Arquitetura MultiGPU Distribuída Utilizando rCUDA: Um Estudo de Caso através da Avaliação de Desempenho de Estênceis Computacionais

Nielsen A. Gonçalves[1], Josivaldo de S. Araújo[1], Raimundo Viegas Jr.[1]

[1]Instituto de Ciências Exatas e Naturais – Universidade Federal do Pará (UFPA)
Caixa Postal 479 – 66.075-110 – Belém – PA – Brasil

{nielsen,josivaldo,rviegas}@ufpa.br

Resumo. *O desenvolvimento de plataformas computacionais heterogêneas tem tornado mais acessíveis os recursos capazes de aumentar o desempenho de aplicações, porém, tais recursos, por vezes, não oferecem a escalabilidade desejada, tornando necessário o uso de soluções distribuídas. No entanto, a implementação de soluções de computação heterogênea distribuída torna necessária a combinação de ferramentas de programação específicas, o que aumenta a complexidade e dificulta a implementação. O presente trabalho apresenta a utilização de uma arquitetura MultiGPU distribuída utilizando o rCUDA, bem como, os ganhos de desempenho obtidos por meio da mesma, em uma aplicação baseada em estênceis computacionais.*

Abstract. *The development of heterogeneous computing platforms has become resources capable of increasing application performance more accessible. However, such resources often do not provide the desired scalability in order to become necessary the use of distributed solutions. Nevertheless, the implementation of distributed heterogeneous computing solutions has made necessary the combination of specific programming tools, which increases complexity and makes implementation difficult. The present work presents the use of a distributed MultiGPU architecture using rCUDA, as well as the performance gains obtained through it, in an application based on computational estênceis.*

1. Introdução

O aprimoramento das técnicas tradicionais, baseadas na evolução das CPUs (*Central Processing Units* - Unidade Central de Processamento) chegou ao seu limite em termos de eficiência energética (FLOPs/Watt), demandando, com isso, o desenvolvimento de novas alternativas computacionais. A computação baseada em sistemas heterogêneos, como as GPUs (*Graphical Processing Unit* - Unidade Gráficas de Processamento), popularizou-se rapidamente, ao oferecer dispositivos de baixo consumo de energia e níveis massivos de paralelismo [Hwu, 2015].

Porém, o desenvolvimento de aplicações é limitado pela complexidade envolvida no processo de compreensão dos detalhes das arquiteturas, de modo a adaptar as abordagens originais para seus equivalentes mais eficientes. Essa limitação, torna necessário o desenvolvimento de técnicas que permitam "habilitar mais usuários a obter vantagens dessas arquiteturas sem a necessidade de conhecimento detalhado do hardware" [Reyes et al., 2012].

O presente trabalho está organizado da seguinte forma: Na seção 2 serão apresentados os trabalhos relacionados. Na seção 3 será apresentado o método de estênceis computacionais. A seção 4 discorrerá sobre as tecnologias utilizadas na implementação: A arquitetura CUDA e a ferramenta rCUDA. A Proposta da Arquitetura Distribuída aparece na seção 5. Já os Resultados Experimentais dos testes implementados serão apresentados na seção 6. Na seção 7 serão expostas as conclusões e os possíveis trabalhos futuros.

2. Trabalhos relacionados

A utilização de processadores gráficos de propósito geral em aplicações científicas tem sido o alvo de diversos trabalhos. Lorenzoni et al. [2017] apresentam melhorias no desempenho e na eficiência energética de aplicações baseadas em estênceis, pela otimização do uso do subsistema de memória de placas GPUs.

Experimentos multiGPU utilizando rCUDA são apresentados por Reaño et al. [2015], desenvolvedores do *framework*, utilizando uma implementação do algoritmo de Montecarlo disponibilizada com a instalação do kit de desenvolvimento CUDA.

Wang [2014] apresenta o uso de ferramentas de simulação de dinâmica de fluidos utilizando GPUs, aplicadas à avaliação das condições da aerodinâmica de aeronaves, com o objetivo de melhorar a eficiência das mesmas, reduzindo o consumo de combustível necessário à propulsão, diminuindo consequentemente o impacto ambiental. O uso de GPUs reduz o tempo de processamento, permitindo que a simulação utilize estênceis computacionais de alta ordem, melhorando a precisão dos modelos.

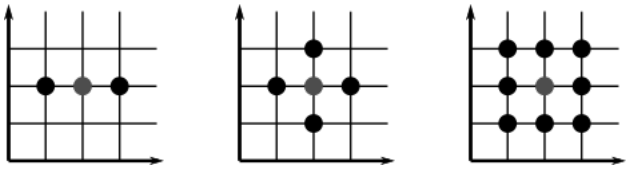
Santos et al. [2013] apresenta o uso de GPUs aplicadas ao processamento de tráfego em redes de alta velocidade, demonstrando o aumento na vazão de processamento para o sistema apresentado. O trabalho indica que o uso de GPUs, apesar de alguns problemas relativos ao *overhead* de entrada e saída, apresenta ao fim uma capacidade de gerar uma vazão de dados superior em relação às implementações de análise de tráfego baseadas em CPU.

O *framework* rCUDA, e seu uso é apresentado em Duato et al. [2011] e Reaño et al. [2013] como uma alternativa para aumentar o aproveitamento de GPUs em clusters de Alto desempenho. Ambos demonstram que o *framework* pode ser utilizado para diminuir os custos com a aquisição e o consumo de energia em clusters, ao tornar um conjunto de GPUs disponível em um nó acessível aos demais por meio da rede.

3. Estênceis Computacionais

Os estênceis computacionais podem ser definidos como operadores aplicados sobre cada um dos elementos de uma matriz, de modo a atualizar seu valor baseando-se nos valores dos elementos vizinhos, utilizando um padrão fixo [Yang and Guo, 2005]. A operação é normalmente repetida em sucessivos instantes de tempo, a partir de uma condição inicial definida em todos os elementos da matriz, de modo a simular a difusão ou outro processo físico ao longo do tempo[Rahman, 2013].

A Figura 1 apresenta graficamente três formatos de estênceis comumente utilizados em aplicações científicas: Um estêncil de 3 pontos, 1(a); o estêncil de Von Neuman, com 5 pontos, 1(b); e o estêncil de Moore, com 9 pontos, 1(c).



(a) 3 pontos (b) Von Neuman (c) Moore

Figura 1. Formatos de estênceis comumente utilizados

Operações com estênceis são utilizadas em diversas aplicações da computação científica. Exemplos, incluem, algoritmos empregados no processamento de imagens, nos quais as células são pixels; métodos numéricos utilizados para calcular soluções aproximadas de equações diferenciais parciais sobre um domínio físico, no qual as células de caso são pequenas regiões contíguas do domínio e simulações de autômatos celulares complexos, dos quais Jogo da Vida de Conway é um exemplo simples bastante conhecido [Casanova et al., 2008].

Em algumas das aplicações citadas, podem ocorrer cenários nos quais o domínio é grande o suficiente para extrapolar os limites de memória compartilhada disponível, tornando necessária uma implementação em memória distribuída.

Em termos de paralelismo, o *loop* de iterações de aplicação do estêncil não é paralelizável, uma vez que ocorre dependência de valores entre as iterações, porém, o cálculo de cada elemento é independente dentro da mesma iteração, permitindo um elevado nível de paralelismo, o que o torna um algoritmo apropriado à execução em GPU. A complexidade computacional de uma operação de estêncil pode ser expressa como $\mathcal{O}(k \cdot N^2)$, dado um número k de iterações sobre uma matriz de ordem N [Cecilia et al., 2012].

4. Utilização de GPUs em Ambiente distribuído

A utilização distribuída de GPUs é uma técnica utilizada em diversos sistemas de alto desempenho e serve para aumentar a escalabilidade das aplicações. Neste trabalho foi utilizada uma plataforma NVIDIA CUDA e o software rCUDA para obter uma arquitetura multiGPU em ambiente distribuído.

4.1. CUDA

A *Compute Unified Device Architecture* (CUDA) fornece uma plataforma de computação paralela que permite uma programação, em processadores gráficos, menos complexa, por meio de um conjunto de instruções próprias para a arquitetura GPGPU (*General Purpose Graphics Processing Unit*) [Cook, 2013]. Dessa forma, os trechos de código que correspondem ao paradigma SPMD (*Single Program Multiple Data*) são implementados em uma função denominada *kernel*. Durante a execução, os dados a serem processados são copiados para a memória da GPU e a operação definida no *kernel* é aplicada em simultâneo a uma quantidade massiva de dados por meio de múltiplas *threads*, o que permite a redução do tempo de processamento. Ao final do processamento, os dados referentes aos resultados são transferidos para a memória principal do sistema *host* [Rauber and Rünger, 2013].

Em termos lógicos, CUDA apresenta as *threads* agrupadas em *Blocks*. Os *Blocks*,

por sua vez, são organizados em *Grids*, desse modo, é necessário, ao executar um *kernel* CUDA, estabelecer quantos *Blocks* de *threads* irão executar a mesma operação, e quantas *threads* comporão cada *Block*. Apenas *threads* de um mesmo *Block* são capazes de trocar informações em memória compartilhada entre si. Os limites de tamanho dos *Blocks* e *Grids* são estabelecidos de acordo com as especificações de hardware de cada placa, conforme ilustrado na Figura 2.

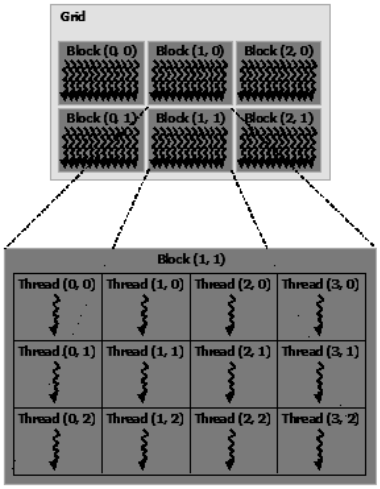


Figura 2. Arquitetura CUDA - Fonte: [NVIDIA, 2013]

Fisicamente, cada GPU é composta por um conjunto de *Streaming Multiprocessors* (SM ou SMX). Cada SM dispõe de um espaço de memória de alta velocidade de acesso, que pode ser compartilhada dentro dos *Blocks* da *Grid* entre as *threads* que compõem o respectivo *Block*. Uma representação esquemática da arquitetura de um sistema GPU/CPU é apresentado na Figura 3, na qual são apresentados os SMs, as estruturas de cache, e a possível conexão de múltiplas GPUs em um mesmo *host*, por meio de barramento.

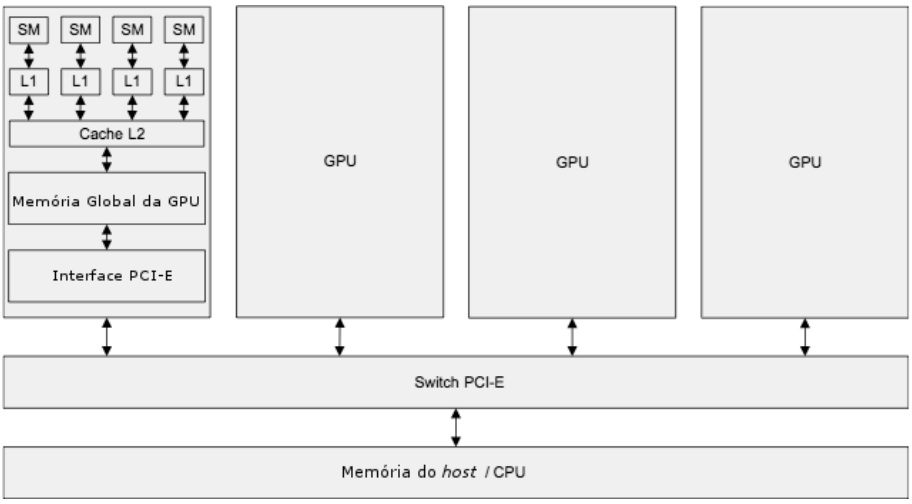


Figura 3. Representação de uma GPGPU - Adaptado de [Cook, 2013]

4.2. rCUDA

Desenvolvido por um grupo de pesquisa da *Universitat Politecnica de Valencia*, o rCUDA consiste em um *framework* para virtualização remota de GPUs [rCUDA Team, 2016]. A solução permite que o acesso remoto a placas gráficas compatíveis com CUDA seja feito de forma transparente à aplicação.

A ferramenta funciona por meio de uma arquitetura cliente/servidor. No cliente, as bibliotecas de execução do CUDA são substituídas pelas versões rCUDA, que recebem as requisições a recursos da GPU e enviam a um servidor. O servidor correspondente a cada placa gráfica virtual é definido por meio de variáveis de ambiente. O servidor rCUDA executa um *daemon*, que entrega as requisições à GPU e responde ao cliente.

A solução suporta a utilização de redes baseadas nas arquiteturas TCP/IP e Infiniband®.

A proposta original da aplicação é disponibilizar GPUs CUDA de forma concorrente entre dispositivos, desacoplando-as dos *hosts* nos quais estão instaladas. Segundo [rCUDA Team, 2014], o objetivo do *framework* é prover três tipos de cenários:

- Aumentar a taxa de utilização dos nós dotados de GPUs em *Clusters* e *Datacenters*
- Oferecer acesso compartilhado de poucas GPUs de alto desempenho, disponibilizando-as a equipamentos de baixo custo em laboratórios acadêmicos
- Habilitar o acesso a GPUs a partir de máquinas virtuais

5. Arquitetura MultiGPU Distribuída Proposta

A arquitetura distribuída proposta, foi definida em um ambiente de testes composto por três computadores *desktop* de configuração idêntica:

- 1 Processador Intel® Core i5 3.10GHz
- 8 GB de memória RAM
- 1 Placa Gráfica NVIDIA® GeForce GT 630
- Sistema Ubuntu Linux 14.04 (x86_64)

As especificações da Placa Gráfica NVIDIA GT630 são apresentadas na Tabela 1:

Tabela 1. Especificações da Placa Gráfica Utilizada na Equação de Laplace

Especificação	Valor
Multiprocessadores	2
CUDA Cores	96
GPU Clock	1400 MHz
Quantidade de memória Global	4096 MB
CUDA Capability	2.1

Foram feitas as configurações iniciais em uma rede Gigabit Ethernet e o *daemon* do rCUDA foi iniciado nos computadores designados a disponibilizar as GPUs. Algumas variáveis de ambiente foram definidas no computador designado a acessar as GPUs remotas, desabilitando o acesso à GPU local. Como resultado, foi obtida uma arquitetura multiGPU baseada nas GPUs remotas disponibilizadas. Como procedimento de verificação, foi executado o programa *deviceQuery*, que detecta GPUs CUDA. O programa

identificou a disponibilidade dos dispositivos compatíveis. A Figura 4 apresenta um diagrama com a representação da arquitetura proposta.

Foi elaborado, também, um algoritmo que aplica um estêncil sobre uma matriz quadrada de determinada ordem, repetindo-o por 1000 iterações. A ordem da matriz quadrada variou entre os valores 1024, 2048 e 4096, a fim de observar o comportamento da aplicação a medida em que é aumentado o tamanho do problema. Com o objetivo de avaliar a variação do desempenho em função da intensidade computacional, foram utilizados três formatos de estênceis: Um com 3, outro com 5 e outro com 9 pontos.

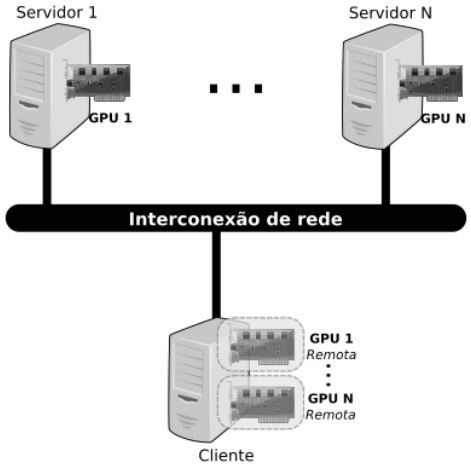


Figura 4. Representação do ambiente rCUDA proposto

6. Resultados Experimentais

Os resultados experimentais obtidos são apresentados por meio das Tabelas 2, 3 e 4. As colunas "M" e "DP", referenciam, respectivamente o valor médio e o desvio padrão obtidos para cada conjunto de resultados. As séries de dados identificadas como "Serial O3" referem-se aos resultados obtidos a partir do código serial compilado utilizando as otimizações disponíveis para a *flag* O3 do compilador GCC e para o processador utilizado.

Os tempos de processamento coletados nos testes com estêncil de 3 pontos são apresentados na Tabela 2.

Tabela 2. Tempos de processamento para o Estêncil de 3 pontos (segundos)

	1024		2048		4096	
	M	**DP**	**M**	**DP**	**M**	**DP**
Serial	5,69	0,02	22,83	0,04	91,54	0,04
Serial O3	2,04	0,01	8,16	0,01	36,54	0,01
CUDA	1,01	0,01	3,89	0,02	15,59	0,02
rCUDA 2GPUs	0,57	0,01	2,15	0,01	8,50	0,02
rCUDA 3GPUs	0,56	0,01	2,16	0,01	8,40	0,04

A Tabela 3 apresenta os tempos de processamento obtidos na execução do Estêncil de 5 pontos.

Tabela 3. Tempos de processamento para o Estêncil de 5 pontos (segundos)

	1024		2048		4096	
	M	**DP**	**M**	**DP**	**M**	**DP**
Serial	9,15	0,03	36,83	0,05	146,75	0,05
Serial O3	8,73	0,07	34,63	0,01	138,95	0,02
CUDA	1,33	0,01	5,26	0,02	21,35	0,03
rCUDA 2GPUs	0,77	0,01	2,99	0,01	11,98	0,03
rCUDA 3GPUs	0,67	0,01	2,51	0,01	9,53	0,05

Na Tabela 4 são mostrados os tempos de processamento obtidos na execução do estêncil de 9 pontos.

Tabela 4. Tempos de processamento para o Estêncil de 9 pontos (segundos)

	1024		2048		**4096**	
	M	**DP**	**M**	**DP**	**M**	**DP**
Serial	13,85	0,01	55,75	0,07	222,58	0,05
Serial O3	10,55	0,01	42,31	0,03	170,28	0,04
CUDA	2,08	0,01	8,26	0,02	33,25	0,02
rCUDA 2GPUs	1,15	0,01	4,48	0,01	17,94	0,02
rCUDA 3GPUs	0,93	0,01	3,52	0,01	13,53	0,03

O valores de *speedup* correspondentes, obtidos para os estênceis de 3, 5 e 9 pontos, em relação às versões seriais otimizadas, são respectivamente apresentados por meio dos gráficos da Figura 5, da Figura 6 e da Figura 7.

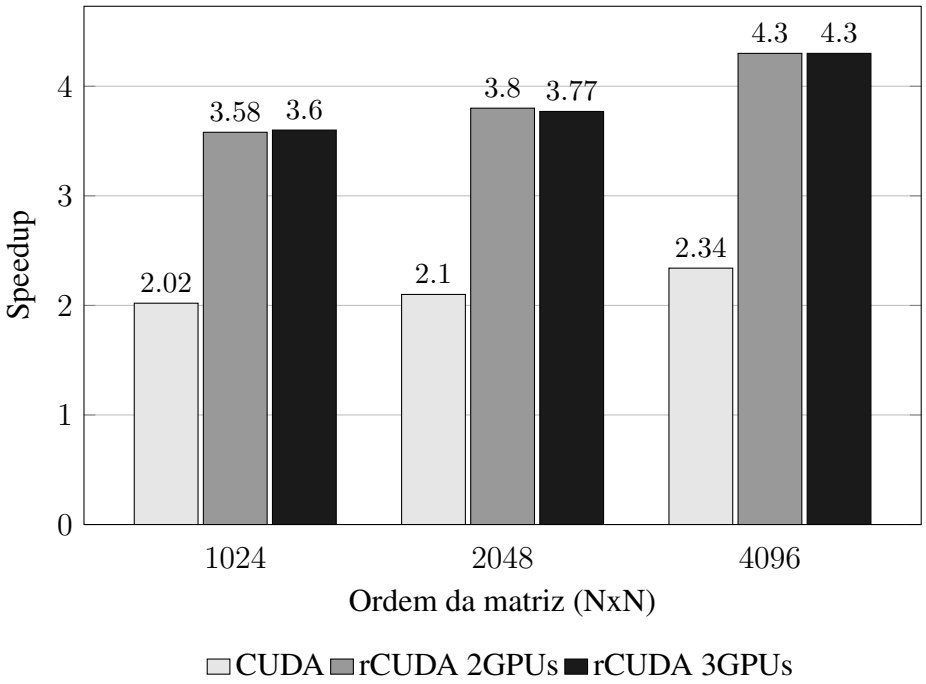


Figura 5. Valores de *speedup* CUDA e rCUDA para as aplicações de Estêncil (3 pontos)

A análise das informações permite observar um ganho constante de desempenho das versões CUDA a medida em que o tamanho do problema é aumentado.

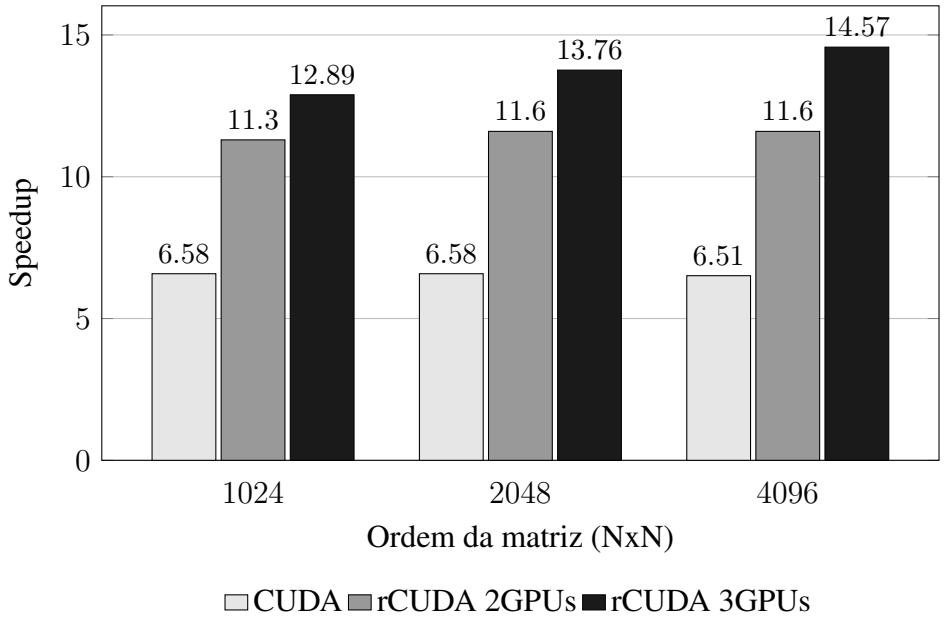


Figura 6. Valores de *speedup* CUDA e rCUDA para as aplicações de Estêncil (5 pontos)

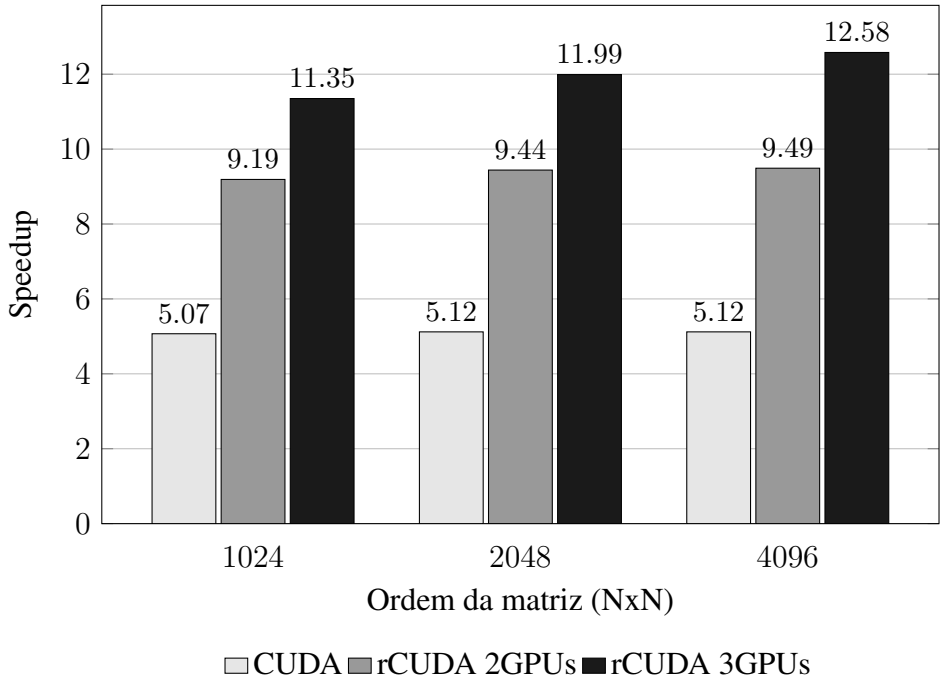


Figura 7. Valores de *speedup* CUDA e rCUDA para as aplicações de Estêncil (9 pontos)

O aumento do formato do estêncil de 3 para 5 pontos implicou na ampliação do *speedup* obtido pelas versões CUDA, porém o mesmo não foi observado após o aumento para 9 pontos, indicando a necessidade de otimizações no *kernel* CUDA, de modo a promover o reuso das informações entre as *threads*.

Observa-se também uma redução entre 54 e 58% do tempo de processamento ao dobrar a quantidade de placas por meio do rCUDA. Ao utilizar três GPUs foi possível obter valores de *speedup* superiores a 12.

Outro fator importante a ser observado foi que a utilização das múltiplas placas requereu, além do particionamento do domínio do problema, apenas a inserção de chamadas ao método *cudaSetDevice()* e definição de *streams* paralelos para a execução simultânea entre as GPUs.

7. Conclusões

A partir dos resultados obtidos percebe-se que a utilização de um sistema multiGPU permitiu a execução do processamento em menor tempo, indicando, na aplicação utilizada, a vantagem do uso da técnica.

Foi possível perceber também o aumento relativo da eficiência no uso da GPU ao aumentar a intensidade computacional do problema: Ao aumentar o estêncil de 3 para 5 pontos, o aumento de desempenho da versão serial otimizada foi proporcionalmente menor, enquanto as versões CUDA apresentaram uma redução constante no tempo de processamento, o que resultou em valores de *speedup* ainda maiores na versão com o estêncil de 5 pontos, mesmo considerando que a implementação CUDA não utilizou qualquer tipo de otimização, como o uso de *Shared Memory*.

Os valores de *speedup* foram compatíveis com o das versões não otimizadas encontradas em outros casos na literatura consultada. Não foram obtidas métricas específicas sobre a utilização da GPU, pois a utilização de ferramentas de *GPU profiling* não foram objeto de estudo deste trabalho.

Com os resultados obtidos no presente trabalho foi possível explorar, por meio do rCUDA, a possibilidade de executar códigos multiGPU associando as GPUs de múltiplas máquinas a um único cliente, contornando a limitação de uma única placa, comumente encontrada em equipamentos de baixo custo.

Ao aumentar a quantidade de GPUs foi possível observar uma redução no tempo de processamento. O desempenho obtido com a disponibilização das GPUs por meio da arquitetura de rede viabilizada pelo rCUDA indica um baixo *overhead* de comunicação, sugerindo a possibilidade de ganhos de desempenho constantes ao utilizar a técnica para aumento de escala de aplicações.

Como trabalho futuro, pretende-se analisar o comportamento do desempenho ao usar um número maior de GPUs, a fim de investigar os limites de escalabilidade da técnica apresentada.

Um outro trabalho futuro possível consiste em efetuar uma comparação dos resultados obtidos no presente estudo com outras técnicas de computação multiGPU, como o uso de sistemas com múltiplas placas ou implementações utilizando MPI.

Referências

Casanova, H., Legrand, A., and Robert, Y. (2008). *Parallel Algorithms*. Chapman & Hall/CRC Numerical Analysis and Scientific Computing Series. CRC Press.

Cecilia, J. M., García, J. M., and Ujaldón, M. (2012). Cuda 2d stencil computations for the jacobi method. In *Proceedings of the 10th International Conference on Applied Parallel and Scientific Computing - Volume Part I*, PARA'10, pages 173–183, Berlin, Heidelberg. Springer-Verlag.

Cook, S. (2013). *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Applications of GPU Computing Series. Morgan Kaufmann.

Duato, J., Pena, A. J., Silla, F., Mayo, R., and Quintana-Orti, E. S. (2011). Performance of cuda virtualized remote gpus in high performance clusters. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 365–374. IEEE.

Hwu, W. (2015). *Heterogeneous System Architecture: A new compute platform infrastructure*. Elsevier Science.

Lorenzoni, R. K., Serpa, M. S., Panetta, J., Navaux, P. O. A., and Méhaut, J.-F. (2017). Otimizando o uso do subsistema de memória de gpus para aplicações baseadas em estênceis. In *XVI Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, pages 1768–1773. SBC.

NVIDIA (2013). Cuda toolkit documentation. http://docs.nvidia.com/cuda/.

Rahman, R. (2013). *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. The expert's voice in microprocessors. Apress.

Rauber, T. and Rünger, G. (2013). *Parallel Programming: For Multicore and Cluster Systems*. Springer Berlin Heidelberg.

rCUDA Team (2014). *rCUDA 4.2 User's Guide*. Universitat Politècnica de València y Universitat Jaume I.

rCUDA Team (2016). rcuda. http://www.rcuda.net.

Reaño, C., Mayo, R., Quintana-Ortí, E. S., Silla, F., Duato, J., and Peña, A. J. (2013). Influence of infiniband fdr on the performance of remote gpu virtualization. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8. IEEE.

Reaño, C., Silla, F., Castelló, A., Peña, A. J., Mayo, R., Quintana-Ortí, E. S., and Duato, J. (2015). Improving the user experience of the rcuda remote gpu virtualization framework. *Concurrency and Computation: Practice and Experience*, 27(14):3746–3770.

Reyes, R., López-Rodríguez, I., Fumero, J. J., and de Sande, F. (2012). accull: An openacc implementation with cuda and opencl support. In Kaklamanis, C., Papatheodorou, T. S., and Spirakis, P. G., editors, *Euro-Par*, volume 7484 of *Lecture Notes in Computer Science*, pages 871–882. Springer.

Santos, A. F., Júnior, P. G. L., Fernandes, S. F. L., and Sadok, D. F. H. (2013). Explorando o processamento paralelo na classificação de tráfego em redes de alta velocidade. In de Computação (SBC), S. B., editor, *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 47–60.

Wang, Z. J. (2014). High-order computational fluid dynamics tools for aircraft design. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 372(2022).

Yang, L. and Guo, M. (2005). *High-Performance Computing: Paradigm and Infrastructure*. Wiley Series on Parallel and Distributed Computing. Wiley.