Uso de Rotinas StarPU em Simulação de Secagem de Grãos*

Hígor Uélinton da Silva, Claudio Schepke

[1]Ciência da Computação – Universidade Federal do Pampa (UNIPAMPA)
Alegrete – RS – Brazil

{higorsilva.aluno,claudioschepke}@unipampa.edu.br

Resumo. *O foco deste trabalho é achar um modo para acelerar o tempo de processamento de uma aplicação de simulação de secagem de grãos utilizando uma abordagem orientada a tarefas. Para tal, optou-se pelo uso da interface de programação paralela StarPU. Como resultados obtidos foi possível gerado um código paralelo, mas redução no tempo de execução.*

1. Introdução

A secagem é um processo feito para retirar a umidade de um material na produção de alimentos. A secagem do grão consiste na remoção da umidade do núcleo até que o teor de umidade seguro seja geralmente 12-14% em base úmida [Bala 2017].

Existem alguns tipos de secadores artificiais que são usados na produção de grãos. Entre eles está o silo de secagem. O silo de secagem é uma estrutura cilíndrica com piso perfurado, preenchido por um espalhador de grãos, uma unidade de ventilação quente e recursos para varrer e descarregar os grãos sob o piso.A ventoinha do aquecedor inicia quando os grãos são colocados dentro do silo e, desde que não atinja o menor teor de umidade médio dos grãos, o processo não termina [Krokida et al. 2006]. Todo esse processo de secagem em leito fixo apresenta diferentes zonas de secagem: camada seca, camada de secagem e camada úmida, que devem ser consideradas para determinar o período de secagem. A zona seca está localizada no fundo da caixa e é a primeira camada a ser seca. O processo termina quando a zona úmida seca.

Na formulação matemática do problema usou-se a lei de Darcy na parte porosa e Navier-Stokes na parte aberta [Cornelissen 2016]. Deve-se levar em consideração a mudança abrupta no fluxo livre e meio poroso, criando uma zona de transição.

Um esquema Dinâmico de Fluídos Computacional foi implementado em FORTRAN usando o método de Volumes Finitos para simular e computar as soluções numéricas [de Oliveira 2020]. Esse tipo de aplicação demanda muito tempo de execução para qualquer simulação simples. Para reduzir esse tempo à um tempo aceitável, explora-se a simultaneidade das instruções em arquiteturas multicore [Silva et al. 2021].

O presente trabalho tem como objetivo obter um melhor desempenho da aplicação, contribuindo para a redução do seu tempo de execução. Além disso, uma solução melhor permite o cálculo com malhas maiores, isto é, que particionam o domínio de maneira mais fina. Nesse caso, há uma carga de trabalho computacional mais alta, mas o fornecimento de uma simulação mais precisa. Para atingir esse objetivo, adota-se estratégias de paralelização para criar uma versão da aplicação que roda em multicore e explora seus recursos computacionais.

*Bolsista PROBITI/FAPERGS 2021/2022

2. StarPU

StarPU é uma interface baseada no paradigma de programação paralela orientada a tarefas para arquiteturas híbridas. "Ao invés de lidar com problemas de mais baixo-nível, programadores podem se concentrar nos problemas do algoritmo!" [StarPU 2018]. A interface oferece tanto implementações em CPU como em GPU. Para programação em GPU ela fornece suporte as interfaces CUDA e OpenCL. Está disponível em C/C++, mas possui algum suporte a FORTRAN, sendo esse nativo ou usando *marshalling wrappers* C.

O *runtime* da interface lida com: dependência de tarefas; *scheduling* heterogêneo otimizado; transferência de dados e replicação otimizada entre memória principal e memórias específicas; e comunicação em *cluster* otimizado.

3. Metodologia

O código da aplicação, feito em FORTRAN, consiste em etapas de pré-processamento, iteração e pós-processamento [Silva et al. 2022]. Todas essas etapas são chamadas explicitamente na rotina principal. Na etapa de pré-processamento, os arquivos de configuração são lidos, os dados são alocados e as variáveis são inicializadas. A etapa iterativa consiste em um *loop* para o cálculo do pseudo-tempo. Nesse *loop* são chamadas as rotinas `solve_u` e `solve_v` para resolver a equação de momento, `solve_p` para resolver a equação de continuidade, e `solve_z` para resolver a equação de energia. No final do *loop*, a convergência é calculada e algumas variáveis são atualizadas. O pós-processamento consiste na persistência, em arquivos, dos resultados físicos obtidos.

Um estudo de performance preliminar foi conduzido usando a ferramenta Perf. A Tabela 1 apresenta as rotinas que mais demandam tempo na execução da aplicação. As rotinas `solve_u`, `solve_v`, `solve_p`, e `solve_z` são essencialmente usadas para chamar outras rotinas, incluindo as que mais gastam tempo de execução como `resv`, `resu`, `resz`, `upwind_v` e `upwind_u`. O ambiente onde os testes foram executados é apresentado na Tabela 2.

Com base no estudo de performance, decidiu-se paralelizar as 4 rotinas, `solve_u`, `solve_v`, `solve_p`, e `solve_z`, que operam as 3 equações principais da aplicação, momento, continuidade e energia. Cada rotina foi transformada em 4 tarefas, e para

Tabela 1. Análise de performance com Perf

Rotina	% de tempo
`resv`	42,53
`resu`	41,28
`resz`	7,21
`upwind_v`	2,60
`upwind_u`	1,80
`solve_p`	1,25
`solve_u`	1,10
`solve_v`	1,10
`solve_z`	0,45

Tabela 2. Recursos da CPU

Recursos	Xeon E5-2650 ($\times$2)
Frequência	2.00 GHz
Núcleos / *Threads*	8 ($\times$2) / 16 ($\times$2)
Cache L1	32 KB
Cache L2	256 KB
Cache L3	20 MB
Memória RAM	128 GB

cada rotina criou-se um *codelet*. *Codelets* são estrutura do StarPU que armazenam as configurações das tarefas.

A interface não oferece tratamento dos dados de entrada das tarefas de forma automática. As matrizes de entrada cujos dados são alterados dentro das rotinas foram particionadas em 4 partes, divididas horizontalmente, para evitar conflitos de escrita e possibilitar a execução paralela. As matrizes não modificadas nas rotinas não precisaram sofrer particionamento. Com isso, as regiões onde cada tarefa trabalha precisam ser definidas. O controle dessas regiões é manual: linhas iniciais e finais são previamente calculadas para cada tarefa. A paralelização da rotina `solve_u` é apresentada no Código 1.

Código 1. Paralelização da rotina `solve_u` com StarPU

```
! Alocacao do codelet para rotina
codelet = fstarpu_codelet_ALLOCATE()
CALL fstarpu_codelet_add_cpu_func(codelet, C_FUNLOC(solve_U))

! Definindo numero de buffers e adicionando as variaveis ao descritor
nbuffers = 7
DO i=1,4
  desc(i) = fstarpu_data_descr_array_alloc(nbuffers)
  !feito o mesmo com: h_um_tau, h_vm_tau, h_pn
  CALL fstarpu_data_descr_array_set(desc(i), 0, h_um, FSTARPU_R)
  !feito o mesmo com: hp_vm_tau, h_residual_us
  CALL fstarpu_data_descr_array_set(desc(i), 1,
    fstarpu_data_get_sub_data(hp_um_n_tau, 1, (/i-1/)), FSTARPU_RW)
ENDDO

!Resolucao do momento
DO i=1,4
  CALL fstarpu_insert_task((/ codelet, FSTARPU_DATA_MODE_ARRAY, desc(i)
    , C_LOC(nbuffers), C_NULL_PTR /))
ENDDO
CALL fstarpu_task_wait_for_all()
```

Após realizar o tratamento e avaliação dos dados de entrada das rotinas, os descritores de dados foram alocados e as tarefas puderam ser submetidas. Como escalonador das tarefas, `lws` foi utilizado. Após submeter todas as tarefas, um bloqueio/barreira foi criado. Esse bloqueio evita que erros sejam cometidos, como acesso indevido a dados com resultados incompletos e/ou inválidos.

Para medir o tempo de cada rotina foi utilizada a rotina `cpu_time`. Foram realizadas 2000 medições para se obter uma média aritmética do tempo gasto em cada rotina. Com essas medições, também se calculou a variância e desvio padrão dos tempos obtidos.

4. Resultados

Os resultados das rotinas paralelizadas não apresentaram redução de tempo de execução em relação a execução sequencial. Uma das rotinas paralelizadas chegou a ficar 20 vezes mais lenta, sendo a pior rotina em termos de paralelização. Todas essas medições se encontram na Tabela 3.

Os resultados preliminares, embora negativos, precisam ser melhor investigados. Até o momento, somente pode-se supor o motivo dos resultados não estarem bons. Um

Tabela 3. Performance: Tempo, em segundos, variância e desvio padrão da execução para malha 51x63

Rotina	Serial	σ^2	σ	StarPU	σ^2	σ
solve_u	$6{,}81.10^{-4}$	$1{,}55.10^{-9}$	$3{,}94.10^{-5}$	$2{,}77.10^{-3}$	$1{,}30.10^{-5}$	$3{,}61.10^{-3}$
solve_v	$8{,}97.10^{-4}$	$2{,}17.10^{-9}$	$4{,}66.10^{-5}$	$2{,}97.10^{-3}$	$1{,}33.10^{-5}$	$3{,}65.10^{-3}$
solve_p	$3{,}44.10^{-5}$	$5{,}87.10^{-12}$	$2{,}42.10^{-6}$	$9{,}12.10^{-4}$	$8{,}56.10^{-6}$	$2{,}93.10^{-3}$
solve_z	$1{,}80.10^{-4}$	$5{,}31.10^{-9}$	$7{,}29.10^{-5}$	$3{,}62.10^{-3}$	$6{,}60.10^{-6}$	$2{,}57.10^{-3}$

deles é o domínio do problema, onde os cálculos em um ponto da malha necessitam de suas vizinhanças, o que dificulta o particionamento dos dados, e consequentemente, na eficiência das tarefas. Como mencionado anteriormente, algumas matrizes não foram particionadas, tendo-se que controlar as regiões manualmente, podendo ocasionar uma carga de trabalho replicada, gerando custos extras desnecessários. O tamanho do domínio pode afetar o ganho de desempenho, sendo necessário mais testes com malhas maiores.

5. Considerações Finais

Neste trabalho é apresentado uma aplicação de simulação de secagem de grãos. Aplicações deste tipo demandam de um tempo de processamento expressivo. Devido à isso, optou-se por realizar a paralelização da aplicação utilizando a interface StarPU com o intuito de explorar o paralelismo de tarefas.

Como sequência do trabalho, mais estudos sobre o domínio do problema devem ser empregados com o intuito de se obter uma melhor divisão dos dados e, assim, conseguir melhorar o desempenho na paralelização. Além disso, testes com malhas maiores deverão ser feitos.

Referências

Bala, B. K. (2017). *Drying and Storage of Cereal Grains*. Wesley Blackwell.

Cornelissen, P. (2016). *Coupled free-flow and porous media flow: a numerical and experimental investigation*. PhD thesis, Faculty of Geosciences - Utrecht University.

de Oliveira, D. P. (2020). Fluid Flow Through Porous Media with the One Domain Approach: A Simple Model for Grains Drying. Dissertação de mestrado, Universidade Federal do Pampa.

Krokida, M., Marinos-Kouris, D., and Mujumdar, A. S. (2006). *Handbook of industrial drying. Rotary Drying*. Taylor & Francis Group.

Silva, H. U., Lucca, N., Schepke, C., Oliveira, D. P., and Cruz, C. F. (2022). Parallel OpenMP and OpenACC Mixing Layer Simulation. In *Proceedings of Euromicro International Conference on Parallel, Distributed and Network-Based Processing*. IEEE.

Silva, H. U., Schepke, C., Cruz, C. F., Oliveira, D. P., and Lucca, N. (2021). An Efficient Parallel Model for Coupled Open-Porous Medium Problem Applied to Grain Drying Processing. In *Proceedings of Latin America High Performance Computing Conference*.

StarPU (2018). A Unified Runtime System for Heterogeneous Multicore Architectures. https://starpu.gitlabpages.inria.fr/.