Implementação MPIC++ e HPX dos Kernels NPB

Ricardo Leonarczyk[1], **Dalvan Griebler**[1,2]

[1] Laboratório de Pesquisas Avançadas para Computação em Nuvem (LARCC)
Faculdade Três de Maio (SETREM), Três de Maio, RS, Brasil

[2]Escola Politécnica, Grupo de Modelagem de Aplicações Paralelas (GMAP),
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, Brasil

ricardo.leonarczyk95@gmail.com,dalvan.griebler@pucrs.br

Resumo. *Este artigo apresenta a implementação paralela dos cinco kernels pertencentes ao NAS Parallel Benchmarks (NPB) com MPIC++ e HPX para execução em arquiteturas de cluster. Os resultados demonstraram que o modelo de programação HPX pode ser mais eficiente do que MPIC++ em algoritmos tais como transformada rápida de Fourier, ordenação e Gradiente Conjugado.*

1. Introdução

A suíte de *benchmarks* paralelos da NAS (NPB) é amplamente utilizada para representar aplicações científicas provenientes da área da dinâmica dos fluidos computacional [Bailey et al. 1991]. A utilização destes *benchmarks* na literatura vai desde a avaliação de supercomputadores até a implementação de técnicas de paralelismo. Implementações oficiais dos *benchmarks* são disponibilizadas tendo Fortran77 como linguagem de programação predominante, junto à bibliotecas como OpenMP e MPI. Portar a versão MPI dos NPB para C++ facilita a posterior avaliação de outras bibliotecas distribuídas para esta linguagem, pois provê um ponto de partida para a paralelização. Além disso, torna-se possível fazer comparações com o padrão de fato da indústria (MPI) em C++. Não foi possível encontrar na literatura trabalhos que houvessem portado *benchmarks* da NAS para C++ utilizando MPI diretamente.

Publicações sobre programação distribuída nos NPB em C++ tendem a dar maior ênfase ao modelo PGAS (*Partitioned Global Address Space*) devido a este prover abstrações que facilitam a paralelização de programas, ao passo que exibe um desempenho similar ao modelo de troca de mensagens. O trabalho de [Fürlinger et al. 2016] apresentou uma versão do *benchmark* NAS DT paralelizado com a biblioteca DASH, a qual fornece estruturas de dados distribuídas e algoritmos paralelos através de uma interface familiar a programadores de C++ (na versão 11 ou superior da linguagem). Já o trabalho de [Sakae and Matsuoka 2001] paralelizou os *benchmarks* NAS CG e IS com MPC++, uma extenção PGAS da linguagem C++ provendo recursos para programação em memória compartilhada e distribuída. Ambos os trabalhos reportaram ter atingido um desempenho maior com as operações unilaterais do PGAS, em contraste com as operações bilaterais e bloqueantes (*send/receive*) provindas do MPI na versão 1. No entanto, ainda há modelos de programação que merecem ser explorados em novas implementações dos NPB, devido a terem anteriormente demonstrado resultados promissores em avaliações com outros *benchmarks*. Entre estes está o modelo AGAS (Active Global Address Space), implementado pela biblioteca HPX.

O HPX é um ambiente de execução paralelo para C++ que provê uma interface de programação homogênea, assíncrona e baseada em tarefas para promover o paralelismo

em memória compartilhada e distribuída [Kaiser et al. 2014]. Uma aplicação que utiliza HPX pode ser estruturada em tarefas relacionadas por um grafo de dependências. O grafo é executado conforme a disponibilidade dos recursos computacionais. Quando em ambiente distribuído, é possível manter objetos globalmente acessíveis cujos métodos são invocados remotamente e de forma assíncrona. A criação de uma versão HPX dos NPB provê ao ecossistema da referida biblioteca benchmarks amplamente reconhecidos pela sua eficácia em medir arquiteturas paralelas, desta forma possibilitando avaliar se através do HPX é possível obter ganhos de desempenho em aplicações representadas pelos NPB. Uma vantagem importante que o HPX oferece e que o diferencia das bibliotecas PGAS previamente discutidas é a possibilidade de, transparentemente ao código, ajustar a carga de trabalho dinamicamente entre os nós. No entanto, tais recursos não foram utilizados para esse artigo. Deste modo, nossa abordagem pode ser considerada semelhante à dos trabalhos PGAS no uso de operações assíncronas unilaterais, que no HPX convertem-se em chamadas remotas de métodos.

Neste artigo são apresentadas duas novas versões dos NPB, nas quais utilizam-se (respectivamente) as bibliotecas C++ MPI e HPX para arquiteturas de memória distribuída. Os *kernels* paralelizados foram CG, EP, IS, FT e MG, sendo os três primeiros inicialmente portados para MPIC++ por [Leonarczyk and Griebler 2020]. Para os experimentos compara-se a versão MPI original (em Fortran/C) com a versão MPI C++, e a versão MPI C++ com a versão HPX. O restante do artigo está subdividido em três seções. Na Seção 2 a implementação é descrita, na Seção 3 os resultados são discutidos, e conclui-se o artigo com a seção 4.

2. Implementação

O trabalho de implementação foi realizado em duas etapas. Na primeira os 5 *kernels* foram portados de sua versão Fortran MPI para C++ MPI. Para agilizar este processo a versão C++ sequencial dos NPB provida por [Griebler et al. 2018] foi utilizada como ponto de partida, restando então a adição das chamadas MPI e outras reestruturações necessárias ao código para introduzir o paradigma distribuído.

Na segunda etapa, a versão MPIC++ foi utilizada como base para realizar a paralelização com a biblioteca HPX. Procurou-se manter as estratégias de paralelismo presentes na versão MPI através do uso de recursos equivalentes no HPX. Devido à natureza assíncrona do HPX, barreiras foram usadas para aproximar as operações MPI. Porém as barreiras HPX utilizadas são globais, portanto seu uso não é equivalente as barreiras implícitas existentes na comunicação de processos MPI. Pares *send/receive* foram convertidos em chamadas remotas, nas quais um nó x invoca um método no nó y, e x bloqueia a execução até que o resultado seja recebido. Ao receber a chamada, o nó y então retorna uma *future* pela qual é possível aguardar o resultado e recuperá-lo uma vez que tenha sido computado. Como não é possível saber em que momento um método será chamado, foi preciso armazenar as *futures* e seus resultados para retornar a qualquer momento entre duas barreiras globais. As operações coletivas do MPI utilizadas nos NPB possuem versões alternativas no HPX, não sendo necessário implementá-las manualmente. No entanto, verificamos que o uso do `alltoall` realizava cópias desnecessárias de parâmetros, e por isso estas coletivas foram implementadas utilizando chamadas remotas.

A respeito especificamente das implementações dos benchmarks, CG resolve uma equação linear pelo uso do algoritmo distribuído do Gradiente Conjugado, no qual diversas reduções são realizadas em uma matriz particionada entre processos utilizando o

padrão `MPI_Irecv`, `MPI_Send` e `MPI_Wait` (padrão expressado por chamadas remotas em HPX). EP exercita a computação de ponto flutuante através da geração de pares de desvios gaussianos aleatórios em cada processo, seguido por reduções de soma realizadas globalmente por coletivas. IS realiza o ordenamento distribuído de um vetor de números aleatórios, estressando a comunicação por meio de coletivas `alltoall` e `alltoallv`. FT aplica o algoritmo da transformada rápida de Fourier para resolver uma equação diferencial 3D, requerendo diversas cópias de vetores e transposições globais operadas por coletivas `alltoall`. MG aproxima uma solução para a equação discreta de Poisson em uma grade 3D, empregando comunicação no padrão *stencil* com chamadas MPI (e HPX) realizadas da mesma forma como descrito para o CG.

3. Resultados

Os experimentos foram executados em um *cluster* de 4 nós, cada qual equipado com 2 processadores AMD Opteron(tm) de 6 núcleos físicos a 2.1GHz com suporte a *hyperthreading* desabilitado, 32GB de memória RAM e tendo o Ubuntu Server 18.04 como sistema operacional. Cada nó do *cluster* possui agregação de 4 links (modo *bonding round-robin*) Gigabit Ethernet (largura de banda teórica é de 4Gb). O mapeamento dos processos MPI foi em *round robin*, no qual sempre um processo é atribuído a um nó diferente. Assim, sempre mais de um nó (e por consequência a rede) é utilizado. O *speed-up* teve como base a versão sequencial dos *benchmarks* originais e da versão C++ de [Griebler et al. 2018]. A versão sequencial para Fortran foi obtida através da compilação da versão OpenMP com o OpenMP desativado, como sugerido pela NAS para o NPB 3.4. Para o cálculo do *speed-up* foi considerada a média aritmética de um conjunto de 40 execuções para cada combinação de *benchmark*, classe e número de processos. Devido à limitações de espaço, o *speed-up* não é mostrado na tabela. É possível obtê-lo através dos valores das médias de tempo. Os *benchmarks* foram compilados com GFortran, GCC e G++, na versão 7.4 e com nível de otimização 3 (*flag* -O3). As bibliotecas utilizadas foram OpenMPI-2.1.1 e HPX-1.4.1. O algoritmo para geração de números aleatórios utilizado nos experimentos está presente na função `randdp` na suíte NPB. Utilizou-se nos experimentos as classes A, B e C, porém no artigo apenas é mostrado o resultado para a classe C, por esta possuir uma carga de trabalho maior. Os resultados do tempo de execução em segundos podem ser observados na Tabela 1. Observou-se em média um desvio padrão inferior a 1.1.

No CG a versão MPIC++ obteve tempos de execução maiores do que a versão MPIFortran em todos os casos. A maior diferença foi de 4,82s (1,85%) com 2 processos, e a menor foi de 0,25s (0,08%) com 16 processos. A versão HPX do CG apresentou o menor desempenho com 2 processos, requerendo cerca de 20s a mais para executar do que o MPIC++. Melhores resultados começam a aparecer com números de processos maiores. Com 16 processos o HPX alcançou um speed-up de 6,47 enquanto o MPIC++ atingiu 2,13, resultando em cerca de 200 segundos de diferença. No EP o MPIC++ exibiu a mesma tendência de sua versão sequencial, com um tempo de execução 1% menor que a versão Fortran. Os tempos do EP-HPX são comparáveis aos da versão MPIC++, exceto quando executado com 32 processos. As versões FT-MPI são comparáveis em tempo de execução, porém o speed-up foi sempre maior para o C++ devido às diferenças entre as versões sequenciais. O HPX apresentou menores tempos de execução do que C++-MPI para todos os casos exceto com 2 processos. A maior diferença foi obtida com 32 processos, em que o HPX executou em 22% do tempo do MPIC++. As versões MPI do IS foram as mais semelhantes em tempo de execução, o que já era esperado pelo IS ser escrito na linguagem C. A versão IS-HPX comportou-se de forma semelhante ao FT-

HPX, devido a comunicação se concentrar no uso de coletivas `all-to-all`. A maior diferença foi obtida com 16 processos, em que o HPX executou em 23% do tempo do MPIC++. MG-MPIC++ obteve um tempo de execução semelhante à versão Fortran. A maior diferença de tempo alcançada foi de 0,57s (0,7%) com dois processos. Devido ao uso intenso de barreiras globais na versão HPX, o MG-HPX foi o benchmark que obteve o menor desempenho em comparação com o C++. O particionamento incorreto de um vetor impediu que MG-HPX executasse com números de processos maiores que 8 neste experimento.

Tabela 1. Resultados dos experimentos usando a classe C. São apresentados a média aritmética do tempo de 40 execuções.

	CG			EP			FT			IS			MG		
Seq.	575.53	632.6	-	976.51	965.89	-	385.43	1034.31	-	63.8	64.51	-	131.47	135.97	-
Proc.	FORT	CPP	HPX	FORT	CPP	HPX	FORT	CPP	HPX	FORT	CPP	HPX	FORT	CPP	HPX
2	254.36	259.18	279.04	488.5	481.29	482.68	302.68	324.61	505.8	20.28	20.29	34.72	77.61	77.04	132.78
4	240.07	242.0	286.74	244.47	240.81	241.46	584.7	591.8	312.33	60.0	60.52	17.14	49.16	49.05	176.7
8	197.43	198.62	159.57	122.41	120.82	120.99	425.85	430.85	170.48	47.7	47.68	10.89	28.32	28.61	96.57
16	297.06	297.31	97.75	61.37	60.5	60.59	396.73	399.2	114.93	46.23	46.15	10.65	-	-	-
32	265.48	265.93	120.19	30.86	30.49	43.14	380.15	381.38	84.51	45.4	45.4	18.49	-	-	-

4. Conclusões

Este trabalho contribuiu com a paralelização dos cinco kernels do NPB usando os modelos de programação MPI e HPX em C++. Como trabalho futuro, além de executar experimentos em outros clusters, pretende-se paralelizar as pseudo-aplicações e avaliar outros modelos de programação paralela distribuída na suíte NPB.

Agradecimentos Os autores agradecem ao Laboratório de Pesquisa Avançada em Computação em Nuvem (LARCC/SETREM, Brasil) por fornecer recursos de computação em nuvem, que contribuíram para os resultados das pesquisas relatadas neste artigo.

Referências

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., et al. (1991). The NAS parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73.

Fürlinger, K., Fuchs, T., and Kowalewski, R. (2016). Dash: A c++ pgas library for distributed data structures and parallel algorithms. In *IEEE 18th Intern. Conf. on High Performance Computing and Communications*, pages 983–990. IEEE.

Griebler, D., Loff, J., Mencagli, G., Danelutto, M., and Fernandes, L. G. (2018). Efficient NAS benchmark kernels with C++ parallel programming. In *26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 733–740. IEEE.

Kaiser, H., Heller, T., Adelstein-Lelbach, B., Serio, A., and Fey, D. (2014). Hpx: A task based programming model in a global address space. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, page 6. ACM.

Leonarczyk, R. and Griebler, D. (2020). Implementação MPIC++ dos kernels NPB EP, IS e CG. In *20th Escola Regional de Alto Desempenho da Região Sul (ERAD-RS)*, pages 101–104, Santa Maria, RS, Brazil. Sociedade Brasileira de Computação.

Sakae, Y. and Matsuoka, S. (2001). MPC++ Performance for Commodity Clustering. In *International Conference on High-Performance Computing and Networking*, pages 503–512. Springer.