou com o apoio do Núcleo de Biologia Computacional e Gestão de Informações Biotecnológicas (**NBCGIB**) da UESC e do Centro de Pesquisa Desenvolvimento e Inovação (**CEPEDI**).

## Referências


Correia, A. M., Lerche, E. A., Eester, D. V., Segundo, G. S. A., and Orellana, E. T. V. (2018). Aprimoramento do Código CYRANO de Simulação de Aquecimento de Pasma por Ondas RF Utilizando Técnicas de Processamento Paralelo. *Revista Mundi Engenharia, Tecnologia e Gestão*, 3(3):1–14.

Dongarra, J. J., Croz, J. D., Hammarling, S., and Hanson, R. J. (1985). A proposal for an extended set of fortran basic linear algebra subprograms. *SIGNUM Newsl.*, 20(1):2–18.

Doucet, K. and Zhang, J. (2017). Learning Cluster Computing by Creating a Raspberry Pi Cluster.

Durand, Y., Carpenter, P. M., Adami, S., Bilas, A., Dutoit, D., Farcy, A., Gaydadjiev, G., Goodacre, J., Katevenis, M., Marazakis, M., Matus, E., Mavroidis, I., and Thomson, J. (2014). EUROSERVER: Energy efficient node for European micro-servers. In *Proceedings - 2014 17th Euromicro Conference on Digital System Design, DSD 2014*, pages 206–213.

Filho, S. E. A., Burlamaqui, A. M. F., Aroca, R. V., and Goncalves, L. M. G. (2017). NPi-cluster: A low power energy-proportional computing cluster architecture. *IEEE Access*, 5:16297–16313.

Geerling, J. Networking benchmarks. `https://www.pidramble.com/wiki/benchmarks/networking`. Acesso em 7 de Março de 2019.

Gustafson, J. L. (1988). Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533.

Hunger, R. (2007). Floating Point Operations in {Matrix-Vector} Calculus. Technical Report TUM-LNS-TR-05-05, Munich University of Technology.

Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., and Chapman, B. (2011). High performance computing using mpi and openmp on multi-core parallel systems. *Parallel Computing*, 37(9):562 – 575. Emerging Programming Paradigms for Large-Scale Scientific Computing.

Kumar, D., Memon, S., and Thebo, L. A. (2019). Design, Implementation & Performance Analysis of Low Cost High Performance Computing (HPC) Clusters. *2018 12th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–6.

Matthews, S. J., Adams, J. C., Brown, R. A., and Shoop, E. (2018). Portable Parallel Computing with the Raspberry Pi. pages 92–97.

Matthews, S. J., Blaine, R. W., and Brantly, A. F. (2016). Evaluating Single Board Computer Clusters for Cyber Operations. In *2016 INTERNATIONAL CONFERENCE ON CYBER CONFLICT (CYCONUS)*, number Octover 2016, pages 1–8.

Moreira, I. A., Sales, C. M. M., and Orellana, E. T. V. (2018). Desenvolvimento de Cluster de HPC de baixo custo e baixo consumo. In *5º Simpósio de Ensino, Extensão, Inovação, Pesquisa e Pós-Graduação e 24º Seminário de Iniciação Científica*, Ilhéus.

NBCGIB. Centro de armazenamento de dados e computação avançada da uesc. `http://nbcgib.uesc.br/cacau/`. Acesso em 7 de Março de 2019.

Vasudevan, V., Franklin, J., Andersen, D., Phanishayee, A., Tan, L., Kaminsky, M., and Moraru, I. (2009). FAWNdamentally Power-efficient Clusters. In *12th Workshop on Hot Topics in Operating Systems (HotOS XII)*, pages 1–14, Monte Verita.

Willenbring, J. M. and Laboratories, S. N. (2015). Replicated Computational Results ( RCR ) Report for " BLIS : A Framework for Rapidly Instantiating BLAS Functionality ". 41(3):0–3.