Paralelização do NAS-PB usando Do Concurrent*

Anna V. G. Marciano†, Artur dos Santos Antunes, Claudio Schepke

[1]Laboratório de Estudos Avançados em Computação (LEA)
Universidade Federal do Pampa (UNIPAMPA) – Alegrete – RS – Brazil

{annamarciano,arturantures}.aluno@unipampa.edu.br
{claudioschepke}@unipampa.edu.br

Resumo. *Neste artigo, exploramos e comparamos o paralelismo nativo de FORTRAN com as diretivas fornecidas pela interface de programação paralela OpenMP em algoritmos do NAS Parallel Benchmark (CG, FT e MG). Os resultados mostram que Do Concurrent pode produzir código paralelo de CPU com compatibilidade numérica e resultados de desempenho semelhantes aos laços paralelos de OpenMP.*

1. Introdução

Uma Interface de Programação Paralela pode ser uma extensão de linguagem de programação, biblioteca, *framework* ou linguagem específica de domínio, que oferece recursos para criação e instanciação de *threads*. Se a execução distribuída for incluída, há ainda outras alternativas, como Interface de Programação de Aplicativos, Processos Sequenciais de Comunicação e Espaço de Endereço Global Particionado. Ferramentas de programação paralela beneficiam diferentes tipos de aplicativos ao permitir a geração de código por meio de processos, *threads*, aceleradores ou instruções vetoriais [Vetter 2013].

Por outro lado, a ideia de ter uma linguagem específica para programação simultânea para qualquer classe de aplicativos não teve sucesso [Ozen 2018]. Por exemplo, a tentativa de definir a extensão Fortran de Alto Desempenho (HPF [Koelbel et al. 1993]) como um padrão de linguagem de programação paralela acabou. A maioria dos programadores preferiu seguir em direção a uma linha de programação orientada a OpenMP e não usar HPF como uma linguagem de programação paralela [Kennedy et al. 2007]. Além disso, especificações FORTRAN sucessivas incorporam algumas das ideias do HPF.

Desde a especificação Fortran 2008, e pelo aumento de especificadores de localidade em 2018 [Reid 2018], é possível criar simultaneidade em loops pelas palavras reservadas da linguagem Do Concurrent. Assim, um compilador gera paralelismo de loop sem nenhuma modificação no código-fonte. A ideia é distinta do paralelismo fornecido por uma abordagem de diretivas pré-compiladas, na qual um código-fonte instrumentado (pragmas), em uma primeira etapa de compilação, inclui rotinas de thread para a compilação subsequente, como ocorre nos PPIs OpenMP e OpenACC.

Este artigo investiga o impacto do paralelismo da cláusula Do Concurrent da linguagem Fortran contra o paralelismo PPI tradicional por meio de diretivas de compilação em 3 algoritmos do NAS Parallel Benchmarks (Gradiente Conjugado, Multi-Grid e Transformada Rápida de Fourier). Queremos ver se Do Concurrent pode substituir diretivas e

*Trabalho parcialmente financiado pelo Edital CNPq: Projeto 135963/2023-0.
†Bolsista PIBIC/CNPq 2023/2024.

manter o mesmo desempenho. As contribuições deste artigo são: (a) Avaliar o paralelismo nativo de Fortran com abordagens de diretivas paralelas e (b) Apresentar as etapas necessárias e as modificações no código para acelerá-lo com essas abordagens.

2. NAS Parallel Benchmark

NAS Parallel Benchmarks (NPB) é uma coleção de algoritmos destinados a calcular o desempenho de supercomputadores paralelos. Originários de aplicativos de Dinâmica de Fluidos Computacional (DFC), esses benchmarks simulam características de computação e movimentação de dados para aplicativos DFC. O conjunto inicial, NPB 1, inclui cinco kernels computacionais e três pseudo-aplicativos, cada um representando um aspecto diferente dos cálculos DFC [Löff et al. 2021]. Neste trabalho, avaliamos três desses aplicativos (CG, MG, FT), que serão abordados a seguir [Maliszewski et al. 2018]:

- **CG (Conjugate Gradient):** Avalia o acesso irregular à memória e os padrões de comunicação, usando uma técnica de simulação de sistemas lineares.
- **MG (Multi-Grid):** Avalia o desempenho do método multi-grid envolvendo comunicação de longa e curta distância, enfatizando o uso da memória.
- **FT (Fast Fourier Transform):** Avalia a comunicação entre todos os fluxos concorrentes para a resolução de uma equação diferencial parcial.

Esses benchmarks são organizados em oito classes com base no tamanho do problema, cada uma aumentando progressivamente, sendo elas: tamanho pequeno(para avaliações rápidas), tamanho da estação de trabalho(indicativo das capacidades dessas estações dos anos 1990), problemas de teste padrão(onde cada classe é, aproximadamente, quatro vezes maior que sua antecessora) e problemas de teste em larga escala(onde cada classe é cerca de dezesseis vezes maior que a anterior).

3. Metodologia

A metodologia consiste em melhorar paralelamente o código e avaliar o desempenho da execução das implementações [Tremarin et al. 2024]. Os testes medem os resultados numéricos de saída e o tempo de execução. Compilamos o código com pgf90 (Nvfortran), usando o kit de ferramentas HPC_SDK 23.5 NVidia, com os seguintes sinalizadores: `-O3` para otimização sequencial, `Minfo=all` para diagnósticos detalhados do compilador e `-fopenmp` para paralelismo OpenMP. Executamos 20 vezes cada caso, usando a classe C para testes. O desvio padrão obtido foi insignificante.

A composição do ambiente computacional usado neste trabalho para executar os testes é de dois processadores Intel Xeon CPU E5-2420 de seis núcleos (1,90 GHz) e uma GPU Nvidia TITAN V. Ubuntu 22.04.1 é o sistema operacional com GNU/Linux versão 5.19.0-1025-oracle. A versão CUDA é 12.1 e a versão do driver é 525.105.17. Para automatizar o processo experimental, implementamos um shell script que compila e executa os algoritmos em um laço, gerando os arquivos de saída para cada execução. Uma vez que o laço de execução é concluído, um script Python é acionado para processar os arquivos de saída, extraindo os resultados para calcular a média e o desvio padrão.

4. Implementação

Os laços que usavam estruturas como `OMP PRIVATE` foram substituidos pelo especificador de localidade `Local` de Do Concurrent nos algoritmos de NPB. O Algoritmo 1

Algoritmo 1: Trecho de código do kernel FT demonstrando o uso da cláusula `private`

```
!$omp parallel do default(shared) private(i,j,k) collapse(2)
do k = 1, d3
 do j = 1, d2
    do i = 1, d1
       u0(i,j,k) = 0.d0
       u1(i,j,k) = 0.d0
       twiddle(i,j,k) = 0.d0
    end do
 end do
end do
```

Algoritmo 2: Trecho de código do kernel FT demonstrando o uso do especificador de localidade `local` de Do Concurrent

```
do concurrent (k = 1 : d3) local (i, j)
 do j = 1, d2
    do i = 1, d1
       u0(i,j,k) = 0.d0
       u1(i,j,k) = 0.d0
       twiddle(i,j,k) = 0.d0
    end do
 end do
end do
```

apresenta um exemplo ilustrando o uso da cláusula privada de OpenMP, enquanto o Algoritmo 2 fornece a mesma implementação adaptada para `Do Concurrent`. Alguns loops que foram paralelizados com OpenMP não foram modificados porque continham chamadas de função dentro de seus escopos. A construção `Do Concurrent` ainda não lida com tais cenários de forma eficaz, pois luta com chamadas de função que podem introduzir efeitos colaterais ou dependências, limitando sua aplicabilidade nessas situações.

5. Resultados

A Figura 1 mostra os resultados de tempo de execução dos algoritmos CG, FT e MG para as versões OpenMP e Do Concurrent. As diferenças percentuais médias observadas foram inferiores a 5%, o que indica que o Do Concurrent pode ser uma alternativa viável. Entretanto, verificamos que aproximadamente 15% dos loops permaneceram sem conversão devido a chamadas de funções, o que limita a aplicabilidade geral da abordagem.

6. Considerações Finais

Do Concurrent é uma alternativa para automatizar o paralelismo de uma aplicação HPC, presente no padrão FORTRAN ISO 2018; além do mais, é uma abordagem promissora para paralelismo em Fortran. O código-fonte torna-se mais limpo porque não há necessidade de colocar diretivas de pré-compilação. Além disso, a avaliação dos resultados usando o NAS Parallel Benchmarks (NPB) destacou que, embora a construção Do Concurrent ainda não seja tão avançada quanto o OpenMP em termos de otimização e fle-

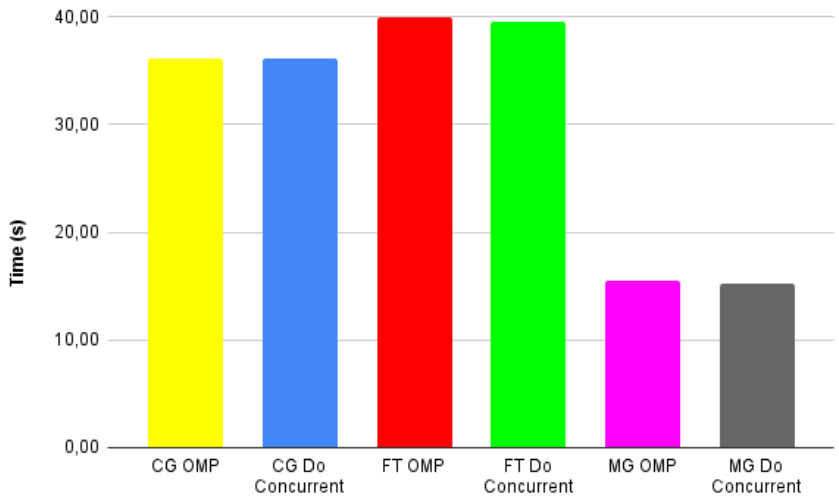

Figura 1. Tempo de execução das implementações

xibilidade, ela demonstrou desempenho paralelo competitivo; porém, ainda há desafios com relação ao suporte da chamada de funções, o que torna o estudo um pouco menos aprofundado. Como abordagem futura, é indispensável a inclusão de benchmarks mais complexos e explorar mais as classes do NAS-PB.

Referências

Kennedy, K., Koelbel, C., and Zima, H. (2007). The Rise and Fall of High Performance Fortran: An Historical Object Lesson. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, HOPL III, page 7–1–7–22, New York, NY, USA. Association for Computing Machinery.

Koelbel, C. H., Loveman, D., Schreiber, R. S., Jr., G. L. S., and Zosel, M. (1993). *High Performance Fortran Handbook*. The MIT Press.

Löff, J., Griebler, D., Mencagli, G., Araujo, G., Torquati, M., Danelutto, M., and Fernandes, L. G. (2021). The NAS Parallel Benchmarks for evaluating C++ parallel programming frameworks on shared-memory architectures. *Future Generation Computer Systems*, 125:743–757.

Maliszewski, A. M., Griebler, D., and Schepke, C. (2018). Desempenho em Instâncias LXC e KVM de Nuvem Privada usando Aplicações Científicas. In *Anais da XVIII ERAD/RS*, Porto Alegre, RS, Brasil. SBC.

Ozen, G. (2018). *Compiler and runtime based parallelization & optimization for GPUs*. PhD thesis, Computer Architecture Department - Universitat Politècnica de Catalunya.

Reid, J. (2018). The New Features of Fortran 2018. *SIGPLAN Fortran Forum*, 37(1):5–43.

Tremarin, G. D., Marciano, A. V. G., Schepke, C., and Vogel, A. (2024). Fortran do concurrent evaluation in multi-core for nas-pb conjugate gradient and a porous media application. In *SSCAD*, pages 133–143. SBC.

Vetter, J. S. (2013). *Contemporary High Performance Computing: from Petascale Toward Exascale*. CRC Press.